NEW METHODS FOR UNDERSTANDING AND CONTROLLING THE SELF-ASSEMBLY OF REACTING SYSTEMS USING COARSE-GRAINED MOLECULAR DYNAMICS

by

Stephen Thomas

A dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Materials Science and Engineering

Boise State University

August 2018

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Stephen Thomas

Dissertation Title: New Methods for Understanding and Controlling the Self-Assembly of Reacting Systems Using Coarse-Grained Molecular Dynamics

Date of Final Oral Examination: 12th August 2018

The following individuals read and discussed the dissertation submitted by student Stephen Thomas, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Eric Jankowski, Ph.D.	Chair, Supervisory Committee
Carla Reynolds, Ph.D.	Member, Supervisory Committee
Peter Mullner, Ph.D.	Member, Supervisory Committee
Scott Phillips, Ph.D.	Member, Supervisory Committee

The final reading approval of the dissertation was granted by Eric Jankowski, Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

dedicated to curiosity, tenacity and our families.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Eric Jankowski for giving me the golden opportunity to work with him and setting me up on a path to success from day one. I want to especially thank him for showing me by example, how to be a good scientist, communicator and a strategist. I am grateful to Dr. Carla Reynolds for guiding me through this project and helping me get the practical perspective on ideas that I came up with. I am also thankful for her for helping me run many of the simulations on the Garcia cluster computer at Boeing and being the first beta tester for `epoxpy`.

I want to thanks Dr. Alex Punnoose for introducing me to the idea of Materials Science in 2006 and then encouraging me to pursue my dream of doing a Ph.D. again in 2012. I consider myself very lucky to have Chad Watson as my first point of contact at Boise State University and for continuing to be my friend. I am grateful to Dr. Yang Lu for giving me the opportunity to come to Boise State University as his first student and for guiding me through the initial years of research.

I want to thank my dear friend Dr. Mathew Swenson for helping me understand the difference between a strategy and a plan and for the fishing lessons which I enjoyed a lot. I want to also thank him, Sarah Swenson, Andrew Swenson and Madeline Swenson for their company without which our life in Boise would have been far less exciting. I want to thank Dr. Mathew Jones for being an extremely uplifting company and for graciously accepting every one of my 'hey Matty, can I have a minute?', which never was. I am very thankful to Mike Henry and Evan Miller for being the most friendly lab mates I have had and without the countless conversations

we had about the science that we do, I do not stand a chance at completing this Ph.D. at this rate. I wish to express my deepest gratitude to Dr. Janet Callahan for being an extremely supportive Department Chair. I am also very grateful to Dr. Peter Mullner for guiding me throughout my life as a graduate student and for being part of my dissertation committee. I want to thank Dr. Will Hughes for being my instructor for the first Materials Science course I attended (MSE 245, Introduction to Materials Science), which continues to influence the way I approach Materials Science and for continually encouraging me by showing confidence in me whenever we ran into each other.

I greatly appreciate the Department of Materials Science and Engineering staff, especially Jessica Economy for her support throughout the program and for all those reminders you sent. I acknowledge the administrators of the R2, kestrel and fry cluster computers, especially Jason Cook and Kelly Byrne at Boise State University for maintaining those machines, without which my Ph.D. would have taken much longer to finish.

I want to extend my thanks to Raju Chacko and Elizabeth Chacko, my adopted parents in Boise with whom we spent every Friday night. I extend my thanks to my extended "Boise family"- Nitin, Suvarna, Rohan, Arjun (Aju), Rohit, Bhavana, Meera, Nandi and soon to be "Dr." Tony Varghese for all the wonderful memories. I also want to thanks our "American" cousins Sanju, Reshma, Vinu, Megha and our "American" aunt and uncle, Geetha and Dr. Mathew for being our Santa through our college life.

Even though I have not met my paternal grandfather K. C. Kurian, his stories about being a chronic experimentalist and a self taught inventor influenced my love for science, for which I am thankful. To my father K. K. Thomas, my mother Isha

Thomas, my father-in-law Mathew Pottamkulam and mother-in-law Latha Mathew, without your unconditional love, affection and encouragement, I would not have followed my dreams. And to all my siblings (Vinay, Rhea, Kurian, Payal, Clifford and Diya), your support means a lot.

My sons Thomas and Mathew, I appreciate your patience for bearing with your dad's decision to go back to college and taking you away from your dearest cousins and grandparents back in India. Finally, my dearest wife Anuya, this Ph.D. is as much yours as it is mine and I am in debt. I cannot express in words my love and thanks to you for being alongside me during this adventure and many more exciting ones to come.

ABSTRACT

This research aims at developing new computational methods to understand the molecular self-assembly of reacting systems whose complex structures depend on the thermodynamics of mixing, reaction kinetics, and diffusion kinetics. The specific reacting system examined in this study is epoxy, cured with linear chain thermoplastic toughening whose complex microstructure is known from experiments to affect mechanical properties and to be sensitive to processing conditions. Mesoscale simulation techniques have helped to bridge the length and time scales needed to predict the microstructures of cured epoxies, but the prohibitive computational cost of simulating experimentally relevant system sizes has limited their impact. In this work we develop an open-source plugin for the molecular dynamics code HOOMD-Blue that permits epoxy crosslinking simulations of millions of particles to be routinely performed on a single modern graphics card. Using these capabilities, we are able to use ensembles of epoxy processing pathways to obtain realistic bond kinetics and relaxation times that sensitively depend on stochastic bonding rates and a diffusive drag parameter respectively. This work also demonstrates the first implementation of fully customizable temperature-time curing profiles and the largest cross-linked structures obtained using molecular dynamics simulation. We evaluate coarse-grained models based on Dissipative Particle Dynamics (DPD) and compare with Lennard-Jones(LJ) models for their suitability to study glassy dynamics which is important for modeling epoxies or any other glassy material. We find that "hard" particle potentials such as the LJ potential are necessary to model glassy materials and characterize multiple

methods for measuring the glass transition temperature (T_g) in simulations. We find that variations in temperature-time curing profiles result in significant differences in the final cured morphologies. Finally, we apply our general techniques to the specific DGEBA/DDS/PES system and validate our predicted glass transition temperatures against experiment.

TABLE OF CONTENTS

ABSTRACT viii

LIST OF TABLES xiv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xxviii

LIST OF SYMBOLS xxx

1 Introduction 1

1.1 Motivation 1

1.2 Outline 3

2 Background 5

2.1 An Abridged Introduction to Polymer Science 5

2.2 Crosslinking Polymers 6

2.3 Characterising Structure from MD Simulation Output 8

2.4 Coarse Graining 11

2.5 Simulation Methods 16

2.6 Data Management and Analysis 18

2.6.1 epoxpy 19

2.6.2 epoxpy-flow 20

2.6.3 `pandas` and `jupyter notebooks` . . . 22

3 Developing Efficient Methods for Reaction Modelling of Epoxy Crosslinking . . . 23

3.1 Introduction . . . 23

3.2 Methods . . . 28

3.2.1 HOOMD-Blue . . . 28

3.2.2 Dissipative particle dynamics . . . 29

3.2.3 Reaction model . . . 33

3.2.4 Enthalpy of Reaction . . . 34

3.3 Results . . . 36

3.3.1 Morphology Characterization . . . 37

3.3.2 Calibration of Reaction Kinetics . . . 39

3.3.3 Enthalpy of Reaction . . . 45

3.3.4 Cure Path Dependence . . . 45

3.4 Conclusions . . . 49

4 Evaluating Molecular Dynamics Models for Studying Glassy Dynamics . . . 52

4.1 Introduction . . . 52

4.2 Propensity for Unphysical Bond Crossing . . . 56

4.3 Methods for Measuring T_g . . . 63

4.3.1 Thermodynamic Variables Used to Measure T_g . . . 63

4.3.2 Quench and Anneal Cooling Methods . . . 64

4.3.3 Data Fitting Models for Detecting T_g from Thermodynamic Data 65

4.3.4 Fiducial Parameters . . . 67

4.4 Comparing T_g Measurements 69
4.4.1 Effect of Choice of Thermodynamic Variable Used to Measure T_g 69
4.4.2 Effect of Data Fitting Method 73
4.4.3 Effect of Cooling Method 79
4.4.4 Effect of Simulation Model 86
4.4.5 Effect of Chain Length 89
4.4.6 Effect of Angle Constraints 91
4.4.7 Effect of Asymmetric Interaction Parameters 93
4.5 Conclusion 99

5 Modelling the Poly (ether sulphone) toughened Diglycidyl Ether Bisphenol-A/4,4'-Diaminodiphenylsulphone system 101
5.1 Introduction 101
5.2 Methods 104
5.2.1 Lennard Jones Parameterization 104
5.3 Results 107
5.3.1 Glass Transition Temperature of DGEBA/DDS/PES 107
5.3.2 Deriving Physical Units Using $T_g(\alpha = 1.0)$ 109
5.3.3 Finite Size Effects 112
5.3.4 Morphology Evolution of the DGEBA/44DDS/PES System . . . 113
5.3.5 Sensitivity to the "Step" Time-Temperature Curing Profiles . . . 117
5.4 Conclusion 120

6 Conclusions and Suggestions for Future Work 126
6.1 Conclusions 126
6.2 Suggestions for Future Work 130

REFERENCES 133

APPENDIX A Performance optimization 151

A.1 Performance Profiling 151

A.2 Performance and System Size 154

APPENDIX B Calibration of Reaction Kinetics 155

APPENDIX C Glass Transition Measurement 158

APPENDIX D Sensitivity to the 'Linear Ramp' Time-Temperature Curing Profiles 160

APPENDIX E LJ Unit Conversions 163

APPENDIX F Post-Simulation Data Analysis Code 166

F.1 Code for Chapter 3 166

F.2 Code for Chapter 4 267

F.3 Code for Chapter 5 641

F.4 Common Code for All Chapters 737

LIST OF TABLES

3.1 Unitless repulsion parameters a_{ij} for amines (A), epoxies (B), and toughener (C) beads determined by Hildebrand solubility parameters from atomistic molecular dynamics. 32

3.2 Fiducial simulation parameters 33

3.3 Fit quality (R^2) for the FO model 44

4.1 Characteristic Features of Glass Forming Systems Modelled using CGMD. 54

4.2 Repulsion parameters a_{ij} for amines (A), epoxies (B), and toughener (C) beads determined by Hildebrand solubility parameters from atomistic molecular dynamics. 58

4.3 DPD/Harmonic Parameters 59

4.4 LJ/Harmonic Parameters 60

4.5 LJ/FENE Parameters 61

4.6 Comparison of Bond Crossing Energy (E_{BC}), Bond Crossing Distance ($D_{BC}[\sigma]$) and Bond Crossing Probability (X_{BC}) 63

4.7 LJ parameters for coarse grained beads A, B and C. 67

4.8 LJ/Harmonic Simulation Parameters 68

4.9 DPD Quench Parameters 86

4.10 Asymmetric LJ interaction parameters. 94

4.11 Simulation Parameters 94

5.1 LJ parameters (ϵ_{ij}) for coarse grained beads representing DGEBA, DDS and PES molecules. 105
5.2 Fiducial Simulation Parameters 106
5.3 Comparison with other DGEBA/44DDS/PES models 124

A1 Comparison of TPS between the python bonding routine, the `freud` bonding routine and the dybond plugin, with no bonding listed as reference (N = 50,000). 153

B2 Solubility parameters obtained from MD for DDS 155
B3 Solubility parameters obtained from MD for DGEBA 156
B4 Solubility parameters obtained from MD for PES 10-mers 157

C5 Custom Data Ranges for PRM for the A-B Binary LJ/Harmonic Model 158
C6 Custom Data Ranges for PRM for the A-B-C Binary LJ/Harmonic Model 159

LIST OF FIGURES

1.1 Depending on the curing conditions of the epoxy-thermoplastic resin, a spectrum of microstructures can be generated depending on composition, chemistry and the crosslinking conditions. 2

2.1 X_{gel} is detected as the divergence of the molecular mass of the largest branched polymer from the second largest. 7

2.2 The intensity indicates the degree to which the coherence length scales appear in the simulated volume. The peak in the scattering pattern indicated by the red dot (q_m) typically corresponds to the distance between the toughener domains or the size of the toughener domain. The white horizontal and vertical lines overlayed on the morphology are 25 σ long corresponding to the detected q_m. 10

2.3 The microstructure of cured epoxy-thermoplastic blends is typically observed at the μm scale, necessitating the use of CGMD. 14

2.4 Coarse graining of the DGEBA/44DDS/PES system where each monomer is represented by a single simulation element referred as a "bead". 15

2.5 Three types of interactions are considered for the coarse-grained beads in this work. 16

2.6 The class hierarchy in `epoxpy`. Note that only the yellow classes can be instantiated as objects and the others are all abstract classes. 21

3.1 Amine, epoxy, and toughener monomers are represented with spherical simulation elements ("beads"). 30

3.2 During a bonding step, candidate bonds (dashed lines) are stochastically converted to bonds (solid lines) between amine/epoxy pairs that have reactive sites remaining and are sufficiently close. 34

3.3 C-C structure factors show $\approx$ 35 nm toughener domains emerge for $N > 1 \times 10^6$ system sizes, while $N = 5 \times 10^4$ systems demonstrate macrophase separation. Blue stars indicate the wavenumber corresponding to half the box length (the largest resolvable length scale with a periodic simulation volume), and red dots indicate local scattering maxima corresponding to the phase-separated feature size. The error bars indicate standard error. 38

3.4 C-C structure factors for $N = 5 \times 10^4$ to $N = 3 \times 10^6$ show that microstructure can consistently be detected for $N \geq 1.2 \times 10^6$. The blue stars indicate the wave vector corresponding to the half box length and the red dot indicates the detected first peak. The structure factor intensities are shifted up in intensity for visibility. The average wave vector corresponding to the characteristic feature size ($\langle q_{max} \rangle = 0.17 nm^{-1}$) is shown in dotted line and the color bar indicates N. 39

3.5 Representative cure fraction (dashed) and amine concentration trajectories. A_n indicates an amine with n formed bonds. 41

3.6 Average fit metric $\left\langle R^2 \right\rangle$ for the four kinetic models as a function of bond frequency A= $\frac{n_B}{\tau_B}$. 42

3.7 FO model fits of simulated α_T for (a) $A = 2.0$ and (b) $A = 0.1$. Reducing A increases R^2 and decreases α_T. 43

3.8 Temperature history of the curing simulation with different ΔT_{rxn} where initial temperature is 2 kT. 46
3.9 Cure fraction as a function of time where the initial temperature is 2 kT and $\Delta T_{rxn} = 1 \times 10^{-5}$ kT. 47
3.10 Quality of fits for the different reaction models for different values of ΔT_{rxn} show that we get cure kinetics that match the SAFO model with $\Delta T_{rxn} > 0$ kT. The error bar show standard error in R^2 value for curing temperature $T = 0.5, 1.0, 2.0, 4.0, 6.0$ kT 48
3.11 Isothermal curing results in higher cure fraction as a function of time during the first half of the simulation at 850 K, while linear ramps allow for more structural rearrangements at the point each cure protocol reaches the same cure fraction. 49
3.12 Except for 200 K, all the samples reach α_{cut}=0.95. We observe $\alpha_{gel} \approx 0.5$ for all temperatures. 50
3.13 Structure evolves differently for the samples cured with different temperature profiles. C-C structure factors are shown for samples taken at 10% cure and 95% cure. The error bars show standard error. 51

4.1 Illustration of two hard-sphere chains stretching to pass through each other. 57
4.2 DPD potential is used for the non-bonded particles and Harmonic bond potential for bonded particles. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a . 59

4.3 The functional form for the LJ/Harmonic potential where E Factor=1.0. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a 60
4.4 The functional form for the LJ/FENE potential where E Factor=1.0. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a 62
4.5 The self-diffusion coefficient of the coarse-grained beads show increased diffusivity with increasing temperature and lower cure fraction. 65
4.6 Comparison of (a) volume and (b) diffusivity (self diffusion of B beads) change as a function of quench temperatures. The systems with lower degrees of cure exhibit a sharp change in volume as a function of quench temperatures rather than a gradual change as seen at higher degrees of cure. 69
4.7 The specific heat capacity at constant pressure (C_p) shows discontinuity for the system cured to 10% and not for the 90% cured system indicating a first order phase transition in the former. 70
4.8 The average molecular mass is below 10 M (M is the dimensionless mass unit) up to gel point where the mass of each bead is 1 M. The low standard deviation in molecular mass before gelation indicates that most of the networks are small until gelation. 71

4.9 The DiBenedetto equation (Equation 4.10) fits the T_g values calculated using (a) volume better than (b) diffusivity. However, the T_g values detected by the volume data below α = 0.6 do not correspond to a glass transition or first-order phase transition, but is an artifact of the PRM-a. 72

4.10 PRM-a is able to detect the T_g (★) well for system with (a) α = 0.0 and (b) α = 0.6, but for (c) α = 0.9, the T_g detection is unstable. The large deviations of T_g data from the DiBenedetto function (c) for α > 0.5 indicate the instability of the PRM. 73

4.11 The DiBenedetto equation is found to fit T_g (★) values detected using the PRM better than the PRM-a. The custom data ranges used are given in Appendix C5 75

4.12 The DiBenedetto equation is found to fit the T_g (★) values detected using the HFM very well. The inset for α = 0.0 (a) shows the magnified low temperature data fitting of the hyperbola tail. 76

4.13 The DiBenedetto equation is found to fit the T_g (★) values detected using the HFM-c almost as well as the HFM. The inset for α = 0.0 (a) shows the magnified low temperature data fitting of the hyperbola tail. 77

4.14 The DiBenedetto equation the T_g (★) values detected using the PLFM is also found to fit well and is similiar to the HFM and HFM-c fitting. The inset for α = 0.0 (a) shows the magnified low temperature data fitting. 78

4.15 A first-order phase transition is observed at $T \approx 0.28\ T^*$ when the 40% cured system is quenched. The sharp change in volume at $T = 0.28\ T^*$ is characteristic of this transition. The A-A radial distribution function and morphology in the inset indicate that the system undergoes a transition from an amorphous structure at $T = 0.3\ T^*$ to a structure with long range order at $T = 0.28\ T^*$. 80

4.16 A glass transition is observed at $T \approx 0.23\ T^*$ when the 60% cured system is quenched indicated by the characteristic gradual change in volume between $T = 0.2\ T^*$ and $T = 0.5\ T^*$. The A-A radial distribution and morphology in the inset shows that the structure before the transition ($T = 0.3\ T^*$) is the very similiar to the structure after the transition ($T = 0.23\ T^*$). The A-A radial distribution function for $T = 0.28\ T^*$ is shown for comparing with Figure 4.15. 81

4.17 A 30% cured system results in amorphous structure at low temperatures when quenched, but results in an ordered structure when annealed. The A-A radial distribution function and morphology of the system at $T = 0.2\ T^*$ (inset) shows the structural difference caused by the different cooling methods. 82

4.18 The self diffusion coefficient of B beads obtained from annealed simulations cured to $\alpha = 0.6$ is fit using PRM, PLFM, HFM and HFM-c. The diffusivities obtained from the two cooling methods are found to be very similiar and hence the detected T_g values for the quenched systems (☆) and annealed systems (★) are also found to be similiar. . . 84

4.19 The quenched and annealed cooling methods does not seem to significantly impact the T_g measurements with the PRM, PLFM, HFM and HFM-c. The T_g detected from quenched simulations for α = 0.9 using the HFM in (b) was discarded as noise since the $T_g(\alpha = 0.9) = 0.47\ T^*$ was unusually high and α = 0.8 was used instead. 85

4.20 Self-diffusivity of A beads of the DPD model fitted using the piecewise linear regression method and the constrained hyperbola to detect T_g (★). The dotted lines show diffusivity for α=0.2 and α=0.6. The solid lines show the fitted functions. 86

4.21 T_g of the DPD model as a function of cure fraction (α) measured using the PRM-a, HFM, HFM-c and PLFM (dots). The solid lines show the DiBenedetto equation fit (Equation 4.10). The star shows the experimentally observed $T_g(\alpha = 1) \approx 480K$[47] 87

4.22 The annealed simulations used to measure (a) diffusivity and (b) the DiBenedetto equation (Equation 4.10) fit for the T_g data as a function of cure fraction. The self diffusion coefficients of the A particles in the A/B/C mixture are used here. 89

4.23 The length of the C-C chain when increased to 100, causes the C beads to undergo glass transition below $T \approx 0.3\ T^*$ ($g_{CC}(r)$ in (b)), but the A and B beads crystallize ($g_{AA}(r)$ in (b)). Above the phase transition temperature all beads loose long range order as seen in the A-A and C-C radial distribution functions in (c). The system shown here is below gel point ($\alpha \approx < 0.5$) so that the crystallization of A and B can be observed. 90

4.24 The FRC and FJC systems are A/B/C ternary systems which contain C 10-mers with constrained (FRC) and unconstrained (FJC) C-C-C bond angles (The inset images show single FJC and FRC C 10-mers). The A/B binary system does not contain the C 10-mers. The solid line shows the DiBenedetto equation fits for the T_g data. The A/B/C system with FJC has lower T_g than the A/B system at high cure fractions. With FRC, the A/B/C system exhibits higher T_g than the A/B system. 92

4.25 The equilibrated volume obtained from the systems that consider the asymmetric energy parameters show discontinuity only for the $\alpha = 0$ system.. 95

4.26 The A-A radial distribution function for the A/B/C ternary mixture with asymmetric interaction parameters show crystallization at 0.7 T^* for the 0% cured system (b), but is glassy for the 30% cured sample (c) unlike the system with symmetric parameters which crystallize for systems under 50% cure ($g_{AA}(r)$ in Figure 4.23b). 96

4.27 The diffusivity measurement obtained from quench simulation of A/B/C ternary mixture of LJ/Harmonic model with asymmetric interaction parameters do not undergoes crystallization for $\alpha \geq 0.2$. The inset figure shows the discontinuity in diffusivity for $\alpha = 0.0$, which is characteristic of the first order phase transition. 97

4.28 Comparing different analysis models to detect T_g from the quench simulations of the A/B/C ternary system with asymmetric parameters. 98

5.1 (a) Diffisivity values obtained from quench simulation of the DGEBA/DDS/PES system were used to detect T_g using HFM. (b) The solid curve shows the DiBenedetto equation fit for the T_g data and the dotted horizontal line is a guide for comparing the experimentally observed $T_g(\alpha = 0.4) \approx 300\ K$. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol. 107

5.2 (a) Diffisivity values obtained from quench simulation of the DGEBA/DDS/PES system were used to detect T_g using PRM. (b) The solid curve shows the DiBenedetto equation fit for the T_g data and the dotted horizontal line is a guide for comparing the experimentally observed $T_g(\alpha = 0.4) \approx 300\ K$. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol. 108

5.3 Adding bond angle constraint to the PES chains when HFM is used to detect T_g results in a low quality of fit for the DiBenedetto equation. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol. 109

5.4 Effect of adding bond angle constraint to the PES chains when piecewise linear regression where the data range was custom selected manually to detect T_g. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol. 110

5.5 The simulated T_g at low and high cure fractions shows close agreement with T_g values measured from an experimental DGEBA-44DDS system[73]. The T_g values are detected using PRM in the simulated systems. 111

5.6 The PES-PES structure factor evolution with time (left) for $N = 1{\times}10^6$ indicates that the morphology (right) has equilibrated. The red dots show the detected first peaks (q_{max}). 113

5.7 The PES-PES structure factor shows emergence of a 0.22 nm^{-1} feature (indicated by the vertical dotted line) at $N \geq 2 \times 10^5$. The color bar indicate system size (N). 114

5.8 The evolution of structure is tracked using q_{max} for one simulation per cure temperature as shown in the legend. The data points in dots are the first peaks (q_{max}) detected in the PES-PES structure factor as a function of time as the system is undergoing curing reaction. The solid curve shows the relaxation time of phase separation function (Equation 5.7) fit for the q_{max} data. 115

5.9 The solid line shows the WLF equation fit to the mean structural relaxation time τ (squares) obtained from 5 replicate simulations. The error bars show the standard error from the replicate simulations and the legend shows the WLF parameters obtained from the WLF fitting. 116

5.10 The "Step" curing profile shown here has three variable times and temperatures indicated by the red dots. 118

5.11 Temperature profiles where the initial ramp up time (t1) is varied 119

5.12 Time to gelation is not affected by $t1 < 2 \times 10^5\ \Delta t$. t1 time denote the time at which the cure temperature is ramped up and held constant. . . 120

5.13 Temperatures profiles (a) and curing profiles (b) for $t2 < t_{gel}$ (t2=$2 \times 10^6\ \Delta t$) and $t2 > t_{gel}$ (t2=$9.5 \times 10^6\ \Delta t$). The hollow squares show gel point. $T2$ is chosen to be higher than and $T3$ is chosen to be slightly lower than the T_g of the fully cured system ($T_g(\alpha = 1.0) = 480\ K$). 121

5.14 PES-PES structure factor shows difference in morphology as a result of varying t2 of the "Step" curing profile. The error bars show standard error from the three replicate simulations. The intensity at $q \approx 0.25\ nm^{-1}$ shows a statistically significant difference indicating that $t2 < t_{gel}(t2 = 2 \times 10^6\ \Delta t)$ resulted in a more mixed microstructure. 122

6.1 The TPS as a function of system size shows performance of the DPD/Harmonic model. 127
6.2 For simulations where $N < 1.2 \times 10^6\ \Delta t$. 127
6.3 A total of 66085 GPU hours of simulations have been run on the 4 clusters during the course of this research. This information is based on the log files produced by the simulations. 129
6.4 The newly developed parts of the software stack used to run the simulations are shown in blue and the green boxes show the software libraries that are leveraged by this software stack. 130

A.1 Distribution of computational cost for our pure python code implementation of Algorithm 1. 151
A.2 Distribution of computational costs for our `freud` implementation of Algorithm 1. 152
A.3 Split of the time taken by different parts of a typical curing simulations. 153
A.4 Performance scaling of TPS vs. number of particles N from 5000 to 2 million where the cure percent has reached 95%. Morphologies shown are for 50,000, 1,000,000 and 2,000,000 particles. 154

D.1 Temperature and cure fraction of an isothermal and linearly ramp profile. 160

D.2 Temperature and cure fraction of an isothermal and linearly ramp profile.161
D.3 PES-PES structure factor from the isothermal cure profile shows microphase separation whereas the linearly ramped cure profile has resulted in a macrophase separated morphology. 162

LIST OF ABBREVIATIONS

44DDS 4,4'-diaminodiphenyl sulphone.

AAMD All Atom Molecular Dynamics.

ABS poly(acrylonitrile-butadiene-styrene).

CA 4-methylhexahydrophthalic anhydride.

CGMD Coarse Grained Molecular Dynamics.

DGEBA diglycidyl-ether of bisphenol A.

DGEBF diglycidyl-ether of bisphenol F.

DPD Dissipative Particle Dynamics.

EP 3,4-Epoxycyclohexylmethyl-3,4-epoxycyclohexanecarboxylate.

FENE Finitely Extensible Non-Linear Elastic.

FJC Freely Joint Chains.

FRC Freely Rotating Chains.

HFM Hyperbola Fitting Method.

HFM-c Constrained Hyperbola Fitting Method.

HOOMD-Blue Highly Optimized Object-oriented Many-particle Dynamics- Blue Edition.

IBI Iterative Boltzmann Inversion.

KG Kremer-Grest.

LJ Lennard-Jones.

MD Molecuar Dynamics.

MSIBI Multistate Iterative Boltzmann Inversion.

MTHPA methyl tetrahydrophthalic anhydride.

PDMS Polydimethylsiloxane.

PE Polyethylene.

PES poly ether sulphone.

PLFM Power Law Fitting Method.

PP Polypropylene.

PRM Piecewise Regression Method.

PRM-a Automatic Piecewise Regression Method.

PS Polystyrene.

RDF radial distribution function.

WLF William-Landel-Ferrel.

LIST OF SYMBOLS

A Frequency factor (n_B/N_B).

D_{BC} The inter-particle distance at which bond crossing occurs.

E_{BC} The energy required for bond crossing.

E_a Activation energy of bond formation.

E_{coh} Cohesive energy density.

F_i^C Conservative force.

F_i^D Dissipative force.

F_i^R Random force.

δ Hildebrand solubility parameter.

K Kelvin.

L_B Length of the cubic simulation box edge.

L Distance unit.

N_B Total bonds possible in the system.

N Total number of particles in the system.

P The barostat step point pressure.

$\mathbf{E}$ Total energy of the system.

T_C Calibration temperature for the DPD model.

ΔT_{rxn} Change in temperature of the system per reaction.

T^a_{high} The highest annealed temperature to which the system is cooled for T_g measurement.

T^a_{low} The lowest annealed temperature to which the system is cooled for T_g measurement.

T_g^{exp} The glass transition temperature found experimentally.

T_g^{sim} The glass transition temperature found from simulation.

T_g Glass transition temperature.

T^q_{high} The highest quenched temperature to which the system is cooled for T_g measurement.

T^q_{low} The lowest quenched temperature to which the system is cooled for T_g measurement.

T^* The dimensionless temperature of the system.

$\mathbf{U}$ Potential energy of the system.

$\mathbf{V}$ System volume.

X_{BC} The probability of bond crossing.

α_{cut} Maximum cure fraction of the simulation after which bonding is stopped.

a_{ij} DPD repulsion parameter.

α_{high} The highest cure fraction considered in the study.

α_{low} The lowest cure fraction considered in the study.

α_{gel} Cure fraction at gelation.

α Cure fraction of the system.

cm centi meter.

$^{\circ}\mathbf{C}$ Degree Celcius.

δ Delta function.

dt Time step size.

Υ E factor, a scaling factor for the LJ energy parameter ϵ.

ϵ The Lennard-Jones interaction energy parameter.

γ Dissipative frag coefficient.

k_1 Primary reaction rate.

k_2 Secondary reaction rate.

k_B Boltzmann's constant.

χ Flory Huggins interaction parameter.

kcal kilocalories.

$k_{\mathbf{harmonic}}$ Harmonic bond constant.

k_θ The equilibrium bond angle.

λ The DiBenedetto equation interaction parameter.

n_B Total bonds made per bond step.

nm nano meter.

ρ_n Number density.

ps pico second.

q_{max} The first peak in the scattering pattern which describes the characteristic feature size in the system.

r_0 Equilibrium harmonic bond distance.

ς Random force amplitude.

τ Relaxation time of phase separation.

β Scaling factor for activation energy of secondary bond formation.

s second.

σ Length scale denoting the size of a particle.

t_a The anneal time at each cooling temperature in time steps.

t_q The quench time at each cooling temperature in time steps.

τ_B Bond period in time steps.

t_{gel} Time taken to reach gelation.

θ_0 The equilibrium bond angle.

μm micro meters.

$\boldsymbol{v}_i$ Velocity of particles i.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Carbon Fiber Composites (CFC) have revolutionized the airline industry making the planes lighter and hence more fuel efficient. Aluminum is increasingly being replaced by these materials due to their weight advantage. Thermoset and thermoplastic resins are the two major families of matrix materials used in these composites and epoxy resins are the most widely used thermoset materials. While thermoset polymers have superior thermal stability compared to thermoplastics, by themselves, thermosets are brittle. Mixing thermoplastic polymers with the thermoset resins increases the fracture toughness of the thermoset resin and of the overall composite. The addition of thermoplastics also have the added advantage of making these resins more recyclable and also decrease the energy consumption in processing[22]. However, the addition of thermoplastics results in the formation of a two-phase material whose microstructure can vary greatly depending on the curing conditions. Figure 1.1 shows the extremes which are the fully mixed and the macrophase separated microstructure as well as a microphase separated microstructure that resembles the co-continuous networks seen in experiments.

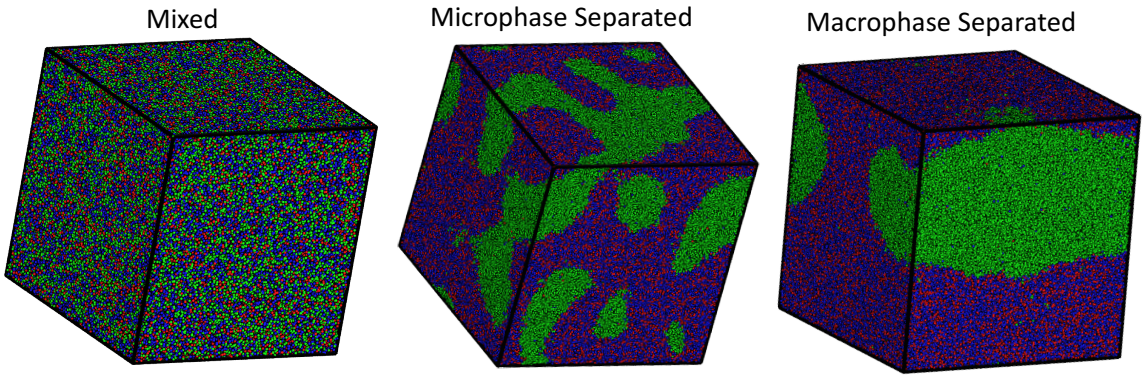


Figure 1.1: Depending on the curing conditions of the epoxy-thermoplastic resin, a spectrum of microstructures can be generated depending on composition, chemistry and the crosslinking conditions.

The increased interfacial area in microphase-separated morphology makes it more suitable than the macrophase separated morphology for applications that require these materials to withstand high mechanical stress. It has been shown in experimental studies that the formation of co-continuous networks of thermoplastic tougheners in the thermoset matrix results in a significant increase in fracture toughness.[13]

The microstructures that can be generated from these two phase materials depend generally on the chemistry, composition and the crosslinking conditions. The amount of thermoplastic tougheners used in the mixture has been shown to affect the fracture toughness[71], but excessive amounts of the thermoplastic toughener also increase the viscosity, making handling more difficult[15]. The number of functional groups in the reacting epoxy monomers and the crosslinking monomer along with the stoichiometry of the mixture is also known to significantly impact the resultant mechanical and thermodynamic properties such as the glass transition temperature (T_g)[61]. The relative reaction rates of the primary (k_1) and secondary amines (k_2) in a reacting mixture of epoxy and amine crosslinkers was shown to influence the

time-temperature-transformation (t-t-t) diagram[61].

The complex nature of this molecular self-assembly motivates the use of computational modeling to efficiently explore the factors that affect the microstructure of this class of materials in the composition-chemistry-crosslinking parameter space. This research aims at developing computational tools and techniques to model the self-assembly of reacting systems, resulting in a better understanding of the structure-properties-processing relationship. These tools and techniques can be widely applicable to a variety of other application areas that depend on the structure-properties-processing relationship of reacting systems.

1.2 Outline

The outline of this dissertation is as follows: Chapter 2 provides some background information on polymer physics relevant to crosslinking polymers followed by the computational methods used to perform simulations as well as for post-processing the output of the simulations.

In Chapter 3, new computational methods are developed for modeling reacting systems. A newly developed reaction algorithm is evaluated for modeling the epoxy curing process using a material model called Dissipative Particle Dynamics (DPD).

In Chapter 4, the DPD model is compared with two other material models for understanding their suitability for studying glassy dynamics, which is an important characteristic feature of epoxy materials that are of interest to this study. In addition to characterizing material models, some data analysis models and molecular dynamics techniques which are intrinsically linked to this study of glassy dynamics are characterized in Chapter 4.

Finally in Chapter 5, the tools and techniques developed are applied to a specific chemistry consisting of diglycidyl-ether of bisphenol A (DGEBA) epoxy, 4,4'-diaminodiphenyl sulphone (44DDS) amine crosslinker and poly ether sulphone (PES) thermoplastic toughener and validated against experimental data because this chemistry is well studied and hence, plenty of data is available in the literature for comparison.

CHAPTER 2

BACKGROUND

2.1 An Abridged Introduction to Polymer Science

Humans have been using natural polymers such as caoutchouc or natural rubber that is extracted from the bark of *Hevea brasiliensis*, the rubber-tree plant much before the advent of the "Polymer Age", as Rubinstein and Colby[86] put it. Although the idea of polymers being macromolecules made of covalently bonded repeat units obtained by polymerizing 'mono'-mers is now commonplace, the common viewpoint in the early twentieth century was that these were colloids of small molecules bound together by non-covalent interaction. The idea of 'macromolecules' as we know it today was first proposed by Staudinger[96], in 1920.

Polymeric substances are known to exits as liquids, solids, and liquid crystals[86]. Liquid polymers are characterized as dilute and semidilute depending on the volume fraction of polymer in the solvent. In the absence of the solution, this liquid is called a polymer melt which is the type of polymer this research focuses on. Polymer melts exhibit liquid flow above their glass transition temperature (T_g) or melting temperature. On cooling, the polymer melt can either transform into a semicrystalline solid material or into an amorphous glass. While the former has a melting temperature, the latter exhibits T_g, the temperature below which the material behaves like a glass and above which it starts to flow.

2.2 Crosslinking Polymers

When the reacting monomers have more than two functional groups, the linear polymer chains connect with each other by forming covalent bonds and form branched polymers. These branched polymers, which are still relatively small molecules, are known as 'sol' since they are soluble in solution. As the crosslinking process continues to form larger branched polymers, it reaches a point in the reaction when a single branched polymer spans the entire system which is no longer soluble in solution. This state of the crosslinked polymer is known as 'gel' and the transition is known as the 'sol-gel' transition or 'gel' transition or 'gelation'. A formulation for the condition for gelation for a specific chemistry purely in terms of the cure fraction, independent of cure conditions such as temperature or amount of catalyst was first developed by Flory[27]. He developed this theory for cases where a difference in reactivity of the same kind of functional group does ($k_2/k_1 \neq 1$) and does not vary ($k_2/k_1 = 1$) due to steric effects. However, experimental validation was done for two chemistries where the difference in reactivity due to steric effects could be ignored. Flory[27] showed that the experimentally observed gel point α_{gel}=0.604 for pentaerythritol, a tertra-functional monomer, polymerized with adipic acid agreed well with the theoretically predicted value of α_{gel}=0.577 when $k_2/k_1 = 1$. For epoxy/amine systems this ratio of reactivities cannot be ignored. The difference in the activation energy of reaction for the primary amine group and the secondary amine group in trimethylene glycol di-p-aminobenzoate is known to be 3.7 kcal/mol[110] and the ratio of the reaction rates of the secondary to the primary amine group (k_2/k_1) is known to be 0.33 at 160°C. The gel point was later derived as the divergence of the average molecular mass in crosslinked polymers by Dusek and Prins[23] as a function of composition of

two reacting species, the number of functional groups in each of polymer species and the ratio of the rate of primary (k_1) and secondary (k_2) reactions. According to their analytical equation, an epoxy monomer with two functional groups reacting with an amine monomer with two functional groups can have a gel point at a cure fraction X_{gel} between 0.5 and 0.618,where X_{gel} = 0.5 is predicted for the ratio of reaction rate $k_2/k_1 \rightarrow \infty$ and X_{gel} = 0.618 is predicted for k_2/k_1 = 0. In simulations performed in this research, the gel point is calculated as the point (X_{gel} in Figure 2.1) at which the molecular weight of the largest branched polymer diverges from that of the second largest branched polymer[26,102].

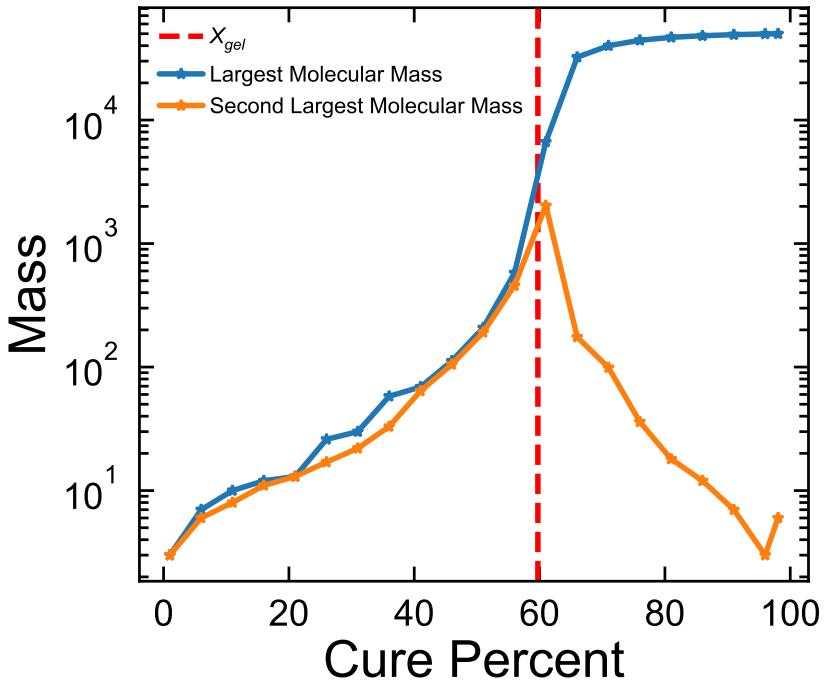


Figure 2.1: X_{gel} is detected as the divergence of the molecular mass of the largest branched polymer from the second largest.

The reaction rate is the rate-limiting mechanism as the reaction starts. As the

reaction progresses, the molecular weight of the polymer chains increase, crosslinking of the chains occur and eventually undergoes the gelation transition. As a result of the increased molecular weight, the viscosity increases and reduces the rate at which potential reactants approach each other. At a high degree of crosslinking past the gelation transition, the diffusion rate reduces significantly and the materials transitions from a gel state to a glassy state. At this state, the diffusion rate is the rate-limiting mechanism of the reaction. For a given chemistry, this transition point occurs at a characteristic temperature known as the glass transition temperature (T_g) and is a function of the cure fraction. The T_g is a very important physical property of epoxy materials making it suitable for applications that require high-temperature resilience. Chapter 4 discussed the detection of T_g in detail.

2.3 Characterising Structure from Molecuar Dynamics (MD) Simulation Output

The radial distribution function (RDF) and virtual scattering experiments are used to quantify the structure obtained from simulations. The RDF ($g(r)$) gives the probability of finding two particles i and j at a distance r away from each other in a given system. This probability is typically normalized to the ideal gas density at the same thermodynamic condition. Given the positions of particles r^N, The RDF at a separation distance r is defined as

$$g(r) = \frac{1}{\rho} \sum_{i}^{N} \sum_{j,j \neq i}^{N} \delta(r - r_{ij}) \tag{2.1}$$

where δ is the delta function, r_{ij} is the distance between particles i and j and $\rho = N/V$ where N is the number of particles and V is the volume of the system. In practice, given the particle positions in a simulated system, $g(r)$ can be calculated by measuring all the inter-particle distances and grouping them into equispaced bins of distances (r) rather than calculating the probability over infinitesimally small distance intervals[28]. The RDF is a very useful tool to characterize crystallinity of the system and for observing order at short distances given by the high probability at those short distances.

The structure factor ($S(\vec{q})$), which can be thought as the reciprocal space representation of the RDF is more suitable to characterize long-range order and is used at several instances in this research. Typically in simulation literature, the structure factor is calculated as

$$S(\vec{q}) = 1 + 4\pi\rho \int_0^\infty \frac{(g(r) - 1)r^2 sin\ qr}{qr} dr \tag{2.2}$$

Since this form of $S(\vec{q})$ is inaccurate for anisotropic systems, $S(\vec{q})$ is calculated according to Ref. 46 where the scattered intensity I at wave vector $\vec{q}$ is related to the structure factor and the form factor $P(\vec{q})$ as:

$$I(\vec{q}) = S(\vec{q})P(\vec{q}) = |FT[\rho(\vec{x})]|^2 \tag{2.3}$$

A spherical form factor $P(\vec{q}) = |FT[\rho p(\vec{x})]|^2$ is considered here where $\rho p(\vec{x})$ is the density distribution a spherical particle. The scattering density ($\rho(\vec{x})$) is calculated from the particles positions ($\vec{x}$) from the simulation and $FT[\rho(\vec{x})]$ is the Fourier transform of $\rho(\vec{x})$. Figure 2.2 shows a sample morphology and the corresponding

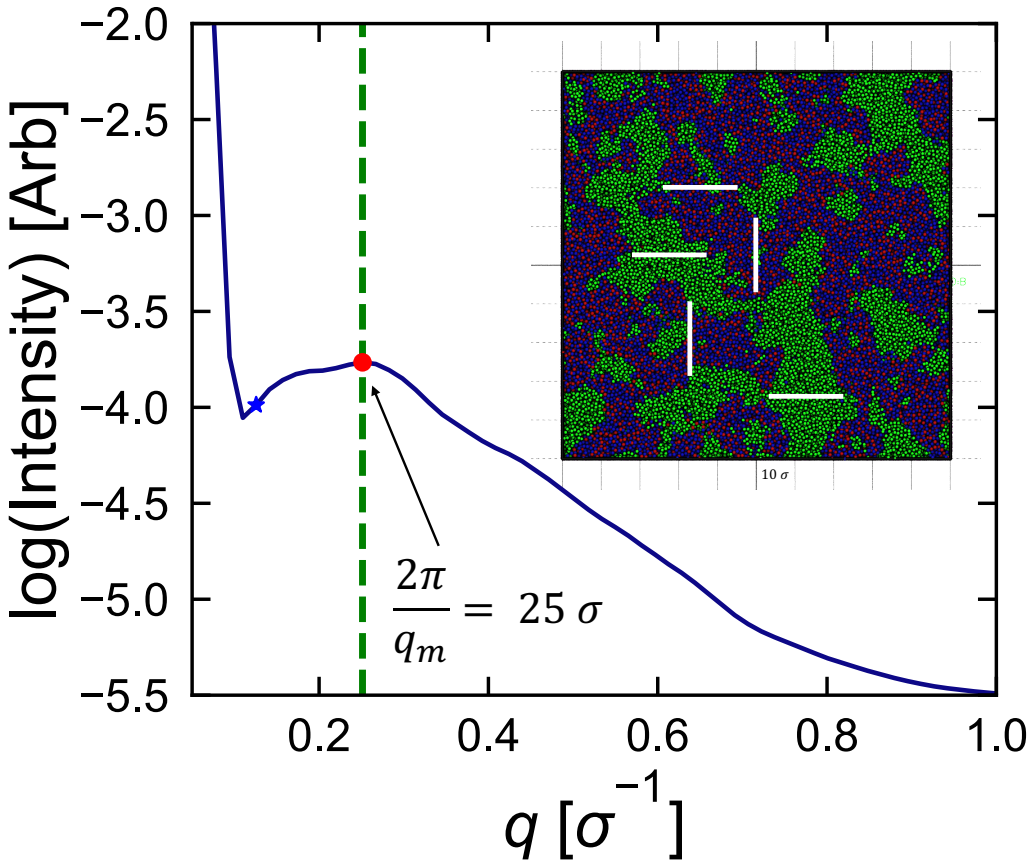


Figure 2.2: The intensity indicates the degree to which the coherence length scales appear in the simulated volume. The peak in the scattering pattern indicated by the red dot (q_m) typically corresponds to the distance between the toughener domains or the size of the toughener domain. The white horizontal and vertical lines overlayed on the morphology are 25 σ long corresponding to the detected q_m.

radially averaged structure factor.

The structure factors are calculated by averaging scattering features obtained from spherically distributed orientations of the simulation snapshot. Typically this radially averaged structure factor is further averaged over multiple snapshots after the simulated structure had stopped evolving. In some cases, replicate simulations with different initial random particle positions are also used to quantify the uncertainty in structure.

The `diffractometer`[46] package is used for calculating the structure factor from simulated structures.

Small-Angle X-ray Scattering of epoxy based materials[70] can be used to compare the simulated scattering data.

2.4 Coarse Graining

The phase separation induced by polymerization of epoxy in an epoxy-thermoplastic toughener mixture is a complex process involving competing mechanisms such as thermodynamically driven demixing and crosslinking reactions. The microstructural features resulting from epoxy curing in the presence of tougheners are typically at the nano meter-micro meter scale and the time taken by the curing is typically in the order of seconds. The computational modeling approach of choice should be able to resolve these length and time scales to effectively study this phenomenon. Germann and Kadau[29] simulated 1×10^{12} Lennard-Jones particles which used 40 TB of memory and represented an edge length of about 2.5 μm. Even though such large simulations of Lennard-Jones particles are possible these days, each timestep was reported to take 45-49 s of wall-clock time. A 10 ps simulation was performed in a couple of days. While large length scales can be parallelly simulated and is mainly limited by the available computer memory and communication overheads, simulating long time scales do not benefit from parallel computing capabilities. According to Germann and Kadau's estimate based on Moore's Law (assuming that computational power doubles every 18 months), 3 dimensional MD simulations at engineering scale (2.5 cm) would be possible in the next 100 years, but the time scale achievable at this rate will only be 1 ms. One of the relatively large scale All Atom Molecular Dynamics (AAMD)

crosslinking simulation of phenolic resins consists of 230,000 atoms and a simulation volume edge length of 13.6 nm which was used to study the crosslinked network structure and compared with experimentally observed network structures[93]. This system did not include a non-reacting component such as a thermoplastic toughener which phase separates from the reacting components. Even larger system sizes might be necessary to observe the thermoplastic phase separation. Based on the two representative examples of AAMD studies, it appears that the study of self-assembly of thermoplastic toughened epoxy system into morphologies such as co-continuous networks is out of reach for typical computational resources available today using AAMD primarily because of the limited timescales accessible as well as the high computational cost of accessing large size scales. Another important aspect of studies such as this is the practical upper limit for the time taken to run these simulations which render the very large scale AAMD simulations ineffective and impractical. Since crosslinking simulations are essentially non-equilibrium simulations, multiple replicate simulations might be necessary to quantify the uncertainty in the resultant morphology. In order to study the effect of curing temperature on morphology, it is necessary to run multiple simulations of curing reactions at different temperatures. In both of these cases, the ability to run many simulations in a short amount of wall-clock time is more beneficial than very large simulations that takes a large amount of wall-clock time.

Higher scale modeling techniques such as finite element analysis lack the resolution necessary to model the diffusion and reaction dynamics involved in the epoxy curing process. Tao et al.[101] was able to model the cure kinetics of CE-688 epoxy using constitutive equations where the degree of cure agreed well with the experimental degree of curing. The predicted degree of cure during isothermal curing deviated

from the experimental degree of cure beyond 70% curing. The authors attributed this discrepancy to the fact that the reaction becomes diffusion controlled at later stages of curing which was not explicitly taken into account by the constitutive model. Another continuum-scale empirical model called the phase field modeling method[45] has been shown to have practical applications for predicting phase separated morphologies. This method uses the Cahn-Hilliard equation to model spinodal decomposition and predicts the spinodal size scale for polymeric mixtures demixing to form bicontinuous structures[17]. The thermodynamics of mixing is modeled by this equation by taking as it's input the Flory-Huggins interaction parameter χ and a chain stiffness parameter k in the framework of random phase approximation. According to the authors, some of the factors that limit the applicability of this method are non-availability of the experimental χ value for polymer pairs, the deviations from predicted spinodal size due to the effect of chemistry specific relaxation timescale of polymers on their demixing behaviour and the empirically obtained k value's applicability being limited to linear chains, weakly interacting mixtures and critical compositions. Furthermore, there is a lower limit to the spinoidal size for the Cahn-Hilliard equation at which the prediction of spinodal size breaks down. Another fundamental limitation of this method is the inability to examine the network structure formed during reaction because of the continuum assumption of the materials, unlike atomic scale modeling methods which model materials as discrete particles.

A mesoscale modeling method called Coarse Grained Molecular Dynamics (CGMD) is used in this research to overcome the limitations associated with AAMD and continuum scale modeling methods such as phase field modeling and finite element analysis. A schematic representation of the relative position of CGMD with respect to AAMD and finite element method is shown in Figure 2.3. The CGMD method is similar to

AAMD in terms of the governing equation and the particulate nature of the material model. These two methods differ in the spatial and temporal resolution. Unlike AAMD where induvidual simulation elements model atoms, in CGMD induvidual simulation elements model more than one atom.

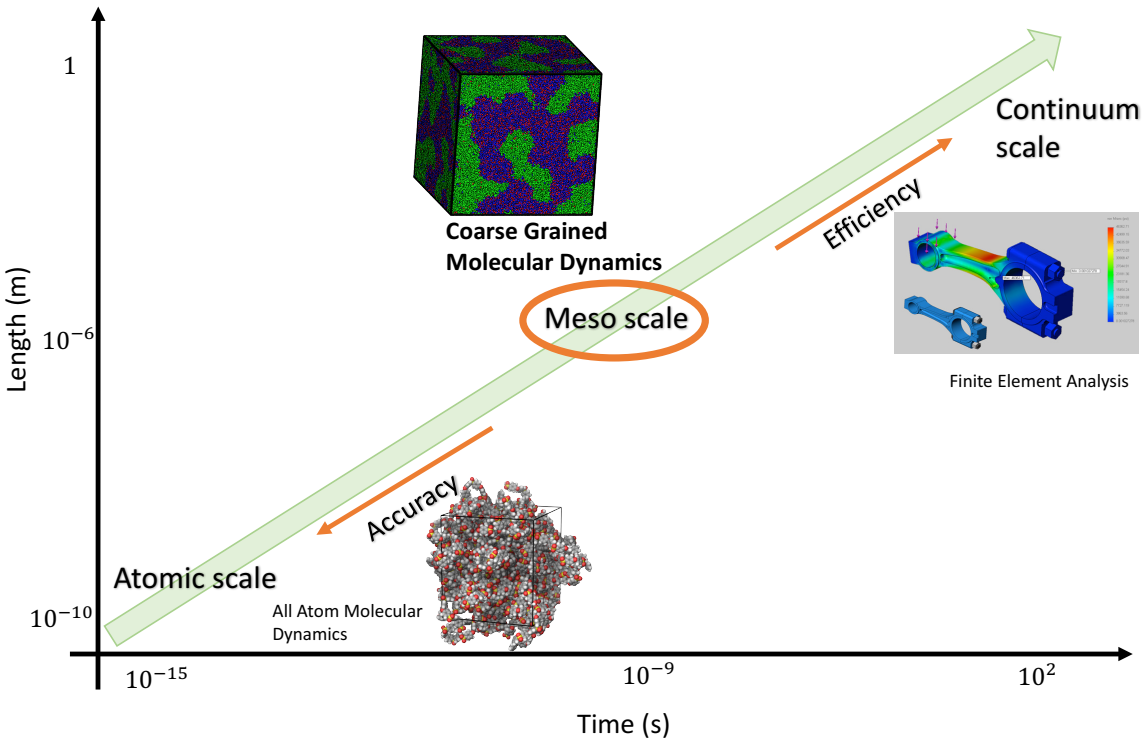


Figure 2.3: The microstructure of cured epoxy-thermoplastic blends is typically observed at the μm scale, necessitating the use of CGMD.

Figure 2.4 shows the coarse graining of a DGEBA, 44DDS and PES monomer where the internal degrees of entire molecules are abstracted away and represented by a single simulation element. Even though it is possible to represent more than one monomer as a single bead[118,119], it will be more difficult to conceptualize the process of reacting epoxy and amine monomers if each bead represents more than

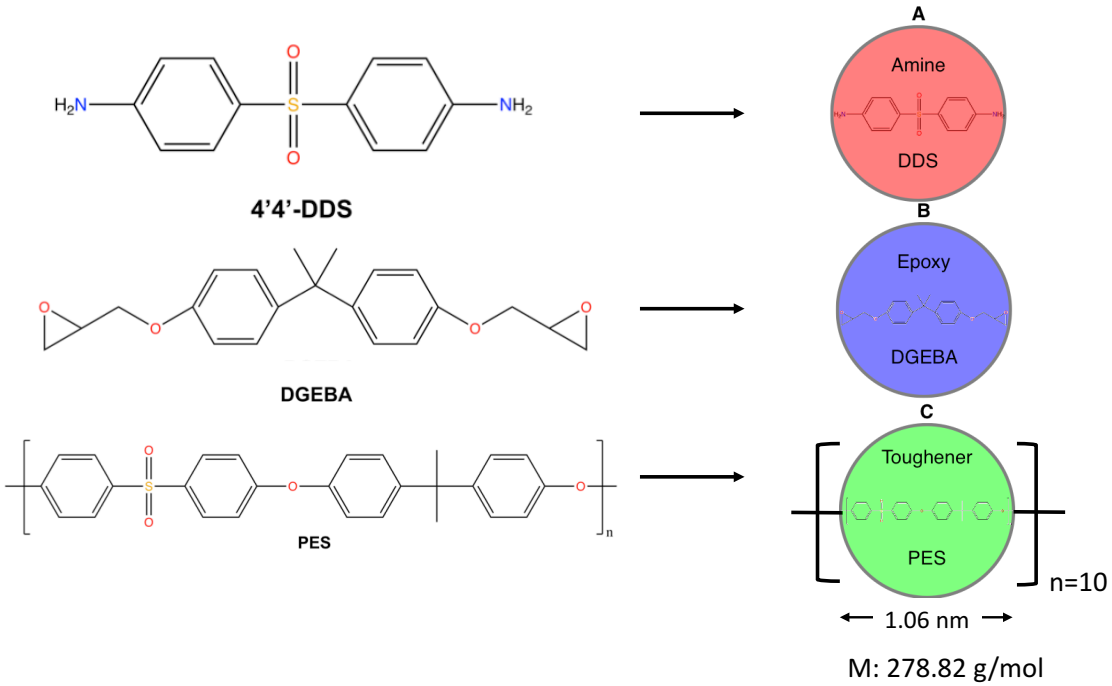


Figure 2.4: Coarse graining of the DGEBA/44DDS/PES system where each monomer is represented by a single simulation element referred as a "bead".

one monomer. Hence the CGMD model in this work considers a single "bead" to represent a monomer units that typically consist of about 20 atoms. Three types of interactions experienced by these beads at the nanometer scale are considered in this work, the non-bonded interactions which capture the van der Waals interactions, the bonded-interactions which capture the covalently bonded interactions and the implicit solvent interactions which capture the interactions of these beads with the environment which include random forces and dissipative forces, shown as arrows and dashpots in Figure 2.5.

The non-bonded interaction is modeled in this work using empirically derived

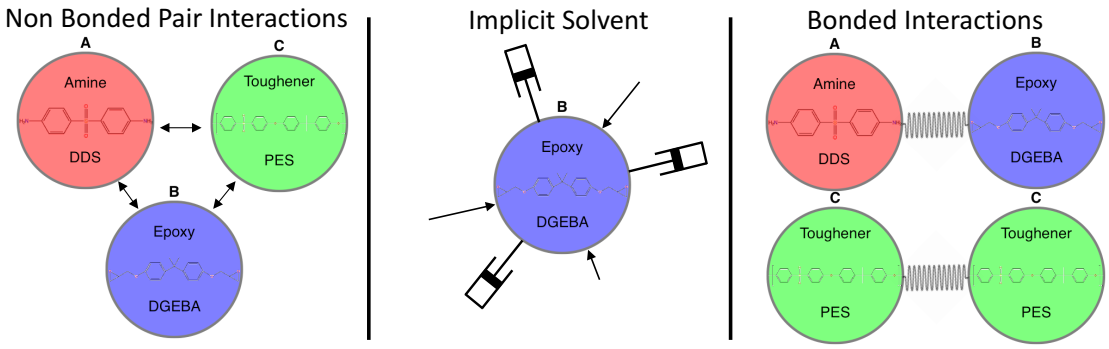


Figure 2.5: Three types of interactions are considered for the coarse-grained beads in this work.

Lennard-Jones (LJ) and DPD force fields. A harmonic bond potential and the Finitely Extensible Non-Linear Elastic (FENE) potential are considered for the bonded interactions. The implicit solvent modeling is done through either the non-conservative part of the DPD or the Langevin thermostat. The specific details of the force fields and the methods used to derive their parameters empirically are discussed in later chapters.

2.5 Simulation Methods

CGMD is performed using a general purpose MD simulation engine called Highly Optimized Object-oriented Many-particle Dynamics- Blue Edition (HOOMD-Blue). In MD, the positions of N particles $\boldsymbol{x}^N = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$ with mass m are integrated forward in time according to Newtons's second law of motion[28].

$$m\ddot{\boldsymbol{x}}_i(t) = \boldsymbol{F}_i^C(\boldsymbol{x}_i(t)) \tag{2.4}$$

The equation states that the acceleration experienced by particle i depends on the conservative forces (F_i^c) acting on it which is given by the negative gradient of the potential energy (U) which is a function of all the particle positions $\boldsymbol{x}^N$. While considering implicit solvent methods such the non-conservative forces are superimposed on the conservative forces (F^c). In the absence of non-conservative forces, solving the MD equations of motion is equivalent to sampling configurations in the microcanonical ensemble (NVE) where the number of particles (N), volume (V) and energy (E) is kept constant while allowing temperature and pressure to vary. In order to compare simulations to experiments, it is desirable to sample the system in other thermodynamic ensembles such as the canonical ensemble (NVT) which considers constant temperature or the isothermal-isobaric ensemble (NPT) which considers constant pressure and temperature. These additional ensembles are sampled by modifying the traditional MD equations of motion and extending it to include terms for thermostats and barostats. The Langevin equation of motion used in this work is such a modified MD equation (Equation 2.5) where F_i^C, the conservative force is obtained as the derivative of the LJ potential energy function, F_i^R and F_i^D, the random and dissipative forces simulates the temperature and viscous forces imparted by the implicit solvent (Figure 2.5).

$$m\ddot{\boldsymbol{x}}_i(t) = \boldsymbol{F}_i^C(\boldsymbol{x}_i(t)) + \boldsymbol{F}_i^R(t) + \boldsymbol{F}_i^D(\dot{\boldsymbol{x}}_i) \tag{2.5}$$

The dissipative force acts in the opposite direction of the particle's motion via the Stoke's equation $\boldsymbol{F}_i^D = -\gamma \boldsymbol{v}_i$, where γ is the dissipative frag coefficient and $\boldsymbol{v}_i$ is the particle velocity. The random force has a mean force of zero ($\langle \boldsymbol{F}_i^R \rangle = 0$) and satisfies the fluctuation-dissipation theorem by relating to the dissipative force according to:

$$\langle \boldsymbol{F}_i^R(t).\boldsymbol{F}_j^R(t')\rangle = 6\gamma k_B T/\Delta t\delta_{ij}\delta(t-t') \tag{2.6}$$

The Langevin equation of motion is very similiar to the DPD equation used in this work. The difference between them is that the random force in DPD is a pairwise force that acts in the direction of the line of centers between the two interacting particles, thereby conserving momentum locally and reproducing hydrodynamic interaction locally whereas the random forces in the Langevin dynamics are unrestricted thereby not reproducing hydrodynamics. Hydrodynamics refer to the condition when particles not directly in contact with each other interact through particles in between them, which could be solvent particles of other solute particles. The bonded particles experience hydrodynamic interactions through the bonded potentials. Hydrodynamics is important to consider for modeling dilute solutions, but for modeling polymer melts or semidilute solutions such as the systems considered in this research, hydrodynamics can be ignored[86]. Further details about MD methods can be found in Ref. 60.

2.6 Data Management and Analysis

Since HOOMD-Blue is built as a general-purpose molecular dynamics package, no system specific functions are provided by it. The initial configurations of the desired epoxy system, with the desired system parameters which include information about the interaction parameters, thermodynamic state points and other simulation parameters such as the time step size and output data saving frequency are all important information that all come together to produce one simulation. It is a common practice to write a monolithic simulation script which contains all of the previously mentioned information. The advantage of this approach is that it is quite easy to make changes quickly to the simulation script. However, in order to explore variations of the

simulated model, it will become necessary to duplicate code which in turn makes the process of bug fixing more tedious since the same bug needs to be fixed in all the duplicated versions of the original simulation script. Overall this approach soon becomes impossible to maintain and intractable for large projects. In order to avoid this problem of duplicated code an object-oriented **python** package called `epoxpy`[102] is developed and used throughout this research.

2.6.1 epoxpy

`epoxpy` is an open-source python package that is essentially a collection of python classes that each represent a single simulation. These simulation classes form a layer of abstraction over HOOMD-Blue and the bonding plugin called `dybond` which is developed as part of this research. These simulation classes also leverage an open-source molecule builder package called `mbuild` to generate initial particle positions and connectivities of the epoxy system being modeled. Every simulation class has a function called `execute` which calls three sub routines: `initialize`, `mix` and `run`. The `initialize` function uses `mbuild` to generate the initial particle positions and connectivities. The `mix` function runs a high-temperature MD simulation with all the prescribed interaction potentials and generates a random initial condition that is representative of a fully miscible polymer blend that is uncured. This phase is important to relax some of the unphysical configurations that may be generated by the initial molecule builder. The `run` function executes the `curing` simulation or `cooling` simulations that are performed on the cured simulation output for purposes such as calculating equilibrium properties as ensemble averages for measuring T_g.

The simulation classes all have a common base class called `Simulation` which encapsulates the common functionality of any simulation. The next level in this

inheritance tree contains the `EpoxySimulation` class which encapsulates methods and variables that are common for all simulations of epoxy materials as shown in Figure 2.6. As this tree grows, the simulation class contains increasingly specific methods and variables that apply to specific simulations. For example, the `ABCTypeEpoxyLJHarmonicSimulation` class is responsible to running a specific epoxy system with "A","B" and "C" beads where the non-bonded interactions are modeled using LJ and the bonded interactions use the Harmonic bond potential unlike the `ABCTypeEpoxyDPDHarmonicSimulation` which uses DPD as its non-bonded interaction. As each of these simulations is added to this class hierarchy, tests cases that run a small simulation and validate the output against predefined expected values are also created. Examples of simple predefined expected value include the number of particles in the system or the number of bonds in the system. More complex expected values include the average equilibrium bond distance of the system. Simple tests can potentially catch coding errors that would break the system and cause exceptions when deployed in production runs. These errors caught right at the time of origination, are easier to fix because the code change that caused the error is still fresh in the developer's mind. Furthermore, these tests are automatically executed when the code is committed to the code repository. This practice of running tests on every incremental code change to make sure that the system has not broken is known as "continuous integration".

2.6.2 epoxpy-flow

Most materials discovery projects such as the present research require running hundreds or even thousands of simulations per study which need to be executed parallelly on supercomputers with many computing nodes. Two components are necessary

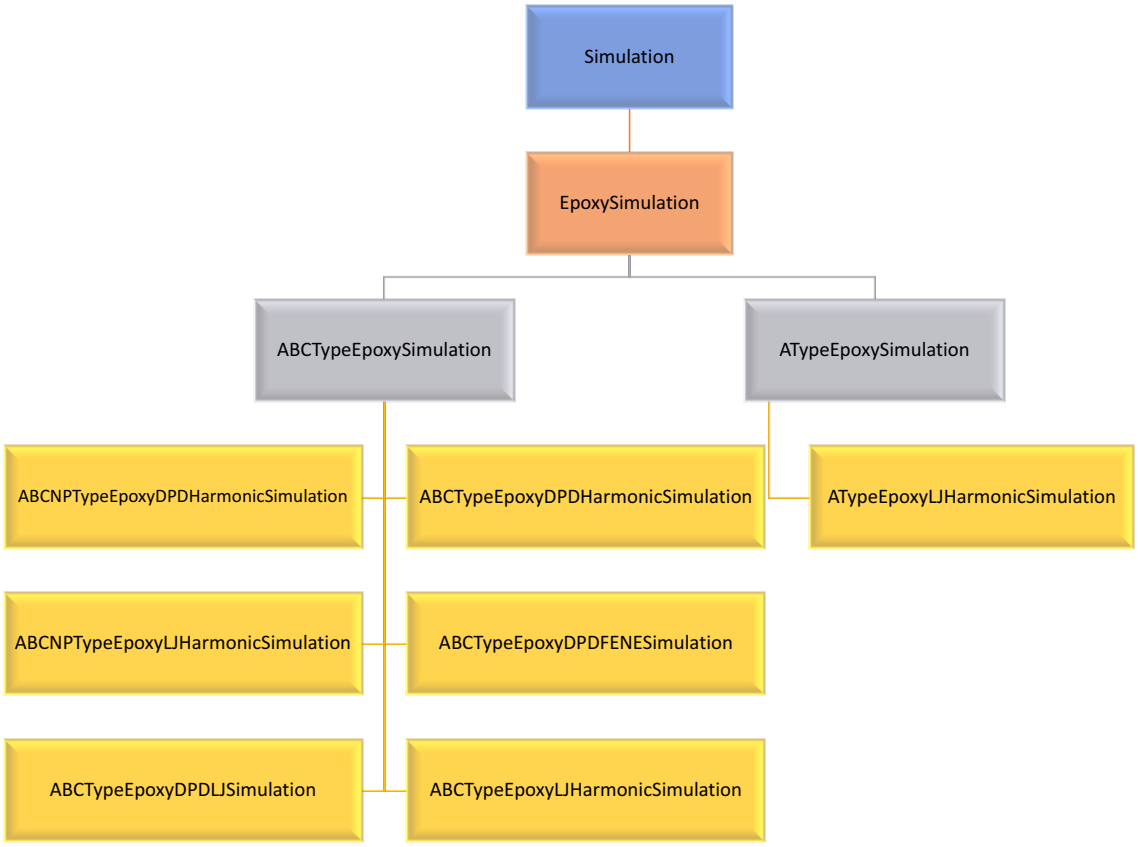


Figure 2.6: The class hierarchy in `epoxpy`. Note that only the yellow classes can be instantiated as objects and the others are all abstract classes.

for this workflow to be seamless. The first one is an efficient data management framework to store, manage and query the data generated by the simulations. The second one is a job submission framework that can create, submit and manage jobs on supercomputers. `epoxpy-flow` is a **python** code that leverages a data management and job management framework called `signac` and `signac-flow` respectively to achieve both of these tasks. Apart from creating jobs for large parameter sweeps, and submitting them, `epoxpy-flow` is also responsible for performing post-processing operations on the simulation output so that data analysis is done in a somewhat decentralized fashion. The output of the postprocessing is typically the aggregate

data obtained from a single simulation. This data includes information such as the mean temperature, pressure, diffusivity, gel point and final cure fraction and are typically stored in a file format known as the JSON (JavaScript Object Notation).

2.6.3 `pandas` and `jupyter notebooks`

Even though most of the post-processing is performed remotely, the data analysis is performed locally by copying the aggregate data in JSON format. An easy to use web-based interactive python editor called `jupyter` notebook is used to write data analysis and visualization code. A **python** package called `pandas` is used to convert the raw JSON data into a table-based data structure called `DataFrame` with which it is very convenient to perform extensive data analysis. All of the data analysis and visualizations produced in this research is obtained using the procedures described here.

CHAPTER 3

DEVELOPING EFFICIENT METHODS FOR REACTION MODELLING OF EPOXY CROSSLINKING[a]

3.1 Introduction

Epoxy thermosets are widely used in industrial applications as adhesives and coatings[3], for encapsulated electronics[3], and as matrices for advanced carbon fiber composite materials[95,104]. The widespread use of epoxies derives from the low-cost precursor components and the ease with which they may be cured into materials with high chemical resistance, high strength, and low density[115]. During cure, epoxy monomers are mixed with monomers of hardening agents, reacting to form a crosslinked network that transforms from a liquid to a gel, and finally to a vitrified glass phase[72,74,110]. The highly crosslinked topology of the epoxy-hardener network gives the thermoset excellent hardness and thermal stability, but with low ductility and low fracture toughness[61]. In order to enhance the fracture toughness, thermo*plastic* toughening agents are added[11,20,43,61,72]. Cure-induced phase separation of toughener from reacting epoxy and amine suggests that thermoset morphology depends on how fast polymerization-induced phase separation occurs in relation to glassy vitrification[117].

[a]This chapter is published in the Journal of Theoretical and Computational Chemistry and is referenced as "Thomas, S., Alberts, M., Henry, M. M., Estridge, C. E. & Jankowski, E. Routine million-particle simulations of epoxy curing with dissipative particle dynamics. J. Theor. Comput. Chem. 0, 1840005 (2018). https://doi.org/10.1142/S0219633618400059"

To engineer composites from toughened epoxy thermosets with customizable mechanical properties, we require a fundamental understanding of how the cured morphology depends on its ingredients and how it was processed.

Understanding how to control epoxy morphology is important because the mechanical properties and reliability of parts made from epoxies depend sensitively on their microstructure[13,117]. Raghava studied the effects of poly (ether sulphone) (PES) molecular weight on phase separation in a tetrafunctional epoxy resin cured with aromatic anhydrides and concluded that the phase separation of the toughening agent from the epoxy matrix was a minimum condition for improved fracture toughness of the thermoset matrix[87]. At weight fractions of 10% PES toughener in a biphenyl epoxy resin, Mimura et al.[71] observed semi-continuous phase separated networks with PES domain sizes of 50-80 nm which corresponded to a 60% increase in fracture toughness compared to the neat epoxy resin. At 20% PES weight fraction a continuous interpenetrating network with domain sizes of 1 μm was formed and the fracture toughness was observed to be 90% greater than the neat epoxy[71]. The differences in the fracture toughness of toughened thermosets have been attributed to the phase separated morphology[11,20,43,72]. Domains ranging from 5 nm to 12 μm have been observed and a number of studies have found that the largest increase in fracture corresponded to a co-continuous interpenetrating network morphology of thermoset and thermoplastic[13,52,71,117,121].

In addition to the composition of toughened epoxy blends, the processing pathway of the material has a significant impact on phase separation of toughening agents, making the ingredient-processing-performance parameter space complicated. Zhang et al. studied the effects of heating rate during cure on the morphology of PES toughened multifunctional epoxy systems and observed as the heating rate is increased

the diameter of microphase-separated PES domains increased from 9.67 μm to 11.41 μm[123] which resulted in an 86% increase in fracture toughness. However, the sample heated at a low heating rate and smaller PES domain size had a much lower degree of cure. Exploring this landscape through synthesis and processing of these materials is costly and labor-intensive and points to a clear need for predictive capabilities to help narrow the scope of viable materials and processes to meet targeted materials performance.

In principle, computer simulations should be able to assist in the exploration of processing protocols, but in practice, it is challenging to predict epoxy morphology because of the disparate time scales and length scales that matter. Using atomistic models to represent toughened epoxy thermoset structures is impractical because tens of millions of atoms are needed to represent structures on 100 nm length scales[13,52,65,71,117,121]. Recent atomistic simulations using ReaxFF and LAMMPS modeled crosslinking polymers networks of 4,284 atoms for which mechanical properties were calculated[77,108]. The cubic volumes in these simulations are around 4 nm long. More efficient polymer-specific schemes such as Polymatic[4] and template-based polymerization[31] have been devised to tackle the issue of high computational cost of atomistic reaction modeling. Both of these models generated crosslinked networks of hundreds of reactive units where system sizes reach a few nm, but these length-scales are far from the experimentally relevant length-scales (100's to 1000's of nm). These models also require customization for simultaneous diffusion dynamics. For epoxy microstructure simulations, we, therefore, require more coarse-grained models.

Two types of coarse-grained models have been used to model epoxy curing. The first type involves mapping specific chemical moieties within a monomer to coarse-grained beads such that a single monomer may be represented by more than one

coarse-grained simulation elements, also known as "beads" in the context of polymer science. Yang et al.[118,119] represented a tetra-functional epoxy phenol novolac (EPN) monomer and bisphenol-A (BPA) monomer using an 8 bead and 3 bead CG model respectively. Komarov et al.[56] simulated the curing of cycloaliphatic epoxy resin (CAER) where the epoxy monomer and curing agent monomer were represented by a 7 site and 3 site CG model respectively. The coarse-grained beads in these models typically use an LJ-like non-bonded interaction. The nature of these "hardcore" models makes it suitable to study mechanical properties of cured epoxies since it allows for entanglements[24]. However, this very nature of "hardcore" models also makes them difficult for modeling reaction-induced phase separation (RIP) of toughened epoxies due to energetic traps that prevent phase separation. The second type of coarse-graining involves mapping entire monomers to coarse-grained beads. These models have typically used either an LJ-like potential[76] or Dissipative Particle Dynamics (DPD)[65,66] for the non-bonded interactions. The DPD potential[34,39] models fluidic elements which can pass through each other making it suitable for modeling RIP in toughened epoxies. Liu et al. developed DPD simulations with stochastic bonding routines with 248,832 coarse-grained simulation elements, and achieved cures of around 80% in 1×10^6 steps[65]. Stochastic bonding routines have been successfully applied to polystyrene polymerization, where thermostat sensitivity to the bonding model was observed[66]. Li et al.64 takes a similar stochastic reaction approach in DPD, but shows that the conversion profiles in simulations are orders of magnitude too fast with respect to experiments. Langeloth et al. achieve nearly 80% cure with CG simulations accessing as much as 32×10^{-9} s and 10 nm length-scales[58,59]. Free radical living polymerization reaction kinetic sensitivity to bonding rates is shown in systems of 24,000 DPD spheres[120]. Kacar et al.[49] performed DPD simulations with

108,062 particles for 8×10^5 steps and in a later work[50] the same authors achieved 92% crosslinking. In short, reactive models of epoxies are approaching 10-100 nm lengths and experimental cure fractions, but additional work is needed to simultaneously resolve reaction and diffusion dynamics for systems with more than a few hundred thousand particles.

To maximize experimental relevance, it is desirable for epoxy curing simulations to (1) represent dozens, if not thousands, of nanometers, (2) simultaneously model reaction and diffusion, (3) model experimental temperature-time curing profiles, and (4) allow high-throughput screening of thousands of experiments per week. Atomistic simulations cannot meet criterion 1. Criteria 2 and 3 can be met by improving or extending reaction models with mesoscale methods, and is the focus of the present work. Criterion 4 is desirable because isolated simulation trajectories are not adequate for studying nonequilibrium dynamics with equilibrium-based techniques such as DPD. That is, we require high-throughput simulations that enable calculations of uncertainties in simulated results and efficient evaluation of large parameter spaces to validate models and inform engineering processes.

In this work, we implement an open-source plugin to HOOMD-Blue[8,32] that enables high-throughput simulation of crosslinking epoxy thermosets. HOOMD-Blue is a molecular dynamics engine written in C++ and CUDA (A GPU programming Application Programming Interface (API)) with an easy to use python API. This allows users to leverage the easy to use nature of Python and the speed of graphic processing units. We fully describe our crosslinking algorithm and provide access to our plugin's source code. We characterize how different bonding rates influence the overall bonding kinetics and give guidelines for matching experimental rates. We model a classic toughened epoxy thermoset diglycidyl ether of bisphenol A (DGEBA[1])

epoxy with amine hardener 4-4'-diaminodiphenyl sulphone (DDS4-4'[2]), and PES toughener and demonstrate its morphology dependence on processing. In sum, we present a new computational tool that enables for the first time high throughput simulations representing millions of atoms over tens of millions of steps that achieve over 95% cure in a few hours.

3.2 Methods

We implement an open-source (GNU General Public License v3.0) dynamic bonding plugin that stochastically adds epoxy-amine bonds during dissipative particle dynamics performed with the HOOMD-Blue simulation engine. The source code is available at `https://bitbucket.org/cmelab/epoxpy`[103].

3.2.1 HOOMD-Blue

High-throughput molecular simulations now routinely leverage graphics processing units (GPUs) to parallelize and therefore speed up computational bottlenecks. Packages including HOOMD-Blue, LAMMPS, and AMBER have demonstrated speedups between 2x to 10x, depending on which systems are used as benchmarks and how many core kernels are parallelized[8,9,33,106]. We use HOOMD-Blue to perform the DPD simulations implemented here. HOOMD-Blue is used here for its combination of performance, accessibility, and extendibility. After initializing on the CPU, HOOMD-Blue simulations with thousands to millions of simulation elements can easily be performed on a single modern GPU (e.g., NVIDIA Tesla K20 or P100) with negligible communication to the host CPU. This ability to perform large-scale simulations on a single hardware accelerator is favorable for high-throughput simulation studies on

modern supercomputers. Modern supercomputers with multiple GPUs per CPU enable multiple, asynchronous molecular simulations to be performed in parallel on a single node. Because HOOMD-Blue is an importable python module the scientific computing capabilities of other python libraries (e.g. `numpy` and `scipy`) are easily leveraged for structuring simulation set-up and analysis.

While python enables quick implementation of complex modeling ideas, oftentimes performance improvements to python routines can be realized by using machine code optimized and compiled for specific hardware. HOOMD-Blue's plugin API makes it relatively straightforward to add C++ or CUDA routines that impose constraints or add functionality to molecular simulations. In this work, we describe performance improvements necessary for high-throughput simulations of reacting epoxies via python and C++ implementations of our dynamic bonding algorithm.

3.2.2 Dissipative particle dynamics

We model reacting mixtures of DGEBA, DDS and PES toughener using coarse-grained representations and dissipative particle dynamics[39,65]. Difunctional DGEBA epoxides are modeled with a single coarse-grained simulation element ("bead"), as are tetrafunctional amine molecules, and monomers of PES (Figure 3.1). Each bead is a spherical simulation element of the same size (diameter = 1σ). Here we consider PES chains of 10 repeat units. Throughout this work, we will use the colors red, blue, and green to distinguish these three chemical species, respectively (A=amine=red, B=epoxy=blue, C=toughener=green). Throughout this work we consider equifunctional blends of amine and epoxy, with one 10-mer chain of C per 10 beads of A, so the overall species ratios A:B:C are 1:2:2, or 20% A, 40% B, and 40% C.

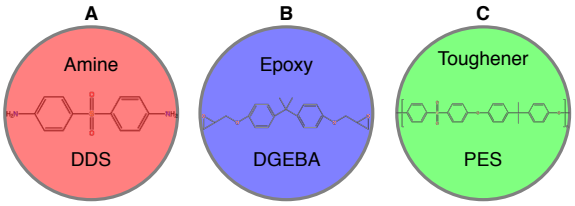


Figure 3.1: Amine, epoxy, and toughener monomers are represented with spherical simulation elements ("beads").

The DPD implementation in HOOMD-Blue[82] provides parallel force calculations and position integrations of the method originally developed by Hoogerbrugge and Koelman[39]. The force on bead i from neighbors j (where $r_{ij} \leq 1$) depends on three types of forces (Equation 3.1).

$$F_i = \sum_{i \neq j} F_{ij}^C + F_{ij}^R + F_{ij}^D \tag{3.1}$$

The conservative force

$$F_{ij}^C = a_{ij}\omega^C(r_{ij})\hat{\mathbf{r}}_{\mathbf{ij}} \tag{3.2}$$

is a soft repulsive force along the center-to-center vector $\hat{\mathbf{r}}_{\mathbf{ij}}$, where $\mathbf{r_{ij}} = \mathbf{r_i} - \mathbf{r_j}$, $r_{ij} = |\mathbf{r_{ij}}|$ and $\hat{\mathbf{r}}_{\mathbf{ij}} = \mathbf{r_{ij}}/|\mathbf{r_{ij}}|$. The force has a maximum value of a_{ij} and ω^C is a weight function, typically

$$\omega^C(r_{ij}) = \begin{cases} 1 - \frac{r}{r_{ij}} & (r \leq r_c) \\ 0 & (r > r_c), \end{cases} \tag{3.3}$$

where $r_c = 1$ is the bead center-to-center cutoff distance past which beads do not interact. This linearly ramping soft repulsive force makes it easy to initialize random configurations for polymer systems and allows for relatively large timesteps ($dt \leq$

0.04)[34].

The random force

$$F_{ij}^{R} = \varsigma\omega^{R}(r_{ij})\zeta_{ij}\Delta t^{-1/2}\hat{\mathbf{r}}_{\mathbf{ij}} \tag{3.4}$$

models random fluctuations due to thermal noise and the dissipative force

$$F_{ij}^{D} = -\gamma\omega^{D}(r_{ij})(\hat{\mathbf{r}}_{\mathbf{ij}}.\mathbf{v}_{\mathbf{ij}}) \tag{3.5}$$

models viscous drag. The amplitudes ς and γ of the random and viscous forces, respectively, are related to each other by the fluctuation-dissipation theorem $\varsigma^2 = 2\gamma k_B T$[25].

Here we determine the repulsion parameters a_{AA}, a_{AB}, a_{AC}, a_{BB}, a_{BC}, and a_{CC} from atomistic molecular dynamics simulations. Solubility parameters (δ)

$$\delta_i = \sqrt{E_{coh}^{i}/V_i} \tag{3.6}$$

are calculated from the cohesive energy density E_{coh} and specific volume V_i of molecules in atomistic NPT simulations equilibrated at 11 temperatures ranging from 273 K to 600 K (Appendix A). These data are used to solve for the Flory-Huggins interaction parameters

$$\chi_{ij} = \frac{\overline{V}}{k_B T}(\delta_i - \delta_j)^2 \tag{3.7}$$

and the DPD interaction parameters

$$a_{ij} = \frac{75 k_B T}{\rho_n} + \Delta a \tag{3.8}$$

via an empirical relationship

$$\chi_{ij} = 0.286\Delta a \tag{3.9}$$

determined for number density ρ_n=3, which we also employ here[34].

The morphologies obtained with all-atom MD simulations using the OPLS-2005 force field are then compared with that of the coarse-grained DPD model for validation[68]. The DPD interaction parameters averaged over the temperatures sampled between 273 K and 600 K are used here (Table 4.2). The mass unit M = 278.82

Table 3.1: Unitless repulsion parameters a_{ij} for amines (A), epoxies (B), and toughener (C) beads determined by Hildebrand solubility parameters from atomistic molecular dynamics.

	A	B	C
A	25.000	30.729	25.003
B		25.000	30.532
C			25.000

g/mol is calculated from the weighted average of the masses of the A, B and C beads $\overline{M} = M_A * \phi_A + M_B * \phi_B + M_C * \phi_C$, where M_A = 248.3 g/mol, M_B = 340.42 g/mol, and M_C = 232.46 g/mol. Ratios of A:B:C are 1:2:2 throughout this work.

The average volume $\overline{V}$ is calculated as $\overline{M}/\overline{\rho}$ for the temperature of interest. The length scale is calculated as L= $(\overline{V}\rho_n)^{1/3}$ where ρ_n is the reduced number density of beads. The energy unit k_BT used here corresponds to T_C = 439 K. The fundamental units of energy, mass and distance are 0.873 kcal/mol, 272.82 g/mol, and 1.06 nm, respectively, where the calibration temperature T_C=439.36 K. The derived units of time (τ) and force (F) are 9.29 ps and 5.7e-12 N respectively.

We use `signac` and `signac flow` for data collection and job submission[5,6]. DPD simulations are initialized using `mBuild`[54] followed by a 5000-step NVE simulation at 1760 K to generate unique random configurations for each run. Velocity distributions

consistent with the starting temperature of the desired run are then set. Bonds between toughener monomers and between epoxies and amines are modeled with harmonic springs, with $k_{\text{harmonic}} = 4k_BT/r_c^2$ and equilibrium spacing $r_0 = 0$ as in Ref. 42 and Ref. 41. We employ velocity-Verlet integration of Newton's equations of motion ($dt = 0.01$) in the NVE ensemble[99]. Unless otherwise stated, the fiducial simulation parameters listed in Table 3.2 describe each simulation in this work.

Table 3.2: Fiducial simulation parameters

N	50,000
L_B	25.54 σ
dt	0.01 τ
ρ_n	3
γ	4.5 M/τ
k_{harmonic}	4 k_BT_C/r_c^2
r_0	0 σ
n_B	0.000025 N_B
N_B	40,000
τ_B	1 dt
E_a	1 k_BT_C

3.2.3 Reaction model

Epoxy-amine crosslinking is modeled by the addition of bonds during DPD simulations, similar to the method of Liu et al. (Algorithm 1)[65]. Every τ_B steps we call the bonding routine, wherein a fraction $I = \frac{n_B}{N_B}$ of the number of the total creatable bonds N_B are added (Figure 3.2). The bonding reactions occur stochastically as in Refs. 65 and 66, where the probability of forming a bond

$$p(E_a) = e^{-\frac{E_a\beta}{k_BT}} \tag{3.10}$$

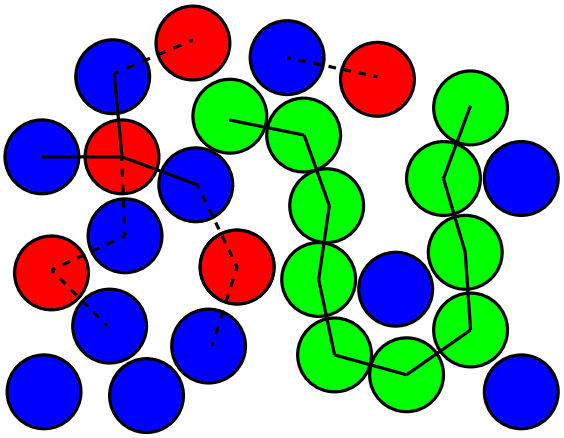


Figure 3.2: During a bonding step, candidate bonds (dashed lines) are stochastically converted to bonds (solid lines) between amine/epoxy pairs that have reactive sites remaining and are sufficiently close.

depends on the activation energy E_a. Here, β describes the relative activation energy of secondary (and higher order) reactions to primary bond formation

$$\beta = \begin{cases} 1 & \text{if } [R(p_i) \; or \; R(p_j)] < 1 \\ \beta_2 & \text{otherwise} \end{cases} \tag{3.11}$$

where $R(p_X)$ gives the bond rank of particle p_X. Here, $\beta_2 = 3$.

The *dist* function in line 8 gives the distance between the particles p_i and p_j. *random* in line 10 produces a uniform random number between 0 and 1.

3.2.4 Enthalpy of Reaction[b]

The simulated curing data in Ref. 102 is known to fit the first order reaction model, whereas experimentally, it is known that the curing profile fits the self-accelerated second order reaction model the best[7]. It is experimentally known that the curing

[b] This section is not published in Ref. 102

Algorithm 1 Amine-Epoxy bonding

```
1: n_B ≥ 1, τ_B ≥ 1
2: repeat every τ_B time steps
3:     for each bond attempt i in n_B do
4:         p_i is a randomly chosen particle of type A or B
5:         if p_i can bond then
6:             Distance sort neighbors of complimentary type to p_i
7:             for each neighbor p_j do
8:                 if p_j can bond and dist(p_i, p_j) < r_bond then
9:                     Calculate p(E_a, β) using Equation 3.10 and Equation 3.11
10:                    if p(E_a) > random(0, 1) then
11:                        Bond p_i and p_j
12:                        break
13:                    end if
14:                end if
15:            end for
16:        end if
17:    end for
18: until t == t_end
```

reaction of epoxides with amines is exothermic. The model described in Ref. 102 however does not capture the heat release during reaction and could potentially explain the different cure kinetics. In this work, the effects of adding heat of reaction on the resultant cure kinetics is studied by adding this functionality to the `epoxpy`[102] package.

This approach has been explored previously[78] with AAMD curing simulation of epoxies where the heat of reaction raises the kinetic energy of the products, causing a local increase in temperature. ReaxFF has previously been used to detect the reaction mechanism from the enthalpy change during the exothermic part of a simulated decomposition reaction of Nitromethane[85] where the enthalpy change was calculated as $\Delta H = -\Delta U_{exo} + V(P(t_\infty) - P(t_{max}))$. ΔU_{exo} is the difference between the equilibrium potential energy at $t = \infty$ and the potential energy at the beginning

of exothermic reaction phase which occurs at $t = t_{max}$ and $P(t_\infty) - P(t_{max})$ is the pressure change during the reaction.

In this work, we take a more coarse-grained approach where heat of reaction causes a global increase in temperature rather than a local increase of kinetic energy of the products. This approach assumes that the released heat gets dissipated to the system before the next time step in the simulation and this assumption is valid for coarse grained systems such as the current work where the time unit is in the order of pico seconds. Another important consideration is about the temperature of the heat bath, simulated by the thermostat. While the reaction increases the system temperature, the thermostat tries to bring it back to the temperature of the heat bath and this can cause ringing temperature profiles which can be very large during early stages of curing. This can potentially affect the expected behavior of the thermostat. In order to avoid this effect, the heat released is also added to the heat bath. We use a rescale thermostatting approach to cause an instantaneous increase in the system temperature as well as an update of the energy of the heat bath. The rescale thermostat[92] increases the instantaneous temperature of the system T to T' by scaling the instantaneous velocity of the particle v_i to $v_i^{'}$ as $v_i^{'} = \lambda v_i$ where $\lambda = \sqrt{\frac{T'}{T}}$. Here $T' = T + \Delta T$ where ΔT is defined as $\sum^{N} \Delta T_{rxn}$ where N is the total number of bonds made and ΔT_{rxn} is the change in temperature due to one reaction.

3.3 Results

We perform DPD simulations of reacting epoxy thermosets with three aims. First, we identify the minimum system size necessary to observe toughener phase separation using simulated scattering experiments. Second, we vary A, a parameter that controls

reaction rate (described in section 3.3.2), to determine which bonding frequencies best match experimental reaction kinetics. Third, we test two curing protocols and demonstrate morphological sensitivity to processing.

We also profile the performance of three implementations of Algorithm 1 to optimize its performance while attempting the least amount of coding. After optimization, simulations with $N = 5 \times 10^4$ achieve 95% cure in about 45 wall-clock minutes and the $N = 2 \times 10^6$ simulations achieve 95% cure in about 7.5 wall-clock hours. We note that this ability to simulate large simulation volumes in reasonable amounts of time enabled the identification of a minimum system size necessary to observe microstructural features using this model. A detailed description of performance profiling and optimization strategies we employ are given in the Supplementary Information (SI).

3.3.1 Morphology Characterization

We characterize the degree of toughener phase separation by inspecting the C-C structure factor at low wave number, calculated using `diffractometer` from Ref. 46. Five independent replicate simulations with system sizes between 5×10^4 and 3×10^6 are run using the fiducial parameters (Table 3.2) to obtain these C-C structure factors.

Figure 3.3 summarizes our finding that large simulation volumes are needed to observe the length scales over which phase separation occurs by comparing C-C structure factors. In all seven cases 95% cure fraction is achieved, but the ≈ 35 nm-wide toughener domains in the $N \geq 1 \times 10^6$ simulations appear as macrophase separation occurring in the $N \leq 5 \times 10^5$ cases where the simulation volume is not large enough to resolve 35 nm features. Figure 3.4 reveals the average feature size ($\langle q_{max} \rangle$) of

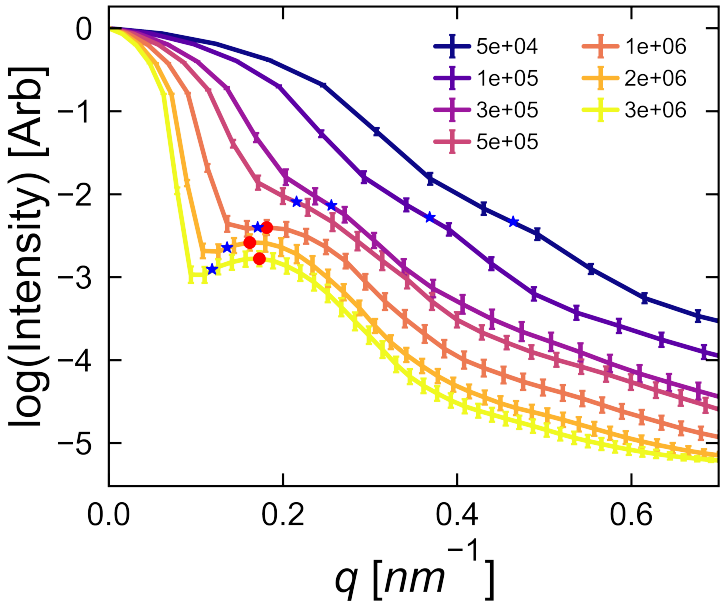


Figure 3.3: C-C structure factors show ≈ 35 nm toughener domains emerge for $N > 1 \times 10^6$ system sizes, while $N = 5 \times 10^4$ systems demonstrate macrophase separation. Blue stars indicate the wavenumber corresponding to half the box length (the largest resolvable length scale with a periodic simulation volume), and red dots indicate local scattering maxima corresponding to the phase-separated feature size. The error bars indicate standard error.

this system to be 0.17 nm^{-1} or 37 nm. In order to resolve this feature size using the `diffractometer`, it is necessary to have a system with $L_B/2 \geq 37$ nm which is satisfied by $N \geq 1.2 \times 10^6$. The fact that this feature size remains intact over a large range of system sizes indicate that this is a characteristic size of this model and not a simulation artifact. These results show that co-continuous domains of thermoset and thermoplastic can be efficiently simulated with the coarse-grained model and bonding algorithm used here. We pause to emphasize that these particular feature sizes and morphologies are not meant to be predictive for DGEBA/DDS/PES, because the model used here lacks key features specific to those chemistries, but serves as a

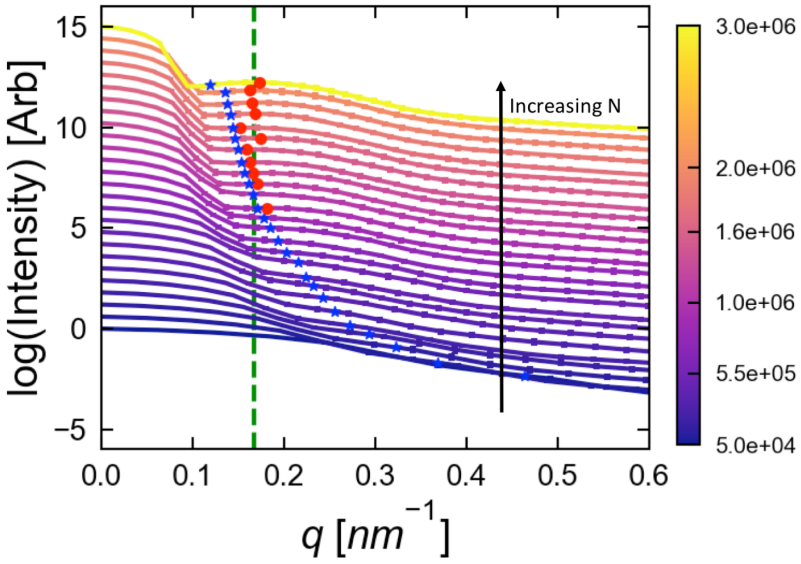


Figure 3.4: C-C structure factors for $N = 5 \times 10^4$ to $N = 3 \times 10^6$ show that microstructure can consistently be detected for $N \geq 1.2 \times 10^6$. The blue stars indicate the wave vector corresponding to the half box length and the red dot indicates the detected first peak. The structure factor intensities are shifted up in intensity for visibility. The average wave vector corresponding to the characteristic feature size ($\langle q_{max} \rangle = 0.17 nm^{-1}$) is shown in dotted line and the color bar indicates N.

qualitative validation that features important to epoxies are accessible. The 37 nm toughener domains observed here are a factor of 40 smaller than the micron-scale domains observed in some experiments but represent the largest domains observed to date in reactive DPD models. This discrepancy in system sizes for microstructure detection reinforces the importance of large-scale volumes for predicting these morphological features.

3.3.2 Calibration of Reaction Kinetics

Simulation of bonding dynamics with coarse-grained models requires the simulated reaction kinetics to be matched to experimental timescales. The reaction rate constant

$$k = He^{\frac{-E_a}{RT}} \tag{3.12}$$

depends on the reaction activation energy E_a and prefactor H. Due to the accelerated dynamics in coarse-grained models, it is not necessarily the case that experimentally-determined E_a and H will model the desired kinetics in a model trajectory. The degree of cure α measures the fraction of possible bonds that have formed in a curing epoxy, and its rate of change

$$\frac{d\alpha}{dt} = k(t)f(\alpha) \tag{3.13}$$

is modeled by the k and $f(\alpha)$, which is a polynomial in α describing reaction kinetics[7]. For example, $f(\alpha) = \alpha_\infty - \alpha$ for first-order (FO) reaction kinetics, $f(\alpha) = (\alpha_\infty - \alpha)^2$ for second-order (SO), $f(\alpha) = (\alpha_\infty - \alpha)(1 + C\alpha)$ for self-accelerated first-order (SAFO), and $f(\alpha) = (1-\alpha)(\alpha_\infty-\alpha)(1+C\alpha)$. Here, α_∞ is the degree of cure at $t = \infty$ and C is a temperature-independent acceleration constant. In the reaction model implemented here, the prefactor H is related to the number of bonds we attempt per call to the bonding routine (n_B) and the steps between bonding routine calls (τ_B).

To determine reaction sensitivity to the ratio $A = \frac{n_B}{\tau_B}$, we perform high-throughput DPD simulations with dynamic bonding. Isothermal curing simulations are performed until $\alpha_\infty = 0.95$ is achieved, using the fiducial parameters, except n_B, τ_B and k_BT are varied. The number of bonds to attempt per bonding-step is expressed as a fraction of the total number of bonds (N_B) that can be formed from initial concentrations of amines and epoxy monomers. Here, $n_B \in \{0.000025N_B, 0.00005N_B, 0.0001N_B, 0.01N_B\}$, $\tau_B \in \{1, 2, 10, 20, 40, 80, 100\}$, and $k_BT \in \{0.2, 0.5, 1, 2, 3, 4, 5, 6\}$, for a total of 224 parameter combinations. At each parameter combination, we perform twenty repeat trials, for a total of 4480 calibration simulations performed over three weeks. After

each simulation completes, the cure profile $\alpha(t)$ is fit with FO, SO, SAFO, and SASO models and the mean square deviation R^2 is calculated for each of the four models. A representative cure profile and associated amine concentrations is shown in Figure 3.5.

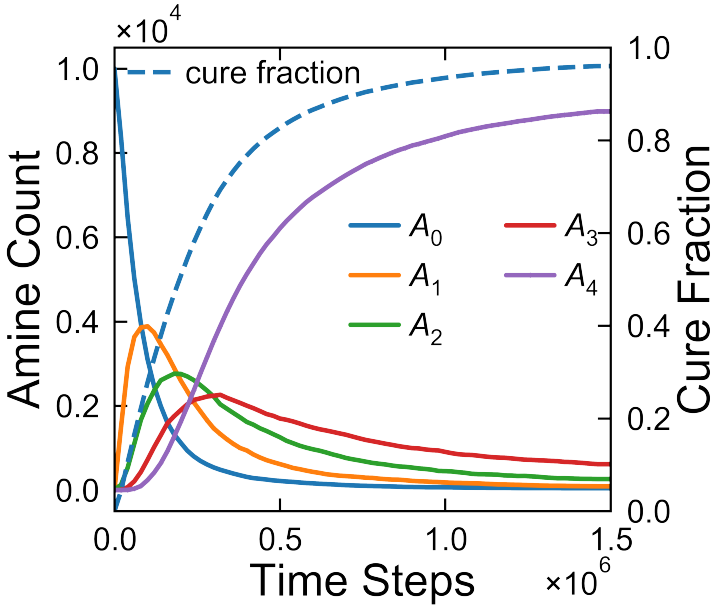


Figure 3.5: Representative cure fraction (dashed) and amine concentration trajectories. A_n indicates an amine with n formed bonds.

The Python plotting library `Matplotlib` is used to generate the plots within this work[40]. For a given bonding rate A we average the R^2 across the simulated temperatures to get an aggregate measure $\langle R^2 \rangle$ for how well each kinetic model matches simulated reaction kinetics. Figure 3.6 summarizes kinetic model sensitivity to bonding rate A.

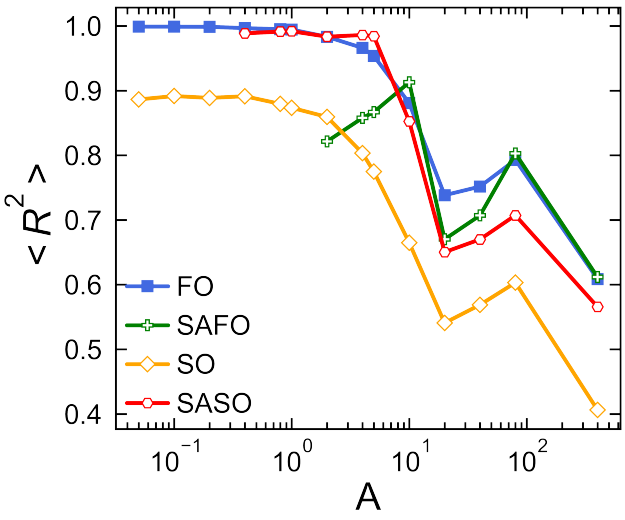
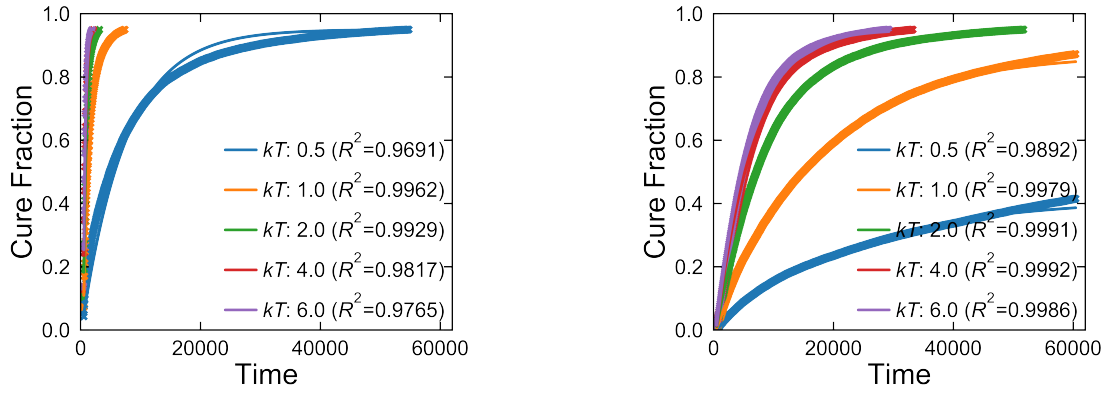


Figure 3.6: Average fit metric $\left\langle R^2 \right\rangle$ for the four kinetic models as a function of bond frequency A= $\frac{n_B}{\tau_B}$.

We find the first-order kinetic model matches our simulation results near perfectly, and has the best fit when $A \leq 1$. It is expected that the FO model would best fit our reaction model because we do not model heat release with each formed bond and because equimolar ratios of unbonded A and B are maintained as the reaction proceeds. We note that experimentally, we would expect the exothermic reactions of the amine and epoxy modeled here to give rise to SASO kinetics[7]. With the exception of the SASO model, the general trend is towards higher accuracy fits with lower A. The observation that simulated bonding kinetics so sensitively depend on A suggests that coarse-grained simulations of dynamically crosslinked epoxies require characterization and justification of stochastic bond frequencies. The sensitivity of bonding kinetics to A is highlighted between Figure 3.7 (a) and Figure 3.7 (b), where changing A from 2 to 0.1 causes cure fractions to drop significantly over the same time scale.



(a) A = 2.0 (b) A = 0.1

Figure 3.7: FO model fits of simulated α_T for (a) A = 2.0 and (b) A = 0.1. Reducing A increases R^2 and decreases α_T.

For the first order reaction model, we find a general trend of improved accuracy with smaller A (Table 3.3). When $A \leq 1$, the average quality of fit is greater than 0.9945, but rapidly decreases for $A > 1$. We find that the accuracy of the FO model is best for temperatures close to the calibration temperature $0.5k_BT_C < k_BT < 4k_BT_C$. We note that the optimal A values obtained in this study are sensitive to the time step (dt). Further studies will aim to explore the sensitivity of A to dt. Again we emphasize that $A < 1$ is not necessarily the optimal choice for modeling DGEBA/DDS crosslinking, but that reaction kinetics can be calibrated to desired experimental kinetics with high throughput simulations.

A fundamental challenge with using DPD to simultaneously model reaction and diffusion arises from the fact that particles diffuse as fast as momentum, rather than a factor of 1000 slower[34], so simulation timescales derived from mass, distance, and energy are not straightforward to interpret.

Table 3.3: Fit quality (R^2) for the FO model

A	k_BT	Best τ_B	Best n_B (Fraction of N_B)	H	R^2	$< R^2 >$
	0.5	40	1.0e-4	0.0164	0.9823	
	1.0	40	1.0e-4	0.0081	0.9979	
0.1	2.0	20	5.0e-4	0.0090	0.9991	0.9954
	4.0	10	2.5e-5	0.0100	0.9992	
	6.0	20	5.0e-5	0.0103	0.9986	
	0.5	5	5.0e-4	0.0194	0.9793	
	1.0	10	5.0e-4	0.0251	0.9928	
0.4	2.0	10	1.0e-4	0.035	0.9992	0.9932
	4.0	10	1.0e-4	0.0389	0.9979	
	6.0	5	1.0e-4	0.04	0.9970	
	0.5	5	1.0e-4	0.0262	0.9785	
	1.0	5	1.0e-4	0.0491	0.9932	
0.8	2.0	5	1.0e-4	0.0672	0.9983	0.9917
	4.0	5	1.0e-4	0.0745	0.9954	
	6.0	5	1.0e-4	0.0764	0.9932	
	0.5	1	2.5e-5	0.0297	0.9759	
	1.0	1	2.5e-5	0.0611	0.9947	
1.0	2.0	1	2.5e-5	0.0830	0.9978	0.9907
	4.0	1	2.5e-5	0.0910	0.9943	
	6.0	1	2.5e-5	0.0940	0.9910	
	0.5	1	5.0e-4	0.0495	0.9691	
	1.0	1	5.0e-4	0.1155	0.9962	
2.0	2.0	1	5.0e-4	0.1538	0.9929	0.9832
	4.0	1	5.0e-4	0.1662	0.9817	
	6.0	1	5.0e-4	0.1691	0.9765	
	0.5	100	1.0e-2	0.0943	0.9776	
	1.0	1	4.0e-2	0.2142	0.9959	
4.0	2.0	100	1.0e-2	0.2650	0.9774	0.9661
	4.0	100	1.0e-2	0.2743	0.9464	
	6.0	100	1.0e-2	0.2729	0.9333	
	0.5	80	4.0e-2	0.1165	0.9791	
	1.0	80	4.0e-2	0.2555	0.9943	
5.0	2.0	80	4.0e-2	0.3114	0.9646	0.9537
	4.0	80	4.0e-2	0.3140	0.9260	
	6.0	80	4.0e-2	0.3151	0.9045	
	0.5	40	4.0e-2	0.2157	0.9879	
	1.0	40	4.0e-2	0.4407	0.9785	
10.0	2.0	40	4.0e-2	0.4646	0.8940	0.8808
	4.0	40	4.0e-2	0.4429	0.7945	
	6.0	40	4.0e-2	0.4293	0.7490	

The ability to independently tune reaction timescales with bonding frequency A and diffusion timescales with γ offer promise for developing reactive DPD simulations that are at least empirically informed and predictive, if not broadly transferable.

3.3.3 Enthalpy of Reaction[c]

Curing simulations starting at a temperature of 2.0 kT are performed with ΔT_{rxn}= $\{\ 0.0, 1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}\ \}\ kT$ and $A = 1.0$ ($n_B = 2.5 \times 10^{-5}, \tau_B = 1.0$). The temperature history shown in Figure 3.8 indicates the maximum temperature increase during the initial reaction stage and stop increasing as the curing reached completion. The cure fraction as a function of time steps shown in Figure 3.9 with $\Delta T = 1 \times 10^{-5}$ is fitted to the four reaction models FO, SAFO, SO and SASO. The curing data obtained with $\Delta T > 0$ fits the SAFO model the best, indicating that this system is representative of the experimental system when the enthalpy change is being captured. A parameter sweep of the system at $T = \{\ 0.5, 1.0, 2.0, 4.0, 6.0\ \}\ kT$ are performed and the standard error in the quality of fit for the different temperatures are shown in Figure 3.10. The trend in the quality of fit shows a trend where the higher quality of fit drops drastically for temperatures $T > 1\ kT$.

3.3.4 Cure Path Dependence

Two temperature profiles are tested to characterize how the structural evolution of epoxy networks depend on the temperature history during cure (Figure 3.11). Isothermally cured samples are initialized at the cure temperature and maintained there throughout the simulation (1×10^7 steps). Linearly ramped samples are initialized at 300 K and then linearly heated to the final cure temperature over 1×10^7 steps.

[c] This section is not published in Ref. 102

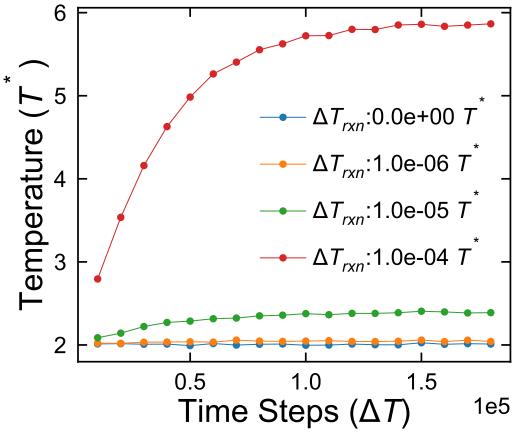


Figure 3.8: Temperature history of the curing simulation with different ΔT_{rxn} where initial temperature is 2 kT.

The bonding is stopped when the sample is cured 95 % (α_{cut}=0.95). The linearly ramped simulation at 850 K reaches this cure fraction at $\approx 5 \times 10^6$ timesteps as seen in Figure 3.11. Curing simulations with each temperature profile are performed with fiducial simulations with the exception of E_a=2 $k_B T_C$ at each of five final temperatures $T \in \{200, 425, 600, 850, 1000\}$ K. Ten independent replicate simulations are performed at each cure temperature for both temperature profiles to quantify uncertainties. As expected, the lower average temperature of the linearly ramped cures results in lower curing at fixed cure time before reaching α_{cut} as seen in Figure 3.11.

In each simulation, we monitor the sizes of the largest and second largest molecules using `NetworkX`[35]. In simulations where the molecular weights of these two molecules diverge, we deem the divergence time the *gel point*. The largest and second-largest molecule sizes are useful metrics for measuring gelation because once a percolating cluster exists it is more likely for clusters to bond to the percolating cluster than to grow independently. Therefore, a divergence in the first and second largest cluster sizes is a good proxy for when a percolating cluster exists. Average final cure

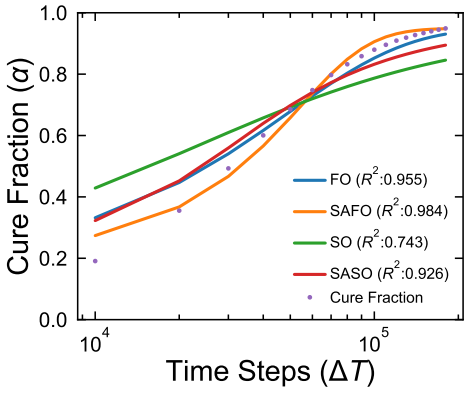


Figure 3.9: Cure fraction as a function of time where the initial temperature is 2 kT and $\Delta T_{rxn} = 1 \times 10^{-5}\ kT$.

percentages and gel points for each temperature are shown in Figure 3.12. Both profiles show $\alpha_{gel} \approx 0.5$ for all temperatures. It is expected that α_{gel} should depend on the functionality and initial concentrations of the reacting molecules, but be independent of temperature and processing.[7,12] Gelation for the chemical species considered in our model is known to occur experimentally around 0.6 cure fraction[12,83] and this property is a consequence of the percolation threshold. The α_{gel} reported here are lower than expected for with DGEBA/DDS blends. Because our current dissipative dynamics do not capture chain entanglements and because we omit the exothermic reaction effects it is not surprising that α_{gel} does not precisely match experiments. At a low temperature of 200 K,the energetic favorability of amines to prefer mixing with tougheners rather than with epoxy (Table 4.2), in combination with the slower diffusion of particles from a fully mixed initial condition, the linear ramp curing resulted in a higher cure fraction (Figure 3.12). Figure 3.13 (b) shows that the isothermal cures give rise to larger feature sizes than the linear ramped case where the toughener completely phase separated from the resin. As expected, the

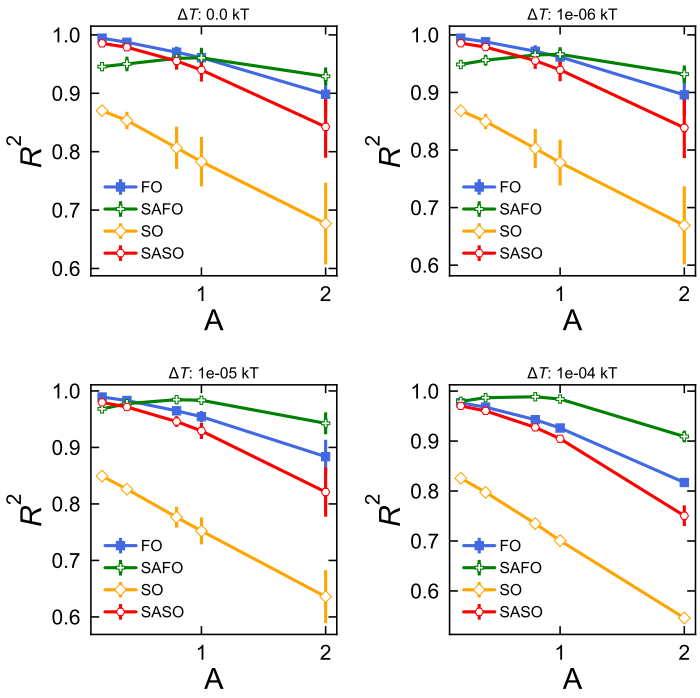


Figure 3.10: Quality of fits for the different reaction models for different values of ΔT_{rxn} show that we get cure kinetics that match the SAFO model with $\Delta T_{rxn} > 0\ kT$. The error bar show standard error in R^2 value for curing temperature $T = 0.5, 1.0, 2.0, 4.0, 6.0\ kT$

standard error of structure factors for the macro phase separated samples in Figure 3.13 are lesser than the samples which are not macro phase separated. We observe differences in how structures evolve between the isothermal and linear ramp cures. At 10% cure, the linearly cured samples have larger feature sizes compared to the isothermally cured samples based on the higher intensity at low wave numbers (Figure 3.13 (a)). This is in contrast to the final structures for the two curing protocols, where isothermal cure results in larger sized features.

The observation that two different temperature histories give rise to different morphologies at the same cure fractions is important because this is a qualitative

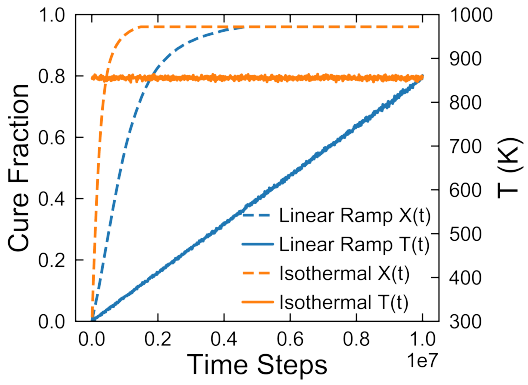


Figure 3.11: Isothermal curing results in higher cure fraction as a function of time during the first half of the simulation at 850 K, while linear ramps allow for more structural rearrangements at the point each cure protocol reaches the same cure fraction.

modeling feature needed to understand via simulations how processing influences structure. The low standard error for the structure factors further reinforces that the temperature histories curing a cure cycle has a strong influence on the resultant microstructure. The ability to set generic temperature-time histories for curing epoxies at 90 nm length scales enables the application of high throughput simulations to this problem of industrial interest. Cure path sensitivities reported here are not expected to hold for DGEBA/DDS/PES systems in particular, though we expect calibrated models using the techniques reported here will advance towards being predictive.

3.4 Conclusions

DPD simulations of millions of reacting particles can be performed with experimentally-relevant temperature profiles in a few hours using `epoxpy` and HOOMD-Blue on K20 and P100 GPUs from NVIDIA. Even though the bonding algorithm is written

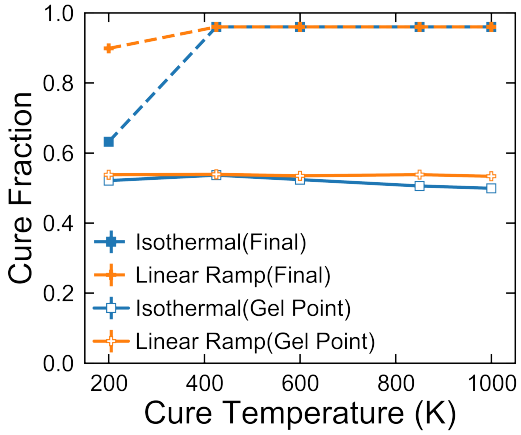
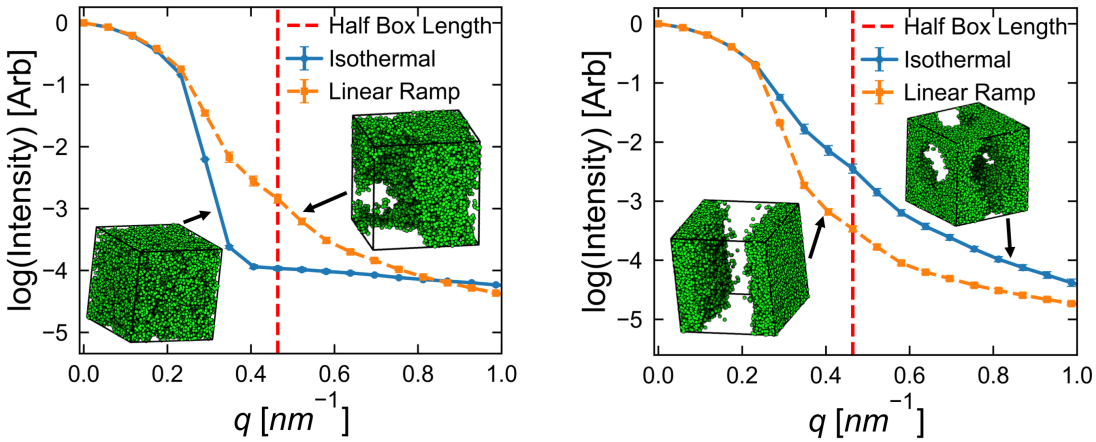


Figure 3.12: Except for 200 K, all the samples reach α_{cut}=0.95. We observe $\alpha_{gel}\approx$ 0.5 for all temperatures.

specifically as a plugin for HOOMD-Blue, it should be fairly straightforward to implement it in other MD tools such as LAMMPS as long they permit adding bonds on-the-fly and provides access to their neighborlist. Given the object-oriented nature of `epoxpy`, extending it for other MD packages is also feasible. With coarse-grained beads representing reactive monomers, million-particle simulations approach the representation of cubic volumes with 100 nm sides, and desired reaction kinetics can be tuned by adjustment of the stochastic bonding rates, an essential validation step. Here we find that to match first-order reaction kinetics, very small bond rates (0.002% of possible bonds) are required. Irrespective of the kinetic model, our findings support the heuristic that low bonding rates are necessary to match cure kinetics because of the fast transport enabled by DPD.

We also find that increasing temperature of the simulated system as new bonds are made captures the enthalpy of reaction and results in reaction kinetics that matches the experimentally observed self-accelerated first order model instead of the first-order



(a) Cure Fraction = 0.10 (b) Cure Fraction = 0.95

Figure 3.13: Structure evolves differently for the samples cured with different temperature profiles. C-C structure factors are shown for samples taken at 10% cure and 95% cure. The error bars show standard error.

model[d].

These observations inform a possible two-step process for calibrating nonequilibrium bonding simulations of reactive polymers: (1) Match cure kinetics to experiments with stochastic reaction rates, and (2) use the dissipative drag parameter γ to match structural relaxation times. We demonstrate the present model captures temperature history dependence on microstructure, that co-continuous domains spontaneously phase separate of crosslinking with gelation transitions that all qualitatively match experiments. We also find that a minimum system size of 1.2×10^6 particles is necessary to clearly detect the peak in the C-C structure factor which characterizes microphase separation. With this ability to capture the relevant structure and dynamics of crosslinking polymers, future work will focus on developing and validating models for specific reacting systems and incorporating interaction potentials that enable chain entanglements to be modeled.

[d]This result is not published in Ref. 102

CHAPTER 4

EVALUATING MOLECULAR DYNAMICS MODELS FOR STUDYING GLASSY DYNAMICS

4.1 Introduction

In Chapter 3, new methods for modeling the crosslinking reactions of epoxy-thermoplastic mixtures have been developed and two types of microstructures were observed as a result of varying the time-temperature curing profiles. The next logical step in this research is to measure the physical properties of these different microstructures which relate to their mechanical behavior. One of the fundamental physical properties of glass-forming liquids such as epoxies that determine their performance is the glass transition temperature (T_g). The ability to measure this property reliably from computational models is a necessary step to relate these models to experimentally relevant chemistries, especially if the purpose of the model is to enable prediction of viscoelastic properties that change during crosslinking.

Unlike the first order phase transition in crystalline materials, this transition is not trivial to understand, both in experiments and simulations. The melting temperature of crystalline materials are well defined for a given thermodynamic state point and the structure of a crystalline material is significantly different from its amorphous state. The T_g is known to depend on experimental cooling rates and the structure of

a glassy material is strikingly similar to its liquid state. The dependence on cooling rates means that simulated cooling rates cannot be directly mapped to experimental rates as the MD timescales are typically several orders of magnitude shorter than experimental timescales. However, the time-temperature superposition principle[94] has been successfully used to relate the T_g of atomistic simulations to experimental T_g [51]. One of the outstanding questions about glass transition is whether this transition is a thermodynamic or kinetic process[97]. The viewpoint of the thermodynamics argument is that the rapid loss of entropy during supercooling and causes the kinetic slowdown and the kinetic viewpoint is that this is purely a result of kinetic constraints causing a loss of ergodicity excluding any thermodynamic explanation. Ergodicity is a feature of systems in equilibrium implying that any microstate that occurred is likely to recur.

AAMD simulations that explicitly consider all the atoms have successfully been used to measure T_g of amorphous polymer such as Polystyrene (PS), Polypropylene (PP)[36] and epoxy systems such as diglycidyl-ether of bisphenol F (DGEBF)/44DDS[26]. The size scales accessible to these systems are limited to a few 10s of nm. Due to the relatively small time and length scales accessible to these systems, it is not possible to fully characterize macromolecules which have features in the 100s of nm and μm scales. CGMD have been used to successfully model glassy dynamics of several common linear chain polymers such as PS[37], Polyethylene (PE),PP and Polydimethylsiloxane (PDMS)[57,98] using a bead-spring called the Kremer-Grest (KG) model. The KG model is a combination of the Weeks-Chandler-Anderson (WCA) potential (a truncated and shifted LJ potential) and FENE for non bonded and bonded interactions respectively along with the Langevin thermostat. Good agreement has been found for the self-diffusion coefficient of PS between the KG model, full resolution atomic-scale molecular dynamics simulation and experiments[37]. The

combination of LJ and Harmonic potential has also been used to model polymer melts[113] using CGMD. Much remains to be done to establish a universal mapping between the atomic scale and different levels of coarse-graining and this is an area of active research[100].

Most epoxy models that examine the properties through the glass transition tend to use AAMD or use CGMD to first generate the crosslinked structure and back-mapped into AAMD[56]. This raises a question about whether CGMD can be used to model glass transition of crosslinked networks. Table 4.1 lists key features of a few representative coarse-grained models of glass forming systems. The parameters that seem to cause glass formation in these systems are the symmetry of the interaction parameter ϵ, the angle constraints on the polymer chains which are either Freely Joint Chains (FJC) which do not have any angle constraints or Freely Rotating Chains (FRC) which have an angle constraint.

Table 4.1: Characteristic Features of Glass Forming Systems Modelled using CGMD.

Reference	No. of Components	ϵ Symmetry	Chain Constraints	Chains Length	Comments
19	2	Asymmetric	FJC	< 100	[a]
109	2	Asymmetric	FJC	> 5	
55	2	Asymmetric	FJC	5	
10	1	Symmeteric	FJC	10	[b]
113, 114	1	NA	FJC	16	[c]
16	1	NA	FJC	≤ 50	[d]
16	1	NA	FRC	≤ 50	

There appear to be different strategies to promote glass formation in CGMD models and it will be advantageous to characterize these factors and study their relative contribution to glass formation. This section attempts to address this gap in the literature by examining the effect of choice of thermodynamic variable to measure T_g, the data fitting model used to detect T_g, the cooling method used, the force field used to model interparticle interactions, chain lengths and symmetry of interaction parameters on glass formation, with a focus on polymer chains that grow and form crosslinked networks. More specifically, the glass transition temperature (T_g) of crosslinked chains are measured and the factors affecting the T_g are characterized. The learning from this study can also be useful for CGMD studies of crystalline materials and not just amorphous materials.

To model glass formation in polymers represented by the bead-spring model using molecular dynamics simulations, the force field needs to be carefully chosen so that unphysical bond crossing does not occur. The DPD potential is known to allow bond crossing given its "soft" interaction potential function[24]. A molecular mechanics approach is first used to qualitatively evaluate the three different models for their probability of unphysical bond crossing which is an indicator of unsuitability for glassy dynamics. We first evaluate the DPD force field which is known to be "soft" potential and a "hard" particle potentials to understand their suitability to model glassy dynamics. The "hard" particle potentials studied in this section are hypothetical systems which are analogous to the epoxy-thermoplastic mixture but

[a] Another example in this work where ϵ is symmetric and chain length=1 is seen to crystallize.

[b] This model uses Asymmetric σ to introduce packing frustration which is known to result in glass formation.

[c] This work mentions the use of a harmonic bond $k = 2000\frac{\epsilon}{\sigma^2}$ to avoid crystallization.

[d] This model which uses a repulsive-only LJ potential. Another model in the same work using an attractive LJ potential indicates crystallization.

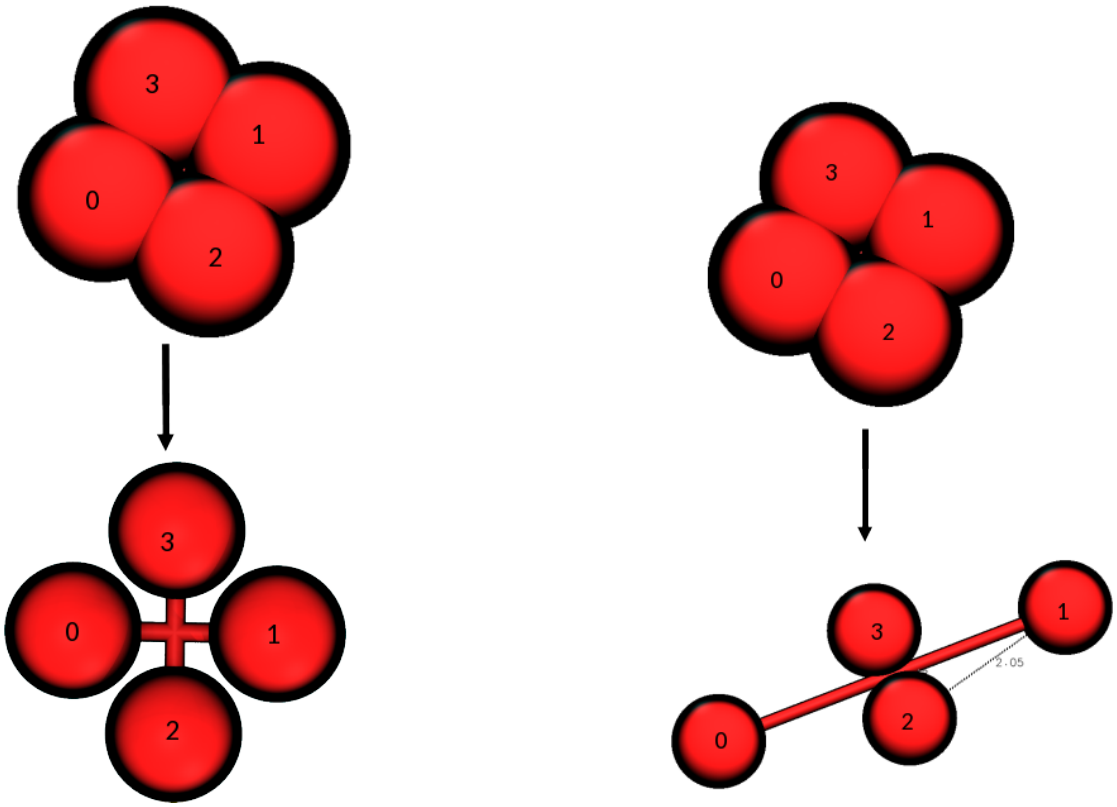
are not representative of any chemistry. In other words, the hard particle potentials studied in this section are toy models that are aids to understanding how they differ from the soft potentials. The well studied LJ potential is chosen as the hard particle potential for non bonded pair interactions and the harmonic potential and FENE potential is considered for the bonded interactions.

4.2 Propensity for Unphysical Bond Crossing

In this section, three potentials are examined to understand the conditions under which the chains can cross each other during simulations. This unphysical bond crossing event is seen as detrimental to modeling entanglements seen in physical polymer systems. These potentials for non-bonded and bonded interactions (written as "non-bonded"/"bonded") are DPD/Harmonic, LJ/Harmonic, LJ/FENE, where the last two use a Langevin integrator to model the interaction of the coarse-grained beads with its environment. Two variants of the last two combinations are explored where the LJ parameter ϵ is multiplied by a scaling factor called E Factor (Υ). The Υ value controls the "stickiness" of the LJ beads by changing the LJ potential "well depth". Υ = 1.0 and Υ = 0.25 are considered in this study where Υ = 0.25 results in less sticky beads compared to Υ = 1.0.

A simple test for understanding the propensity for the unphysical bond-crossing of each potential combination is to take two pairs of polymer chains and arrange them orthogonally and measure the energy of bond crossing and the bond stretch distance. We can either choose to let both chains stretch or stretch one chain as shown in Fig 4.1a and Fig 4.1b. This study uses the former method because it is energetically more favorable given that the stretch distance necessary to cross the bond is lesser.

Geometrically, the stretch distance is $\sqrt{2}\sigma$ for the former case and $\sqrt{3}\sigma$ for the latter where σ is the length scale of the system which is typically the diameter of a particle. Note that the two pairs of chains are not from an actual simulation, but is a test scenario which is likely to occur naturally in one. The purpose of this test scenario is to understand the energetics of bond crossing for a given model characterized by the choice of interparticle potential functions.



(a) stretching both chains

(b) stretching one chain

Figure 4.1: Illustration of two hard-sphere chains stretching to pass through each other.

This test uses a molecular statics simulation rather than a molecular dynamics simulation which implies that the particle positions are manually modified and the

potential energy calculated as a function of their positions, hence the effect of the Langevin equations of motion is not relevant.

In this experiment, we choose to measure the energy starting from an overlapped state to a non overlapped state as shown in Fig 4.1. The stretch distance (D_{BC}) where the total potential energy is minimum is the distance at which the chains cross. The per-particle energy at D_{BC} is the bond crossing energy E_{BC}. Using the Maxwell Boltzmann distribution of energy we can calculate the fraction of particles (X_{BC}) that will have an energy higher than E_{BC} at a given temperature as shown in Equation 4.1.

$$X_{BC}(T) = \frac{2}{\sqrt{\pi}}\sqrt{\frac{E_{BC}}{k_BT}}e^{-\frac{E_{BC}}{k_BT}} + erfc\Big(\sqrt{\frac{E_{BC}}{k_BT}}\Big) \tag{4.1}$$

The X_{BC} is a rough estimate for the fraction of particles that will undergo unphysical bond crossing at a given temperature. Table 4.6 shows the summary of X_{BC} for all the potentials considered.

The parameters used for the DPD potential is shown in Table 4.2 and other simulation parameters are shown in Table 4.3 where the bond distance and bond constant applies for the bonded C beads and the bonded A/B beads.

Table 4.2: Repulsion parameters a_{ij} for amines (A), epoxies (B), and toughener (C) beads determined by Hildebrand solubility parameters from atomistic molecular dynamics.

	A	B	C
A	25.000	30.729	25.003
B		25.000	30.532
C			25.000

The reduced number density (ρ_n) relates to the length scale (L) as $L = (V\rho_n)^{1/3}$. $\rho_n = 3$ implies that three beads fit into a unit volume in the simulation which is

Table 4.3: DPD/Harmonic Parameters

Parameter	Value
Bond Distance (r_o)	0 σ
Bond Constant (k_{harmonic})	4 $\frac{\epsilon}{\sigma^2}$
Density (ρ_n)	3

typical for DPD simulations. The potential energy fuction for this system resulting from the DPD force (Equation 3.1) is illustrated in 4.2.

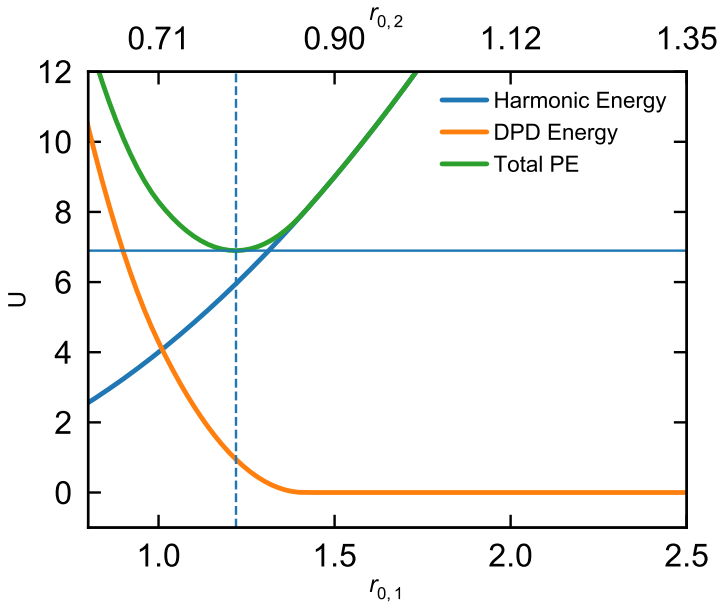


Figure 4.2: DPD potential is used for the non-bonded particles and Harmonic bond potential for bonded particles. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a

The parameters used for the LJ/Harmonic model are shown in Table 4.4 and the potential energy function for this system resulting from the combination of the Lennard-Jones potential (Equation 4.2) and Harmonic Bond potential (Equation 4.3)

is illustrated in Fig. 4.3 for an Υ of 1.0.

Table 4.4: LJ/Harmonic Parameters

Parameter	Value
ϵ_{XX}	1.0 E
ϵ_{XY}	0.2 E
r_{cut}	2.5 σ
Bond Distance (r_o)	1.0 σ
Bond Constant (k_{harmonic})	100 $\frac{\epsilon}{\sigma^2}$
Density (ρ_n)	1.0
E Factor (Υ)	{1.0,0.25}

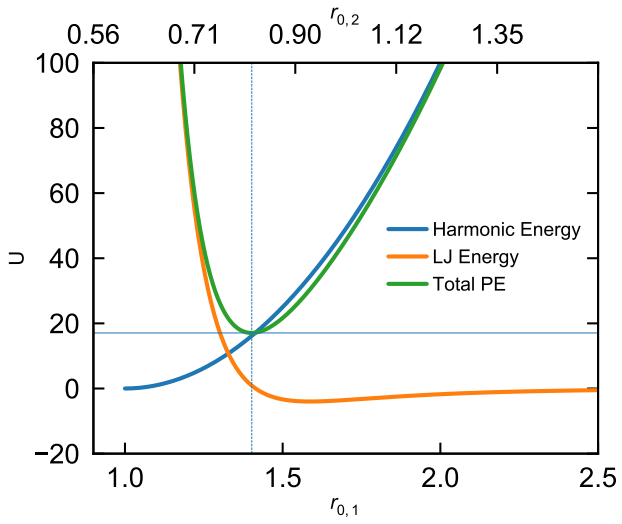


Figure 4.3: The functional form for the LJ/Harmonic potential where E Factor=1.0. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a

A smoothing function is applied to the LJ potential energy function to gradually shift the potential energy to 0 at r_{cut}. The Υ value scales the ϵ value of the LJ potential energy function. This parameter can be used to control the interaction strength between the non-bonded beads of the same or different types.

$$U_{ij} = 4\Upsilon\epsilon\Big(\big(\frac{\sigma}{r_{ij}}\big)^{12} - \big(\frac{\sigma}{r_{ij}}\big)^{6}\Big) \tag{4.2}$$

$$U_{ij} = \frac{1}{2}k_{harmonic}\big(r - r_0\big) \tag{4.3}$$

A combination of LJ with FENE is expected to increase the ability of bonded chains to entangle given the infinite potential barrier of the FENE potential. The parameters used for this model are shown in Table 4.5 and the effective potential energy function for the combination of the Lennard-Jones (Equation 4.2) and the FENE potential (Equation 4.4) is shown in Figure 4.4 where Υ = 1.0.

$$U_{ij} = -\frac{1}{2}kr_0^2 ln\Big(1 - \Big(\frac{r_{ij} - \sigma}{r_0}\Big)\Big) \tag{4.4}$$

Table 4.5: LJ/FENE Parameters

Parameter	Value
ϵ_{XX}	1.0 E
ϵ_{XY}	0.2 E
Max Bond Distance (r_o)	1.5 σ
Particle diameter	1.0 σ
Bond Constant (k)	30 $\frac{\epsilon}{\sigma^2}$
Density (ρ_n)	1.0
E Factor (Υ)	{1.0,0.25}

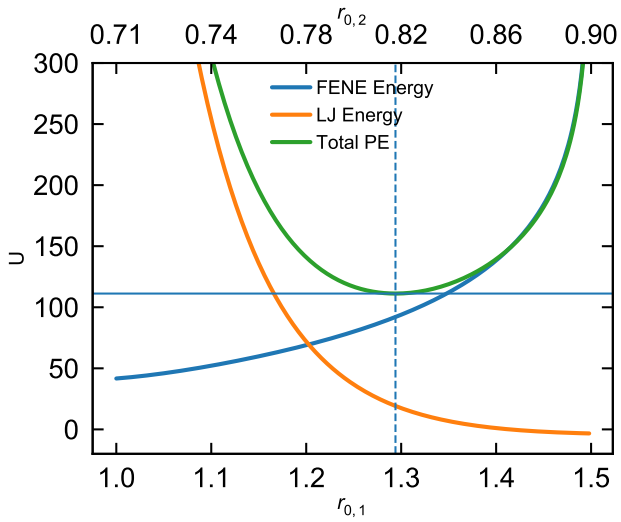


Figure 4.4: The functional form for the LJ/FENE potential where E Factor=1.0. The lower x-axis shows the distance between the bonded particles 0 and 1 and the upper x-axis shows the distance between the non bonded neighbours 0 and 2 from Figure 4.1a

Based on the data presented in Table 4.6 the LJ/FENE potential has a very low probability for unphysical bond crossing whereas about 30% of the particles in the DPD model will have an energy higher than E_{BC} at a temperature of 1 T^* which is typical of most molecular dynamics simulations. T^* is the dimensionless unit of temperature which is related to temperature in real units (T), Boltzmann constant (k_B) and the energy unit (E) as

$$T^* = \frac{k_B T}{E}. \tag{4.5}$$

Even though LJ/FENE has the least probability for bond crossing, the LJ/Harmonic is a more practical choice given its relatively low X_{BC} and ease of producing initial random configurations for simulations. Unlike the FENE potential which can only be stretched to a finite distance, the Harmonic bond can be stretched infinitely, making

it a more preferred bond potential to generate initial structures which are typically generated using random positions or positions that are not energy optimized. Note here that typical temperatures at which glassy dynamics occurs is near or lower than 1 T^* and the probability of two bonded particle pairs having energy $E > E_{BC}$ will be lower than these values of X_{BC}. Therefore, this estimate for bond crossing probability is a conservative overestimation and hence choosing LJ/Harmonic is a safe choice. The

Table 4.6: Comparison of Bond Crossing Energy (E_{BC}), Bond Crossing Distance ($D_{BC}[\sigma]$) and Bond Crossing Probability (X_{BC})

Potential Function	E Factor	E_{BC}	$D_{BC}[\sigma]$	$X_{BC}@1T^*$
DPD/Harmonic	NA	1.724	1.220	3.20e-01
LJ/Harmonic	1.0	4.625	1.402	2.61e-02
LJ/Harmonic	0.25	3.325	1.310	8.39e-02
LJ/FENE	1.0	27.805	1.294	5.09e-12
LJ/FENE	0.25	21.699	1.210	2.01e-09

LJ/Harmonic system with an E Factor of 0.25 will be considered to study the glassy dynamics first using a toy system where the parameters chosen are arbitary and does not model any specific chemistry.

4.3 Methods for Measuring T_g

4.3.1 Thermodynamic Variables Used to Measure T_g

The glass transition temperature measurement procedure starts with crosslinked structures obtained using methods described in Ref. 102. The samples cured to different degrees of cure (α_{low} to α_{high}) are cooled from a high temperature (T_{high}) to a low temperature (T_{low}). During this cooling, thermodynamic properties such as the volume are observed as a function of temperature to detect the T_g, the temperature at

which this property changes significantly indicating that the material transitioned to a glass. An alternate approach to using volume to detect glass transition is to use the mass diffusion coefficient of the particles. Ref. 16 used the self-diffusion coefficient in the viscous fluid regime and detected the glass transition as the temperature at which diffusivity trend reach zero. The advantage of using diffusivity comes from the ability to clearly distinguish the low diffusivity glassy state and the higher diffusivity gel state. However, this approach is experimentally more challenging than the volume approach. The self diffusion coefficient (D) of the beads are calculated using the Einstein relation as shown in Equation 4.6 where $r(t_0)$ is the displacement at initial time t_0 and $r(t)$ is the displacement at final time t.

$$\langle (r(t) - r(t_0))^2 \rangle = 6D(t - t_0) \tag{4.6}$$

4.3.2 Quench and Anneal Cooling Methods

Two different molecular dynamics procedures are available for cooling the cured sample from T_{high} to T_{low}. The first method called quenching involves taking the output of a curing simulation and cooling it instantaneously to a quench temperature (T_q) which is anywhere between T_{high} and T_{low}. These simulations are allowed to equilibrate at the T_q for t_q time steps. Since the input for each quench simulation is the output for the curing simulation, each T_q can be simulated concurrently making this method highly parallelizable. The second method called annealing, on the other hand, involves sequentially cooling the cured system from T_{high} to T_{low} where the output of each T_q is used as the input for the next lower T_q. The annealed simulations are equilibrated for t_a time steps which are typically lesser than t_q. While the latter method is computationally less efficient than the former, it preserves the

temperature history. Properties such as volume or diffusivity are calculated after the system has reached equilibrium in both methods. Both of these cooling methods are performed using a constant pressure, constant temperature NPT ensemble using the Martyna-Tobias-Klein barostat-thermostat[69] where the pressure is maintained at P where P is the average pressure of systems cured between α_{high} and α_{low} which are cured at a constant volume. The coupling constant for the barostat (τ_P) and the coupling constant for the thermostat (τ_T) was set to 1000 Δt and 100 Δt respectively to avoid any unphysical ringing in volume.

4.3.3 Data Fitting Models for Detecting T_g from Thermodynamic Data

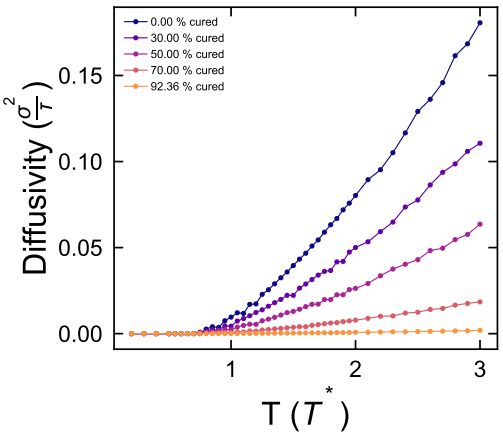


Figure 4.5: The self-diffusion coefficient of the coarse-grained beads show increased diffusivity with increasing temperature and lower cure fraction.

Detecting the T_g from the dynamic and structural data corresponding to this transition is challenging for several reasons and this is an active area of research[90]. The detection of the low temperature and high-temperature asymptotes is particularly difficult when this region is not known in advance as in the case of cured thermoset polymers with different degrees of cure (Figure 4.5). A segmented regression method,

a hyperbola function fitting[80] and a composite function consisting of a linear and power function were evaluated to fit the diffusivity data. An automatically segmented regression detects a minimum number of line segments that best fit the data by monitoring the increase in the sum of squared errors when line segments are merged. In this approach, N/2 line segments are considered first where N is the number of data points. The number of line segments decreases from N/2 to 1 by iteratively merging line segments causing an increase in the sum of the squared errors of individual line segments. Merging is stopped when the increase in the sum of squared errors is more than 3% and the line segments from the previous merge are considered as the best fit lines for the data. This method will be henceforth called the Automatic Piecewise Regression Method (PRM-a) and a variation of this method where segmented regression is done with manual data range selection will be called the Piecewise Regression Method (PRM). The hyperbola fitting uses a model similiar to Ref. 80 where the equation of the half branch hyperbola is modified to suit the diffusivity data as shown in Equation 4.7 where T_0 is taken as T_g and other fitted parameters are D_0, a, b and c. Two variations of the hyperbola fitting methods considered are Hyperbola Fitting Method (HFM) and Constrained Hyperbola Fitting Method (HFM-c) where a, the slope of the low-temperature asymptote is constrained to take only positive values or allowed to take any value respectively.

$$D = D_0 + a(T - T_0) + b\mathcal{H}_0(T, T_0, c) \tag{4.7}$$

$$\mathcal{H}_0(T, T_0, c) = \frac{1}{2}(T - T_0) + \sqrt{\frac{(T - T_0)^2}{4}e^c} \tag{4.8}$$

The composite function which will be called Power Law Fitting Method (PLFM) has the form shown in Equation 4.9 where a, b and p are the fitted parameters.

$$D = \frac{aT^p}{(b - T^{p-1})} \tag{4.9}$$

The modified DiBenedetto Equation[21,79] (Equation 4.10) known to fit the experimental T_gs[88,89] as a function of cure fraction is used to validate the T_gs measured through simulation. Here the data provided for the fitting are the $T_g(\alpha)$ and the interaction parameter λ.. The interaction parameter is a chemistry-specific value between 0 and 1 and for this study, we chose $\lambda = 0.5$. The parameters that we obtain from the fitting are the glass transition temperature of the uncured system (T_{g0}), and the glass transition temperature of the fully cured system (T_{g1}).

$$T_g(\alpha) = \frac{\lambda\alpha(T_{g1} - T_{g0})}{1 - \alpha(1 - \lambda)} + T_{g0} \tag{4.10}$$

4.3.4 Fiducial Parameters

All the simulations in this section except for Section 4.4.4 is performed with an A/B binary mixture of coarse-grained Lennard-Jones bead-spring model described as the LJ/Harmonic model (Table 4.4). In Section 4.4.4, the more complex A/B/C ternary mixtures are considered for T_g measurement. The interaction parameters for the A, B, and C beads are described in Table 4.7. These are the symmetric interaction

Table 4.7: LJ parameters for coarse grained beads A, B and C.

	A	B	C
A	0.25	0.05	0.05
B		0.25	0.05
C			0.25

parameters. Section 4.4.7 discusses the effect of asymmetric interaction parameters

on the T_g. The fiducial simulation parameters used for curing and cooling are shown in Table 4.8.

Table 4.8: LJ/Harmonic Simulation Parameters

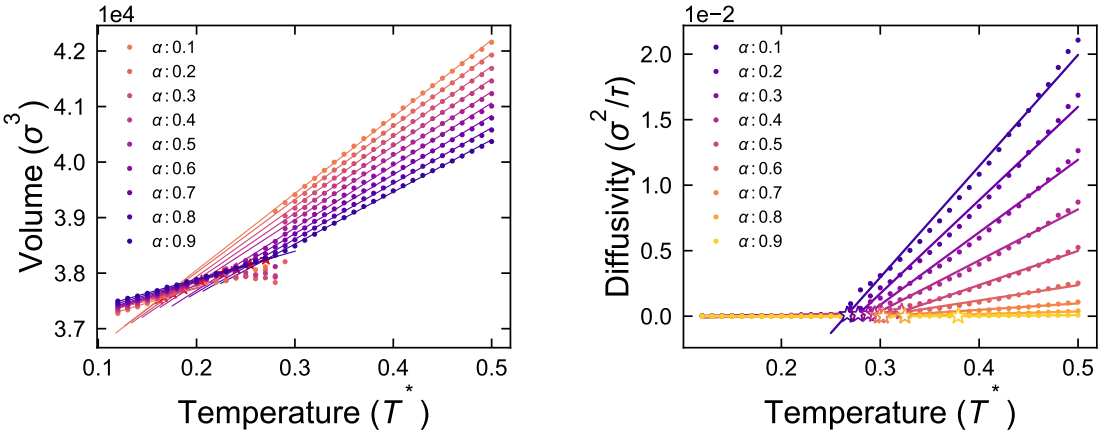
Parameter	Value
Bond Distance (A-B,C-C) (r_o)	1.0 σ
Bond Constant (A-B,C-C) (k)	100 $\frac{\epsilon}{\sigma^2}$
Bond Angle (C-C-C) (θ_0)	109.5°
Bond Angle Constant (C-C-C) (k_θ)	25 E
Density (ρ_n)	1.0
E Factor (Υ)	0.25
Activation Energy (E_A)	2.0 ϵ
Secondary Bond Weight (β)	1.2
Cure Temperature	1.7 T^*
Langevin Diffusive Drag Parameter (γ)	4.5
Time Step Size (Δt)	0.01 τ
Lowest Cure Fraction (α_{low})	0.0
Highest Cure Fraction (α_{high})	1.0
Lowest Anneal Temperature (T^a_{low})	0.2
Highest Anneal Temperature (T^a_{high})	0.4
Anneal Time (t_a)	$6 \times 10^6 \Delta t$
Lowest Quench Temperature (T^q_{low})	0.12
Highest Quench Temperature (T^q_{high})	0.5
Quench Time (t_q)	$1 \times 10^7 \Delta t$
Barostat Pressure Set Point (P)	6 $\frac{E}{D^3}$

The Lennard-Jones interaction parameter ϵ such as the ones shown in Table 4.7 is in units of E. If $T^* = 1.5$ is equivalent to $T = 300\ K$, then the energy unit $E = \frac{k_B 300\ K}{1.5}$.

4.4 Comparing T_g Measurements

4.4.1 Effect of Choice of Thermodynamic Variable Used to Measure T_g

Volume and the self diffusion coefficients of the B beads are considered to measure the T_g of the LJ/Harmonic systems which are quenched from T_{high} to T_{low} as shown in Figure 4.6.



(a) (b)

Figure 4.6: Comparison of (a) volume and (b) diffusivity (self diffusion of B beads) change as a function of quench temperatures. The systems with lower degrees of cure exhibit a sharp change in volume as a function of quench temperatures rather than a gradual change as seen at higher degrees of cure.

The PRM-a method is being used here to detect the T_g. The systems with a higher degree of cure (α) equilibrate to a lower volume due to the highly crosslinked structures. The systems with lower degrees of cure do not exhibit a bi-linear trend

in volume that is typical of a glass transition. The discontinuity in the volume as a function of temperature in Figure 4.6a and the discontinuity in C_p in Figure 4.7 indicate that systems with lower degree of cure undergo a first order phase transition and not a second order phase transition seen in glass transitions. The

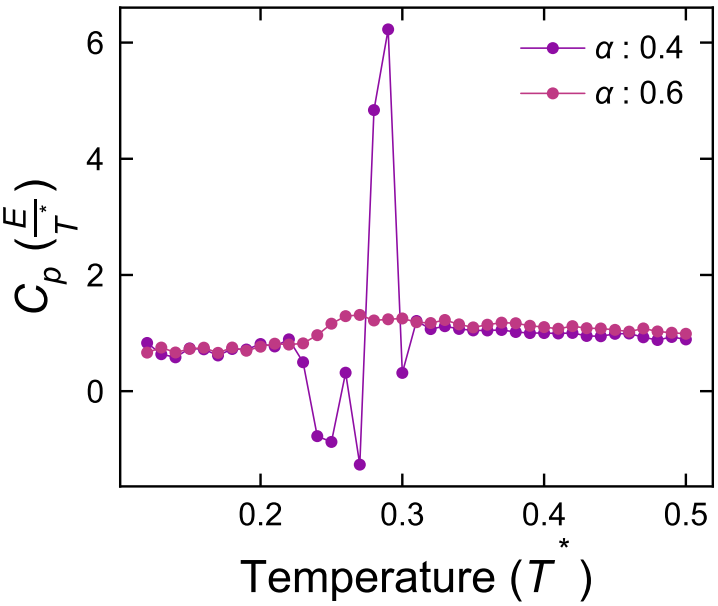


Figure 4.7: The specific heat capacity at constant pressure (C_p) shows discontinuity for the system cured to 10% and not for the 90% cured system indicating a first order phase transition in the former.

first order transition is known to occur with Lennard-Jones mixtures especially when the interaction parameters are symmetric. In Ref. 91 simulations of Lennard-Jones binary mixtures where crystallization occurred were discarded from T_g measurements. In a symmetric interaction parameter, the like interactions (e.g. A-A interaction and B-B interaction) are the same and the cross interactions are also the same (e.g. A-B interaction and A-C interaction). In this study, both the particle sizes and interaction parameters (Table 4.7) considered are symmetric. A recent study[19] shows that

the order-disorder transition temperature a block copolymers system modeled using coarse-grained Lennard-Jones beads with chain length less than 10 to overlap with the glass transition temperature of LJ homopolymer beads indicating that low molecular weight chains (chain length< 10) tend to behave like homopolymer beads. Ref. 113 mentions the use of the harmonic bond constant (k) of 2000 $\frac{\epsilon}{\sigma^2}$ to avoid crystallization. It is interesting to note that the emergence of the bi-linear trend coincides with the gelation transition at $\alpha \approx 0.6$ and that the mean molecular mass of networks in this system before gelation is less than 10 as seen in Figure 4.8. The observation that

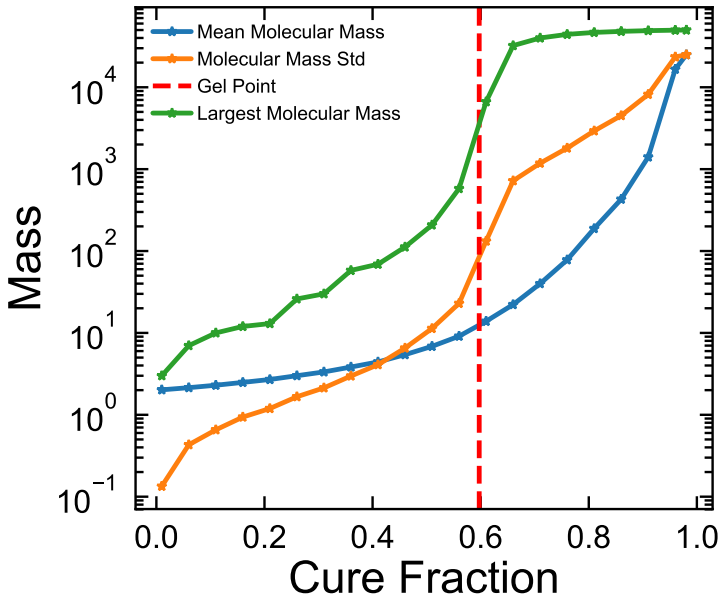
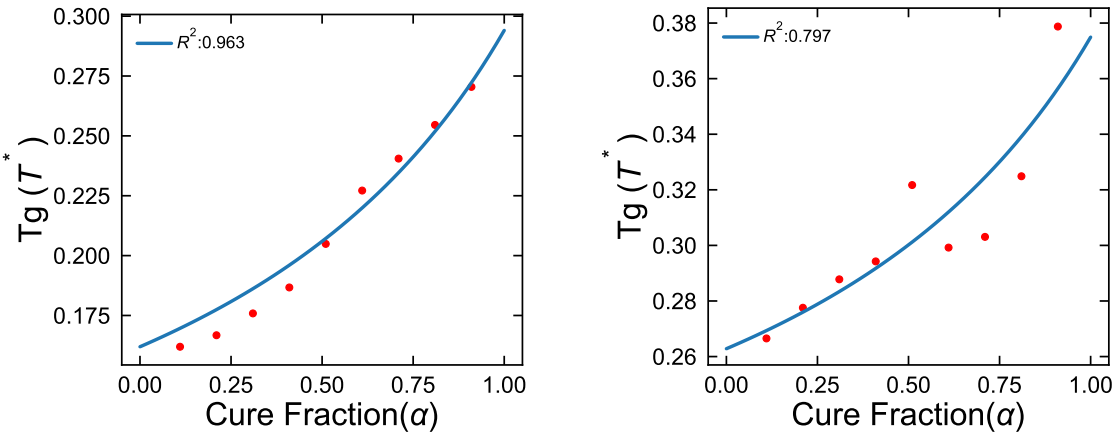


Figure 4.8: The average molecular mass is below 10 M (M is the dimensionless mass unit) up to gel point where the mass of each bead is 1 M. The low standard deviation in molecular mass before gelation indicates that most of the networks are small until gelation.

the gel point coincides with the observation of glass transition combined with the observation of low molecular mass before gelation as seen in Figure 4.8 supports the

notion that longer chains are necessary to cause glassy dynamics rather than the first order transitions observed for systems below gelation. Figure 4.9 shows the T_g measured using these two thermodynamic variables fit to the DiBenedetto equation (Equation 4.10) to validate the physical relevance of these measurements.



(a) Using Volume

(b) Using Diffusivity

Figure 4.9: The DiBenedetto equation (Equation 4.10) fits the T_g values calculated using (a) volume better than (b) diffusivity. However, the T_g values detected by the volume data below α = 0.6 do not correspond to a glass transition or first-order phase transition, but is an artifact of the PRM-a.

The volume-based method appears to result in a better DiBenedetto equation fit than the diffusivity based method. However, the data points for the volume based T_g (a) for $\alpha \leq 0.5$ are not actual T_g, but just an artifact of the data fitting. Note that the actual first-order phase transitions in the low α systems are in fact at a higher temperature than the T_g of the high α systems. The diffusivity based method does indicate the slow kinetics at the transition points for all cure fractions and the T_g values detected are higher than the T_g values observed with the volume based method. Based on the observation that the volume based method only provides useful data

above gelation for coarse-grained systems such as the present one, the diffusivity based method is more suitable than the volume-based method. The PRM-a is observed to result in a bad DiBenedetto equation fit for the diffusivity data.

4.4.2 Effect of Data Fitting Method

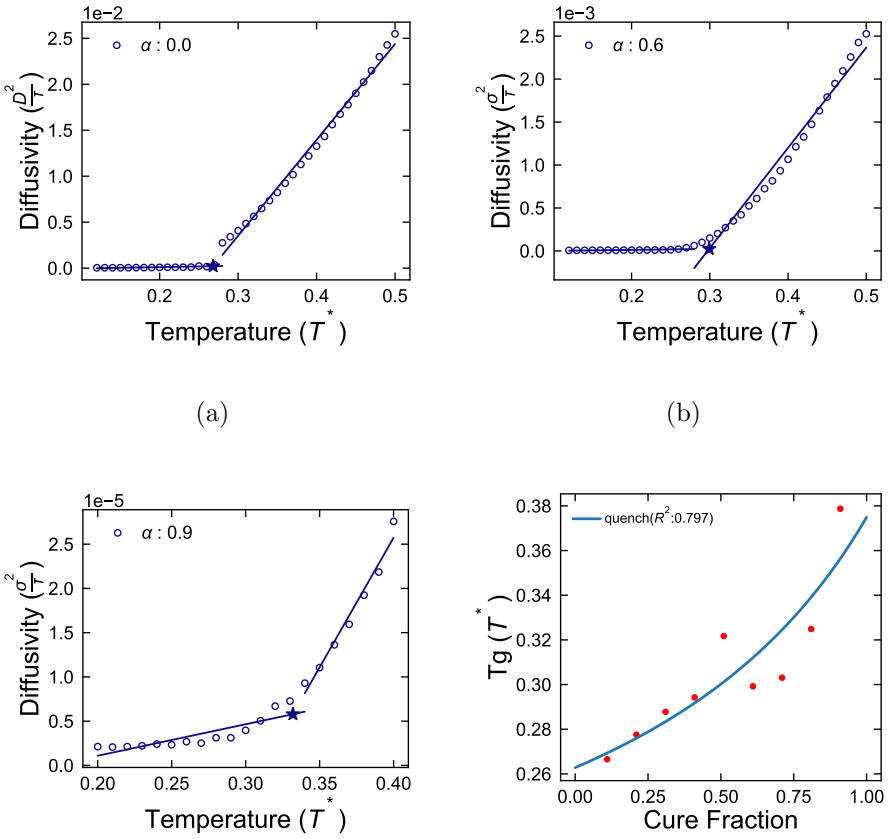


Figure 4.10: PRM-a is able to detect the T_g (★) well for system with (a) α = 0.0 and (b) α = 0.6, but for (c) α = 0.9, the T_g detection is unstable. The large deviations of T_g data from the DiBenedetto function (c) for $\alpha > 0.5$ indicate the instability of the PRM.

The PRM-a, PRM, HFM, HFM-c and PLFM were evaluated for detecting T_g from diffusivity data using simulations that used the quench cooling strategy. The self-diffusion coefficient of the B beads was used because at high cure fractions, the diffusivities of the A beads which are relatively lower than the B beads were noisier and hence unstable for data fitting. Though PRM-a is the simplest, it is unstable when the transition between the low-temperature asymptote and the high-temperature asymptote is a gradual one as seen in a 90% cured system (Figure 4.10c). The PRM-a detects the T_g for diffusivity data where the α is lower as seen in the α = 0.0 (Figure 4.10a) and α = 0.6 (Figure 4.10b). The PRM-a is capable of detecting any number of line segments that minimize the error in regression, but the α = 0.9 data was not detected as three segments without modification to the default algorithm which we do not attempt in this study. The observation that the DiBeneditto Equation does not fit the T_g values well, especially for α $\approx>$ 0.5 highlight the deficiency of this method for automatically detecting gradually varying dependent variables such as temperature dependent diffusivity data.

In the PRM two data ranges are selected such that the intersection of the two line segments fitting the two data ranges is close to the beginning of the very low diffusivity region. This method is more tedious since the data range for different α has to be individually configured such that the low temperature (glass) regime does not include the data from the transition region or the high temperature (gel) regime. The data range for the high temperature (gel) regime is chosen such that this data does not include the low-temperature regime as well as the high-temperature regime where the system is past the viscous gel phase and behaves more like a liquid. The custom data ranges chosen for each cure fraction is given in Appendix C5 and examples of such cherry picked data is shown in Figure 4.11 for quenched simulations.

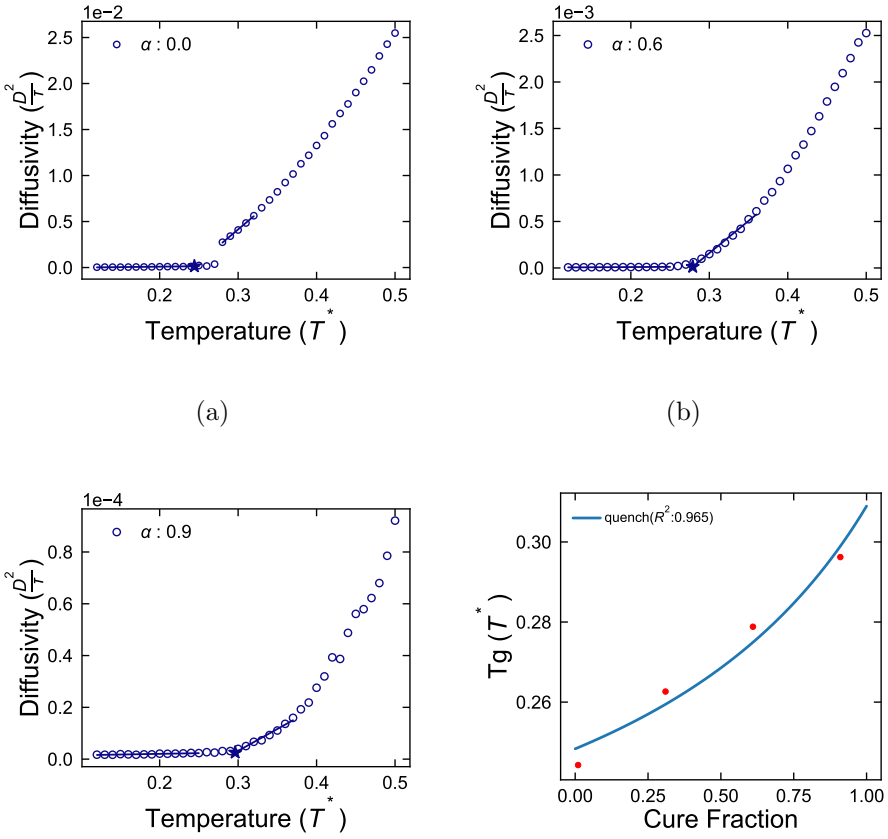


Figure 4.11: The DiBenedetto equation is found to fit T_g (★) values detected using the PRM better than the PRM-a. The custom data ranges used are given in Appendix C5

The DiBenedetto Equation fit for the T_g values detected using PRM show that the PRM is a reliable method for detecting T_g.

Motivated by the convenience of automatically detecting the transition region, other shape functions such as the hyperbola function is explored to evaluate T_g. Two variants of the hyperbola function are examined here, namely the HFM and HFM-c. The HFM is the unconstrained hyperbola function fitting method which was fit to

the diffusivity data from the quench simulations. The systems with cure fraction $\alpha = 0.0, 0.6, 0.9$ are shown along with DiBenedetto equation fit in Figure 4.12.

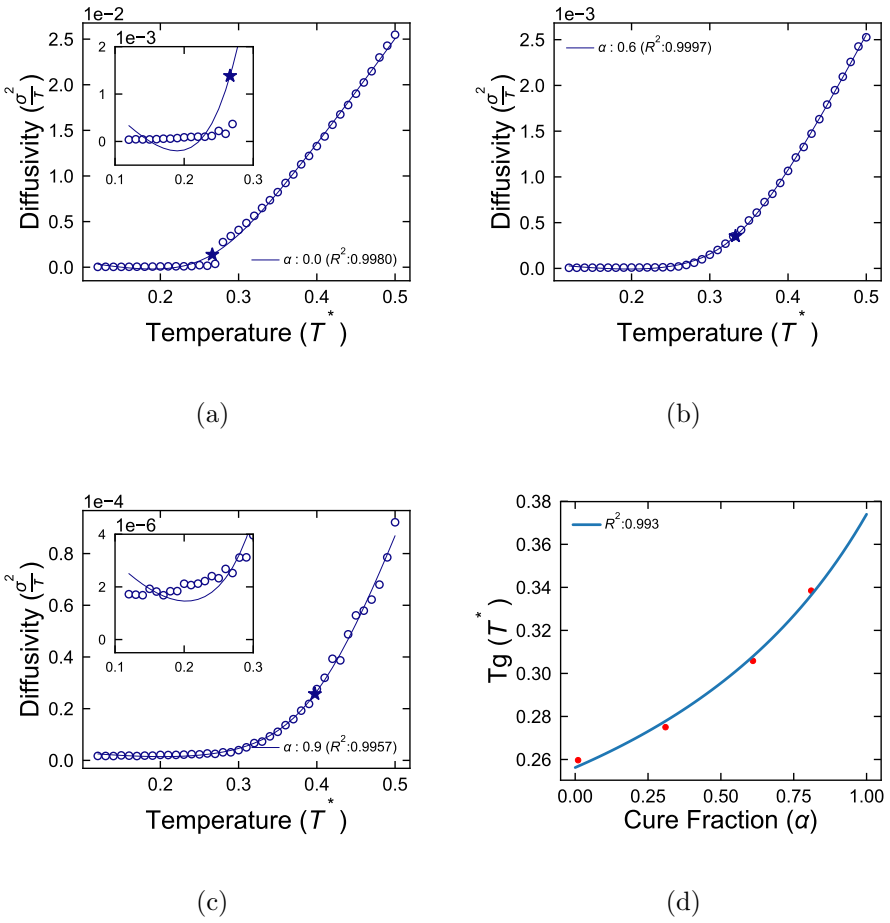


Figure 4.12: The DiBenedetto equation is found to fit the T_g (★) values detected using the HFM very well. The inset for $\alpha = 0.0$ (a) shows the magnified low temperature data fitting of the hyperbola tail.

The low temperature data fitting of the HFM shown in the inset of Figure 4.12 shows that the function does not fit these low temperature data very well. This is a result of not constraining the slope of the low-temperature asymptote (a) in Equation 4.7 and the T_0 value to be positive. However, the HFM fits the diffusivity data at the

transition region and high-temperature data well and results in a good DiBenedetto equation fit.

The second variant of the hyperbola function, the HFM-c was also fit to the diffusivity data from the quench simulations as shown in Figure 4.13.

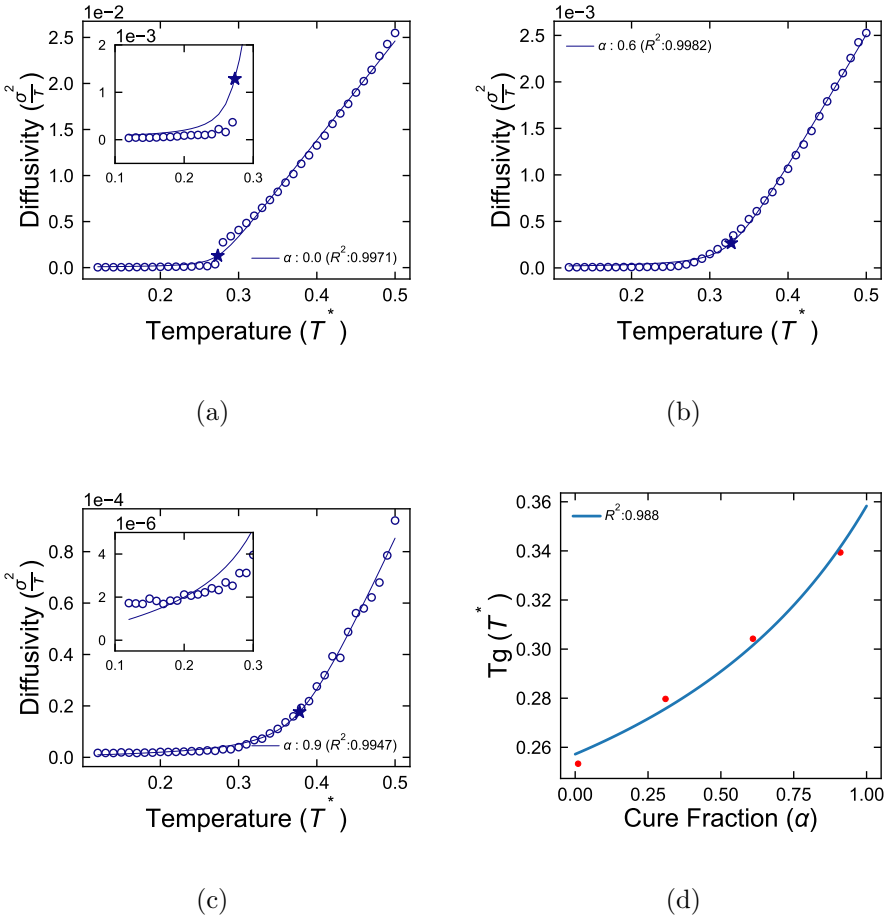


Figure 4.13: The DiBenedetto equation is found to fit the T_g (★) values detected using the HFM-c almost as well as the HFM. The inset for α = 0.0 (a) shows the magnified low temperature data fitting of the hyperbola tail.

The T_g values detected using this function was found to be similiar to the HFM. Unlike the HFM, the HFM-c constraints the a and T_0 in Equation 4.7 to be positive.

The inset images in 4.13a shows the hyperbola function fitting in the low-temperature region. Although the unphysical fit is avoided, the shape does not fit the data exactly and this shape does not predict a diffusivity of 0 at 0 K. The DiBenedetto fit for the T_g data shows that the HFM-c performs as well as HFM.

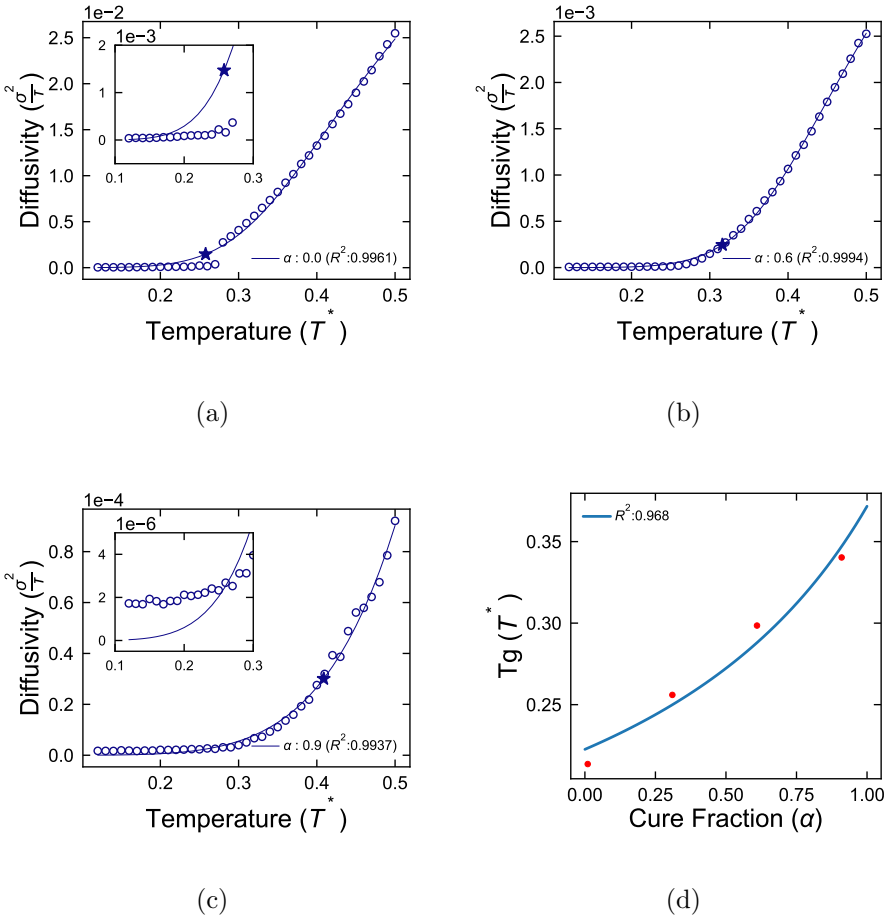


Figure 4.14: The DiBenedetto equation the T_g (★) values detected using the PLFM is also found to fit well and is similiar to the HFM and HFM-c fitting. The inset for α = 0.0 (a) shows the magnified low temperature data fitting.

Finally, the PLFM is evaluated for its effectiveness to detect T_g. The PLFM is similar to the HFM-c because the diffusivity can only take positive values, but unlike

the HFM-c a diffusivity value of 0 is produced by the PLFM at a temperature of 0 K. The inset images in Figure 4.14c shows that even this method does not produce an exact fit for the data even though the deviations are very small in magnitude.

Based on the A/B binary mixture diffusivity data fitting using the different models, it can be concluded that the PRM, PLFM, HFM and HFM-c are suitable for detecting the T_g. However, the hyperbola based methods and the PLFM do not detect the start of the glassy region as well as the PRM which is what we consider as the temperature at which the molecules stop moving and becomes a glass.

4.4.3 Effect of Cooling Method

The two cooling methods quenching and annealing described in Section 4.3.2 are compared using the LJ/Harmonic system for differences in the structures and its relevance to the T_g measurement. In the annealing strategy, the cured systems were first quenched to 0.4 T^* and subsequently annealed to 0.2 T^* with a temperature interval of 0.01 T^* for computational efficiency. The effect of the cooling method on the crystal structure obtained at different temperatures in the vicinity of T_g is examined both using snapshots of the system and the A-A radial distribution function. The volume change as a function of temperature for the 40% cured (below gel point) and 60% cured (above gel point) systems cooled using the quench method is shown in Figure 4.15 and 4.16. A first-order phase transition characterized by the sharp change in volume is seen in the 40% cured system in Figure 4.15. The A-A radial distribution function averaged over the equilibrated frames and morphology from the final snapshot of the system reveals that the system undergoes a first-order transition at the transition temperature of $T \approx 0.28\, T^*$. The amorphous structure at $T = 0.3\, T^*$ transitions to a structure with long range order at $T = 0.28\, T^*$.

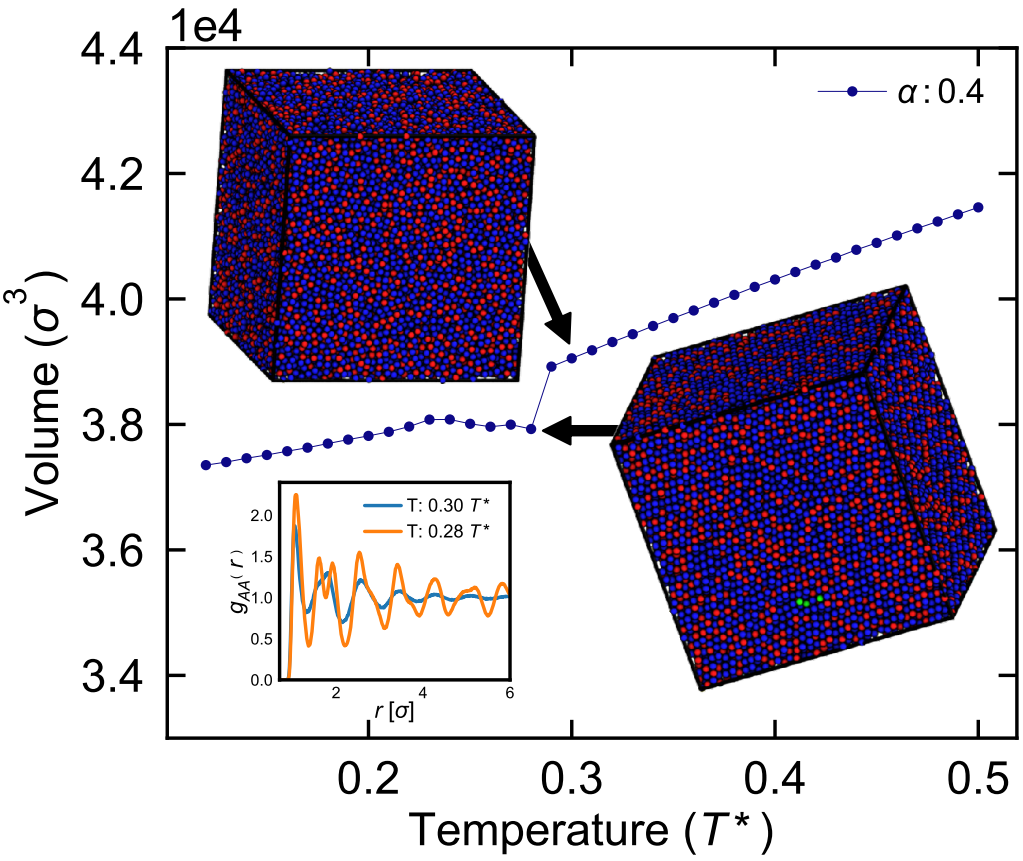


Figure 4.15: A first-order phase transition is observed at $T \approx 0.28\ T^*$ when the 40% cured system is quenched. The sharp change in volume at $T = 0.28\ T^*$ is characteristic of this transition. The A-A radial distribution function and morphology in the inset indicate that the system undergoes a transition from an amorphous structure at $T = 0.3\ T^*$ to a structure with long range order at $T = 0.28\ T^*$.

However, the 60% cured system which is above the gel point undergoes a glass transition at $T \approx 0.23\ T^*$ as shown in Figure 4.16. The A-A radial distribution function and the morphology shows no change in the structure above ($T = 0.3\ T^*$) and below ($T = 0.23\ T^*$) the transition temperature. It is interesting that the structure of the system before and after glass transition is strikingly similar even though their

dynamic properties are expected to be very different based on the diffusivity data in Figure 4.18.

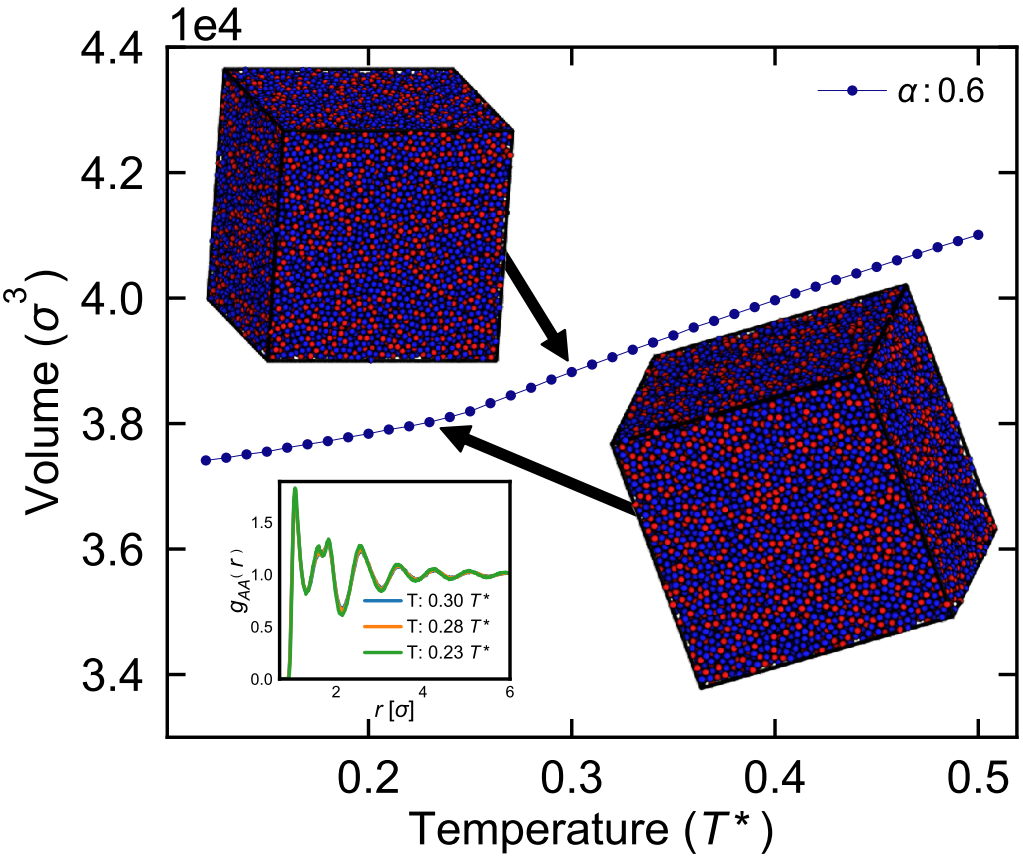


Figure 4.16: A glass transition is observed at $T \approx 0.23\ T^*$ when the 60% cured system is quenched indicated by the characteristic gradual change in volume between $T = 0.2\ T^*$ and $T = 0.5\ T^*$. The A-A radial distribution and morphology in the inset shows that the structure before the transition ($T = 0.3\ T^*$) is the very similiar to the structure after the transition ($T = 0.23\ T^*$). The A-A radial distribution function for $T = 0.28\ T^*$ is shown for comparing with Figure 4.15.

Even though the systems cured less than gel point were observed to be crystalline at the first-order transition temperature, at even lower temperatures, amorphous

structure was observed. This amorphous structure is a result of kinetic arrest that occurs at low quench temperatures when the particles do not have enough kinetic energy to rearrange themselves from an initial mixed structure to a more ordered structure. Unlike the quenched systems, the annealed systems below gelation do not show amorphous structure at low temperatures. This is an effect of temperature history retained during the annealed cooling method and is considered to be important for measuring T_g of MD systems[14,30,51].

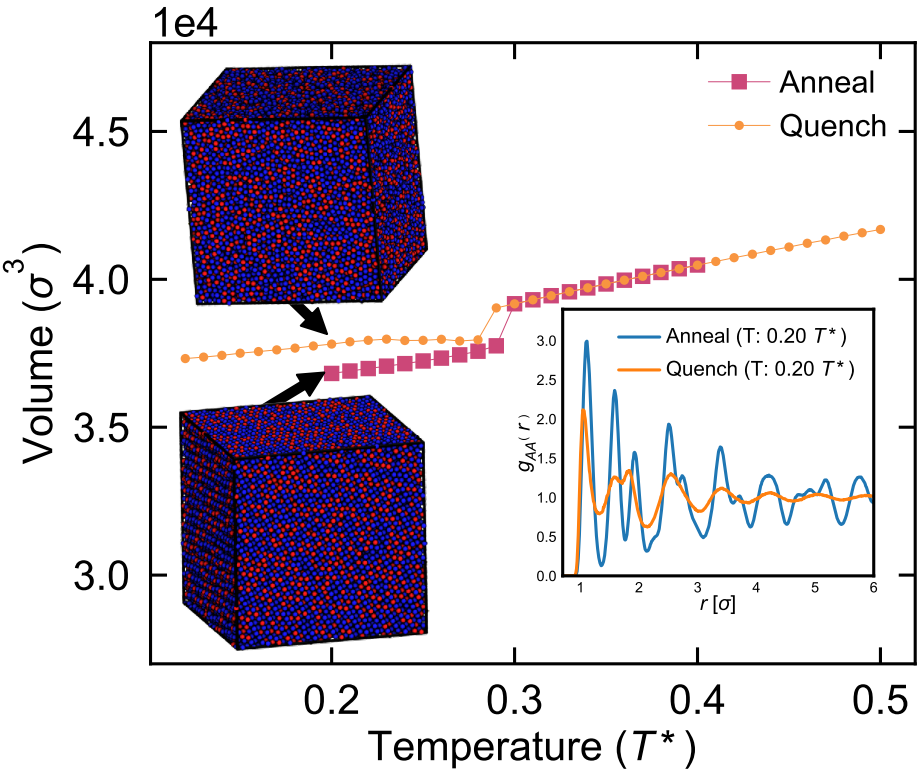
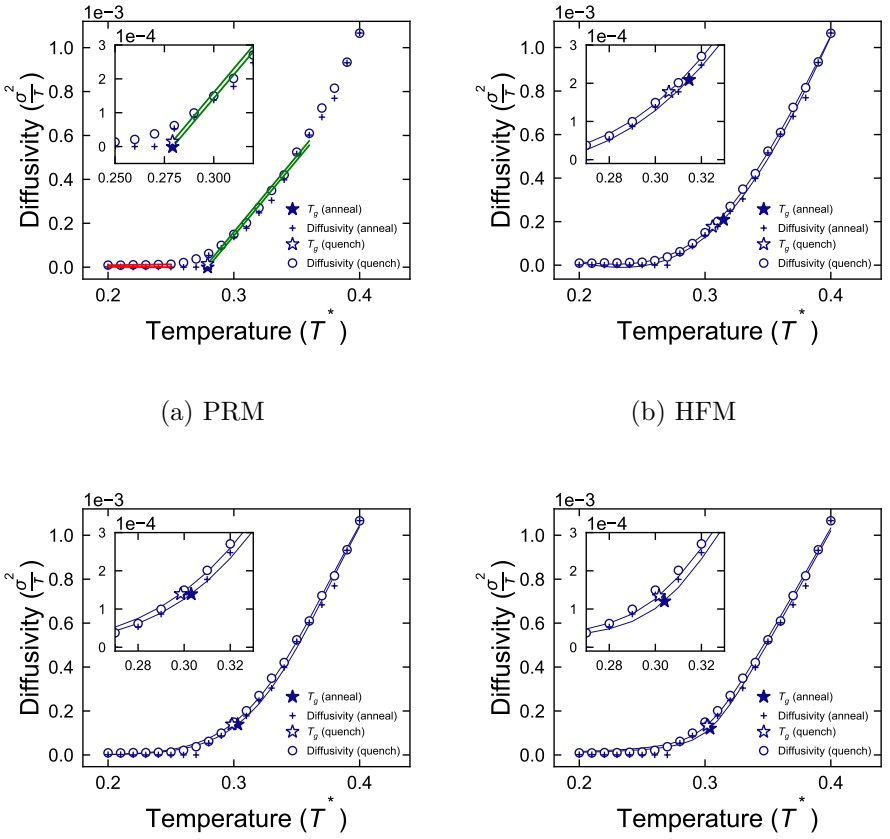


Figure 4.17: A 30% cured system results in amorphous structure at low temperatures when quenched, but results in an ordered structure when annealed. The A-A radial distribution function and morphology of the system at T = 0.2 T^* (inset) shows the structural difference caused by the different cooling methods.

The A-A radial distribution function and morphology at a low quench temperature (T = 0.2 T^*) and a low anneal temperature (T = 0.2 T^*) for a 30% cured system shown in Figure 4.17 show this difference in structure resulting from the cooling method. Since the quenched systems are kinetically arrested at low temperatures, longer simulation time will allow the quenched systems to reach the low energy states as the annealed systems. The volume data below gelation cannot be used to detect the transition using intersecting lines fitted to the low temperature and high-temperature volume data because the lines are almost parallel. The typical T_g detection methods such as the PRM-a will detect an unphysical T_g for systems below gelation because the lines fitting the volume data below and above the first-order transition temperature will have only a slight difference in slope and hence intersects at a very low temperature.

The diffusivity data is examined for its effectiveness to detect the transition temperature. The cooling method's influence on the measured T_g is examined by first comparing the fitting of the diffusivity data from the quenched and annealed simulations with PRM,PLFM, HFM, and HFM-c for a reference cure fraction of α = 0.6. The DiBenedetto equation is then fitted to the T_g values obtained from annealed and quenched simulation with $\alpha = \{0.0, 0.3, 0.6, 0.9\}$ to examine the trend with respect to α. The inset in Figure 4.18a show that the diffusivity data from the annealed simulation and quenched simulation differ in the transition region (T = 0.26 T^* − 0.27 T^*), but otherwise, the two cooling methods result in essentially the same diffusivity. Since the T_g value detected from diffusivity data from quenched simulations for α = 0.9 using the HFM was unusually high (T^* = 0.47 T^*), this T_g value was discarded as noise and $T_g(\alpha = 0.8)$ was considered instead as seen in Figure 4.19b.



(a) PRM

(b) HFM

(c) PLFM

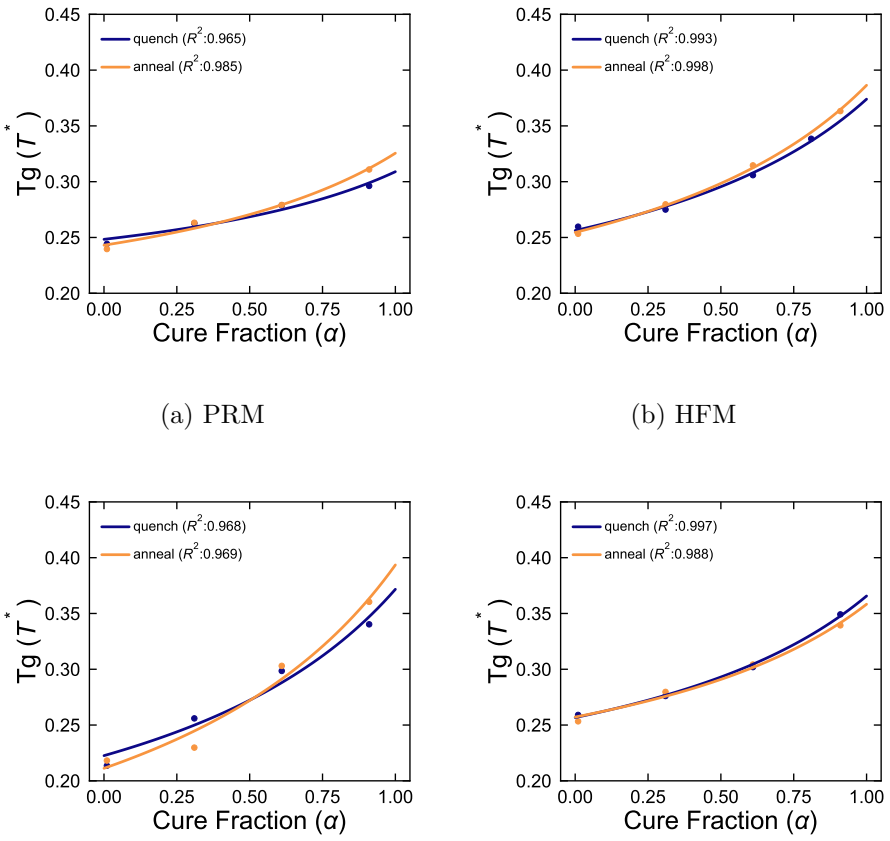
(d) HFM-c

Figure 4.18: The self diffusion coefficient of B beads obtained from annealed simulations cured to α = 0.6 is fit using PRM, PLFM, HFM and HFM-c. The diffusivities obtained from the two cooling methods are found to be very similiar and hence the detected T_g values for the quenched systems (☆) and annealed systems (★) are also found to be similiar.

This unusual jump in the detected T_g indicates that the HFM is sensitive to differences in the diffusivity data obtained from the two cooling methods. The HFM has been observed to be less sensitive when more data points are provided on either side of the transition region.

The DiBenedetto fits for the T_g calculated from the annealed and quenched sim-

ulations in Figure 4.19 shows good fits for all the four methods and the differences in T_g because of the cooling methods are minor. This observation suggests that the choice of cooling method is not a very important factor for T_g measurements from CGMD. The quench method is preferred in this work because of its computational efficiency.



(a) PRM

(b) HFM

(c) PLFM

(d) HFM-c

Figure 4.19: The quenched and annealed cooling methods does not seem to significantly impact the T_g measurements with the PRM, PLFM, HFM and HFM-c. The T_g detected from quenched simulations for α = 0.9 using the HFM in (b) was discarded as noise since the $T_g(\alpha = 0.9) = 0.47\ T^*$ was unusually high and α = 0.8 was used instead.

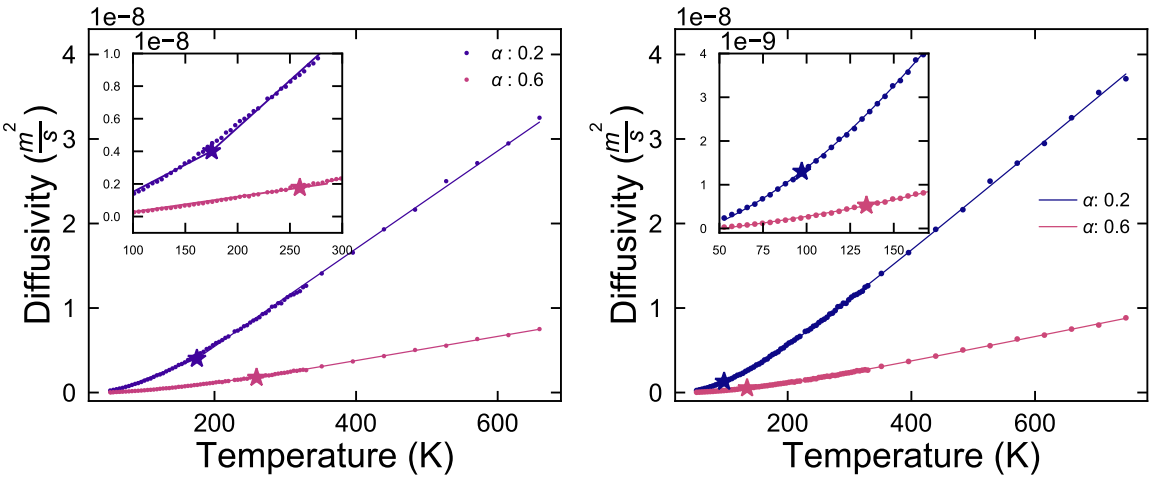
4.4.4 Effect of Simulation Model

DPD Model

The T_g of the DGEBA/DDS/PES system modeled using the DPD force field was measured using the self-diffusion coefficient of the amine monomers represented by the A beads.

Table 4.9: DPD Quench Parameters

Parameter	Value
α_{low}	0.2
α_{high}	0.8
t_q	5e6 Δt
T_{low}	0.12 T_C
T_{high}	1.7 T_C
P	23 $\frac{E}{D^3}$

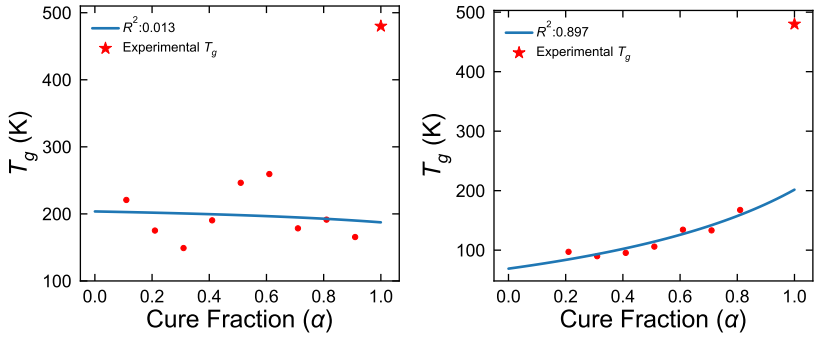


(a) PRM-a

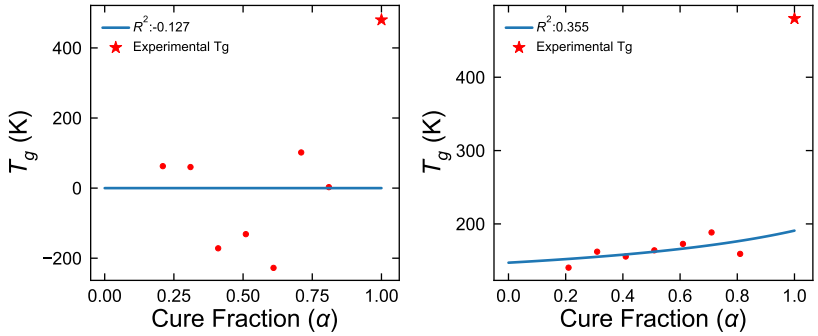
(b) HFM-c

Figure 4.20: Self-diffusivity of A beads of the DPD model fitted using the piecewise linear regression method and the constrained hyperbola to detect T_g (★). The dotted lines show diffusivity for α=0.2 and α=0.6. The solid lines show the fitted functions.

The model parameters used for the curing are the fiducial parameters shown in Table 3.2 except for E_a = 2.0 $k_B T_C$ and a cure temperature of 1.7 T_C (T_C=439.36 K is the calibration temperature as described in Chapter 3). The cooling method used here is quenching and the quench simulation parameters used are shown in Table 4.9. The diffusivity is seen to be increasing as a function of temperatures in Figure 4.20. However, this data does not indicate a significant slow down of the beads characteristic of the glass transition.



(a) PRM-a (b) HFM-c



(c) HFM (d) PLFM

Figure 4.21: T_g of the DPD model as a function of cure fraction (α) measured using the PRM-a, HFM, HFM-c and PLFM (dots). The solid lines show the DiBenedetto equation fit (Equation 4.10). The star shows the experimentally observed $T_g(\alpha = 1) \approx 480K$[47]

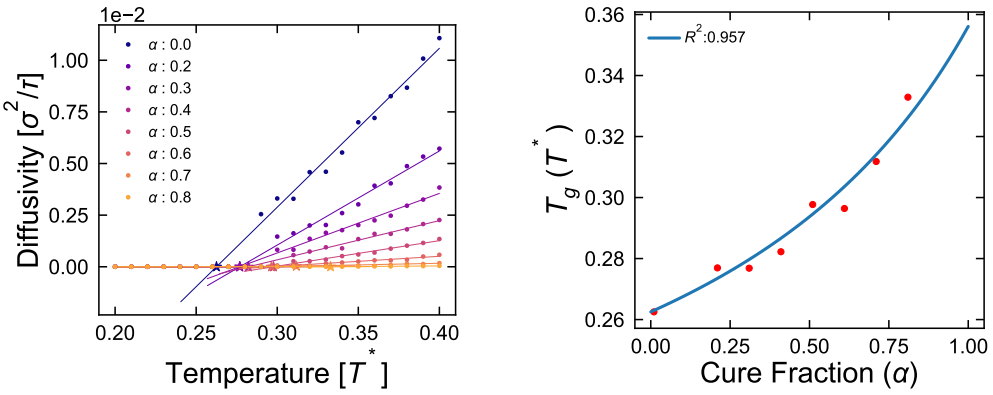
The HFM was found to be unsuitable to fit this data because some of the detected T_g values were unphysical negative values and PRM was found to be impractical because unlike the LJ/Harmonic model's diffusivity data, these diffusivity values were relatively large even at low temperatures. The PLFM was found to be very similar to the HFM-c and hence not shown. The PRM-a (Figure 4.20a) and HFM-c (Figure 4.20b) fit the data well. However, the poor DiBenedetto fits (Figure 4.21) indicate that the DPD model is not suitable to produce the expected trend of decreasing diffusivity with increasing cure fraction (α). The HFM-c (Figure 4.21b) shows a slight increase in T_g for $\alpha > \alpha_{gel}$ where α_{gel}, the cure fraction at gel points is ≈ 0.5. However, the T_g at α=1.0 is much lower than the experimentally observed $T_g(\alpha = 1.0) \approx 480K$[47] (red star in Figure 4.21b). The PLFM (Figure 4.21d) also detects very lower T_g values similiar to HFM-c.

Based on the observation that T_g does not follow a physical trend and the high bond crossing probability ($X_{BC} = 0.32$) as seen in Table 4.6, it is clear that the DPD model is not suitable for simulating glassy dynamics, specifically to measure T_g.

LJ/Harmonic Model

The LJ/Harmonic model described in Table 4.8 is used to simulate glass transition where the LJ interaction parameters used are given in Table 4.7. Since the simpler A/B binary mixture has already been studied in the previous sections, the A/B/C ternary mixtures are now considered where the A and B beads are reacting beads and the C beads are non-reacting 10-mer chains similar to the DPD/Harmonic model in Section 3. The ratio of A:B:C considered here is 1:2:2. The diffusivities of the A/B/C ternary mixture (Figure 4.22a follow a similar trend as the A/B binary mixtures. The T_g values are calculated using PRM, similiar to the A/B binary mixtures except that

the upper temperature as seen in Figure 4.22a and the DiBenedetto equation fits the T_g data well as shown in Figure 4.22b.



(a) (b)

Figure 4.22: The annealed simulations used to measure (a) diffusivity and (b) the DiBenedetto equation (Equation 4.10) fit for the T_g data as a function of cure fraction. The self diffusion coefficients of the A particles in the A/B/C mixture are used here.

4.4.5 Effect of Chain Length

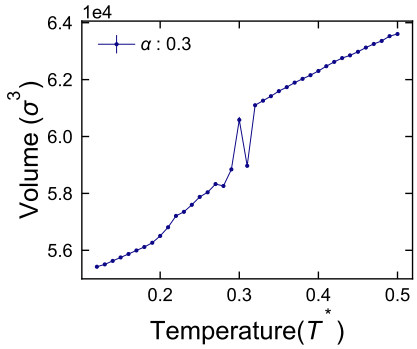
It is well known that T_g decreases with decreasing chain length of polymer chains. The Flory-Fox equation[111] quantifies the relation between T_g and the number averaged molar mass ($\overline{M}_n$) as

$$T_g = T_g^{\infty} - \frac{K}{\overline{M}_n}, \tag{4.11}$$

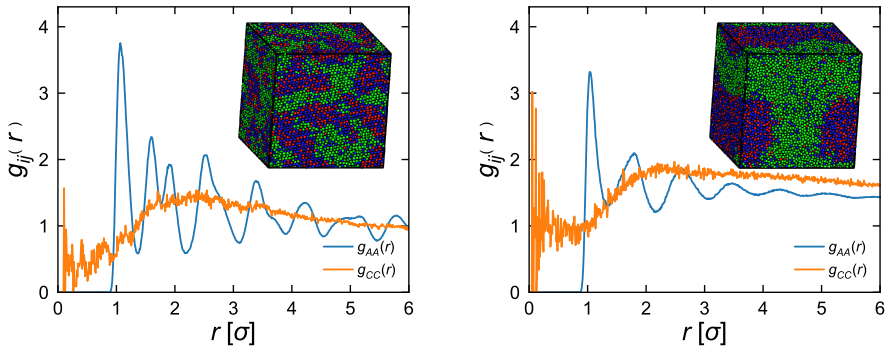
where T_g^{∞} is the T_g of infinitely long chains and K is a positive constant value. The notion of "free volume" can be used to describe the relation between T_g and chain length. The concentration of chain-ends increase with decreasing average chain length in a system of constant density. Chain-ends result in more free volume than segments in the middle of chains because the chain-ends have fewer constraints for motion and

more free volume allows the system to shrink more at lower temperatures, resulting in lower T_g. Free volume is sometimes referred to as the maximum amount of potentially compressible space in a system[111].

The chain length of the C-C linear chains in the A/B/C ternary mixtures were increased from 10 to 100 to observe its effect on the glass transition behavior.



(a)



(b) 0.28 T^* (c) 0.35 T^*

Figure 4.23: The length of the C-C chain when increased to 100, causes the C beads to undergo glass transition below $T \approx 0.3\ T^*$ ($g_{CC}(r)$ in (b)), but the A and B beads crystallize ($g_{AA}(r)$ in (b)). Above the phase transition temperature all beads loose long range order as seen in the A-A and C-C radial distribution functions in (c). The system shown here is below gel point ($\alpha \approx< 0.5$) so that the crystallization of A and B can be observed.

We know from previous experiments that the A and B beads exhibit crystallization below gel point and at a phase transition temperature of $T \approx 0.28\ T^*$ and at much lower temperatures the structure remains amorphous because of a kinetic arrest. In Figure 4.23 we see a similar trend for a 30% cured system toughened with 100-mers of C beads where the quench method was used to cool the system from $0.5\ T^*$ to $0.1\ T^*$. The discontinuity in volume at $T \approx 0.28\ T^*$ in Figure 4.23a indicate a first order phase transition. However, the discontinuity appears to be noisy indicating that there might be more than one type of transition occurring at $T \approx 0.28\ T^*$. Above the phase transition temperature of $T = 0.35\ T^*$, both A and C beads are amorphous as seen in the radial distribution functions ($g_{AA}(r)$ and $g_{CC}(r)$) in Figure 4.23c. But at the phase transition temperature of $T = 0.28\ T^*$, only the A and B beads crystallize ($g_{AA}(r)$), whereas the C beads remain amorphous ($g_{CC}(r)$) as seen in Figure 4.23b because the C-C chains are long enough to avoid crystallization and the A-B chains in this system are short enough to result in crystallization. The A-B-rich phase in this system goes through a first-order phase transition while the C-C-rich phase goes through a glass transition at approximately the same temperature which explains the noisy transition observed in Figure 4.23a.

4.4.6 Effect of Angle Constraints

Two variants of the A/B/C ternary mixture are considered to understand the effect of angle constraint on T_g. The first variant where no angle constraints are imposed on the C-C-C bonds is called the Freely Joint Chains (FJC) and the variant where the C-C-C bond angle is constrained is called the Freely Rotating Chains (FRC). The bond angle in FRC is constrained using a cosine squared angle potential of the form

$$U(\theta) = \frac{1}{2}k_\theta(cos\ \theta - cos\ \theta_0)^2 \tag{4.12}$$

where the equilibrium angle θ_0 = 109.6° and the angle constant k_θ = 25 E. This angle is typical of linear polymer chains[16,116]. The diffusivity data obtained from annealed simulations is used to measure T_g using the PRM method.

The DiBenedetto equation fit for the two systems shown in Figure 4.24 indicates that the T_g increases with angle constraints on the C-C-C bond angle.

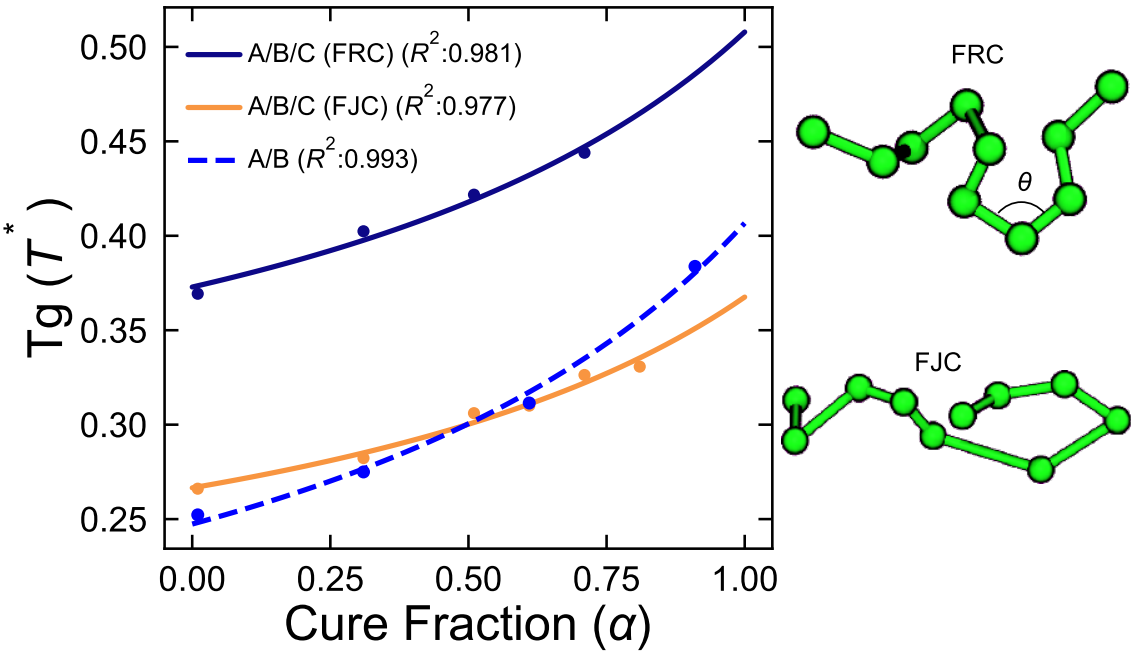


Figure 4.24: The FRC and FJC systems are A/B/C ternary systems which contain C 10-mers with constrained (FRC) and unconstrained (FJC) C-C-C bond angles (The inset images show single FJC and FRC C 10-mers). The A/B binary system does not contain the C 10-mers. The solid line shows the DiBenedetto equation fits for the T_g data. The A/B/C system with FJC has lower T_g than the A/B system at high cure fractions. With FRC, the A/B/C system exhibits higher T_g than the A/B system.

The increase in T_g with angle constraints can be understood using the concept of free volume. The amount of free volume accessible by the FRC is lower than the FJC

because of the constrained motion similar to the constrained motion of chain interiors compared to chain-ends.

The above result provides a direct observation of the effect of bond angle on T_g. The utility and the relevance of imposing bond angle constraint on these models can be understood by examining the T_g of the untoughened A/B system with the T_g of the toughened A/B/C system. The comparison of T_g data for the A/B binary mixture and the A/B/C ternary mixture reveals that the T_g of the A/B/C mixture at higher cure fractions are lower than that of the A/B binary mixture as shown in Figure 4.24. This is contrary to typical experimental results where toughened systems exhibit a similar or higher T_g than untoughened system[44,47]. A plausible explanation for this discrepancy is the relatively low chain length of the toughener chains used in this model. At cure fractions larger than the gel point, the average chain length of the untoughened system is larger than the toughened system because 40% of mixture consists of 10-mer chains. When an angle constraint of 109.5° is imposed on the bond angle as shown in Figure 4.24, the T_g of the A/B/C system is higher than the A/B system at all cure fractions. This result shows a path to reproducing experimentally observed T_g behaviour of toughened systems using the bond angle constraint without altering the toughener chain length.

4.4.7 Effect of Asymmetric Interaction Parameters

Another factor that affects the crystallization of LJ mixtures is the symmetry of the interaction parameters. Unlike the symmetric parameters used in this section, asymmetric parameters in an LJ mixture such as the well studied Kob-Anderson model where the interaction parameters ϵ_{AA}=1.0, ϵ_{AB}=1.5 and ϵ_{BB}=0.5 do not result in crystallization at low temperatures[55,90,91].

Table 4.10: Asymmetric LJ interaction parameters.

	A	B	C
A	0.9216	0.9600	0.9026
B		1.0000	0.9402
C			0.8840

Table 4.11: Simulation Parameters

Parameter	Value
Bond Distance (A-B,C-C) r_o	1.0 σ
Bond Constant k (A-B,C-C)	100 $\frac{\epsilon}{\sigma^2}$
Density (ρ_n)	1.0
E Factor (Υ)	1.0
Activation Energy (E_A)	3.0 ϵ
Secondary Bond Weight	1.2
Cure Temperature	3.0 T^*
Langevin Diffusive Drag Parameter (γ)	4.5
Δt	0.01 τ
α_{low}	0.0
α_{high}	1.0
T_{low}	0.1
T_{high}	2.0
t_q	$1 \times 10^7 \Delta t$
t_a	$6 \times 10^6 \Delta t$
P	10 $\frac{E}{D^3}$

The asymmetric LJ interaction parameters given in Table 4.10 is examined for crystallization behaviour. The motivation for selecting these parameters are explained in detail in Chapter 5. The simulation parameters used for the curing and cooling simulations are described in Table 4.11. Unlike the LJ/Harmonic system, this system uses an E Factor of 1.0.

The uncured system (α = 0) experiences a first-order transition indicated by the discontinuity in the volume data in Figure 4.25. The increased peak intensities at

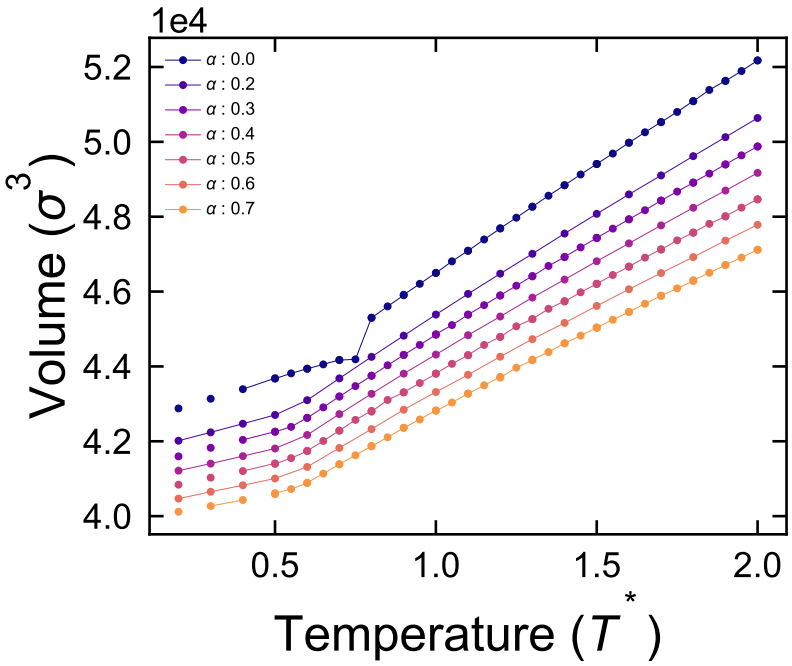
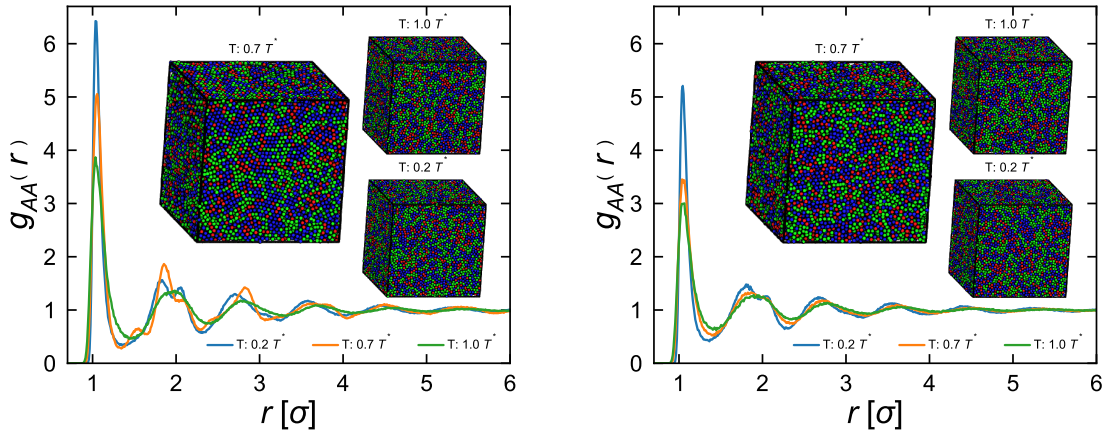


Figure 4.25: The equilibrated volume obtained from the systems that consider the asymmetric energy parameters show discontinuity only for the α = 0 system..

small distances ($r \approx 1.4\ \sigma, 1.8\ \sigma$) in A-A radial distribution function in Figure 4.26a at 0.7 T^* also indicate crystallization and a first order transition. However, the cured systems where $\alpha \geq 0.2$ experience a glass transition. The volume plots as a function of temperature from quench simulations in Figure 4.25 shows continuity for samples where $\alpha \geq 0.2$. The A-A radial distribution of the 30% cured system shown in Figure



(a) 0% cured

(b) 30% cured

Figure 4.26: The A-A radial distribution function for the A/B/C ternary mixture with asymmetric interaction parameters show crystallization at 0.7 T^* for the 0% cured system (b), but is glassy for the 30% cured sample (c) unlike the system with symmetric parameters which crystallize for systems under 50% cure ($g_{AA}(r)$ in Figure 4.23b).

4.26b indicates lack of long range order because there are no significant peaks for interparticle distances $r > \approx 3\sigma$. The morphologies shown in the inset of Figure 4.26b also indicate the amorphous structure at low temperatures (T = 0.2 T^*).

Systems with symmetric parameters in this study experienced the first-order transition at low cure fractions (below gel point) except when long-chain toughener chains (C 100-mers) were used. But with asymmetric interaction parameters, glass transition is observed for systems with cure fractions lower than the gel point even with 10-mer C chains. 10-mer chains using the KG model have been previously shown to undergo glass transition with symmetric energy parameters in Ref. 10. They used asymmetric particle sizes to introduce packing frustration. The asymmetry in the energy parameter introduces a similar effect which is not present in the symmetric energy system resulting in glass transitions rather than crystallization.

To understand the trend in the diffusivity trend for the asymmetric system, the diffusivity values obtained by quenching these systems from 2.0 T^* to 0.1 T^* using simulation parameters shown in Table 4.11 plotted for α = 0.0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 in Figure 4.27. These cure fractions are chosen to examine the behavior of the

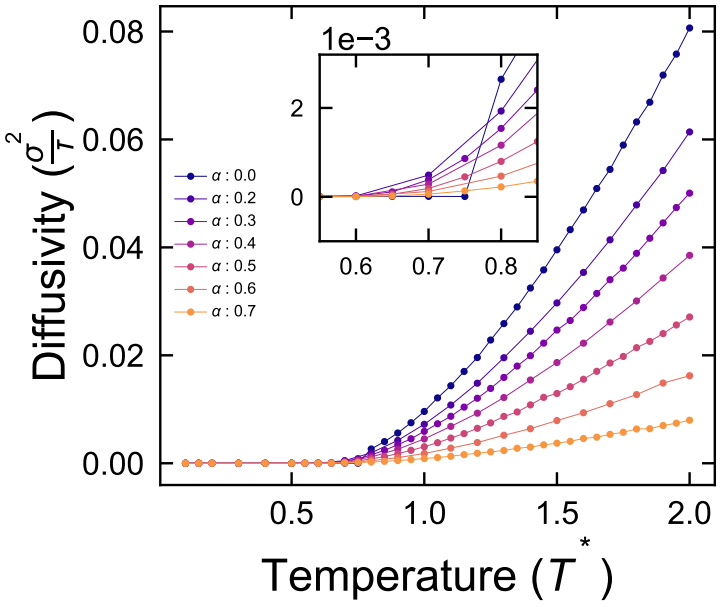


Figure 4.27: The diffusivity measurement obtained from quench simulation of A/B/C ternary mixture of LJ/Harmonic model with asymmetric interaction parameters do not undergoes crystallization for $\alpha \geq 0.2$. The inset figure shows the discontinuity in diffusivity for α = 0.0, which is characteristic of the first order phase transition.

asymmetric system near and below the gel point ($\alpha \approx 0.6$). Both the volume data (Figure 4.25) and the diffusivity data (Figure 4.27) for this system indicate the absence of first order phase transition unlike the A/B and A/B/C systems with symetric interaction parameters. However, the systems where $\alpha < 0.2$ are exceptions and the first order phase transition is observed in both the volume and diffusivity plots for α = 0.0.

Introducing one or more of the other factors that influence glass formation might facilitate glass formation in this system for $\alpha < 0.2$. For the current system composition (40 wt.% toughener chains), it is known that modifying chain length or bond angle does not cause glass formation of the entire system. Asymmetric particle size could result in glass formation at low cure fractions. Finally, the model fitting methods are evaluated for the diffusivity data obtained from asymmetry systems. The T_g values are measured from the diffusivity data using the data fitting methods PRM, HFM, HFM-c and PLFM. The DiBenedetto equation is fit to the T_g data thus obtained and compared in Figure 4.28. The custom data ranges used for the PRM is given in Appendix C6. The quality of fit (R^2 value shown in Figure 4.28) is the highest for the PRM and lowest for the HFM-c.

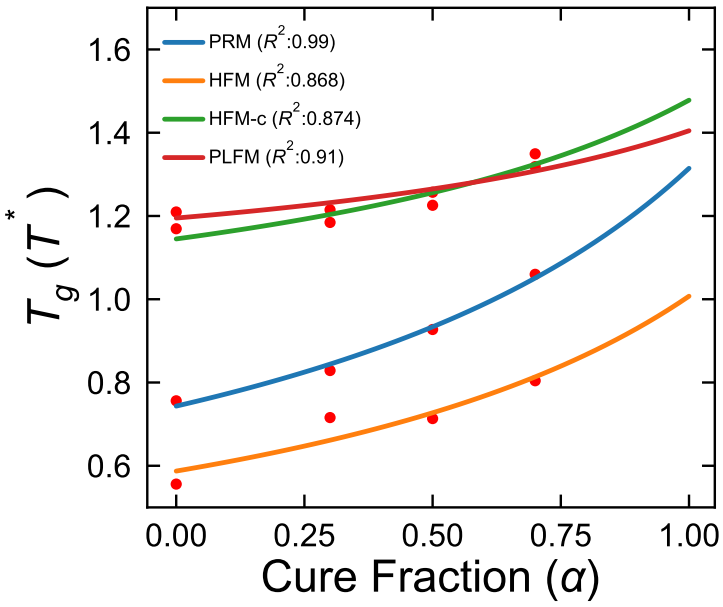


Figure 4.28: Comparing different analysis models to detect T_g from the quench simulations of the A/B/C ternary system with asymmetric parameters.

The PRM and the HFM gives a similar slope in T_g as a function of α even though the absolute values are different. In general, these results are in agreement with the observation for the symmetric systems.

4.5 Conclusion

The bond crossing probability calculation along with the unphysical T_g trend indicates that DPD/Harmonic model is not suitable for modeling the glass transition behavior of crosslinked thermoset polymers. The LJ/Harmonic system is found to be more suitable for modeling glassy dynamics because it has a lower bond crossing probability and agrees well with the DiBenedetto equation which motivates further work to parameterize this model to a specific chemistry of interest. The detection of T_g using molecular dynamics is sensitive to simulation related factors such as the data fitting model used and crystallization. It has been observed that except the PRM-a method, all other methods appear to detect T_g equally well for the toy systems explored in this section. The T_g detected using HFM has been observed to be sensitive to small differences in the diffusivity data between quenched simulation and annealed simulations. Detecting T_g from the structural and dynamic data continues to be a challenging task and the PRM is observed to be the most reliable method. However, the PRM is very tedious because the data range for the diffusivities need to manually selected.

Chain length and degree of cure influence crystallization behavior. It is interesting that the observation of glass transformation is observed only in systems which have undergone gel transition and that the mean chain length of systems below gel transition is ≈ 10 which has previously been observed as a characteristic chain

length below which order-disorder transitions occur[19]. However, this pre-requisite for crystallization is observed only for systems where symmetric interaction parameter is considered. Systems with asymmetric interaction parameters only undergo crystallization for very low cure fractions ($\alpha < 0.2$) compared to the symmetric systems ($\alpha < 0.6$). The asymmetric interaction parameter is found to be the most important factor for avoiding crystallization since none of the other factors cause glassy behavior in systems cured less than the gel point and reproducing the glassy dynamics during early stages of curing is considered important for this research. This observation that the interaction energy is the most important factor for crystallization at low temperatures is congruent with the Flory Huggins theory which predicts that the enthalpic interaction term is more dominant than the entropic term at low temperatures such as the temperatures at which glassy behavior is observed.

The cooling method appears to have no effect on the measured T_g values based on the T_g values obtained from annealing and quenching. The absence of an appreciable difference between the two cooling methods is expected because the missing degrees of freedom in the CGMD model does not cause "jamming" or kinetically arrested states of these molecules as it does in similar AAMD models.

CHAPTER 5

MODELLING THE POLY (ETHER SULPHONE) TOUGHENED DIGLYCIDYL ETHER BISPHENOL-A/4,4'-DIAMINODIPHENYLSULPHONE SYSTEM

5.1 Introduction

The LJ/Harmonic models examined in Chapter 4 were not chemistry specific and the purpose of these models were to understand how these "hard" particle potentials differ from the DPD potential used commonly for modeling epoxy crosslinking[49,65,102]. Mapping coarse-grained molecular dynamics models to physical systems is a long-standing challenge and most parameterization strategies can be categorized generally into either top-down coarse-graining or bottom-up coarse-graining. The bottom-up coarse-graining strategy derives coarse-grained potentials from higher resolution methods such as all-atom molecular dynamics (AAMD) simulations where the main focus is to faithfully reproduce the structural features such as the radial distribution function. Examples of this approach include the Boltzmann Inversion[107], Iterative Boltzmann Inversion (IBI)[84], Multistate Iterative Boltzmann Inversion (MSIBI)[75] and Force Matching (FM) methods[67].

On the other hand methods such as Dissipative Particle Dynamics[34,105] use the top

down coarse-graining strategy where experimentally-relevant thermodynamic properties such as the Flory-Huggins interaction parameter (χ) is used to inform the coarse-grained potential. The top-down approaches in general aim at reproducing the macroscopic properties such as the free energy of mixing rather than reproducing structural detail. The top-down approaches are particularly useful when the purpose of the simulation is to reproduce mesoscale structure rather than atomic scale structure because the relevant size scales are larger than what is derivable from lower scale simulation methods and are easily obtained from experiments or thermodynamic calculations. An example of such an easily obtainable thermodynamic variable is the cohesive energy. Cohesive energy is readily available for most chemistries and has been used extensively to calculate the Hildebrand Solubility Parameter (δ) which can be used quite effectively for estimating macroscopic properties such as miscibility[63]. The Hildebrand Solubility Parameter is also related to the χ parameter through a straightforward relation making it easy to incorporate in CGMD models[102]. In the absence of cohesive energy parameter data for particular chemistry, it can be obtained through fairly straightforward AAMD simulations[102,114]. In essence, this workflow allows the use of a lower scale simulation method to perform top-down coarse-graining. The cohesive energy has also been used to parameterize LJ models in AAMD simulations[113]. The ability to use such readily available chemistry specific data to parameterize coarse-grained molecular dynamics models capable of reproducing glassy dynamics is the main motivation of this study. A recent work by Chremos et al.[19] established a mapping between the LJ interaction parameter ϵ and the experimentally-relevant Flory-Huggins interaction parameter χ and this method was demonstrated for parameterizing the poly (styrene-b-methyl methacrylate) system, a diblock copolymer. They approached the problem of parameterizing the model

by using two different strategies for the interaction between like species ϵ_{ii} and the cross-species interaction ϵ_{ij}. The interaction between coarse-grained beads of like species was obtained from the critical temperature calculated using an analytical method called the group contribution theory. The cross-species interaction term was obtained by deriving a relationship between the χ and ϵ. This is a significant development for parameterizing LJ based coarse-grained models given that this method reproduced the order-disorder transition temperature. However, this method requires that the experimentally obtained empirical constants in the relationship for the χ for the cross-species interaction be available for the systems of interest. Another study[124] used a thermodynamic integration scheme to extract the relation between the cross-species interaction energy ϵ_{XY} and χ. An important distinction between Ref. 19 and Ref. 124 is the choice of mixing rule for the cross interaction. While Ref. 124 used the Lorentz-Berthelot (L-B) mixing rule (Equation 5.1), Ref. 19 chose to derive a more complex mixing rule based on the criterion of reproducing the order-disorder transition temperature because the diblock copolymer system is very sensitive to the choice of the mixing rule.

$$\epsilon_{AB} = \sqrt{\epsilon_{AA}\epsilon_{BB}} \tag{5.1}$$

Here ϵ_{AA} and ϵ_{BB} represents the interaction parameter for A and B particles respecively and ϵ_{AB} is the interaction parameter for cross interaction between species A and B. Ref. 124 also found that systems using the L-B mixing rule showed demixing behaviour when $\frac{\epsilon_{BB}}{\epsilon_{AA}} < 0.88$.

In this chapter, we develop a simple coarse-grain parameterization method based on the L-B mixing rule for reproducing the experimentally observed trend in glass transition temperatures to model the DGEBA/DDS/PES epoxy-thermoplastic mix-

ture. The DiBenedetto equation (Equation 4.10) is fitted to the T_g values to validate the simulations against experimental T_g data. This model of DGEBA/44DDS/PES is then validated against experimental structural relaxation rates. Finally, this model is used to study the sensitivity of the final morphology on the variations of a time-temperature curing profile called the "Step" curing profile.

5.2 Methods

5.2.1 Lennard Jones Parameterization

The coarse-grained molecular dynamics model used in this study is based on the LJ potential function. In this scheme the non bonded interactions use the LJ potential function as shown in Equation 4.2 and the bonded interactions are prescribed the harmonic potential function (Equation 4.3). The σ and ϵ in LJ equation model the particle "size" and the minimum potential energy for particles i and j. r_{ij} is the distance between the particles that correspond to the potential energy U_{ij}. σ and ϵ are constants that need to be determined for specific atom types in an all-atom MD (AAMD) simulation using first-principles calculations or by using empirical methods. Similarly, in coarse-grained molecular dynamics, these values are determined either from lower scale methods such as all-atom MD or by fitting experimental data. We use AAMD simulations to obtain σ using methods described in Ref. 102. The interaction energy between like pairs of particles (ϵ_{AA}) is obtained using the cohesive energy (e_{coh}) between the molecules represented by the corresponding coarse-grained particles. In liquids, e_{coh} represents the energy required to separate molecules from the liquid state into isolated molecules in the vapor phase. Since obtaining these values are not trivial for solids, we estimate it using MD simulations. The values are obtained by

running AAMD simulations of just the molecules represented by the corresponding coarse-grained particle where the e_{coh} is calculated as shown in Equation 5.2.

$$e_{coh} = E_{system} - E_{isolated} \tag{5.2}$$

Here E_{system} is the average potential energy of one molecule in the bulk and $E_{isolated}$ is the average potential energy of one molecule when isolated from all of its neighbors. Essentially, e_{coh} gives us the non-bonded potential energy between two molecules. The e_{coh} values previously calculated[102] from AAMD simulations for DGEBA, DDS and PES monomers are 30.36 kcal/mol, 27.98 kcal/mol and 26.84 kcal/mol respectively. We define the LJ parameter ϵ such that they are normalized with respect to the largest e_{coh} which is that of DGEBA. In this scheme the like interactions (ϵ_{AA}) is defined as

$$\epsilon_{AA} = \frac{e^{A}_{coh}}{e^{B}_{coh}} \tag{5.3}$$

, where e^{B}_{coh} is the largest value (DGEBA) of e_{coh}. The cross interaction between molecules (ϵ_{AB}) are obtained using the Lorrentz-Berthelot mixing rule where

$$\epsilon_{AB} = \sqrt{\epsilon_{AA} + \epsilon_{BB}}. \tag{5.4}$$

The ϵ values thus defined for this system is shown in Table 5.1.

Table 5.1: LJ parameters (ϵ_{ij}) for coarse grained beads representing DGEBA, DDS and PES molecules.

	DDS	DGEBA	PES
DDS	0.9216	0.9600	0.9026
DGEBA		1.0000	0.9402
PES			0.8840

Table 5.2: Fiducial Simulation Parameters

Parameter	Value
Bond Distance (A-B,C-C) (r_o)	1.0 σ
Bond Constant (A-B,C-C) (k)	100 $\frac{\epsilon}{\sigma^2}$
Density (ρ_n)	1.0
E Factor (Υ)	1.0
Activation Energy (E_A)	3.0 ϵ
Secondary Bond Weight	1.2
Cure Temperature	3.0 T^*
Langevin Diffusive Drag Parameter (γ)	4.5
Δt	0.01 τ
α_{low}	0.0
α_{high}	1.0
T_{low}	0.1
T_{high}	3.0
t_q	$1 \times 10^7 \Delta t$
t_a	$6 \times 10^6 \Delta t$
P_q	10 $\frac{E}{D^3}$

We note here that e_{coh} is not directly used as the energy scale but as a relative scale of energy between the different interacting species of molecules. This is not done because it is known that e_{coh} is not a direct measure of the heat of vaporization of molecules in the solid state. If we use the e_{coh} of DGEBA (E=30.36 kcal/mol) as the energy scale, the temperature unit (T^*) turns out to be a very large value of T^* = $1kT$ => $15278K$. In order to derive an appropriate energy scale for our model, we strategically choose a well-defined temperature of the experimental system ($T_g^{exp}(\alpha)$) and equate it to the corresponding dimensionless temperature ($T_g^{sim}(\alpha)$) in the simulated system. The $T_g^{exp}(\alpha$=1.0) of the DGEBA/DDS/PES system is obtained from Ref. 47 and Ref. 73 as 480 K. This value is typically obtained by fitting the DiBenedetto equation[21,79] to the glass transition data for lower cure fractions and extrapolating to 100 % (α = 1.0). Then the energy unit is given by Equation 5.5.

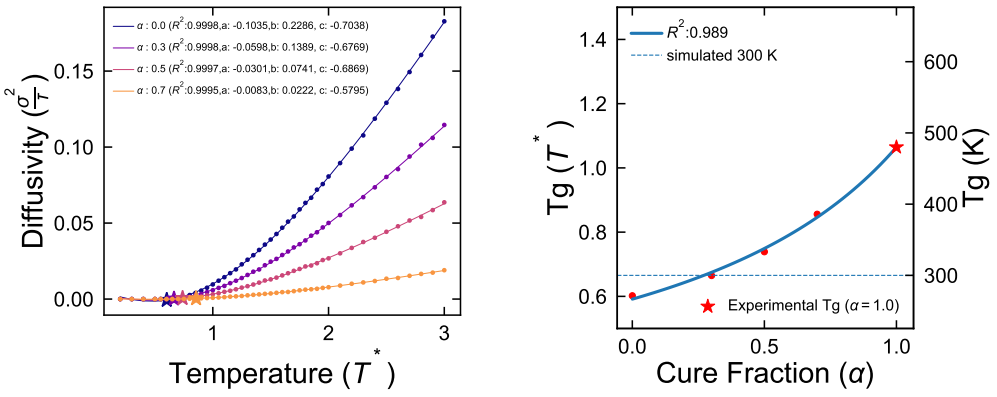
$$E = \frac{k_B T_g^{exp}(\alpha = 1.0)}{T_g^{sim}(\alpha = 1.0)} \tag{5.5}$$

Alternatively the T_g extrapolated to some other reference cure fraction can also be used. As a validation step we compare $T_g^{exp}(\alpha = 0.4) \approx 300\ K$[44] with $T_g^{sim}(\alpha$=0.4) for a similar system (DGEBA/DDM/PES).

5.3 Results

5.3.1 Glass Transition Temperature of DGEBA/DDS/PES

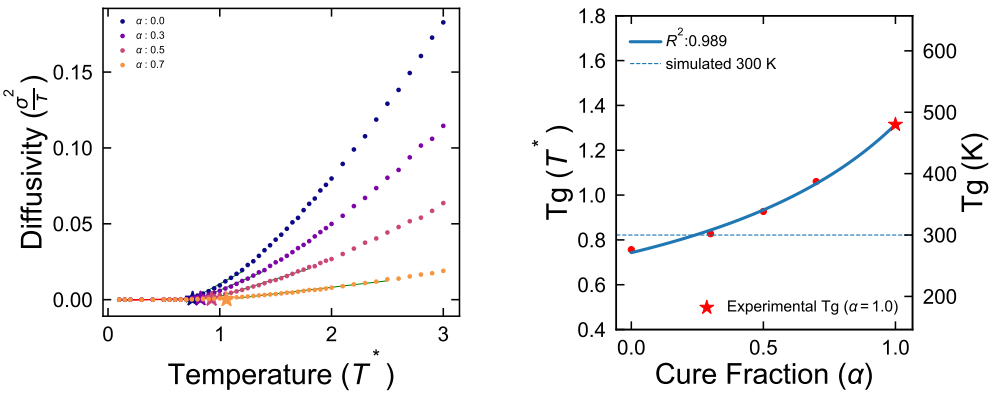
The LJ based system described in Section 5.2.1 has previously been described in Section 4.4.7 and the T_g of the system was measured. The measured $T_g^{sim}(\alpha = 1.0)$ is used to determine the energy scale using Equation 5.5 and the $T_g^{sim}(\alpha = 0.4)$ is compared to see if it is close to the experimentally observed value of 300 K. The



(a) (b)

Figure 5.1: (a) Diffisivity values obtained from quench simulation of the DGEBA/DDS/PES system were used to detect T_g using HFM. (b) The solid curve shows the DiBenedetto equation fit for the T_g data and the dotted horizontal line is a guide for comparing the experimentally observed $T_g(\alpha = 0.4) \approx 300\ K$. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol.

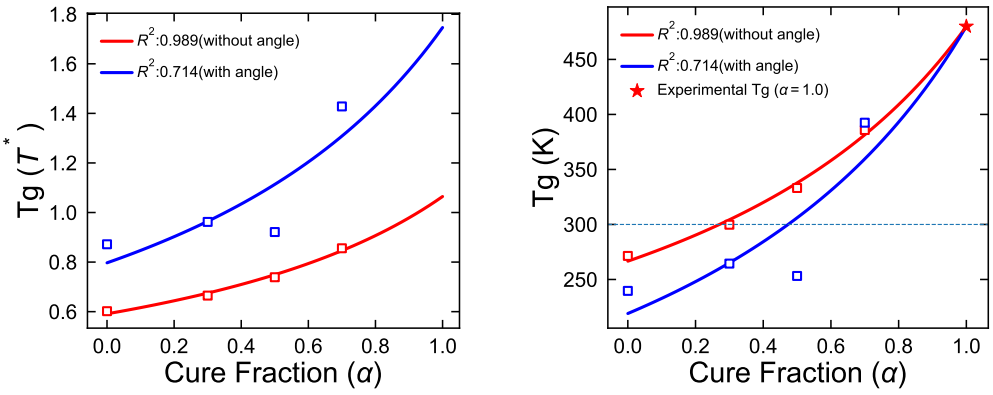
DiBenedetto equation is shown to fit the T_g measured using HFM (Figure 5.1) and PRM (Figure 5.2) well and show close agreement with the experimental T_g of 300 K at α = 0.4. The custom data range given in Appendix C6 is used for fitting the diffusivity data in Figure 5.2a. Since the unconstrained hyperbola fitting



(a) (b)

Figure 5.2: (a) Diffisivity values obtained from quench simulation of the DGEBA/DDS/PES system were used to detect T_g using PRM. (b) The solid curve shows the DiBenedetto equation fit for the T_g data and the dotted horizontal line is a guide for comparing the experimentally observed $T_g(\alpha = 0.4) \approx 300\ K$. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol.

method and the piecewise regression with custom data range selection showed good agreement with the DiBenedetto equation, both methods were used to examine the effect of adding the C-C-C angle potential. It was observed that the unconstrained hyperbola fitting method (Figure 5.3) is sensitive to the variations in the diffusivity data resulting in a low quality of fit for the DiBenedetto function. Figure 5.4a shows that the piecewise regression detected the expected trend that the inclusion of angle potential increases the T_g. The resulting T_g in physical units (Figure 5.4b) show close agreement of the system with angle potential with the experimental value of



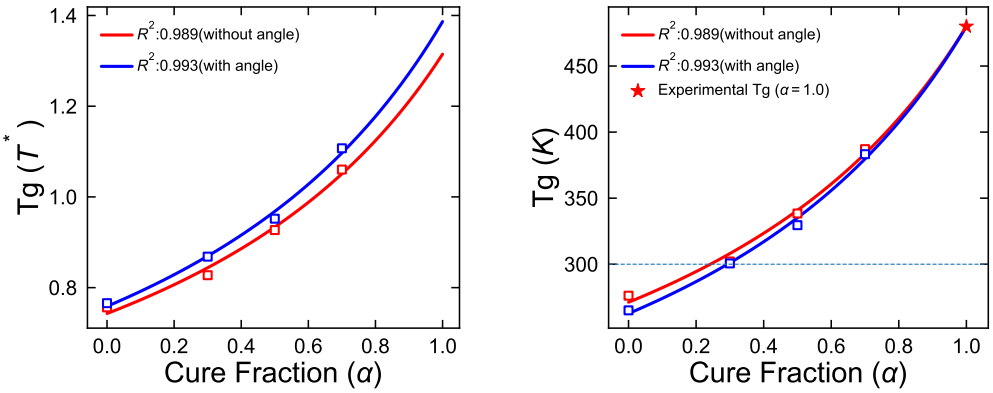
(a) (b)

Figure 5.3: Adding bond angle constraint to the PES chains when HFM is used to detect T_g results in a low quality of fit for the DiBenedetto equation. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol.

$T_g^{exp}(\alpha = 0.4) \approx 300\ K$[44]. Several other untoughened epoxy systems which have a similar epoxy/amine chemistry also shows a similar trend in the DiBenedetto equation where the $T_g(\alpha = 0.4) \approx 300\ K$[73,83,89]. It is known from experiments that the uncured DGEBA/DDS/PES system is completely miscible and flows at room temperature. Both of these conditions are satisfied by the current model. Figure 5.5 shows close agreement of the simulated T_g with that of the experimentally measured T_g for untoughened DGEBA/44DDS[73]. Hence, this condition to validate the physical validity of the T_g measurements from this model is considered well motivation by experiments.

5.3.2 Deriving Physical Units Using $T_g(\alpha = 1.0)$

The $T_g^{sim}(\alpha = 1.0)$ obtained without and with bond angle constraints (Figure 5.4a) are 1.32 T^* and 1.39 T^* respectively. Substituting $T_g^{sim}(\alpha = 1.0) = 1.39\ T^*$ in Equation 5.5, the energy unit E is calculated as $4.77 \times 10^{-21}\ J$ and a temperature



(a) (b)

Figure 5.4: Effect of adding bond angle constraint to the PES chains when piecewise linear regression where the data range was custom selected manually to detect T_g. The $T_g^{exp}(\alpha = 1.0) = 480\ K$ used to derive the energy scale is indicated by the ★ symbol.

unit of $T^* = 345.32\ K$. Similiarly, $T_g^{sim}(\alpha = 1.0) = 1.39\ T^*$ gives an energy unit $E = 5.04 \times 10^{-21}\ J$ and a temperature unit of $T^* = 365.01\ K$. Depending on the model being used (with or without bond angle), the respective units will be used for unit conversions. The unit of mass (M) and distance (D) calculated for this system in Chapter 3 are $4.63 \times 10^{-25}\ kg$ and $1.06 \times 10^{-9}\ m$ respectively. The derived unit of time for this system is calculated as $\tau = 1.05 \times 10^{-11}\ s$.

The curing simulations are run for $6 \times 10^6 \Delta t$ which is equivalent to $\approx 0.6 \mu s$ of time in physical units. This is not completely unrealistic for an MD simulation given that the activation energy used for these simulations ($E_a^{sim} = 3\ E = 1.43 \times 10^{-20}\ J$) are much lower than experimental activation energy of similiar systems ($E_a^{exp} = 1.78 \times 10^{-19}\ J$)[7] and the curing temperature used in the simulations ($T = 3\ T^* = 1035\ K$) is much higher than experimental curing temperature ($T = 313\ K$)[7]. To roughly validate the speedup achieved by using higher curing temperatures (T_{MD})

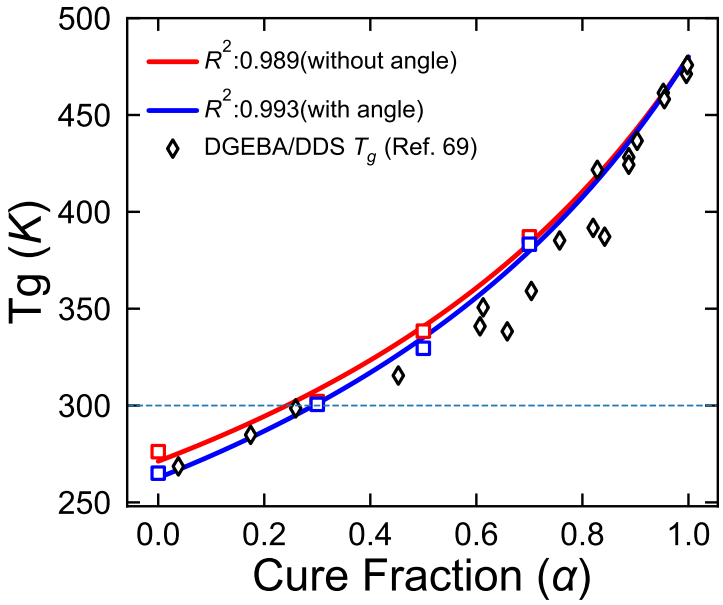


Figure 5.5: The simulated T_g at low and high cure fractions shows close agreement with T_g values measured from an experimental DGEBA-44DDS system[73]. The T_g values are detected using PRM in the simulated systems.

and lower activation energies (E_a^{MD}), a calculation similar to Ref. 81 is done as shown in Equation 5.6 to estimate the reduced activation energy (E_a^{MD}) that will be required to reach similar cure percents at the MD temperature. To make this estimate, the simulated curing experiment which reached a cure fraction of 75% in $t_{MD} = 2.14 \times 10^{-7}$ s at temperature $T_{MD} = 1035$ K was compared with a similiar experiment[7] where 75% cure was achieved in $t_{exp} \approx 1$ h when cured at $T_{exp} = 313$ K and the activation energy was known to be $E_a^{exp} = 1.78 \times 10^{-19}$ J.

$$E_a^{MD} = \left(\ln\left(\frac{t_{MD}}{t_{exp}}\right) - ln\left(\frac{A_{exp}}{A_{sim}}\right) + \frac{E_a^{exp}}{k_B T_{exp}}\right) k_B T_{MD} \tag{5.6}$$

The experimentally observed pre-exponential frequency factor ($A_{exp} = 2.51 \times 10^{14} s^{-1}$)

and the simulated frequency factor ($A_{MD} = 0.026\tau^{-1} = 2.49 \times 10^{9} s^{-1}$) obtained by fitting the cure profile with the first order reaction model[102] is considered in this estimation. The estimated $E_a^{MD} = 8.83 \times 10^{-20}$ J is found to be close to the E_a used in the simulation (1.43×10^{-20} J) and an order of magnitude lower than the experimental value ($E_a^{exp} = 1.78 \times 10^{-19}$ J). Equation 5.6 is convenient to roughly estimate the E_a^{MD} given the amount of simulation time available. For example, if a curing simulation is allowed to run $t_{MD} = 1 \times 10^{9} \Delta t = 0.104$ ms to reach 75 % curing at 313 K, $E_a^{MD} = 5.34 \times 10^{-20}$ $J = 10.59$ E. With a simulation speed of 800 time steps per second, this simulation takes roughly two weeks of GPU time to complete.

5.3.3 Finite Size Effects

As seen previously in Chapter 3, to observe the finite size effects of the model, it is instructive to compare the PES-PES structure factor as a function of system size (N). This study is necessary to understand the minimum system size necessary to observe microstructural features characteristic of this model. System sizes with N ranging from 5×10^{4} to 1×10^{6} was cured to 90% where the fiducial parameters shown in Table 5.2 and simulations were run for 1×10^{7} Δt.

The PES-PES structure factor shown as a function of time in Figure 5.6 indicates that the structure has stopped evolving for $t > 7 \times 10^{6}$ Δt and the final morphology is shown along with the structure factor. On examining the PES-PES structure factor as a function of N in Figure 5.7, a 28.56 nm feature ($q = 0.22$ nm^{-1}) emerges for systems where $N \geq 2 \times 10^{5}$. The feature size seen in the LJ model is slightly smaller than the 35 nm feature size ($q = 0.17$ nm^{-1}) seen in DPD systems in Chapter 3.

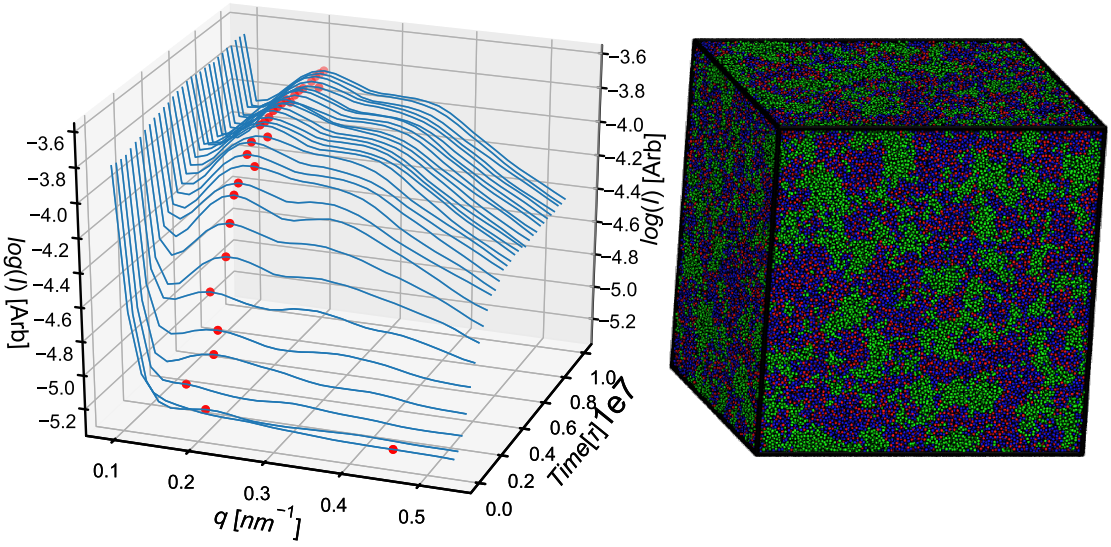


Figure 5.6: The PES-PES structure factor evolution with time (left) for $N = 1 \times 10^6$ indicates that the morphology (right) has equilibrated. The red dots show the detected first peaks (q_{max}).

5.3.4 Morphology Evolution of the DGEBA/44DDS/PES System

The evolution of the first peak (red dots in the PES-PES structure factor in Figure 5.6) is used as an indicator of the morphology evolution motivated by a similar method used to quantify the rate of morphology evolution in experiments[48,122]. In order to study the morphology evolution during curing of the DGEBA/44DDS/PES system, isothermal curing simulations were performed using fiducial parameters shown in Table 5.2 for 1×10^7 Δt. The PES-PES-PES bond angle constraint is not considered in this model. Based on the results of Section 5.3.3, a system size of $N = 4 \times 10^5$ is chosen to be able to observe q_{max}. Five replicate simulations of isothermal curing were run for curing temperature $T = \{1.0, 1.2, ..., 2.8, 3.0\}$ T^*. The $q_m(t)$ values of replicate 1 are shown in Figure 5.8 All the simulations were cured to $\alpha = 0.9$. The simulations cured at $T < 2.0$ T^* did not cure to $\alpha = 0.9$ in 1×10^7 Δt and hence not considered

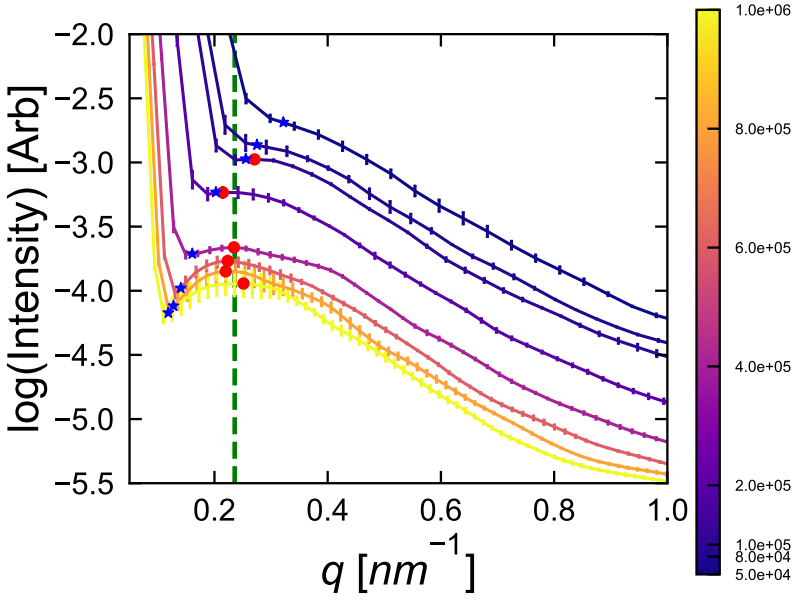


Figure 5.7: The PES-PES structure factor shows emergence of a 0.22 nm^{-1} feature (indicated by the vertical dotted line) at $N \geq 2 \times 10^5$. The color bar indicate system size (N).

for calculating τ. The large q_{max} at low times in Figure 5.8 indicate the formation of small PES domains during the initial stages of curing. As the PES domains grows larger in size, the wave vector q_{max} decreases until it reaches q_0, the equilibrium q_{max}. The relaxation time (τ) of phase separation, a measure of the coarsening time scale of the PES domains is calculated by fitting the simulated $q_m(t)$ data to Equation 5.7 where A_0, q_0 and τ are fitted parameters.

$$q_{max}(t) = q_0 + A_0 \; exp\left(\frac{-t}{\tau}\right) \tag{5.7}$$

The William-Landel-Ferrel (WLF) equation given in Equation 5.8 is used to qualitatively compare the τ as a function of cure temperatures to the experimentally measured τ values.

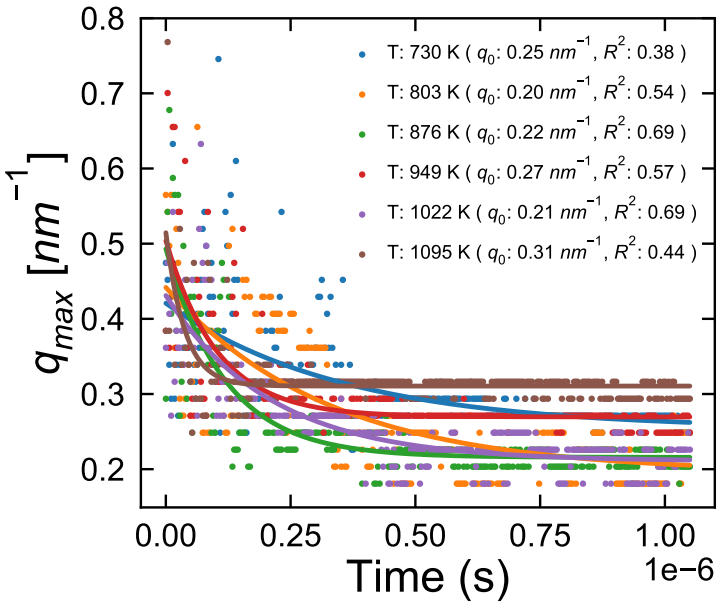


Figure 5.8: The evolution of structure is tracked using q_{max} for one simulation per cure temperature as shown in the legend. The data points in dots are the first peaks (q_{max}) detected in the PES-PES structure factor as a function of time as the system is undergoing curing reaction. The solid curve shows the relaxation time of phase separation function (Equation 5.7) fit for the q_{max} data.

$$log\left(\frac{\tau}{\tau_s}\right) = \left(\frac{-C1(T - T_s)}{C2 + (T - T_s)}\right) \tag{5.8}$$

$C1 = 8.86\ K$ and $C2 = 101.6\ K$ are the chemistry independent universal WLF constants[112] that are known to hold true for a variety of glass forming organic and inorganic materials in the temperature range $T = T_g$ to $T = T_g + 100\ K$. T_s and τ_s are obtained by fitting the simulated $\tau(T)$ data and is used to compare with the experimentally observed value of τ_s and T_s. T_s is known to be $\approx T_g + 50\ K$ and is useful to check the validity of the WLF function against the observed T_g value. The relaxation time of phase separation (τ) has been measured and shown to fit the WLF equation for a similar system where the DGEBA/44DDS system was toughened with

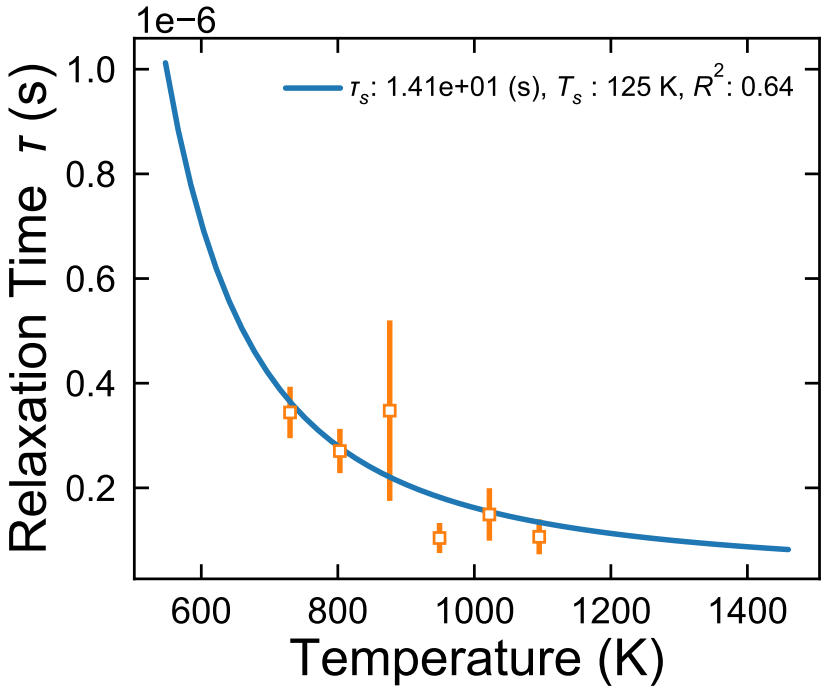


Figure 5.9: The solid line shows the WLF equation fit to the mean structural relaxation time τ (squares) obtained from 5 replicate simulations. The error bars show the standard error from the replicate simulations and the legend shows the WLF parameters obtained from the WLF fitting.

poly(acrylonitrile-butadiene-styrene) (ABS)[48]. Ref. 48 reported $\tau_s = 1.63 \times 10^6$ s and $T_s = 321$ K for a DGEBA/DDS/ABS system. Ref. 122 reported $\tau_s = 1.76 \times 10^6$ s and $T_s = 308$ K for a PES toughened DGEBA system where the crosslinker species was methyl tetrahydrophthalic anhydride (MTHPA).

Both of these systems were reported to have a $T_g \approx 280$ K which is significantly lower than the system being modeled in this study. In the absence of experimental τ values for the DGEBA/44DDS/PES system, data from these two toughened epoxy systems serves well for qualitative comparison. The τ_s values obtained from the simulation as shown in Figure 5.9 is 14.1 s which is $\approx$ 5 orders of magnitude faster

than experiments. The faster curing times of this model is expected because of the lower activation energy and higher curing temperatures compared to the experiment. The fitted value $T_s = 125\ K$ is found to be much lower then the expected value of $T_s \approx T_g + 50\ K = 530\ K$. Much longer simulation times will be necessary to obtain τ values for lower cure temperatures so that q_{max} can reach the equilibrium value of q_0. The agreement of the WLF function with the T_g value indicates that this system obeys the time-temperature superposition principle.

5.3.5 Sensitivity to the "Step" Time-Temperature Curing Profiles

A non-isothermal curing profile referred to as the "Step" profile depicted in Figure 5.10 is examined to understand the sensitivity of the variable times and temperatures of this curing profile on the resultant microstructure. The final microstructure obtained for different curing profiles are compared while keeping the final cure temperature and final cure fraction the same. The LJ/Harmonic model with interaction parameters described in Table 5.1 is used with fiducial parameters shown in Table 5.2 for the curing simulations and the PES-PES-PES bond angle constraint is ignored. A system size of $N = 4 \times 10^5$ is chosen such that finite size effects are not observed and the PES-PES structure factor shows q_{max}. Two different curing times are varied to understand time-temperature sensitivity, namely $t1$ and $t2$. The $t1$ time is the time at which the curing temperature is ramped up from the initial curing temperature ($T1$) to a higher temperature ($T2$). The $t2$ time is the time at which the $T2$ curing phase ends and is ramped down to a lower temperature ($T3$) and held constant till time $t3$. The temperature increase between $T1$ and $T2$ and the temperature decrease between $T2$ and $T3$ is almost instantaneous in the simulations.

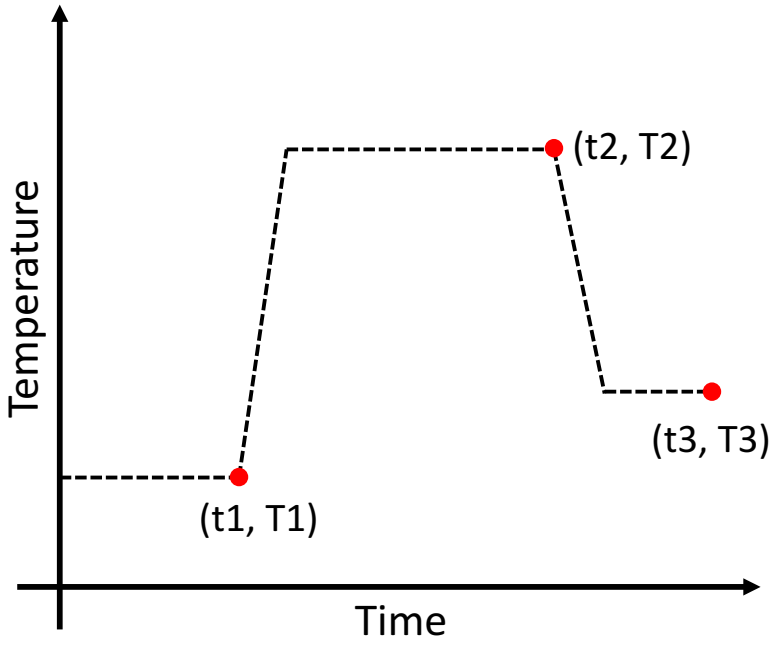


Figure 5.10: The "Step" curing profile shown here has three variable times and temperatures indicated by the red dots.

Sensitivity to t1

The initial ramp up time $t1$ is varied between 1.5×10^5 Δt and 4×10^6 Δt as shown in the temperature plot as a fuction of time in Figure 5.11 to understand how the gelation time (t_{gel}) changes. The curing profiles for the $t1$ study considers $T1 = 2\ T^* = 730\ K$, $T2 = 3.5\ T^* = 1278\ K$, $t3 = 1 \times 10^7\ \Delta t$ and does not include a ramp down to $T3$. Based on the observation of t_{gel} in Figure 5.12, $t1 < 2 \times 10^5\ \Delta t$ has no effect on t_{gel} and ramping up temperature to $T2$ at times later then $t2 = 2 \times 10^5\ \Delta t$ delays the gelation significantly. The cure fraction plot in the inset of Figure 5.12 where the time is in log scale shows that the sigmoidal shape of the curing profile ramps up at $t \approx 1 \times 10^5\ \Delta t$. The curing rate is observed to be significantly lower before the start of the sigmoidal ramp. Based on this study we chose $t1 = 1 \times 10^5\ \Delta t$ conservatively.

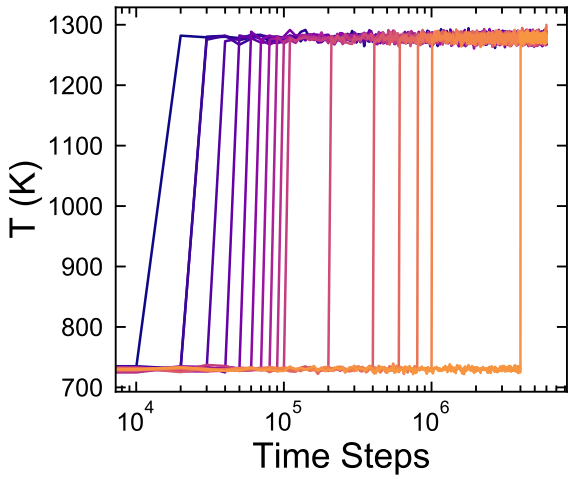


Figure 5.11: Temperature profiles where the initial ramp up time (t1) is varied

It was also observed that the final cured morphology quantified by the PES-PES structure factor (not shown here) has no significant difference for simulations where $t1 < 1 \times 10^5$.

Sensitivity to t2

To study the effect of $t2$ on the "Step" curing profiles, two different $t2$ times are considered where $t2 < t_{gel}$ ($t2 : 2 \times 10^6\ \Delta t$) and $t2 > t_{gel}$ ($t2 : 9.5 \times 10^6\ \Delta t$) as shown in Figure 5.13a and 5.13b. The temperature set points used are $T1 = 1\ T^* = 365\ K$, $T2 = 2.0\ T^8 = 730\ K$ and $T3 = 1.2\ T^* = 438\ K$. $T2$ is chosen such that it is much higher than $T_g(\alpha = 1.0) = 480\ K$ and $T3$ is chosen such that it is lower than $T_g(\alpha = 1.0)$. The time set points used are $t1 = 1 \times 10^5\ \Delta t$ and $t3 = 1 \times 10^7$ for the simulation where $t2 > t_{gel}$ ($t2 = 9.51 \times 10^6$ as shown in Figure 5.13a) and $t3 = 3 \times 10^7\ \Delta t$ ($t2 = 2.01 \times 10^6$ as shown in Figure 5.13a) such that the final cure fraction of both the simulations are greater than 0.85. The PES-PES structure factor shown in Figure 5.14 indicates that the first peak for $t2 = 2 \times 10^6\ \Delta t$ has

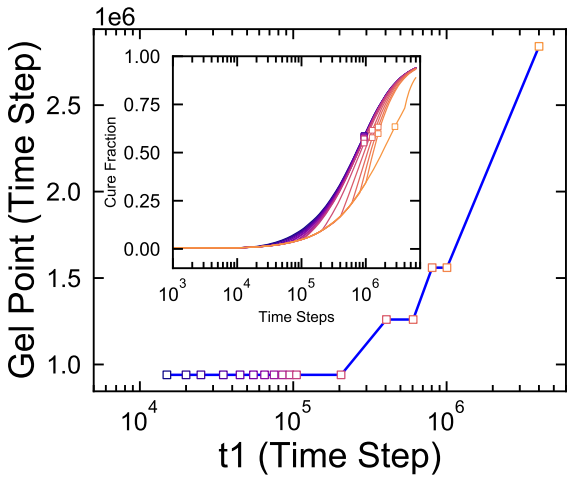
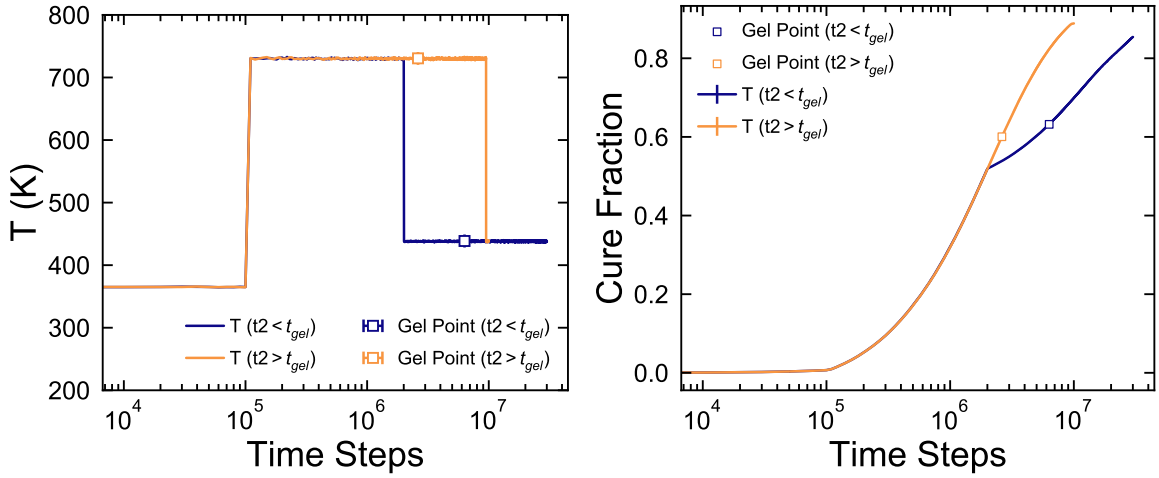


Figure 5.12: Time to gelation is not affected by $t1 < 2 \times 10^5\ \Delta t$. t1 time denote the time at which the cure temperature is ramped up and held constant.

a higher wave vector q_{max} than $t2 = 9.5 \times 10^6\ \Delta t$ indicating that quenching before gelation results in smaller PES domain sizes and less phase separation of the PES toughenners. The visualization of the final morphologies in Figure 5.14 also suggests that the $t2 = 2 \times 10^6\ \Delta t$ resulted in a microstructure where the PES toughereners are less phase seperated than $t2 = 9.5 \times 10^6\ \Delta t$. The reduced phase separation of PES toughenners for $t2 < t_{gel}(t2 = 2 \times 10^6)$ can be attributed to the suppressed diffusion rates caused by the low-temperature ($T3 = 438\ k$) curing phase between the t2 to t3.

5.4 Conclusion

The T_g values of the LJ/Harmonic based model fits the DiBenedetto equation well using both PRM and HFM. The energy scale of the coarse grained LJ model can be obtained by equating the simulated T_g to the experimental T_g using Equation 5.5. The T_g at the reference cure fraction of $\alpha = 0.4$ shows close agreement with experiments (300 K) indicating that the uncured system will be liquid at room temperature as



(a) (b)

Figure 5.13: Temperatures profiles (a) and curing profiles (b) for $t2 < t_{gel}$ (t2=2×10^6 Δt) and $t2 > t_{gel}$ (t2=9.5×10^6 Δt). The hollow squares show gel point. $T2$ is chosen to be higher than and $T3$ is chosen to be slightly lower than the T_g of the fully cured system ($T_g(\alpha = 1.0) = 480\ K$).

seen in experiments. The T_g detection methods have all shown to challenging. The HFM is sensitive to the diffusive data and the PRM needs to be manually provided with the data range for fitting the high temperature and low-temperature diffusivities which is very tedious and error-prone. Further research is required to develop more stable and efficient analysis methods to detect T_g.

The morphology evolution due to phase separation was quantified using the relaxation time of phase separation (τ) measured using the first peak in the PES-PES structure factor (q_{max}). The William-Landel-Ferrel (WLF) equation (Equation 5.8) which is known to describe several temperature dependent properties of glass forming materials fits the τ values for high temperature curing simulations.The WLF parameter $T_s = 125\ K$ obtained by fitting the τ values is found to lower than the T_s value calculated as $T_s = T_g + 50\ K = 530\ K$ which suggests that the current model obeys

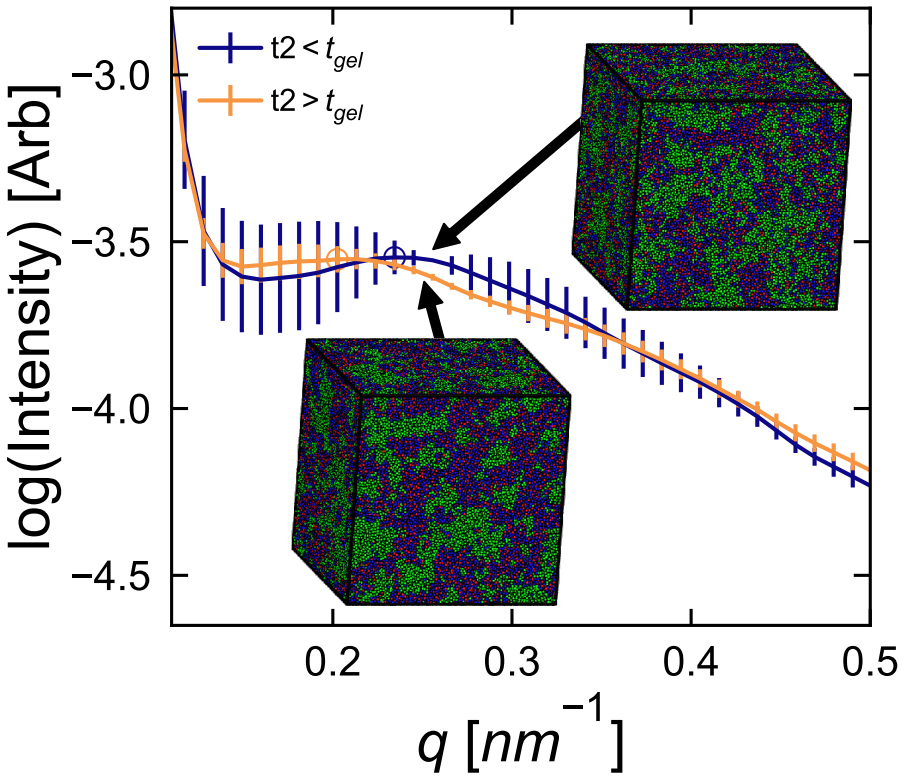


Figure 5.14: PES-PES structure factor shows difference in morphology as a result of varying t2 of the "Step" curing profile. The error bars show standard error from the three replicate simulations. The intensity at $q \approx 0.25\ nm^{-1}$ shows a statistically significant difference indicating that $t2 < t_{gel}(t2 = 2 \times 10^6\ \Delta t)$ resulted in a more mixed microstructure.

the time-temperature superposition principle. The time scales in the WLF fitting are currently much smaller than the experimental time scales as the MD simulations are carried out at much lower activation energy and higher curing temperatures. The diffusive drag coefficient (γ) in the Langevin equation of motion and E factor (Υ) are two possible means of adjusting the diffusion rate. The 29 nm PES-PES feature sizes observed in this model is the largest MD or CGMD simulated PES domain size reported.

The time and temperature variability of the "Step" cure profile is studied for

its effect on the final cured morphology. The $t1$ analysis shows the ramping up of temperature can be delayed to $t1 \geq 1 \times 10^5 \ \Delta t$ without affecting the gelation time and the final morphology which is a more energy efficient curing profile. Based on the statistically significant difference in morphology indicated by the PES-PES structure factors in the $t2$ sensitivity study, the suppression of diffusion rate with a low-temperature cure phase right before the gelation transition results in a less phase separated microstructure which could potentially give rise to better mechanical properties such as toughness. A similar study of a non-isothermal curing profile called "Linear Ramp" using the DPD model (see Appendix D) showed that the "Linear Ramp" produced a macrophase separated morphology in contrast to a microphase-separated morphology produced by the isothermal cure profile which is comparable to the morphology produced by the "Step" profile.

Overall, the present model of DGEBA/44DDS/PES stands out from similar models found in the literature for the following reasons: 1) the reaction rates are calibrated against experimentally observed reaction models, 2) the diffusion rate is validated against experimental rates through the relaxation rate of phase separation and fit against the WLF equation, 3) the computational efficiency of this model allows simulating large enough simulation volume in order to observe the PES toughener feature sizes, 4) this model is able to reproduce the T_g values at different degrees of cure and this is validated against two commonly observed reference T_g values at α = 0.4 and α = 1.0, and finally, 5) the ability to reproduce experimentally-relevant sensitivity to non-isothermal cure profiles that consider knowledge of gel point and glass transition temperature. Table 5.3 compares the current model with several other epoxy curing models found in the literature against these five features of the current model. The first column of the table above shows the reference to the work, the

Table 5.3: Comparison with other DGEBA/44DDS/PES models

{Reference} {Model} {Chemistry}	Reaction Calibration	Diffusion Calibration	PES Feature Size	Matching T_g at reference α	"Step" Profiles
this work CGMD/DPD (DGEBA/44DDS/PES)	Calibrated by fitting reaction models	Calibrated by fitting the WLF Equation	29 nm	α = 0.4, 1.0	Yes
Ref. 65 DPD (DDS/RA/SA)	Uses fixed bond probability P=0.001	No	NA	No	No
Ref. 50 DPD (DGEBA/DETA)	Uses fixed bond probability P=1	No	NA	No	No
Ref. 62 AAMD (DGEBA/33DDS)	Uses fixed bond probability P=1	No	NA	No	No
Ref. 59 CGMD/AAMD (DGEBA/DETA)	Uses fixed bond probability P=1	No	NA	α = 0.4[a]	No
Ref. 51 AAMD (DGEBA/44DDS)	Uses fixed bond probability P=1	No[b]	NA	No	No
Ref. 31 AAMD (DGEBA/DETA)	Uses arbitary bond probability	No	NA	No	No
Ref. 56 EP/CA[c]	Uses arbitary bond probability	No	NA	No	No

model type used and the system being modelled. The second column describes the method used to control the reaction rate. Most models use a fixed probability of 1, which means these models only consider distance criteria for bond formation. Other models use an arbitrary bond probability without any means to calibrate against experimental reaction rates. The third column describes the method used by the

[a] The DiBenedetto equation fit for the T_g values are not shown

[b] The cooling rate dependence of T_g is shown to fit the WLF equation

[c] 3,4-Epoxycyclohexylmethyl-3,4-epoxycyclohexanecarboxylate (EP)/4-methylhexahydrophthalic anhydride (CA)

model to calibrate the diffusion rate of the system against experiments. The fourth column describes the characteristic feature sizes measured in the simulated system. None of the other crosslinked systems characterized microstructure. This is perhaps also because most of these models are neat epoxy systems rather than a toughened system. The fifth column compares the cure fractions (α) at which the simulated T_g values were validates against experimentally observed T_g. The sixth column compares the ability of the model to show sensitivity to non-isothermal cure profiles such as the "Step" profile. None of the other crosslinking models for epoxy systems compared here attempted to characterize the sensitivity of morphology to the curing profiles.

Based on the data presented in Table 5.3, there is only one other model that considers a toughened epoxy system and only one other model matched T_g with experimental data at any other reference cure fraction other than α = 1.0.

CHAPTER 6

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

6.1 Conclusions

The main objective of this research is to develop a computational model of the epoxy curing process such that the model accuracy can be maximized while minimizing the model complexity and computational cost. The major outcomes of this research can be linked to these three guiding principles.

In Chapter 3, an open-source plugin was developed for HOOMD-Blue[8,32] that enables high-throughput simulation of crosslinking epoxy thermosets. A typical DPD simulation of system size, $N = 5 \times 10^4$ can now be crosslinked in under an hour of GPU time. Figure 6.1 shows the scaling of the performance with system size. These capabilities lead to the ability to routinely simulate large simulation sizes and which are necessary to understand the finite size effects associated with these simulations. Figure 6.2 shows that in order to observe feature sizes that are about 40 nm, the system size should be larger than 1.2×10^6 for the DPD model described in Chapter 3. The low performance-overhead makes it possible to run multiple replicate simulations of epoxy crosslinking which is invaluable for quantifying the uncertainty in structure (measured using structure factor) that is sensitive to the cure path.

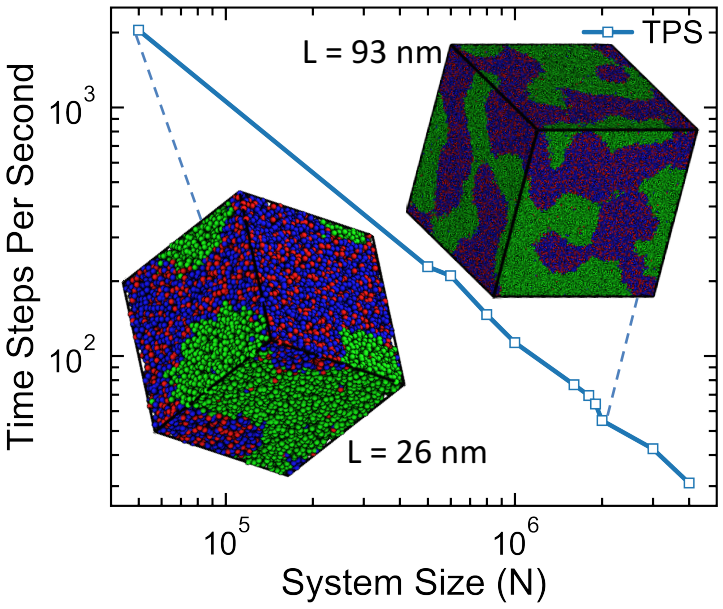


Figure 6.1: The TPS as a function of system size shows performance of the DPD/Harmonic model.

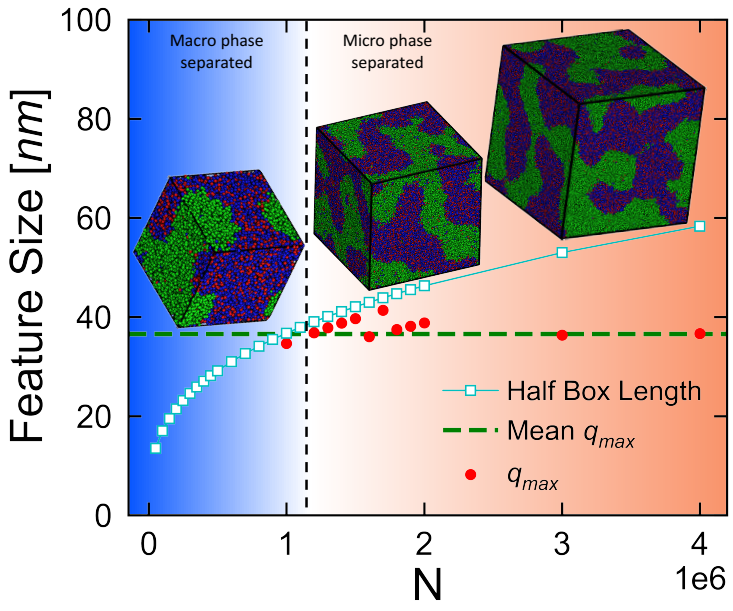


Figure 6.2: For simulations where $N < 1.2 \times 10^6$ Δt.

In Chapter 4, the DPD/Harmonic model, the LJ/Harmonic model and the LJ/FENE

model were evaluated for the ability to not avoid unphysical bond crossing using a molecular statics method. The LJ/Harmonic model is found to be more suitable than the DPD/Harmonic model for avoiding unphysical bond crossing because the LJ/Harmonic model has a relatively low probability of bond crossing (X_{BC} = 0.026) compared to the DPD/Harmonic model (X_{BC} = 0.026) at a typical simulation temperature (T = 1 T^*). To completely avoid bond crossing and hence entanglements, it is necessary to use the LJ/FENE model which has a much lower bond crossing probability (X_{BC} = 5.09×10^{-12}) at T = 1 T^*. Because of its computational efficiency, the LJ/Harmonic model is utilized to study glass transitions given that entanglements are not a necessary condition for reproducing glass transition as seen in LJ homopolymer models (see Table 4.1). Four different analysis models were evaluated to detect the glass transition temperature (T_g) and found that the PRM is more reliable than PRM-a, HFM, HFM-c and PLFM even though the PRM is very inconvenient because of the manual data range selection requirement. Apart from the data fitting method used, the choice of thermodynamic data used to measure T_g, the cooling method, the simulation model, angle constraints, chain length and asymmetric interaction parameters were also examined. Even though the entropic factors such as chain length and angle constraints causes glass transitions, it only influences systems cured beyond gel point, limiting the applicability of this coarse-grained model to only highly crosslinked systems and not the early cure states. But the enthalpic factor such as the asymmetric interaction parameter enables glass transitions in systems cured less than the gel point which is a necessary condition for modeling epoxy systems using CGMD.

In Chapter 5, a coarse-grained molecular dynamics model based on the LJ/Harmonic model is parameterized to represent a specific DGEBA/44DDS/PES chemistry

by matching the T_g of the fully cured experimental system with that of the simulated system. The T_g of the simulated model is found to closely match the experimental T_g at another reference cure fraction (α = 0.4). Furthermore, the simulated model is found to fit the William-Landel-Ferrel (WLF) equation implying that the model satisfies the time-temperature superposition principle. The effect of a non-isothermal time-temperature curing profile called the "Step" profile on the final morphology is also examined using the newly developed model and found that the knowledge of the gel point (t_{gel}) and glass transition temperature (T_g) can be leveraged to obtain significantly different microstructures reliably.

Over 15,000 simulations were performed during the course of this research on four different computing cluster Fry, R2, Kestrel and Garcia.

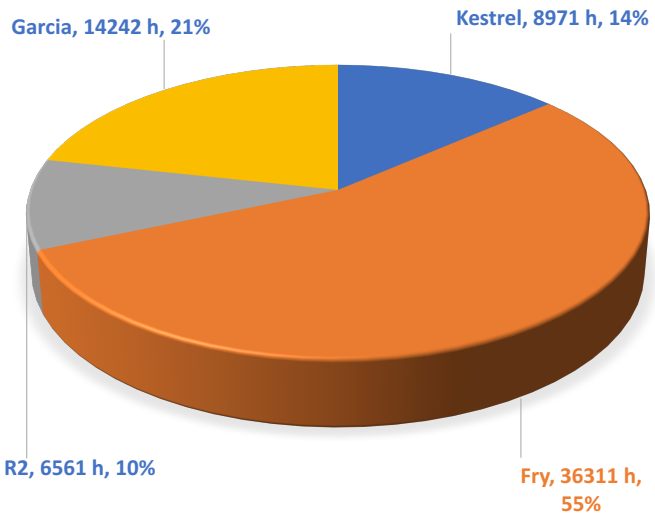


Figure 6.3: A total of 66085 GPU hours of simulations have been run on the 4 clusters during the course of this research. This information is based on the log files produced by the simulations.

This amounts to a total simulation time of roughly 66,000 GPU hours (equivalent to 7.5 years of simulation time on a single GPU) distributed among the four different

supercomputers as shown in Figure 6.3 which does not include the CPU hours used for post-processing the simulation results. In order to manage and process large amounts of data produced by these simulations in a reproducible and sustainable fashion, robust software engineering strategies are imperative. An open-source software stack depicted in Figure 6.4 is developed as part of this research which is all freely available at `https://bitbucket.org/cmelab`.

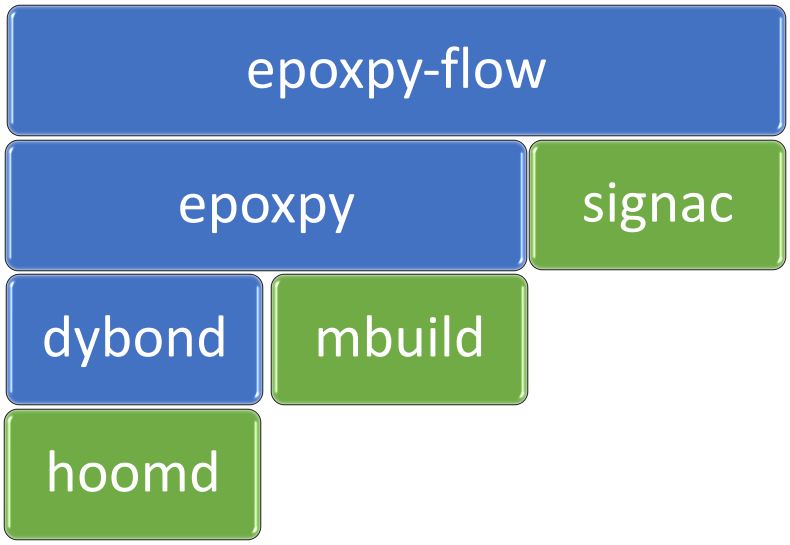


Figure 6.4: The newly developed parts of the software stack used to run the simulations are shown in blue and the green boxes show the software libraries that are leveraged by this software stack.

6.2 Suggestions for Future Work

Plenty of further research and developments opportunities exist for this research for the development of the methods as well as the scientific aspects. The performance profiling (Appendix A.3) results using 50000 particles indicated that the neighbor list creation is the most time-consuming part of the simulation code. However, it is

observed that for larger systems (observed for $N = 4 \times 10^5$) the "dybond plugin" is the slowest part of the simulation code and this presents an opportunity to further improve the "dybond" plugin by performing the scan for bondable particles parallelly on the GPU rather than on the CPU as it is currently done. Another opportunity for expanding the applicability of the "dybond" plugin is to allow bond breaking capability. More method development opportunities are certain to emerge as a result of the application of this method to completely different material phenomena. Two examples of such application areas for the "dybond" plugin are the synthesis of two two-dimensional polymers[53] and adsorption of gas particles on surfaces. A new bottom-up route for synthesizing two-dimensional polymers involves a two-step process of first forming the crystal lattice followed by formation of covalent bonds[53] which is both easily modeled using the current model.

It will be interesting to understand the importance of the fine-grained details of the atomic resolution for further probing mechanical properties such as elastic modulus of these epoxy systems. A test would be to back map the coarse-grained simulation elements into its constituent atoms and compare the property measure from the full resolution system with the coarse-grained system. Another area of opportunity is to explore other methods such as MSIBI[75] to systematically coarse grain using a bottom-up coarse-graining approach. It will be highly beneficial to this research to find alternate means to reliably detect T_g from simulations. A recently developed approach is to apply machine learning to structural information to detect T_g[18,90].

From an epoxy science perspective, it will be interesting to explore the properties of off-stoichiometric ratios of DGEBA/44DDS and varying compositions of the PES toughener. Furthermore, other chemistries with a different number of functional groups resulting in different gel points would also be worth studying. It will also be

interesting to study the effect of other tougheners such as nanoparticle tougheners and functional linear chains.

REFERENCES

[1] BISPHENOL A DIGLYCIDYL ETHER — C21H24O4 - PubChem. https://pubchem.ncbi.nlm.nih.gov/compound/2286.

[2] dapsone — C12H12N2O2S - PubChem. https://pubchem.ncbi.nlm.nih.gov/compound/2955.

[3] Epoxy Resin Market Analysis By Application (Paints & Coatings, Wind Turbine, Composites, Construction, Electrical & Electronics, Adhesives) And Segment Forecasts To 2024. http://www.grandviewresearch.com/industry-analysis/epoxy-resins-market.

[4] Lauren J. Abbott, Justin E. Hughes, and Coray M. Colina. Virtual synthesis of thermally cross-linked copolymers from a novel implementation of polymatic. *Journal of Physical Chemistry B*, 118(7):1916–1924, 2014.

[5] Carl S Adorf, Paul M Dodd, and Sharon C Glotzer. signac - {A} Simple Data Management Framework. *CoRR*, abs/1611.0, 2016.

[6] Carl S. Adorf, Paul M. Dodd, Vyas Ramasubramani, and Benjamin Swerdlow. csadorf/signac: v0.7.0. https://doi.org/10.5281/zenodo.230356, January 2017.

[7] Michael Aldridge, Alan Wineman, Anthony Waas, and John Kieffer. In situ analysis of the relationship between cure kinetics and the mechanical modulus of an epoxy resin. *Macromolecules*, 47(23):8368–8376, 2014.

[8] J. A. Anderson, C. D. Lorenz, and A. Travesset. General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units. *Journal of Computational Physics*, 227(10):5342–5359, 2008.

[9] Joshua A Anderson and Sharon C Glotzer. The development and expansion of HOOMD-blue through six years of GPU proliferation. *arXiv*, 1308.5587, aug 2013.

[10] Christoph Bennemann, Kurt Binder, B Dünweg, and Wolfgang Paul. Molecular Dynamics Simulations of the Thermal Glass Transition in Polymer Melts. *Physical Review E*, 57(1):843–851, 1997.

[11] Ignazio Blanco, Gianluca Cicala, Carmelo Lo Faro, and Antonio Recca. Development of a toughened DGEBS/DDS system toward improved thermal and mechanical properties by the addition of a tetrafunctional epoxy resin and a novel thermoplastic. *Journal of Applied Polymer Science*, 89(1):268–273, 2003.

[12] A. Bonnet, J. P. Pascault, H. Sautereau, M. Taha, and Y. Camberlin. Epoxy-diamine thermoset/thermoplastic blends. 1. Rates of reactions before and after phase separation. *Macromolecules*, 32(25):8517–8523, 1999.

[13] R. D. Brooker, A. J. Kinloch, and A. C. Taylor. The Morphology and Fracture Properties of Thermoplastic-Toughened Epoxy Polymers. *The Journal of Adhesion*, 86(7):726–741, 2010.

[14] Joachim Buchholz, Wolfgang Paul, Fathollah Varnik, and Kurt Binder. Cooling rate dependence of the glass transition temperature of polymer melts: Molecular dynamics study. *The Journal of Chemical Physics*, 117(15):7364–7372, 2002.

[15] Clive B Bucknall, Clara M Gomezt, and Isabelle Quintard. Phase separation from solutions of poly (ether sulfone) in epoxy resins. *Wiley Interdisciplinary Reviews*, 35(2):353–359, 1994.

[16] Monica Bulacu and Erik Van Der Giessen. Molecular-dynamics simulation study of the glass transition in amorphous polymers with controlled chain stiffness. pages 1–11, 2007.

[17] João T. Cabral and Julia S. Higgins. Spinodal nanostructures in polymer blends: On the validity of the Cahn-Hilliard length scale prediction. *Progress in Polymer Science*, 81:1–21, 2018.

[18] Michele Ceriotti and Vincenzo Vitelli. Vitrification: Machines learn to recognize glasses. *Nature Physics*, 12(5):377–378, 2016.

[19] Alexandros Chremos, Arash Nikoubashman, and Athanassios Z. Panagiotopoulos. Flory-Huggins parameter χ, from binary mixtures of Lennard-Jones particles to block copolymer melts. *Journal of Chemical Physics*, 140(5):1–10, 2014.

[20] G. Di Pasquale, O. Motto, A. Rocca, J.T. Carter, P.T. McGrail, and D. Acierno. New high-performance thermoplastic toughened epoxy thermosets. *Polymer*, 38(17):4345–4348, 1997.

[21] A. T. DiBenedetto. Prediction of the glass transition temperature of polymers: A model based on the principle of corresponding states. *Journal of Polymer Science Part B: Polymer Physics*, 25(9):1949–1969, 1987.

[22] Office of Energy Efficiency Doe and Renewable Energy (EERE)'s Advanced Manufacturing Office. Bandwidth Study on Energy Use and Potential

Energy Saving Opportunities in U.S. Pulp and Paper Manufacturing. (September):108, 2015.

[23] K. Dušek and W. Prins. Structure and elasticity of non-crystalline polymer networks. *Advances in Polymer Science*, 6:1–102, 1969.

[24] Pep Español and Patrick B Warren. Perspective : Dissipative particle dynamics. *The Journal of chemical physics*, 150901, 2017.

[25] P Espanol P. Warren. Statistical Mechanics of Dissipative Particle Dynamics. *Europhysics Letters (EPL)*, 30:191, 1995.

[26] Carla E. Estridge. The effects of competitive primary and secondary amine reactivity on the structural evolution and properties of an epoxy thermoset resin during cure: A molecular dynamics study. *Polymer (United Kingdom)*, 141:12–20, 2018.

[27] Paul J Flory. Molecular Size Distribution in Three Dimensional Polymers. I. Gelation 1. *Journal of the American Chemical Society*, 63(11):3083–3090, nov 1941.

[28] Daan Frenkel and Berend Smit. Understanding Molecular Simulation, 2002.

[29] TIMOTHY C. GERMANN and KAI KADAU. Trillion-Atom Molecular Dynamics Becomes a Reality. *International Journal of Modern Physics C*, 19(09):1315–1319, 2008.

[30] Nicolas Giovambattista, C. Austen Angell, Francesco Sciortino, and H. Eugene Stanley. Glass-transition temperature of water: A simulation study. *Physical Review Letters*, 93(4):047801–1, 2004.

[31] Jacob R Gissinger, Benjamin D Jensen, and Kristopher E Wise. Modeling chemical reactions in classical molecular dynamics simulations. *Polymer*, 128:211–217, 2017.

[32] Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications*, 192:97–107, 2015.

[33] Andreas W. Götz, Mark J. Williamson, Dong Xu, Duncan Poole, Scott Le Grand, and Ross C Walker. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *Journal of Chemical Theory and Computation*, 8(5):1542–1555, may 2012.

[34] Robert D. Groot and Patrick B. Warren. Dissipative Particle Dynamics: Bridging the Gap between Atomistic and Mesoscopic Simulation. *The Journal of Chemical Physics*, 107(11):4423, 1997.

[35] Aric a. Hagberg, Daniel a. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 836:11—-15, 2008.

[36] Jie Han, Richard H. Gee, and Richard H. Boyd. Glass Transition Temperatures of Polymers from Molecular Dynamics Simulations. *Macromolecules*, 27(26):7781–7784, 1994.

[37] Vagelis A. Harmandaris and Kurt Kremer. Dynamics of polystyrene melts through hierarchical multiscale simulations. *Macromolecules*, 42(3):791–802, 2009.

[38] Eric S. Harper, Matthew Spellings, Joshua Anderson, and Sharon C. Glotzer. harperic/freud: Zenodo DOI release. nov 2016.

[39] P. J Hoogerbrugge and J. M. V. A Koelman. Simulating Microscopic Hydrodynamic Phenomena with Dissipative Particle Dynamics. *Europhys. Lett*, 19(1):155–160, jun 1992.

[40] J D Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[41] Cyrille Ibergay, Patrice Malfreyt, and Dominic J Tildesley. Electrostatic Interactions in Dissipative Particle Dynamics : Toward a Mesoscale Modeling of the Polyelectrolyte Brushes. pages 3245–3259, 2009.

[42] Cyrille Ibergay, Patrice Malfreyt, and Dominic J Tildesley. Interaction between two polyelectrolyte brushes: a mesoscale modelling of the compression. *Soft Matter*, 7(10):4900–4907, 2011.

[43] Takao Iijima, Satoru Miura, Wakichi Fukuda, and Masao Tomoi. Effect of cross-link density on modification of epoxy resins by N-phenylmaleimide-styrene copolymers. *European Polymer Journal*, 29(8):1103–1113, 1993.

[44] W Jenninger, J. E.K. Schawe, and I Alig. Calorimetric studies of isothermal curing of phase separating epoxy networks. *Polymer*, 41(4):1577–1588, 2000.

[45] A. M. Jokisaari, P. W. Voorhees, J. E. Guyer, J. Warren, and O. G. Heinonen. Benchmark problems for numerical implementations of phase field models. *Computational Materials Science*, 126:139–151, 2017.

[46] M. L. Jones and E. Jankowski. Computationally Connecting Organic Photovoltaic Performance to Atomistic Arrangements and Bulk Morphology. *Molecular Simulation*, 43(10-11):756–773, 2017.

[47] Shi Ru Jong and Tzyy Lung Yu. Physical aging of epoxy resin blended with a medium molecular weight poly(ether sulfone), 1999.

[48] P Jyotishkumar, Ceren Özdilek, Paula Moldenaers, Christophe Sinturel, Andreas Janke, Jürgen Pionteck, and Sabu Thomas. Dynamics of phase separation in poly(acrylonitrile-butadiene-styrene)-modified epoxy/DDS system: kinetics and viscoelastic effects. *The journal of physical chemistry. B*, 114(42):13271–81, 2010.

[49] Gokhan Kacar, Elias A. J. F. Peters, and Gijsbertus de With. Mesoscopic simulations for the molecular and network structure of a thermoset polymer. *Soft Matter*, 9(24):5785, 2013.

[50] Gokhan Kacar, Elias A J F Peters, and Gijsbertus De With. Multi-scale simulations for predicting material properties of a cross-linked polymer. *Computational Materials Science*, 102:68–77, 2015.

[51] Ketan S. Khare and Frederick R. Phelan. Quantitative Comparison of Atomistic Simulations with Experiment for a Cross-Linked Epoxy: A Specific Volume-Cooling Rate Analysis. *Macromolecules*, 51(2):564–575, 2018.

[52] Yu Seung Kim and Sung Chul Kim. Properties of polyetherimide/dicyanate semi-interpenetrating polymer network having the morphology spectrum. *Macromolecules*, 32(7):2334–2341, 1999.

[53] Patrick Kissel, Rolf Erni, W. Bernd Schweizer, Marta D. Rossell, Benjamin T. King, Thomas Bauer, Stephan Götzinger, A. Dieter Schlüter, and Junji Sakamoto. A two-dimensional polymer prepared by organic synthesis. *Nature Chemistry*, 4(4):287–291, 2012.

[54] Christoph Klein, János Sallai, Trevor J Jones, Christopher R Iacovella, Clare McCabe, and Peter T Cummings. A Hierarchical, Component Based Approach to Screening Properties of Soft Matter. *Foundations of Molecular Modeling and Simulation*, 2016.

[55] Walter Kob and Hans C. Andersen. Testing mode-coupling theory for a supercooled binary Lennard-Jones mixture. II. Intermediate scattering function and dynamic susceptibility. *Physical Review E*, 52(4):4134–4153, 1995.

[56] Pavel V Komarov, Chiu Yu-Tsung, Chen Shih-Ming, Pavel G Khalatur, and Peter Reineker. Highly Cross-Linked Epoxy Resins: An Atomistic Molecular Dynamics Simulation Combined with a Mapping/Reverse Mapping Procedure. *Macromolecules*, 40(22):8104–8113, 2007.

[57] Kurt Kremer, Gary S Grest, Kurt Kremer, Institutfor Festkorperforschung, Forschungszentrum Julich, West Germanjf, Institutfor Physik, Universitat Mainz, D Mainz, West Germany, and Gary S Grest. Dynamics of entangled linear polymer melts : A moleculardynamics simulation Dynamics of entangled linear polymer melts : A molecular-dynamics simulation. 5057(May 2013), 1990.

[58] Michael Langeloth, Taisuke Sugii, Michael C Böhm, and Florian Müller-Plathe. Formation of the interphase of a cured epoxy resin near a metal surface: Coarse-

grained reactive molecular dynamics simulations. *Soft Materials*, 12(ja):71–79, 2014.

[59] Michael Langeloth, Taisuke Sugii, Michael C. Böhm, and Florian Müller-Plathe. The glass transition in cured epoxy thermosets: A comparative molecular dynamics study in coarse-grained and atomistic resolution. *The Journal of Chemical Physics*, 143(24):243158, 2015.

[60] Mirjam E Leunissen and Daan Frenkel. Numerical study of DNA-functionalized microparticles and nanoparticles: Explicit pair potentials and their implications for phase behavior. *The Journal of Chemical Physics*, 134(8):84702, 2011.

[61] G. Levita, S. Petris, a. Marchetti, and a. Lazzeri. Crosslink density and fracture toughness of epoxy resins. *Journal of Materials Science*, 26(9):2348–2352, 1991.

[62] Chunyu Li, Grigori A. Medvedev, Eun Woong Lee, Jaewoo Kim, James M. Caruthers, and Alejandro Strachan. Molecular dynamics simulations and experimental studies of the thermomechanical response of an epoxy thermoset polymer. *Polymer (United Kingdom)*, 53(19):4222–4230, 2012.

[63] Chunyu Li and Alejandro Strachan. Cohesive energy density and solubility parameter evolution during the curing of thermoset. *Polymer (United Kingdom)*, 135:162–170, 2018.

[64] Min Li, Yi Zhuo Gu, Hong Liu, Yan Xia Li, Shao Kai Wang, Qing Wu, and Zuo Guang Zhang. Investigation the interphase formation process of carbon fiber/epoxy composites using a multiscale simulation method. *Composites Science and Technology*, 86:117–121, 2013.

[65] Hong Liu, Min Li, Zhong-Yuan Lu, Zuo-Guang Zhang, Chia-Chung Sun, and Tian Cui. Multiscale Simulation Study on the Curing Reaction and the Network Structure in a Typical Epoxy System. *Macromolecules*, 44(21):8650–8660, nov 2011.

[66] Hong Liu, You Liang Zhu, Zhong Yuan Lu, and Florian Müller-Plathe. A kinetic chain growth algorithm in coarse-grained simulations. *Journal of Computational Chemistry*, pages 2634–2646, 2016.

[67] Xiang-Yang Liu, James B Adams, Furio Ercolessi, James B Adams -, I J Robertson, M C Payne, V Heine -, Ying Yuan, al, I-Shou Huang, Ming-Kang Tsai, F Ercolessi, and J B Ad Am. Interatomic Potentials from First-Principles Calculations: The Force-Matching Method. *Europhys. Lett*, 26(8):583–588, 1994.

[68] Nusrat Lubna, Ganesh Kamath, Jeffrey J Potoff, Neeraj Rai, and J Ilja Siepmann. Transferable Potentials for Phase Equilibria. 8. United-atom Description for Thiols, Sulfides, Disulfides, and Thiophene. *The Journal of Physical Chemistry B*, 109(50):24100–24107, dec 2005.

[69] Glenn J. Martyna, Douglas J. Tobias, and Michael L. Klein. Constant pressure molecular dynamics algorithms. *The Journal of Chemical Physics*, 101(5):4177–4189, 1994.

[70] H. Medhioub, C. Zerrouki, N. Fourati, H. Smaoui, H. Guermazi, and J. J. Bonnet. Towards a structural characterization of an epoxy based polymer using small-angle x-ray scattering. *Journal of Applied Physics*, 101(4), 2007.

[71] K. Mimura, H. Ito, and H. Fujioka. Improvement of thermal and mechanical properties by control of morphologies in PES-modified epoxy resins. *Polymer*, 41(12):4451–4459, 2000.

[72] K Mimura, H Ito, and H Fujioka. Toughening of epoxy resin modified with in situ polymerized thermoplastic polymers. *Polymer*, 42(22):9223–9233, 2001.

[73] B. G. Min, Z. H. Stachurski, and J. H. Hodgkin. Cure kinetics of elementary reactions of a DGEBA/DDS epoxy resin: 1. Glass transition temperature versus conversion. *Polymer*, 34(23):4908–4912, 1993.

[74] Zhang Ming, An Xuefeng, Tang Bangming, and Yi Xiaosu. TTT Diagram Used to Control Phase Structure of 2/4 Functional Epoxy Blends for Advanced Composites. *Chinese Journal of Aeronautics*, 22(4):449–452, 2009.

[75] Timothy C. Moore, Christopher R. Iacovella, and Clare McCabe. Derivation of coarse-grained potentials via multistate iterative Boltzmann inversion. *The Journal of Chemical Physics*, 140(22):224104, jun 2014.

[76] Debashish Mukherji and Cameron F. Abrams. Mechanical behavior of highly cross-linked polymer networks and its links to microscopic structure. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 79(6):1–10, 2009.

[77] Gregory M. Odegard, Benjamin D. Jensen, S. Gowtham, Jianyang Wu, Jianying He, and Zhiliang Zhang. Predicting mechanical response of crosslinked epoxy using ReaxFF. *Chemical Physics Letters*, 591:175–178, 2014.

[78] Tomonaga Okabe, Yutaka Oya, Koichi Tanabe, Gota Kikugawa, and Kenichi Yoshioka. Molecular Dynamics Simulation of Crosslinked Epoxy Resins: Curing and Mechanical Properties. *European Polymer Journal*, 2016.

[79] J. P. Pascault and R. J.J. Williams. Glass transition temperature versus conversion relationships for thermosetting polymers. *Journal of Polymer Science Part B: Polymer Physics*, 28(1):85–95, 1990.

[80] Paul N. Patrone and Thomas W. Rosch. Beyond histograms: Efficiently estimating radial distribution functions via spectral Monte Carlo. *The Journal of Chemical Physics*, 146(9):094107, mar 2017.

[81] Gorakh Pawar, Paul Meakin, and Hai Huang. Reactive Molecular Dynamics Simulation of Kerogen Thermal Maturation and Crosslinking Pathways. *Energy & Fuels*, page acs.energyfuels.7b01555, 2017.

[82] Carolyn L Phillips, Joshua a. Anderson, and Sharon C Glotzer. Pseudo-random number generation for Brownian Dynamics and Dissipative Particle Dynamics simulations on GPU devices. *Journal of Computational Physics*, 230(19):7191–7201, aug 2011.

[83] Monoj Pramanik, Eric W. Fowler, and James W. Rawlins. Another look at epoxy thermosets correlating structure with mechanical properties. *Polymer Engineering and Science*, 54(9):1990–2004, 2014.

[84] Dirk Reith, Mathias Pütz, and Florian Müller-Plathe. Deriving effective mesoscale potentials from atomistic simulations. *Journal of computational chemistry*, 24(13):1624–1636, oct 2003.

[85] Naomi Rom, Sergey V. Zybin, Adri C.T. Van Duin, William A. Goddard, Yehuda Zeiri, Gil Katz, and Ronnie Kosloff. Density-dependent liquid nitromethane decomposition: Molecular dynamics simulations based on ReaxFF. *Journal of Physical Chemistry A*, 115(36):10181–10202, 2011.

[86] Michael Rubinstein and Ralph H Colby. *Polymer physics*, volume 23. Oxford university press New York, 2003.

[87] Raghava R S. Development and characterization of thermosetting-thermoplastic polymer blends for applications in damage-tolerant composites. *Journal of Polymer Science Part B: Polymer Physics*, 26(1):65–81.

[88] J. E K Schawe. A description of chemical and diffusion control in isothermal kinetics of cure kinetics. *Thermochimica Acta*, 388(1-2):299–312, 2002.

[89] Jürgen E K Schawe. The Interplay between Molecular Dynamics and Reaction Kinetics during Curing Reactions. 100002:1–6, 2017.

[90] S. S. Schoenholz, E. D. Cubuk, D. M. Sussman, E. Kaxiras, and A. J. Liu. A structural approach to relaxation in glassy liquids. *Nature Physics*, 12(5):1–12, 2016.

[91] M. Scott Shell, Pablo G. Debenedetti, and Athanassios Z. Panagiotopoulos. A conformal solution theory for the energy landscape and glass transition of mixtures. *Fluid Phase Equilibria*, 241(1-2):147–154, 2006.

[92] M. S. Shell. Advanced molecular dynamics techniques. *Advanced molecular dynamics techniques*, pages 1–11, 2009.

[93] Yasuyuki Shudo, Atsushi Izumi, Katsumi Hagita, Toshio Nakao, and Mitsuhiro Shibayama. Large-scale molecular dynamics simulation of crosslinked phenolic resins using pseudo-reaction model. *Polymer (United Kingdom)*, 103:261–276, 2016.

[94] Sindee L Simon, Gregory B Mckenna, and Olivier Sindt. Modeling the Evolution of the Dynamic Mechanical Properties of a Commercial Epoxy During Cure after Gelation. *Applied Polymer Science*, 76:495–508, 2000.

[95] C. Soutis. Carbon fiber reinforced plastics in aircraft construction. *Materials Science and Engineering A*, 412(1-2):171–176, 2005.

[96] H. Staudinger. Uber Polymerisation. *Berichte Der Deutschen Chemischen Gesellschafl*, (53):1073–1085, 1920.

[97] Frank H. Stillinger and Pablo G. Debenedetti. Glass Transition Thermodynamics and Kinetics. *Annual Review of Condensed Matter Physics*, 4(1):263–285, 2013.

[98] Carsten Svaneborg, Hossein Ali Karimi-Varzaneh, Nils Hojdis, Frank Fleck, and Ralf Everaers. Kremer-Grest models for universal properties of specific common polymer species. pages 1–38, 2016.

[99] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. A Computer Simulation Method for the Calculation of Equilibrium Constants for the Formation of Physical Clusters of Molecules: Application to Small Water Clusters. *Journal of Chemical Physics*, 76(1):637–649, 1982.

[100] Kazuaki Z. Takahashi, Ryuto Nishimura, Nobuyoshi Yamato, Kenji Yasuoka, and Yuichi Masubuchi. Onset of static and dynamic universality among molecular models of polymers. *Scientific Reports*, 7(1):1–7, 2017.

[101] Qi Tao, Gerald Pinter, Thomas Antretter, Thomas Krivec, and Peter Fuchs. Model free kinetics coupled with finite element method for curing simulation of thermosetting epoxy resins. *Journal of Applied Polymer Science*, 135(27):1–8, 2018.

[102] Stephen Thomas, Monet Alberts, Michael M Henry, Carla E Estridge, and Eric Jankowski. Routine million-particle simulations of epoxy curing with dissipative particle dynamics. *Journal of Theoretical and Computational Chemistry*, 0(0):S0219633618400059, 2018.

[103] Stephen Thomas, Mike Henry, and Monetalberts. amburan/epoxpy: Introducing epoxpy. oct 2017.

[104] Andrew J. Timmis, Alma Hodzic, Lenny Koh, Michael Bonner, Constantinos Soutis, Andreas W. Schäfer, and Lynnette Dray. Environmental impact assessment of aviation emission reduction through the implementation of composite materials. *International Journal of Life Cycle Assessment*, 20(2):233–243, 2015.

[105] Karl P. Travis, Mark Bankhead, Kevin Good, and Scott L. Owens. New parametrization method for dissipative particle dynamics. *Journal of Chemical Physics*, 127(1), 2007.

[106] Christian Robert Trott. *LammpsCuda-a new GPU accelerated Molecular Dynamics Simulations Package and its Application to Ion-Conducting Glasses*. PhD thesis, 2011.

[107] W. Tschöp, K. Kremer, J. Batoulis, T. Bürger, and O. Hahn. Simulation of polymer melts. I. Coarse-graining procedure for polycarbonates. *Acta Polymerica*, 49(2-3):61–74, 1998.

[108] Adri C T van Duin, Siddharth Dasgupta, Francois Lorant, and William A. Goddard. ReaxFF: A Reactive Force Field for Hydrocarbons. *The Journal of Physical Chemistry A*, 105(41):9396–9409, oct 2001.

[109] Peter Virnau, Marcus Müller, L. G. MacDowell, and K. Binder. Phase behavior of n-alkanes in supercritical solution: A Monte Carlo study. *Journal of Chemical Physics*, 121(5):2169–2179, 2004.

[110] Xiaorong Wang and John K. Gillham. Competitive primary amine/epoxy and secondary amine/epoxy reactions: Effect on the isothermal time-to-vitrify. *Journal of Applied Polymer Science*, 43(12):2267–2277, dec 1991.

[111] Ronald P. White and Jane E.G. Lipson. Polymer Free Volume and Its Connection to the Glass Transition. *Macromolecules*, 49(11):3987–4007, 2016.

[112] Malcolm L. Williams, Robert F. Landel, and John D. Ferry. The Temperature Dependence of Relaxation Mechanisms in Amorphous Polymers and Other Glass-forming Liquids. *Journal of the American Chemical Society*, 77(14):3701–3707, 1955.

[113] Wen Sheng Xu, Jack F. Douglas, and Karl F. Freed. Influence of Cohesive Energy on Relaxation in a Model Glass-Forming Polymer Melt. *Macromolecules*, 49(21):8355–8370, 2016.

[114] Wen Sheng Xu, Jack F. Douglas, and Karl F. Freed. Influence of Cohesive Energy on the Thermodynamic Properties of a Model Glass-Forming Polymer Melt. *Macromolecules*, 49(21):8341–8354, 2016.

[115] Yuan Xu and Suong Van Hoa. Mechanical properties of carbon fiber reinforced epoxy/clay nanocomposites. *Composites Science and Technology*, 68(3-4):854–861, 2008.

[116] Hiromasa Yagyu, Yoshikazu Hirai, Akio Uesugi, Yoshihide Makino, Koji Sugano, Toshiyuki Tsuchiya, and Osamu Tabata. Simulation of mechanical properties of epoxy-based chemically amplified resist by coarse-grained molecular dynamics. *Polymer*, 53(21):4834–4842, 2012.

[117] Keizo Yamanaka and Takashi Inoue. Structure development in epoxy resin modified with poly(ether sulphone). *Polymer*, 30(4):662–667, 1989.

[118] Shaorui Yang, Zhiwei Cui, and Jianmin Qu. A coarse-grained model for epoxy molding compound. *Journal of Physical Chemistry B*, 118(6):1660–1669, 2014.

[119] Shaorui Yang and Jianmin Qu. Coarse-grained molecular dynamics simulations of the tensile behavior of a thermosetting polymer. *Physical Review E*, 90(1):012601, 2014.

[120] Xin Yong, Olga Kuksenok, and Anna C. Balazs. Modeling free radical polymerization using dissipative particle dynamics. *Polymer*, 72:217–225, 2015.

[121] Y Yu, Z Zhang, W Gan, M Wang, and S Li. Effect of polyethersulfone on the mechanical and rheological properties of polyetherimide-modified epoxy

systems. *Industrial and Engineering Chemistry Research*, 42(14):3250–3256, 2003.

[122] Yingfeng Yu, Minghai Wang, Wenjun Gan, Qingsheng Tao, and Shanjun Li. Polymerization-Induced Viscoelastic Phase Separation in Polyethersulfone-Modified Epoxy Systems. *J. Phys. Chem. B.*, 108:6208–66215, 2004.

[123] Jin Zhang, Qipeng Guo, and Bronwyn L. Fox. Study on thermoplastic-modified multifunctional epoxies: Influence of heating rate on cure behaviour and phase separation. *Composites Science and Technology*, 69(7-8):1172–1179, 2009.

[124] Wenlin Zhang, Enrique D. Gomez, and Scott T. Milner. Predicting Flory-Huggins χ from Simulations. *Physical Review Letters*, 119(1):1–5, 2017.

APPENDIX A

PERFORMANCE OPTIMIZATION

A.1 Performance Profiling

Profiling runs of our bonding model use τ_B = 10, and otherwise are the same as the fiducial parameters (Table 3.2 in article). The bonding routine was first implemented in python where nearest-neighbor searches were performed using `numpy` array broadcasting routines. An "icicle" style plot of the cpu time for just the bonding routine called *find_pair* is shown in Figure A.1. The methods *find_neighbours_and_bond* and *pbc_diff* are the most expensive ones, and overall simulation performance varied between 45 and 87 time steps per second (TPS).

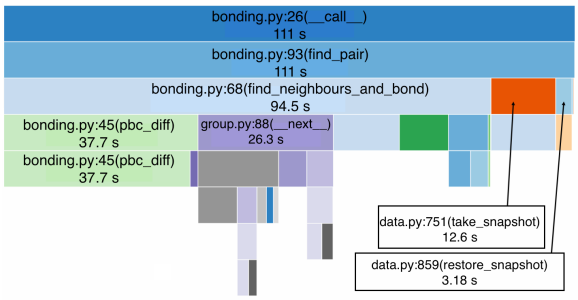


Figure A.1: Distribution of computational cost for our pure python code implementation of Algorithm 1.

These low TPS numbers (in comparison to thousands of TPS when no bonding is occurring) represent a significant degradation of performance. We next implemented

the bonding routine using `freud`, a python package built for MD analysis that implements neighborlist calculations (and many others) in C++[38]. The rationale here is that faster code for the above neighbor-finding bottleneck would make the largest difference to overall performance. Using `freud` reduced the overall time taken by the bonding routine by about 80% (Figure A.2).

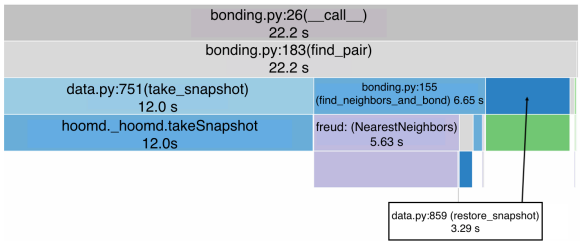


Figure A.2: Distribution of computational costs for our `freud` implementation of Algorithm 1.

In the freud-based code the majority of time spent by the bonding routine is in the *take_snapshot* function in HOOMD-Blue[8,32]. The snapshot is taken every time the callback function is invoked from HOOMD-Blue. During the bonding operation, the bonded group information is modified and the modified snapshot is then restored before the simulation is allowed to continue. Further performance improvements can be achieved by optimising or reducing calls of the *take_snapshot* function.

In principle, there is no reason a running simulation needs to take a snapshot of itself to be used in a bonding routine: all of the information needed for bonding exists within the simulation instance itself. This observation in combination with the low TPS of pure python and `freud` implementations of our bonding algorithm inform our decision to develop a C++ plugin. This plugin allows users to specify the frequency with which bonds are dynamically created between particles with a probability given by Equation 3.10 in the main article. We find that our C++ implementation, which

can use HOOMD-Blue's neighbor list data directly, is a factor of 10 to 70 faster than the python and `freud` implementations, and only adds 7% to 26% overhead compared to no bonding (Table A1). TPS generally decreases with temperature because the neighborlist is updated more frequently at higher temperatures.

Table A1: Comparison of TPS between the python bonding routine, the `freud` bonding routine and the dybond plugin, with no bonding listed as reference (N = 50,000).

Bonding Method	TPS		
	$k_BT = 0.05$	$k_BT = 1$	$k_BT = 5$
No Bonding	3428	2118	1255
Dybond Plugin	3186	1854	946
Freud Bonding	280	156	150
Python Bonding	45	47	87

Figure A.3 shows the split of the time taken by the different parts of the simulation code and it is seen that the slowest part is the neighborlist.

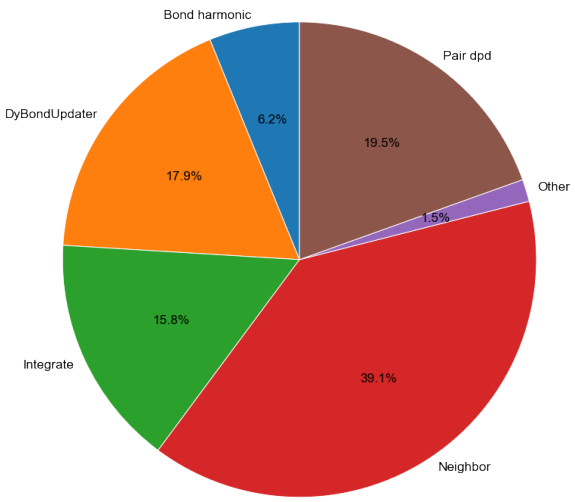


Figure A.3: Split of the time taken by different parts of a typical curing simulations.

A.2 Performance and System Size

We benchmark simulation performance as a function of system size $N \in \{5 \times 10^3, 5 \times 10^4, 1 \times 10^6, 2 \times 10^6\}$ at T = 500 K (Figure A.4). These simulations are performed at the fiducial parameters with the exception of E_A = $1k_BT_C$, τ_B=10, and n_B=2. Simulations with $N = 5 \times 10^4$ achieve 95% cure in about 45 wall-clock minutes and the $N = 2 \times 10^6$ simulations achieve 95% cure in about 7.5 wall-clock hours.

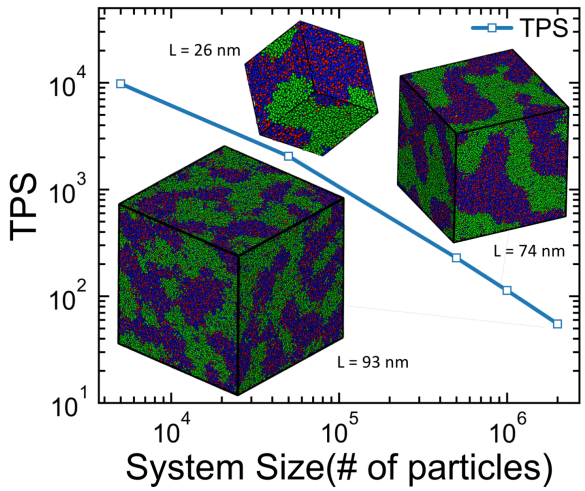


Figure A.4: Performance scaling of TPS vs. number of particles N from 5000 to 2 million where the cure percent has reached 95%. Morphologies shown are for 50,000, 1,000,000 and 2,000,000 particles.

APPENDIX B

CALIBRATION OF REACTION KINETICS

Table B2: Solubility parameters obtained from MD for DDS

T_C (K)	Density (g/cm^3)	Volume ($\mathring{A}^3$)	E_{coh} (kcal/mol)	δ $(cal/cm^3)^{1/2}$
273	1.32	156680.56	28.55	12.300142
305	1.30	157994.45	27.98	12.126000
340	1.29	159528.91	27.32	11.924365
375	1.28	161309.42	26.61	11.703269
405	1.26	163829.46	25.71	11.414836
440	1.24	166750.55	24.68	11.085456
475	1.20	171126.79	23.39	10.652971
505	1.18	174465.55	22.50	10.347872
540	1.15	178998.43	21.42	9.967810
575	1.12	183514.06	20.41	9.609515
600	1.10	187213.51	19.64	9.332904

Table B3: Solubility parameters obtained from MD for DGEBA

T_C (K)	Density (g/cm^3)	Volume ($\mathring{A}^3$)	E_{coh} (kcal/mol)	δ $(cal/cm^3)^{1/2}$)
273	1.14	248901.03	31.23	10.206745
305	1.12	252419.69	30.36	9.993184
340	1.10	257982.95	29.01	9.662577
375	1.07	264520.55	27.40	9.273853
405	1.04	270604.60	26.12	8.952279
440	1.02	277803.22	24.75	8.600696
475	0.99	286160.71	23.30	8.222191
505	0.96	294236.49	22.10	7.897007
540	0.93	303874.10	20.79	7.536939
575	0.90	314288.43	19.52	7.181089
600	0.88	321468.57	18.75	6.958987

Table B4: Solubility parameters obtained from MD for PES 10-mers

T_C (K)	Density (g/cm^3)	Volume ($\mathring{A}^3$)	E_{coh} (kcal/mol)	δ $(cal/cm^3)^{1/2}$)
273	1.30	148734.47	172.55	9.814457
305	1.29	149686.25	169.97	9.709789
340	1.29	149914.22	168.87	9.670957
375	1.28	151012.30	165.71	9.545151
405	1.27	152142.69	163.94	9.458702
440	1.25	154379.68	159.49	9.261606
475	1.24	155133.86	158.79	9.218768
505	1.23	156336.04	156.07	9.104263
540	1.23	157532.01	152.91	8.977350
575	1.22	158735.31	150.68	8.877806
600	1.20	161192.93	147.77	8.724384

APPENDIX C

GLASS TRANSITION MEASUREMENT

Table C5: Custom Data Ranges for PRM for the A-B Binary LJ/Harmonic Model

Cooling Method	α	Low Temperature Range (min,max)[T^*]	High Temperature Range (min,max)[T^*]
Quench	0.0	(0.1,0.25)	(0.28,0.32)
Quench	0.3	(0.1,0.25)	(0.29,0.33)
Quench	0.6	(0.1,0.25)	(0.28,0.34)
Quench	0.9	(0.1,0.25)	(0.30,0.35)
Anneal	0.0	(0.1,0.25)	(0.26,0.31)
Anneal	0.3	(0.1,0.25)	(0.29,0.35)
Anneal	0.6	(0.1,0.25)	(0.28,0.36)
Anneal	0.9	(0.1,0.30)	(0.30,0.37)

Table C6: Custom Data Ranges for PRM for the A-B-C Binary LJ/Harmonic Model

Cooling Method	α	Low Temperature Range (min,max)[T^*]	High Temperature Range (min,max)[T^*]
Quench	0.0	(0.1,0.8)	(0.7,1.2)
Quench	0.3	(0.1,0.8)	(0.85,1.4)
Quench	0.5	(0.1,0.8)	(1.0,1.8)
Quench	0.7	(0.1,0.8)	(1.15,2.5)

APPENDIX D

SENSITIVITY TO THE 'LINEAR RAMP' TIME-TEMPERATURE CURING PROFILES

To understand the effect of a linearly ramped temperature profile as shown in Figure D.1 the final microstructure obtained from this curing schedule is compared with an isothermal cure profile with the same final cure percent and cure temperature.

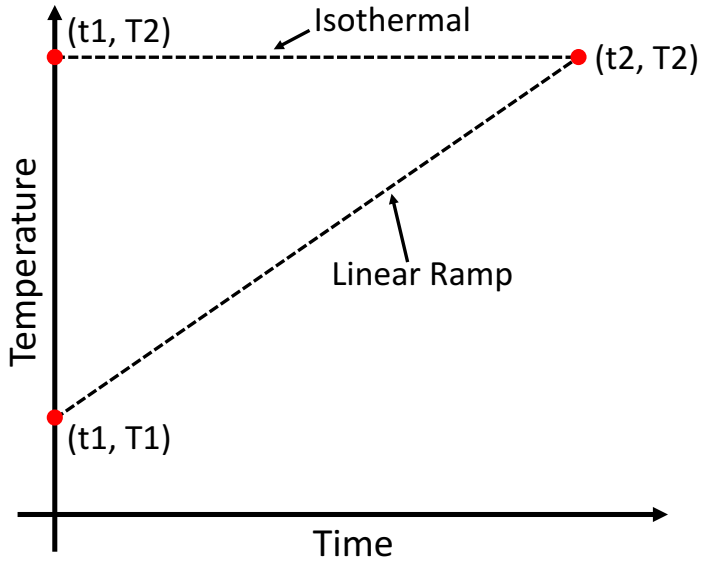


Figure D.1: Temperature and cure fraction of an isothermal and linearly ramp profile.

The DPD/Harmonic model with interaction parameters described in Table 4.2 is used with fiducial parameters shown in Table 3.2 for the curing simulations with the exception of an activation energy of E_a = 2.0 $k_B T_C$ and system size (N). The system

size was chosen to be large enough to avoid finite size effect ($N = 2 \times 10^6$) such that q_m can be detected reliability as seen in Figure 3.4. The PES-PES structure factor was obtained as an average of 5 replicate simulations and the standard error calculated to quantify the uncertainties in the final morphology. As seen in Figure D.2, both profiles used a final setpoint temperature $T2 = 850\ K$ and a final cure fraction $\alpha = 0.8$. The linear ramped profile starts at $T0 = 300\ K$ and the isothermal profile starts at $T0 = 850\ K$.

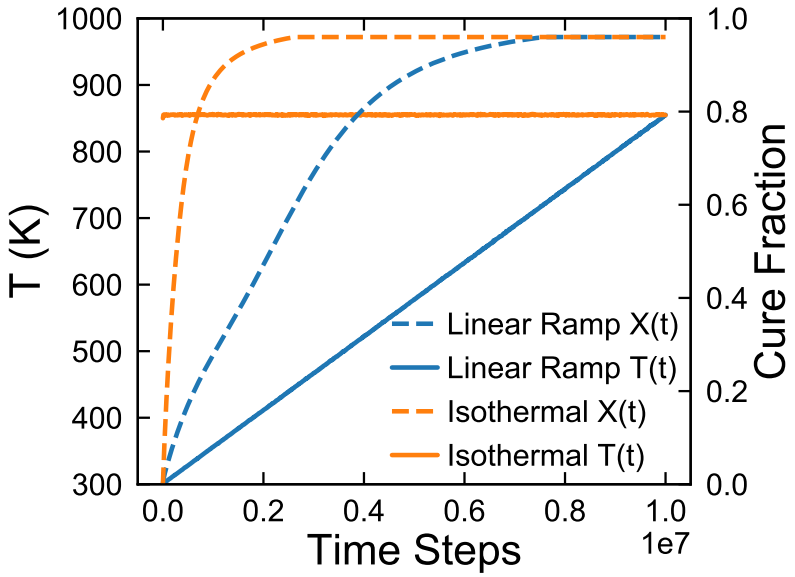


Figure D.2: Temperature and cure fraction of an isothermal and linearly ramp profile.

The PES-PES structure factor in Figure D.3 shows that the linear ramped cure profile resulted in a macrophase separated morphology which is also clearly seen in the final snapshot of the simulation shown in Figure D.3. The isothermal cure profile resulted in a microphase separated morphology that is similar to a co-continuous morphology. The microphase separation in the isothermal cure is a result of rapid crosslinking during the initial stages of curing causing the mixed initial structure to be locked in. The linear ramped profile avoided this locking of the initial mixed

structure due to the low rate of crosslinking (as seen in Figure D.2) and allowed the PES tougheners to diffuse and phase separate.

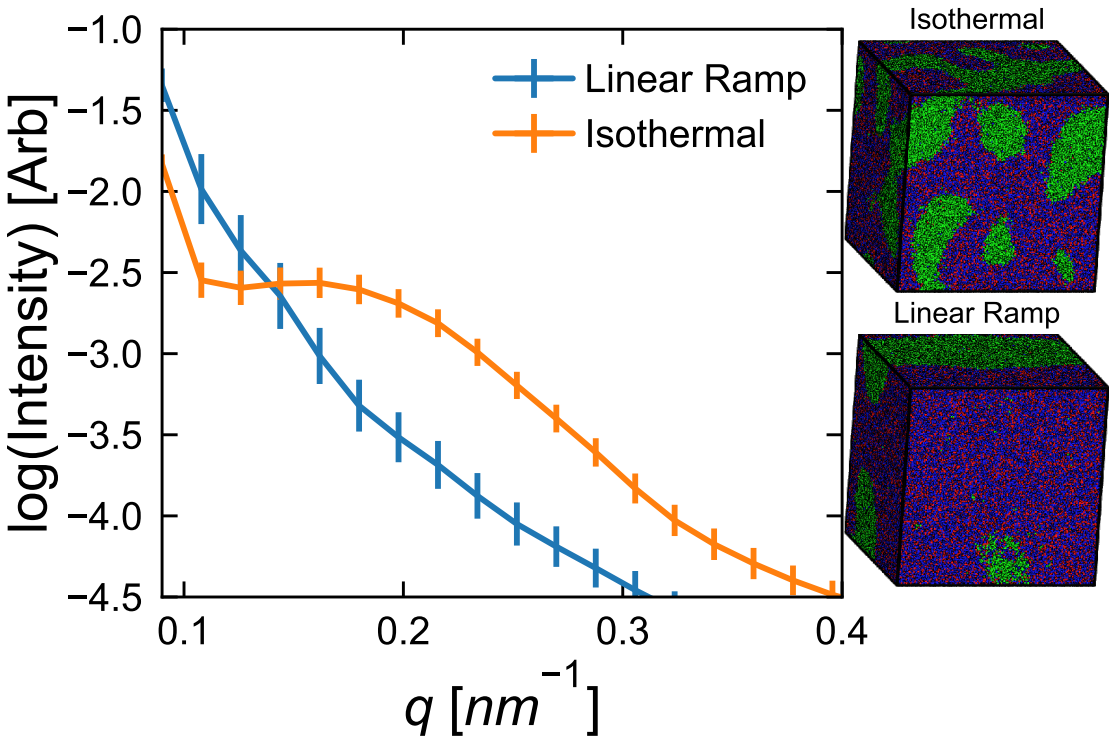


Figure D.3: PES-PES structure factor from the isothermal cure profile shows microphase separation whereas the linearly ramped cure profile has resulted in a macrophase separated morphology.

APPENDIX E

LJ UNIT CONVERSIONS

In [1]:
```
import pandas as pd
import math
import numpy as np

#constants
Na = 6.022e23
cal_cm_cubed_to_MPa = 4.187
kB = 1.381e-23 #JK^-1
J_to_cal = 0.239
cal_cm_cube_to_SI = 4.187e6 #to J/m^3
MPa_to_SI = 1e6 #to J/m^3
```

1 Obtaining the like interaction energies (ϵ_{AA}) from e_{coh} and cross interaction (ϵ_{AB}) from Lorrentz-Berthelot mixing rule

In [2]:
```
#data
molecule_dict = {}
molecule_dict['DEBGF(273 K)'] = {'kT':273,'E_coh':31.23,'V':248901.03,'rho':1.14,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(305 K)'] = {'kT':305,'E_coh':30.36,'V':252419.69,'rho':1.12,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(340 K)'] = {'kT':340,'E_coh':29.01,'V':257982.95,'rho':1.10,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(375 K)'] = {'kT':375,'E_coh':27.40,'V':264520.55,'rho':1.07,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(405 K)'] = {'kT':405,'E_coh':26.12,'V':270604.60,'rho':1.04,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(440 K)'] = {'kT':440,'E_coh':24.75,'V':277803.22,'rho':1.02,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(475 K)'] = {'kT':475,'E_coh':23.30,'V':286160.71,'rho':0.99,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(505 K)'] = {'kT':505,'E_coh':22.10,'V':294236.49,'rho':0.96,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(540 K)'] = {'kT':540,'E_coh':20.79,'V':303874.10,'rho':0.93,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(575 K)'] = {'kT':575,'E_coh':19.52,'V':314288.43,'rho':0.90,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}
molecule_dict['DEBGF(600 K)'] = {'kT':600,'E_coh':18.75,'V':321468.57,'rho':0.88,'nmol':
500,'mol_weight':340.423,'name':'DGEBA'}

molecule_dict['DDS(273 K)'] = {'kT':273,'E_coh':28.55,'V':156680.56,'rho':1.32,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(305 K)'] = {'kT':305,'E_coh':27.98,'V':157994.45,'rho':1.30,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(340 K)'] = {'kT':340,'E_coh':27.32,'V':159528.91,'rho':1.29,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(375 K)'] = {'kT':375,'E_coh':26.61,'V':161309.42,'rho':1.28,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(405 K)'] = {'kT':405,'E_coh':25.71,'V':163829.46,'rho':1.26,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(440 K)'] = {'kT':440,'E_coh':24.68,'V':166750.55,'rho':1.24,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(475 K)'] = {'kT':475,'E_coh':23.39,'V':171126.79,'rho':1.20,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(505 K)'] = {'kT':505,'E_coh':22.50,'V':174465.55,'rho':1.18,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(540 K)'] = {'kT':540,'E_coh':21.42,'V':178998.43,'rho':1.15,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(575 K)'] = {'kT':575,'E_coh':20.41,'V':183514.06,'rho':1.12,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}
molecule_dict['DDS(600 K)'] = {'kT':600,'E_coh':19.64,'V':187213.51,'rho':1.10,'nmol':50
0,'mol_weight':248.306,'name':'DDS'}

molecule_dict['PES10(273 K)'] = {'kT':273,'E_coh':172.55,'V':148734.47,'rho':1.30,'nmol'
:50,'mol_weight':2324.614,'name':'PES10'}
```

```
In [41]: import math
         Ea_ex = 107.3e3#J/mol
         Na=6.023e23
         R = 8.314#J/mol.K
         E_md = 5.043e-21#J
         T_sim=3.0
         T_ex = 313#K (40C)

         t_ex = 60*60#seconds
         T_g_exp = 480 #K
         T_g_sim = 1.39 #T*
         TimeConversion=1.0444e-11#s
         EnergyConversion = 4.768920863309353e-21  # 5.043e-21#J
         TemperatureConversion = T_g_exp/T_g_sim # 365.297 #K 480/1.314

         T_md = 3*TemperatureConversion#K

         deltat=1e-2
         simulated_timesteps =2.05e6# (time to cure to 75% at 3 kT  for LJ where Ea of 3 was
         used) see plot below
         t_md = TimeConversion*deltat*simulated_timesteps#seconds
         print('simulated time:',t_md,'s')
         print('experimental time:',t_ex,'s')
         print('simulated T',T_md,'K')

         J_to_kcal = 2.39e-4
         kcal_to_J = 4184
         R = 8.314
         print('experimental E_a:',Ea_ex/Na,'J')
```

```
simulated time: 2.1410200000000002e-07 s
experimental time: 3600 s
simulated T 1035.9712230215828 K
experimental E_a: 1.7815042337705463e-19 J
```

```
In [3]: kB=1.38e-23#J/K
        #T=2*TemperatureConversion
        Ea=Ea_ex/Na
        print('experimental probability at {} K:{}'.format(T_ex,math.exp(-Ea/(kB*T_ex))))
```

```
experimental probability at 313 K:1.2241899692027386e-18
```

$$k_{exp} = A_{exp} e^{-\frac{E_a^{exp}}{k_B T_{exp}}}$$

$$k_{MD} = A_{MD} e^{-\frac{E_a^{MD}}{k_B T_{MD}}}$$

$$\frac{1}{t_{exp}} = A_{exp} e^{-\frac{E_a^{exp}}{k_B T_{exp}}} \text{—(1)}$$

$$\frac{1}{t_{MD}} = A_{MD} e^{-\frac{E_a^{MD}}{k_B T_{MD}}} \text{—(2)}$$

$$\frac{t_{MD}}{t_{exp}} = \frac{A_{exp} e^{-\frac{E_a^{exp}}{k_B T_{exp}}}}{A_{MD} e^{-\frac{E_a^{MD}}{k_B T_{MD}}}}$$

$$\frac{t_{MD}}{t_{exp}} = \frac{A_{exp}}{A_{MD}} e^{\frac{E_a^{MD}}{k_B T_{MD}} - \frac{E_a^{exp}}{k_B T_{exp}}}$$

$$ln\left(\frac{t_{MD}}{t_{exp}}\right) = ln\left(\frac{A_{exp}}{A_{MD}}\right) + \left(\frac{E_a^{MD}}{k_B T_{MD}} - \frac{E_a^{exp}}{k_B T_{exp}}\right)$$

We know that the fitted values of $A_{MD} \approx 0.026\tau^{-1}$ (for frequency factor of 1.0) (see simulated A calculation below)

$$A_{MD} \approx \frac{0.026}{\tau \frac{1.045e-11s}{\tau}}$$

APPENDIX F

POST-SIMULATION DATA ANALYSIS CODE

This section lists all the code used to analyze data collected from simulations. The analysis code also includes code used to generate the figures used in the dissertation. Since the analysis code was mostly used to generate figures for Chapters 3, 4 and 5, only those are shown in Appendix F.1, F.2 and F.3 and the common code are listed in Appendix F.4.

F.1 Code for Chapter 3

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_2_vary_Ea'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_4'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/ttt'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_path'

import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
project = signac.get_project(root=data_path)
markers={'iso':'s','lin_ramp':'P','step':'>'}

df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df['T'] = df['kT']*df['calibrationT']
df['gel_point']
#df[df.signac_id=='c7fdb95108dc87248b9c01adfb570150']['kT']
```

Out[1]:
```
01a3ab33afee81ae515c6614a382659b     180000.0
02b8ff9fb87f34069ef653835ec95659          NaN
0436e48eb688e1eef506bbdde9746868          NaN
04cdff02e7213cdb436ce8437ac5a620          NaN
04de1682430c81512a76e8dd8c74ed77     640000.0
06049b31a13cde94b7d5cb68f84eed5a     160000.0
08178e8713ab89438e0f775a497c63b1     180000.0
081cdc9337c0540a0f12fa64cfcbda56     200000.0
0935e56c0977196c5a62ca9fdcefa49e          NaN
0a783807a143ee23df287cd9d8eba2bd     720000.0
0ee693531ab96496d24d82d5e3b368a1          NaN
0f7fd024dc1fb578da29b67db35cdba9    1400000.0
0fcf0719b7130185c948bcea7858029f     200000.0
10496049a30e7b00d66742e4abe8733d    1400000.0
10613b754c05ce3781a6b0bb79bddb67    1320000.0
10a46274a0266f4830c1ec8a12a7b9a8          NaN
10a7886d4c389c6334fffcbf10f2578b     940000.0
11b0d0606c0523af209baa0269e21969     500000.0
1292ad177409d7155da5ec8d89a46587    1160000.0
1294d7fda4d5d22a1a91ce2cb7748fda     960000.0
1359b139cc62da34129512fa343104eb     180000.0
14406bcbbd5e2466a952a7c64be414a6     940000.0
15d77d92c25b1ec79b8a96607ae6a070     180000.0
15e02651ea127a672b8838aab601b161     180000.0
179251f800bfbfad51c27618010c8b6c    1060000.0
194293248e59d051e0fddb98a200b8ae          NaN
```

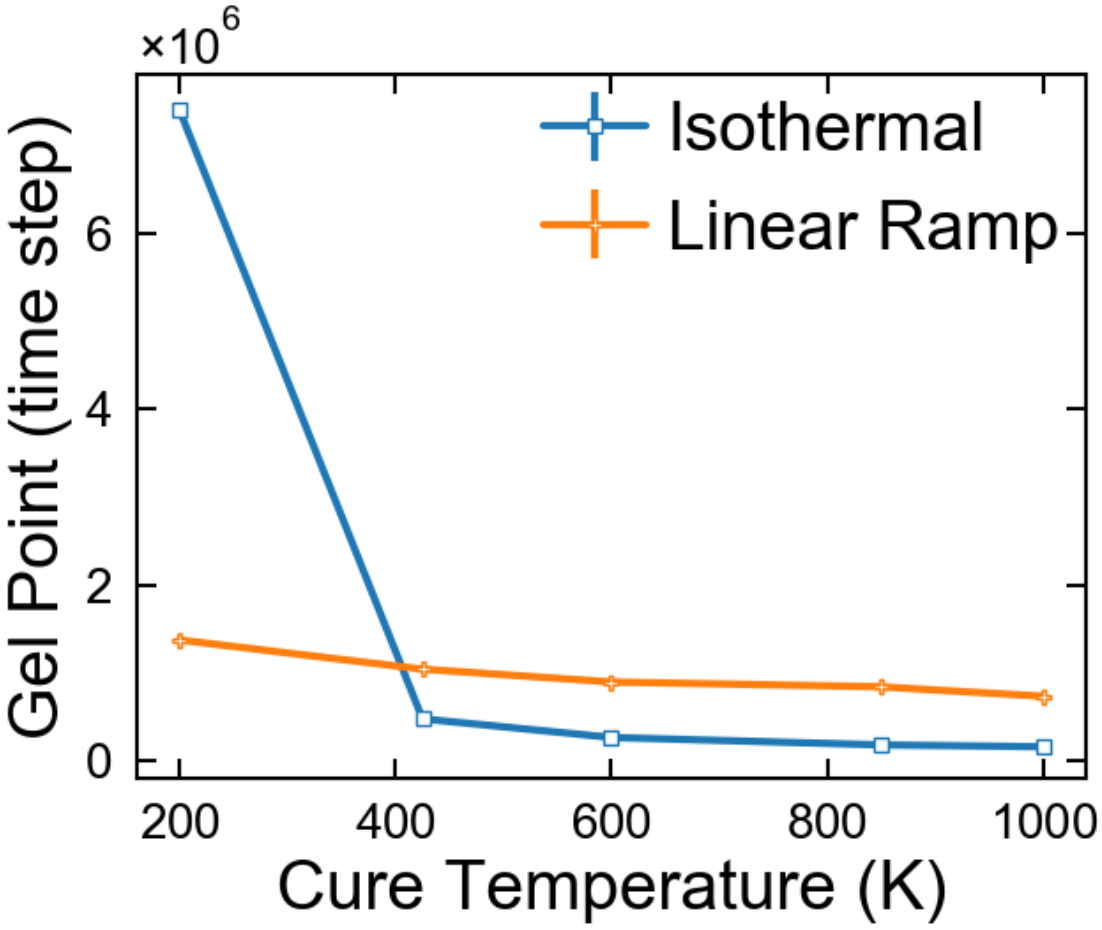
```
1a72bd88f5395d4c1f3895d32742841a          NaN
1b3ae1750a55b9f8b4f8332fbd444d3c     760000.0
1b657198ba16741b2ff9d4b04cf4c42e     960000.0
1c3ded4fc9ca63881b3323f842753fda          NaN
                                          ...
e777d968b4deb060aa88517797c37a34          NaN
e84f8b95b62921855430b25df7845a4f     460000.0
e98b92ebbb4a88e34cec0825146e754e     160000.0
e9ba2af0bca4572f6a52e0dc23d64434     920000.0
e9c0fc880e9f27b779319aeaa194ac37          NaN
e9ec6145e00b7085c3714e5c6376c9d6     180000.0
ea6b154bea8bfbaae27873026e1e681c          NaN
ed5e524919a818922d3282028cea51de          NaN
ee5599f90c87f6ea02609ac5d8d553d0    1180000.0
f2b56ca77e0625d63a5f5b695c75a4ae     920000.0
f2e4020fc3a8b09a0fd531a9b0e6ad15          NaN
f35b8c6b0d22304f1d7909f7a76f9e1c     560000.0
f3894b2b718a380478a4d2291f371e54     160000.0
f45a83e2671d8d4579662aa0b6c3a8c6          NaN
f55aa0a48d39328591df304e28c1b8e0          NaN
f698ffbf7edc172339e923a77732ca08     700000.0
f78d6582323aee95c7ac66419b457bb7          NaN
f92426b10aaa1eec0a7c27d83d076f35     480000.0
f9a7eee9a114ba26cb9f06d04c8f12af          NaN
f9d6fa4ba7df8c50b7a657a29c557288          NaN
fa8fc68be0adb846cc0389940aad17b2          NaN
fab1359a98daab94891423f34cff89d8     920000.0
fac7a6acb8c97a1dbea358f84786dbb2    1400000.0
fcb2485b26040bb36f3eead4b794f279          NaN
fcc60ffe9ac44c88450b57e9acb9417d          NaN
fd1939890ebbec5954159730d41ae2dd     160000.0
fe43c8ca9a0dac39964b60c89c3874ef          NaN
fe94d9d63016dcf82ffbc04ff7d878d3     280000.0
feb66f4458b0527aed402b54c3ec5c49     160000.0
ff993d7dad921bbb452481b796e19534    1060000.0
Name: gel_point, Length: 340, dtype: float64
```

```
In [2]: df_bonded = df[(df.bond==True)&
                       (df.activation_energy==2)&
                       (df.dcd_write==20000)&
                       (df['T']==850)&
                       (df.trial==0)&
                       (df.t_Final==1e7)&
                       #(df.log_write==1e3)&
                       (df.profile=='iso')&
                       (df.profile !='step')].sort_values('kT')
        df_bonded.gel_point
```

```
Out[2]: 97c56a93e39946f3400b4abc85d57e1c    180000.0
        e62f6074db41878eae6d283e578fdc7a    180000.0
        Name: gel_point, dtype: float64
```

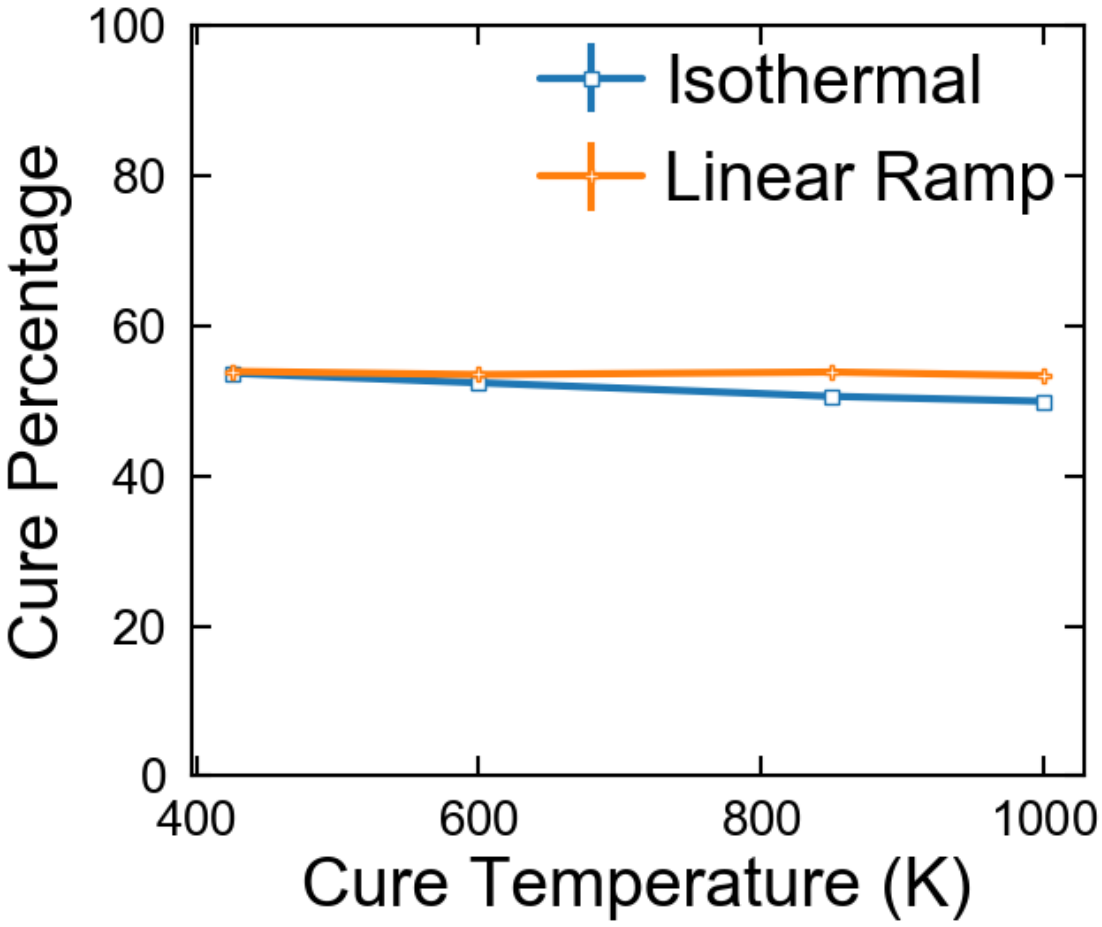
In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline


fig, ax = plt.subplots()
df_bonded = df[(df.bond==True)&
               (df.activation_energy==2)&
               (df.dcd_write==20000)&
               (df.cure_percent>60)&
               (df.profile !='step')].sort_values('kT')
#df_T =
df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')].sort_values('kT')
#print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
#print(df_T['T'],df_T['cure_percent'])
#break
#plt.rcParams["font.family"] = "sans serif"
for key, group in df_bonded.groupby('profile'):
    print(key)
    meanVals = group.groupby(['T']).mean()
    #print('mean',meanVals)
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,
                y=meanVals['gel_point'],
                yerr=stdVals['gel_point'],
                label=names[key],
                marker = markers[key],
                markeredgewidth=1,
                markerfacecolor="white",
                #markersize=12,
                    #linestyle='--',
                color=colors[key], markeredgecolor=colors[key])
                #marker='*',
                #barsabove=True,capsize=5)

legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
ax.set_xlabel('Cure Temperature (K)')
ax.set_ylabel('Gel Point (time step)')

ax.xaxis.major.formatter._useMathText = True
ax.yaxis.major.formatter._useMathText = True
#ax.set_ylim(0,1.0e7)
#ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/gel_poi
nt.png',transparent=True)
plt.show()
```

```
iso
lin_ramp


/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [4]: fig, ax = plt.subplots()
        df_bonded = df[(df.bond==True)&
                       (df.activation_energy==2)&
                       (df.dcd_write==20000)&
                       (df['T']>200)& #because except one of the tajectories out of 10 did not
        gel for T=200 K
                       (df.profile !='step')].sort_values('kT')
        #df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
        !='step')].sort_values('kT')
        #print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
        #print(df_T['T'],df_T['cure_percent'])
        #break
        linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
        colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
        for key, group in df_bonded.groupby('profile'):
            print(key)
            meanVals = group.groupby(['T']).mean()
            #print('mean',meanVals)
            stdVals = group.groupby(['T']).sem()
            #print('std',stdVals)
            ax.errorbar(x=meanVals.index,y=meanVals['curing_at_gel_point'],yerr=stdVals['curing_
        at_gel_point'],
                        label=names[key],
                        markeredgewidth=1,
                        markerfacecolor="white" ,
                        #markersize=12,
```

```
                    marker = markers[key],
                        #linestyle='--',
                    color=colors[key], markeredgecolor=colors[key])
                    #marker='*',
                    #barsabove=True,capsize=5,linestyle=linestyles[key],color=colors[key])

        #ax.legend(fontsize=40)
        legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
        ax.set_xlabel('Cure Temperature (K)')
        ax.set_ylabel('Cure Percentage')
        ax.set_ylim(0,100)
        ax.xaxis.major.formatter._useMathText = True
        ax.yaxis.major.formatter._useMathText = True
        #ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
        #ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/cure_at
        _gel.png',transparent=True)

iso
lin_ramp


/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

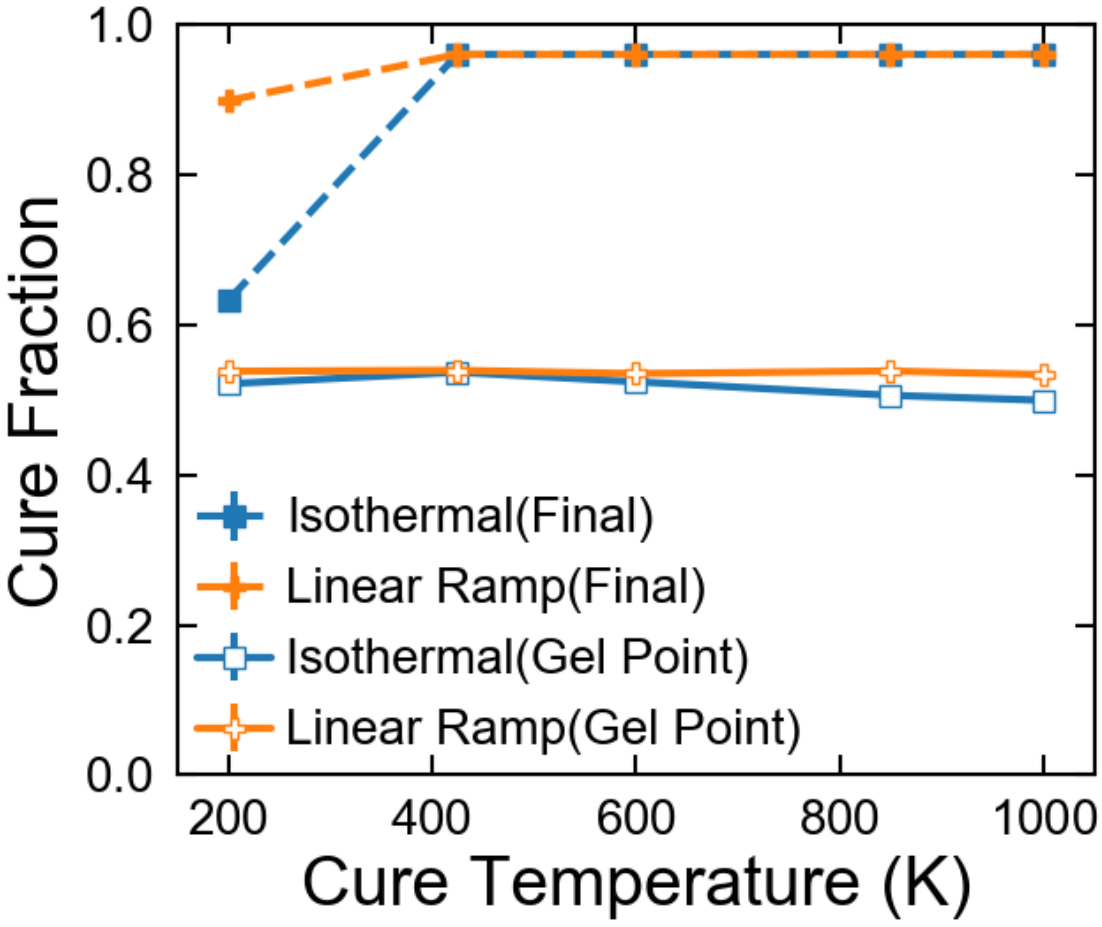


In [13]:
```
fig, ax = plt.subplots()
df_bonded = df[(df.bond==True)&
               (df.activation_energy==2)&
               (df.dcd_write==20000)&
               (df.t_Final==1e7)&
               (df.profile !='step')].sort_values('kT')
#df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
!='step')].sort_values('kT')
#print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
#print(df_T['T'],df_T['cure_percent'])
#break
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
for key, group in df_bonded.groupby('profile'):
    print(key)
    meanVals = group.groupby(['T']).mean()
    #print('mean',meanVals)
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,
                y=meanVals['cure_percent']/100,
                yerr=stdVals['cure_percent']/100,
                label='{}(Final)'.format(names[key]),
                marker = markers[key],
                markersize=8,
                linestyle='--',
                markeredgewidth=1,
                markerfacecolor=colors[key],
                color=colors[key], markeredgecolor=colors[key])

df_bonded = df[(df.bond==True)&
               (df.activation_energy==2)&
               #(df['T']>200)& #because except one of the tajectories out of 10 did not
gel for T=200 K
               (df.dcd_write==20000)&
               (df.profile !='step')].sort_values('kT')
#df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
!='step')].sort_values('kT')
#print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
#print(df_T['T'],df_T['cure_percent'])
#break
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
for key, group in df_bonded.groupby('profile'):
    print(key)
    meanVals = group.groupby(['T']).mean()
    #print('mean',meanVals)
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,
                y=meanVals['curing_at_gel_point']/100,
                yerr=stdVals['curing_at_gel_point']/100,
                label='{}(Gel Point)'.format(names[key]),
                markeredgewidth=1,
                markerfacecolor="white" ,
                markersize=8,
                marker = markers[key],
                color=colors[key],
                markeredgecolor=colors[key])

#ax.legend(fontsize=40)
legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
ax.set_xlabel('Cure Temperature (K)')
ax.set_ylabel('Cure Fraction')
ax.set_ylim(0,1)
```

```
        ax.set_xlim(150,1050)
        ax.xaxis.major.formatter._useMathText = True
        ax.yaxis.major.formatter._useMathText = True
        legend = ax.legend(loc='best', shadow=False, prop={'size':20}, handlelength=1.5,
        borderaxespad = 0)

        #ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
        #ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/cure_at
        _gel_and_final.png',transparent=True)
```

```
iso
lin_ramp
iso
lin_ramp
```

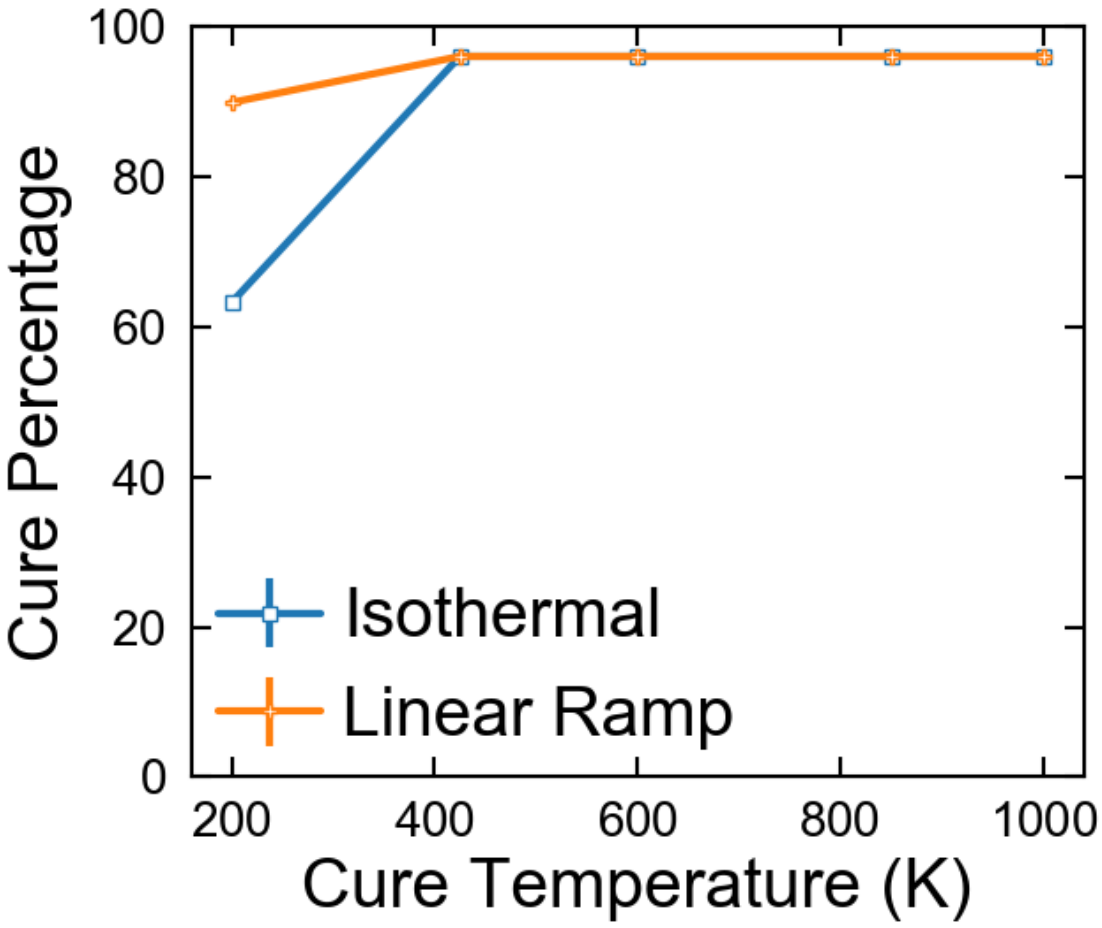
```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [6]:
```
fig, ax = plt.subplots()
df_bonded = df[(df.bond==True)&
               (df.activation_energy==2)&
               (df.dcd_write==20000)&
               (df.t_Final==1e7)&
               (df.profile !='step')].sort_values('kT')
#df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
!='step')].sort_values('kT')
#print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
#print(df_T['T'],df_T['cure_percent'])
#break
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
for key, group in df_bonded.groupby('profile'):
    print(key)
    meanVals = group.groupby(['T']).mean()
    #print('mean',meanVals)
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,y=meanVals['cure_percent'],yerr=stdVals['cure_percent']
,label=names[key],
                marker = markers[key],
                #markersize=12,
                    markeredgewidth=1, markerfacecolor="white" ,
                    #linestyle='--',
                color=colors[key], markeredgecolor=colors[key])
                #marker='*',
                #barsabove=True,capsize=5,linestyle=linestyles[key],color=colors[key])

#ax.legend(fontsize=40)
legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
ax.set_xlabel('Cure Temperature (K)')
ax.set_ylabel('Cure Percentage')
ax.set_ylim(0,100)
ax.xaxis.major.formatter._useMathText = True
ax.yaxis.major.formatter._useMathText = True
#ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
#ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/final_c
ure.png',transparent=True)
```

```
iso
lin_ramp
```

```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

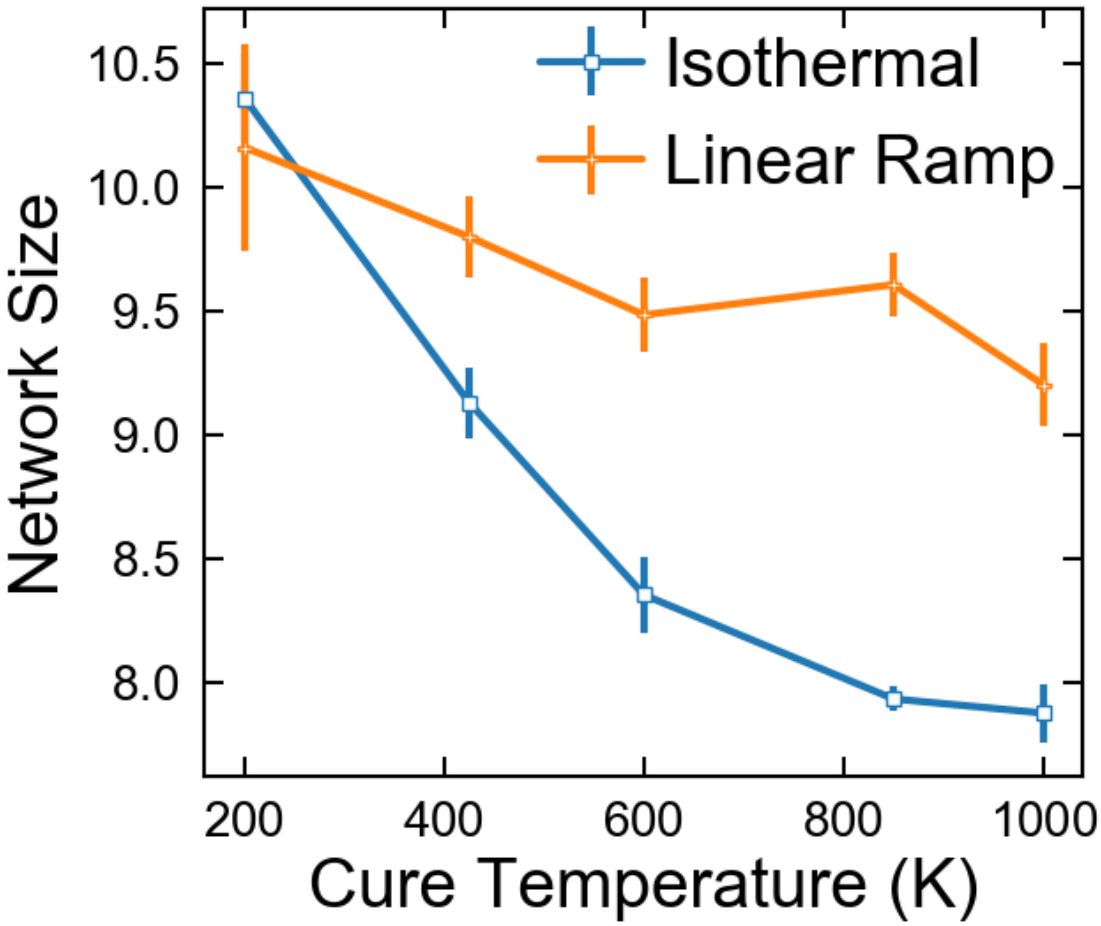



```
In [7]: fig, ax = plt.subplots()
        df_bonded = df[(df.bond==True)&
                       (df.activation_energy==2)&
                       (df.dcd_write==20000)&
                       (df.profile !='step')].sort_values('kT')
        #df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
        !='step')].sort_values('kT')
        #print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
        #print(df_T['T'],df_T['cure_percent'])
        #break
        linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
        colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
        for key, group in df_bonded.groupby('profile'):
            print(key)
            meanVals = group.groupby(['T']).mean()
            #print('mean',meanVals)
            stdVals = group.groupby(['T']).sem()
            #print('std',stdVals)
            ax.errorbar(x=meanVals.index,y=meanVals['av_network_mass_at_gel_point'],
                        yerr=stdVals['av_network_mass_at_gel_point'],label=names[key],
                            markeredgewidth=1, markerfacecolor="white" ,
                        marker = markers[key],
                        #markersize=12,
                            #linestyle='--',
                        color=colors[key], markeredgecolor=colors[key])
                        #marker='*',
```

```
                #barsabove=True,capsize=5,linestyle=linestyles[key],color=colors[key])

        #ax.legend(fontsize=40)
        legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
        ax.set_xlabel('Cure Temperature (K)')
        ax.set_ylabel('Network Size')
        #ax.set_ylim(0,100)
        ax.xaxis.major.formatter._useMathText = True
        ax.yaxis.major.formatter._useMathText = True
        #ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
        #ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/av_netw
        ork.png',transparent=True)
```

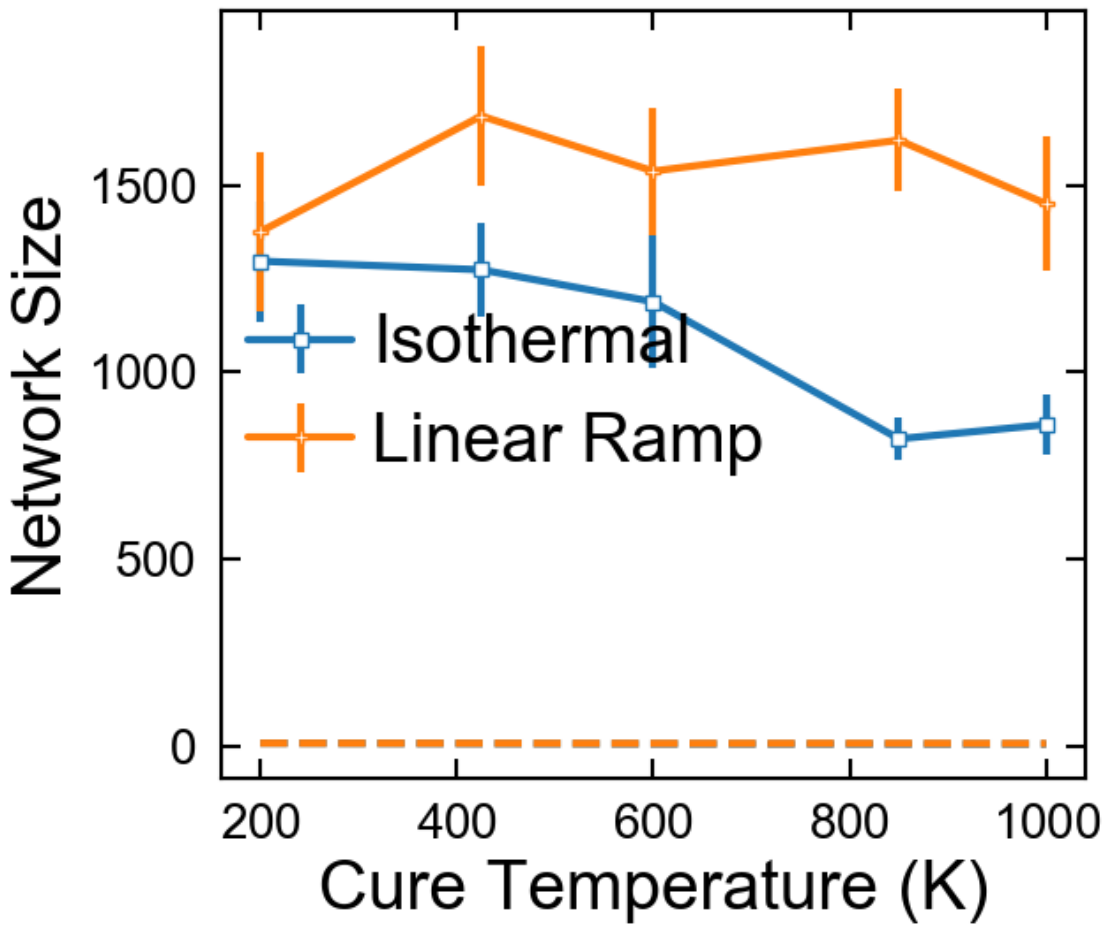
```
iso
lin_ramp
```

```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [8]:
```
fig, ax = plt.subplots()
df_bonded = df[(df.bond==True)&
               (df.activation_energy==2)&
               (df.dcd_write==20000)&
               (df.profile !='step')].sort_values('kT')
#df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
!='step')].sort_values('kT')
#print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
#print(df_T['T'],df_T['cure_percent'])
#break
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
for key, group in df_bonded.groupby('profile'):
    print(key)
    meanVals = group.groupby(['T']).mean()
    #print('mean',meanVals)
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,y=meanVals['second_largest_net_at_gel_point'],
                yerr=stdVals['second_largest_net_at_gel_point'],label=names[key],
                marker = markers[key],
                #markersize=12,
                    markeredgewidth=1, markerfacecolor="white" ,
                    #linestyle='--',
                color=colors[key], markeredgecolor=colors[key])
                #marker='*',
                #barsabove=True,capsize=5,linestyle=linestyles[key],color=colors[key])
ax.plot(meanVals.index,meanVals['av_network_mass_at_gel_point'],linestyle='--',label='')
    print(meanVals['av_network_mass_at_gel_point'])

#ax.legend(fontsize=40)
legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
ax.set_xlabel('Cure Temperature (K)')
ax.set_ylabel('Network Size')
#ax.set_ylim(0,100)
ax.xaxis.major.formatter._useMathText = True
ax.yaxis.major.formatter._useMathText = True
#ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
#ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/second_
network.png',transparent=True)
```

```
iso
T
200.0     10.361242
425.0      9.131079
600.0      8.353872
850.0      7.933194
1000.0     7.876024
Name: av_network_mass_at_gel_point, dtype: float64
lin_ramp
T
200.0     10.162929
425.0      9.801542
600.0      9.486220
850.0      9.608166
1000.0     9.204402
Name: av_network_mass_at_gel_point, dtype: float64


/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [ ]: project = signac.init_project('EpoxpyProject',data_path)

        def getjobprop(jobid):
            job = project.open_job(id=jobid)
            print(job,job.document['curing_at_gel_point'],job.sp.profile,job.sp.T)
        jobids=['cf4b96','e8f383','ce7c2bb','ebb77','c283b2']
        for jobid in jobids:
            getjobprop(jobid)

In [ ]: fig, ax = plt.subplots()
        df_bonded = df[(df.bond==True)&
                       (df.activation_energy==2)&
                       (df.dcd_write==20000)&
                       (df.profile !='step')].sort_values('kT')
        #df_T = df[(df.bond==True)&(df.activation_energy==5)&(df.profile=='iso')&(df.profile
        !='step')].sort_values('kT')
        #print(df_T['gel_point'],df_T['T'],df_T['cure_percent'])
        #print(df_T['T'],df_T['cure_percent'])
        #break
        linestyles={'iso':'-','lin_ramp':'--','step':'-.'}
        colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
        for key, group in df_bonded.groupby('profile'):
            print(key)
            meanVals = group.groupby(['T']).mean()
            #print('mean',meanVals)
```

```
    stdVals = group.groupby(['T']).sem()
    #print('std',stdVals)
    ax.errorbar(x=meanVals.index,y=meanVals['largest_net_at_gel_point'],
                yerr=stdVals['largest_net_at_gel_point'],label=names[key],
                marker = markers[key],
                #markersize=12,
                    markeredgewidth=1, markerfacecolor="white" ,
                    #linestyle='--',
                color=colors[key], markeredgecolor=colors[key])
                #marker='*',
                #barsabove=True,capsize=5,linestyle=linestyles[key],color=colors[key])
ax.plot(meanVals.index,meanVals['av_network_mass_at_gel_point'],linestyle='--',label='')

#ax.legend(fontsize=40)
legend = ax.legend(loc='best', shadow=False, handlelength=1.5, borderaxespad = 0)
ax.set_xlabel('Cure Temperature (K)')
ax.set_ylabel('Network Size')
#ax.set_ylim(0,100)
ax.xaxis.major.formatter._useMathText = True
ax.yaxis.major.formatter._useMathText = True
#ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
#ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/second_
network.png',transparent=True)
```

In [1]:
```
import gsd
import gsd.fl
import gsd.hoomd
from freud import box, density
import signac
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
import hoomd.deprecated
import hoomd.dump
import hoomd
import os
from cme_utils.analyze import diffractometer

def diffract_frame(job,frameid, typeId=2):
    f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
    t = gsd.hoomd.HOOMDTrajectory(f)
    n_frames = len(t)
    if frameid < n_frames and frameid >= 0:
        snapshot = t[frameid]
        sim_box = snapshot.configuration.box
        box_dim=(sim_box[0], sim_box[1], sim_box[2])
        box_dim = np.array(box_dim)
        print('box_dim',box_dim)
        l_pos = snapshot.particles.position
        pos = l_pos[np.where(snapshot.particles.typeid == typeId)]
        diffract_dir = job.fn('diffract_atframe{}_type_{}'.format(frameid,typeId))
        if not os.path.exists(diffract_dir):
                os.makedirs(diffract_dir)
        D = diffractometer.diffractometer(working_dir=diffract_dir)
        D.set(grid_size=512,  peak_width=1, zoom=8, n_views=20,
length_scale=1.0,bot=1e-5,top=1)
        D.load(pos,box_dim)
        D.prep_matrices()
        D.average()
    else:
        print('INVALID FRAME ID GIVEN')


def getFrameAtCurePercent(jobid,cure_p=40):
    thisjob = project.open_job(id=jobid)
    #print(thisjob,thisjob.sp.profile)
    data = np.genfromtxt(thisjob.fn('out.log'))
    for i,this_cure_p in enumerate(data[:,9]):
        if this_cure_p>=cure_p:
            #print('time step:',round(data[i,0]/thisjob.sp.dcd_write))
            return int(round(data[i,0]/thisjob.sp.dcd_write))
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/__init__.py:1405: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
```

In [2]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/ttt/'
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/ttt/'
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_2_vary_Ea/'
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_path/'
```

```
import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_Ea1=df[(df.activation_energy==2)&
          (df['T']==850.0)&
          (df.t_Final==1e7)&
          (df.dcd_write==20000)&
          (df.log_write==10000)&
          (df.trial<10)]#&
          #(df.n_mul==10000)]#&
          #(df.n_dt == 2995000)]
df_Ea1 = df_Ea1.sort_values('n_particles')
jobIds=np.asarray(df_Ea1.signac_id)
jobIds
df_Ea1.profile
```

Out[2]:
```
01a3ab33afee81ae515c6614a382659b         iso
cba7ef45da5e6f6994da398982270e0c    lin_ramp
c94de1c5e06447d55ade313093ab6539    lin_ramp
c07aeff9f77332d189db368d3184f76c    lin_ramp
a9397dd8d86cfebc9edbb4633db41456    lin_ramp
a1ea9b8b8844b7dbdcf69bc7548bc03e         iso
9d6ba4be8d9b6fe9f0a40d661995ebab    lin_ramp
9be87452a95c2d1a7c856fadbb3bb767         iso
9654bbe417c50bcc6c4fee114da3dca6    lin_ramp
7de6d97b3656bc92cb2d1b1493ddebf0         iso
76b32b9209b0fb13bab79b84e5389565         iso
6c5c59f349212ae55daa3da701332248         iso
669a76c3623023288cc6a5ef4f7a37ef    lin_ramp
668bbd5fff155a3f9acce862b1c9e4ee         iso
3d197cde195a941bc75ffbdbdbf18dba    lin_ramp
245885b640323bebb68e024c1d0ab7c3         iso
1b657198ba16741b2ff9d4b04cf4c42e    lin_ramp
08178e8713ab89438e0f775a497c63b1         iso
daf43e56a43528eb5dfc90c848859ae3    lin_ramp
e62f6074db41878eae6d283e578fdc7a         iso
Name: profile, dtype: object
```

1 Cure Percent: 10 %

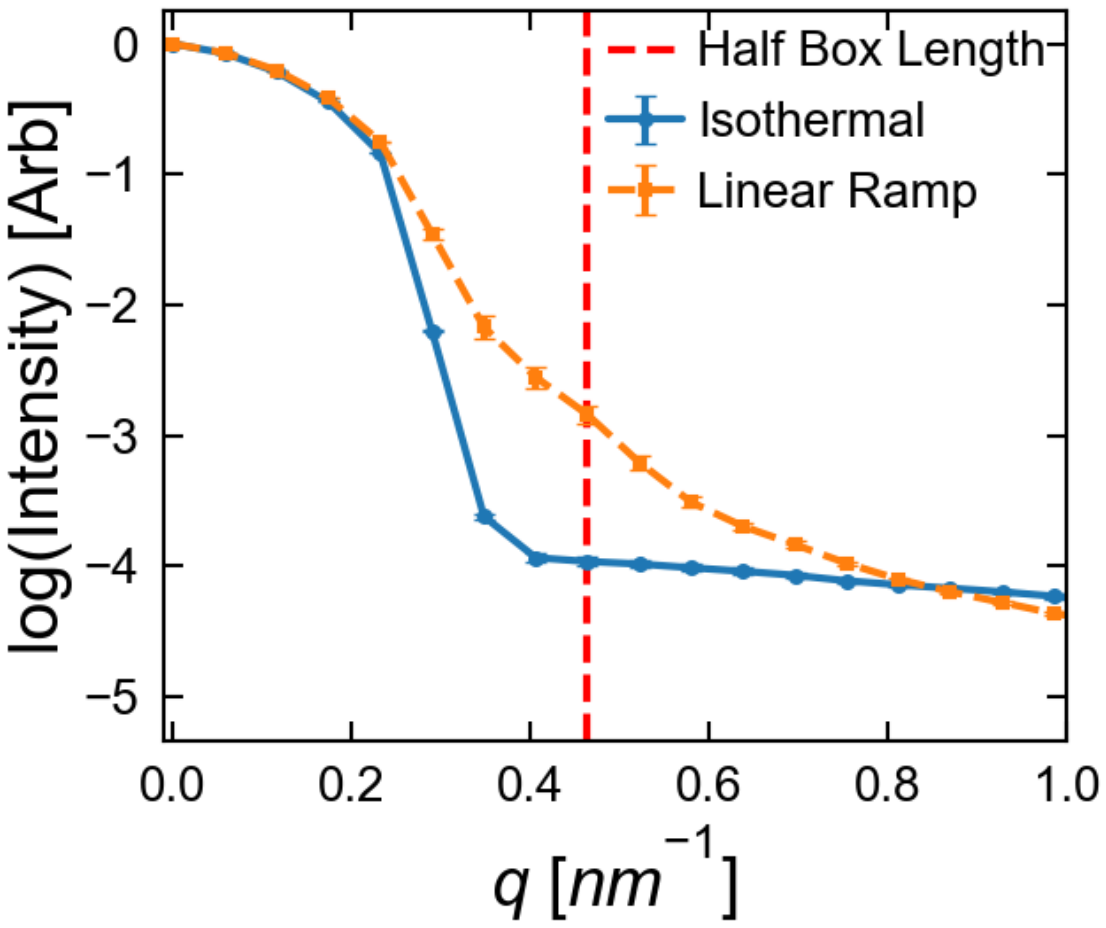
In [ ]:
```
typeId=2
for i,jobId in enumerate(jobIds):
```

```
            job = project.open_job(id=jobId)
            print(job,job.sp.temp_prof)
            hoomd.context.initialize('--mode=cpu')
            snap_at_cure_percent=10
            frame = getFrameAtCurePercent(jobId,snap_at_cure_percent)
            #frame=98
            print('cured to {} percent at frame:{}'.format(snap_at_cure_percent,frame))

            diffr_file_name = 'diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)
            print(job.fn(diffr_file_name),job.isfile(diffr_file_name))
            if not job.isfile(diffr_file_name):
                system = hoomd.init.read_gsd(job.fn('data.gsd'),frame=frame)
                #hoomd.dump.gsd(group=hoomd.group.all(), filename=job.fn('final.gsd'),
        overwrite=True, period=None)
                group_c = hoomd.group.type(name='c-particles', type='C')
                hoomd.deprecated.dump.xml(group=group_c,
        filename=job.fn('CsAtFrame{}.hoomdxml'.format(frame)), position=True)
                diffract_frame(job,frame,typeId=typeId)

In [19]: from cycler import cycler
         from scipy import stats
         import gsd
         import gsd.fl
         import gsd.hoomd

         plt.figure()
         #plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
         #                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
         typeId=2
         iso_qs=[]
         lin_qs=[]
         iso_Is=[]
         lin_Is=[]

         for i,jobId in enumerate(jobIds):
             job = project.open_job(id=jobId)
             #print(job.sp.temp_prof)

             frame = getFrameAtCurePercent(jobId,snap_at_cure_percent)
             #frame=98
             if 'gel_point' in job.document:
                 print(job.sp.profile,job,'gel point',job.document['gel_point'],'diffracting
         @',frame*job.sp['dcd_write'])
             #else:
                 #print(job.sp.profile,job)
             if job.isfile('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)):
         data=np.genfromtxt(job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
                 legend=names[job.sp.profile]#'N={}'.format(job.sp.n_particles)
                 print(job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
                 q = data[:,0]/1.06
                 #print(data[:,0])
                 #print(q)
                 if job.sp.profile=='iso':
                     iso_qs.append(q)
                     iso_Is.append(data[:,1])
                 elif job.sp.profile=='lin_ramp':
                     lin_qs.append(q)
                     lin_Is.append(data[:,1])
                 else:
                     print("**************NOT IMPLEMENTED***************")
                 #plt.plot(q,data[:,1],label=legend)
             else:
                 print('can\'t
         find',job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
             if(job.isfile('data.gsd')):
                 f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
                 t = gsd.hoomd.HOOMDTrajectory(f)
                 n_frames = len(t)
```

```
            end_frame = n_frames-1
            snapshot = t[end_frame]
            sim_box = snapshot.configuration.box
            lx = np.float64(sim_box[0])#JSON serializer does not like float32

        iso_qs=np.asarray(iso_qs)
        lin_qs=np.asarray(lin_qs)
        iso_Is=np.asarray(iso_Is)
        lin_Is=np.asarray(lin_Is)

        iso_Qs_av = np.mean(iso_qs,axis=0)
        iso_Is_av = np.mean(iso_Is,axis=0)
        iso_Is_std = np.std(iso_Is,axis=0)
        iso_Is_sem = stats.sem(iso_Is,axis=0)
        lin_Qs_av = np.mean(lin_qs,axis=0)
        lin_Is_av = np.mean(lin_Is,axis=0)
        lin_Is_std = np.std(lin_Is,axis=0)
        lin_Is_sem = stats.sem(lin_Is,axis=0)

        plt.axvline(x=2*np.pi/(lx*1.06/2),
                    linestyle='--',
                    color='r',
                    label='Half Box Length')
        (_, caps, _) = plt.errorbar(iso_Qs_av,iso_Is_av,iso_Is_sem,linestyle='-',
                                    label='Isothermal',
                                    marker='o',
                                    markersize=5,
                                    capsize=4)
        for cap in caps:
            cap.set_markeredgewidth(1)
        (_, caps, _) = plt.errorbar(lin_Qs_av,lin_Is_av,lin_Is_sem,linestyle='--',label='Linear
        Ramp',
                                    marker='s',
                                    markersize=5,
                                    capsize=4)
        for cap in caps:
            cap.set_markeredgewidth(1)
        plt.xlabel(r"$q$ [$nm^{-1}$]")
        plt.ylabel("log(Intensity) [Arb]")
        plt.xlim(-0.01,1)
        legend = plt.legend(loc='best', shadow=False, prop={'size':20}, handlelength=1.5,
        borderaxespad = 0)
        file_name = 'structure_factor_cure_path_{}_percent'.format(snap_at_cure_percent)
        #plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/{}.png
        '.format(file_name),transparent=True)
        plt.savefig('{}.png'.format(file_name),transparent=True)
        plt.show()
```

```
iso 01a3ab33afee81ae515c6614a382659b gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/01a3ab33afee81ae515c6614a382659b/diffract_atframe2_type_2/asq.txt
lin_ramp cba7ef45da5e6f6994da398982270e0c gel point 920000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/cba7ef45da5e6f6994da398982270e0c/diffract_atframe10_type_2/asq.txt
lin_ramp c94de1c5e06447d55ade313093ab6539 gel point 980000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/c94de1c5e06447d55ade313093ab6539/diffract_atframe10_type_2/asq.txt
lin_ramp c07aeff9f77332d189db368d3184f76c gel point 940000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/c07aeff9f77332d189db368d3184f76c/diffract_atframe10_type_2/asq.txt
lin_ramp a9397dd8d86cfebc9edbb4633db41456 gel point 940000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/a9397dd8d86cfebc9edbb4633db41456/diffract_atframe10_type_2/asq.txt
iso a1ea9b8b8844b7dbdcf69bc7548bc03e gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
```

```
path/workspace/a1ea9b8b8844b7dbdcf69bc7548bc03e/diffract_atframe2_type_2/asq.txt
lin_ramp 9d6ba4be8d9b6fe9f0a40d661995ebab gel point 960000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/9d6ba4be8d9b6fe9f0a40d661995ebab/diffract_atframe10_type_2/asq.txt
iso 9be87452a95c2d1a7c856fadbb3bb767 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/9be87452a95c2d1a7c856fadbb3bb767/diffract_atframe2_type_2/asq.txt
lin_ramp 9654bbe417c50bcc6c4fee114da3dca6 gel point 960000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/9654bbe417c50bcc6c4fee114da3dca6/diffract_atframe10_type_2/asq.txt
iso 7de6d97b3656bc92cb2d1b1493ddebf0 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/7de6d97b3656bc92cb2d1b1493ddebf0/diffract_atframe2_type_2/asq.txt
iso 76b32b9209b0fb13bab79b84e5389565 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/76b32b9209b0fb13bab79b84e5389565/diffract_atframe2_type_2/asq.txt
iso 6c5c59f349212ae55daa3da701332248 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/6c5c59f349212ae55daa3da701332248/diffract_atframe2_type_2/asq.txt
lin_ramp 669a76c3623023288cc6a5ef4f7a37ef gel point 980000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/669a76c3623023288cc6a5ef4f7a37ef/diffract_atframe10_type_2/asq.txt
iso 668bbd5fff155a3f9acce862b1c9e4ee gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/668bbd5fff155a3f9acce862b1c9e4ee/diffract_atframe2_type_2/asq.txt
lin_ramp 3d197cde195a941bc75ffbdbdbf18dba gel point 960000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/3d197cde195a941bc75ffbdbdbf18dba/diffract_atframe10_type_2/asq.txt
iso 245885b640323bebb68e024c1d0ab7c3 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/245885b640323bebb68e024c1d0ab7c3/diffract_atframe2_type_2/asq.txt
lin_ramp 1b657198ba16741b2ff9d4b04cf4c42e gel point 960000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/1b657198ba16741b2ff9d4b04cf4c42e/diffract_atframe10_type_2/asq.txt
iso 08178e8713ab89438e0f775a497c63b1 gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/08178e8713ab89438e0f775a497c63b1/diffract_atframe2_type_2/asq.txt
lin_ramp daf43e56a43528eb5dfc90c848859ae3 gel point 960000.0 diffracting @ 200000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/daf43e56a43528eb5dfc90c848859ae3/diffract_atframe10_type_2/asq.txt
iso e62f6074db41878eae6d283e578fdc7a gel point 180000.0 diffracting @ 40000.0
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_
path/workspace/e62f6074db41878eae6d283e578fdc7a/diffract_atframe2_type_2/asq.txt


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



2 Cure Percent: 95 %

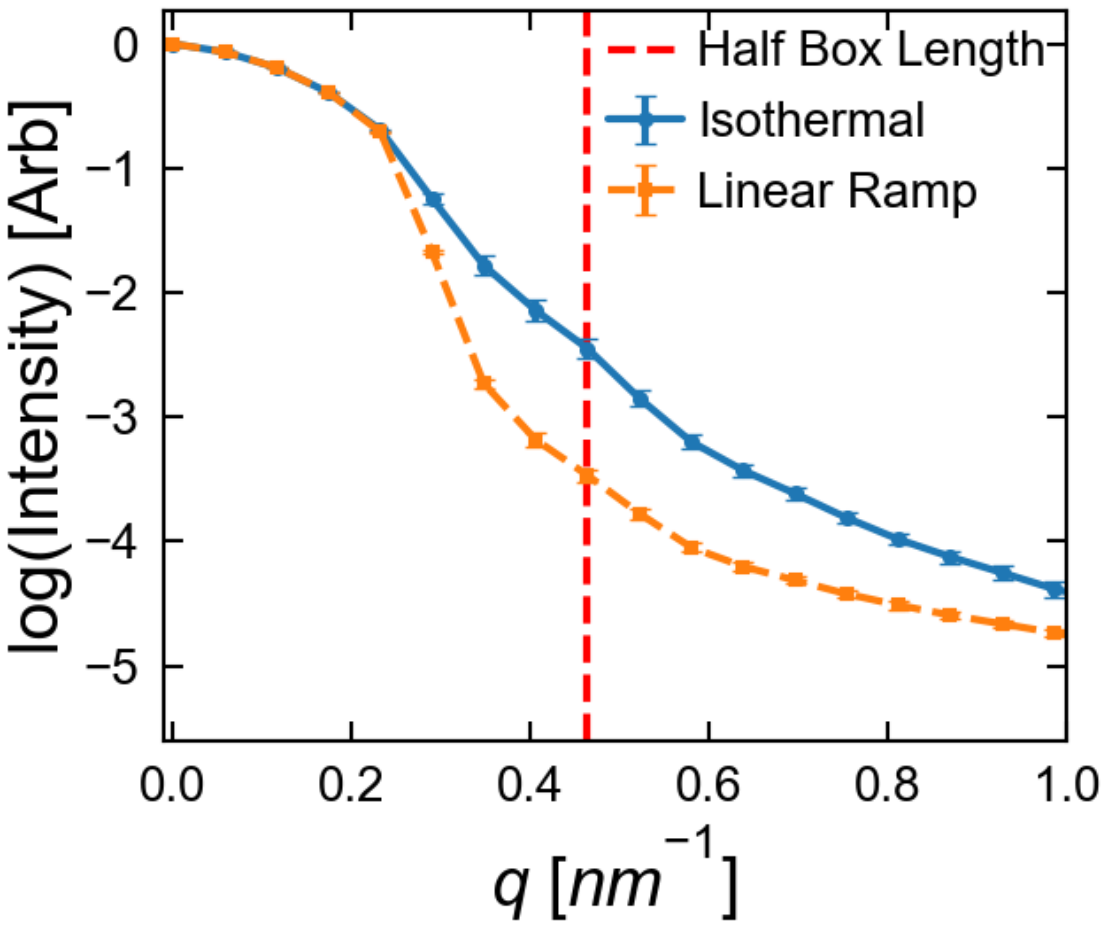
```
In [ ]: typeId=2
        for i,jobId in enumerate(jobIds):
            job = project.open_job(id=jobId)
            print(job,job.sp.temp_prof)
            hoomd.context.initialize('--mode=cpu')
            snap_at_cure_percent=95
            frame = getFrameAtCurePercent(jobId,snap_at_cure_percent)
            #frame=98
            print('cured to {} percent at frame:{}'.format(snap_at_cure_percent,frame))

            system = hoomd.init.read_gsd(job.fn('data.gsd'),frame=frame)
            #hoomd.dump.gsd(group=hoomd.group.all(), filename=job.fn('final.gsd'),
        overwrite=True, period=None)
            group_c = hoomd.group.type(name='c-particles', type='C')
            hoomd.deprecated.dump.xml(group=group_c,
        filename=job.fn('CsAtFrame{}.hoomdxml'.format(frame)), position=True)
            if not job.isfile('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)):
                diffract_frame(job,frame,typeId=typeId)

In [21]: from cycler import cycler
         from scipy import stats
```

```
plt.figure()
#plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
#                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
typeId=2
iso_qs=[]
lin_qs=[]
iso_Is=[]
lin_Is=[]

for i,jobId in enumerate(jobIds):
    job = project.open_job(id=jobId)
    print(job.sp.temp_prof)

    frame = getFrameAtCurePercent(jobId,snap_at_cure_percent)
    #frame=98
    if 'gel_point' in job.document:
        print(job.sp.profile,job,'gel point',job.document['gel_point'],'diffracting
@',frame*job.sp['dcd_write'])
    #else:
        #print(job.sp.profile,job)
    if job.isfile('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)):
data=np.genfromtxt(job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
        legend=names[job.sp.profile]#'N={}'.format(job.sp.n_particles)
        q = data[:,0]/1.06
        #print(data[:,0])
        #print(q)
        if job.sp.profile=='iso':
            iso_qs.append(q)
            iso_Is.append(data[:,1])
        elif job.sp.profile=='lin_ramp':
            lin_qs.append(q)
            lin_Is.append(data[:,1])
        else:
            print("**************NOT IMPLEMENTED***************")
        #plt.plot(q,data[:,1],label=legend)
    else:
        print('can\'t
find',job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
    if(job.isfile('data.gsd')):
        f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
        t = gsd.hoomd.HOOMDTrajectory(f)
        n_frames = len(t)
        end_frame = n_frames-1
        snapshot = t[end_frame]
        sim_box = snapshot.configuration.box
        lx = np.float64(sim_box[0])#JSON serializer does not like float32
iso_qs=np.asarray(iso_qs)
lin_qs=np.asarray(lin_qs)
iso_Is=np.asarray(iso_Is)
lin_Is=np.asarray(lin_Is)

iso_Qs_av = np.mean(iso_qs,axis=0)
iso_Is_av = np.mean(iso_Is,axis=0)
iso_Is_std = np.std(iso_Is,axis=0)
iso_Is_sem = stats.sem(iso_Is,axis=0)
lin_Qs_av = np.mean(lin_qs,axis=0)
lin_Is_av = np.mean(lin_Is,axis=0)
lin_Is_std = np.std(lin_Is,axis=0)
lin_Is_sem = stats.sem(lin_Is,axis=0)

plt.axvline(x=2*np.pi/(lx*1.06/2),
            linestyle='--',
            color='r',
            label='Half Box Length')

(_, caps, _) = plt.errorbar(iso_Qs_av,iso_Is_av,iso_Is_sem,linestyle='-',
                            label='Isothermal',
                            marker='o',
```

```
                                markersize=5,
                                capsize=4)
        for cap in caps:
            cap.set_markeredgewidth(1)
        (_, caps, _) = plt.errorbar(lin_Qs_av,lin_Is_av,lin_Is_sem,linestyle='--',label='Linear
        Ramp',
                                marker='s',
                                markersize=5,
                                capsize=4)
        for cap in caps:
            cap.set_markeredgewidth(1)

        plt.xlabel(r"$q$ [$nm^{-1}$]")
        plt.ylabel("log(Intensity) [Arb]")
        plt.xlim(-0.01,1)
        legend = plt.legend(loc='best', shadow=False, prop={'size':20}, handlelength=1.5,
        borderaxespad = 0)
        file_name = 'structure_factor_cure_path_{}_percent'.format(snap_at_cure_percent)
        #plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/struct
        ure_factor_cure_path_95_percent.png',transparent=True)
        plt.savefig('{}.png'.format(file_name),transparent=True)
        plt.show()
```

```
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 01a3ab33afee81ae515c6614a382659b gel point 180000.0 diffracting @ 1240000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp cba7ef45da5e6f6994da398982270e0c gel point 920000.0 diffracting @ 4080000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp c94de1c5e06447d55ade313093ab6539 gel point 980000.0 diffracting @ 4080000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp c07aeff9f77332d189db368d3184f76c gel point 940000.0 diffracting @ 4100000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp a9397dd8d86cfebc9edbb4633db41456 gel point 940000.0 diffracting @ 4080000.0
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso a1ea9b8b8844b7dbdcf69bc7548bc03e gel point 180000.0 diffracting @ 1200000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp 9d6ba4be8d9b6fe9f0a40d661995ebab gel point 960000.0 diffracting @ 4100000.0
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 9be87452a95c2d1a7c856fadbb3bb767 gel point 180000.0 diffracting @ 1240000.0
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp 9654bbe417c50bcc6c4fee114da3dca6 gel point 960000.0 diffracting @ 4120000.0
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 7de6d97b3656bc92cb2d1b1493ddebf0 gel point 180000.0 diffracting @ 1220000.0
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 76b32b9209b0fb13bab79b84e5389565 gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/76b32b9209b0fb13bab79b84e5389565/diffract_atframe62_type_2/a
sq.txt
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 6c5c59f349212ae55daa3da701332248 gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/6c5c59f349212ae55daa3da701332248/diffract_atframe62_type_2/a
sq.txt
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp 669a76c3623023288cc6a5ef4f7a37ef gel point 980000.0 diffracting @ 4220000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/669a76c3623023288cc6a5ef4f7a37ef/diffract_atframe211_type_2/
asq.txt
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 668bbd5fff155a3f9acce862b1c9e4ee gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/668bbd5fff155a3f9acce862b1c9e4ee/diffract_atframe62_type_2/a
```

```
sq.txt
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp 3d197cde195a941bc75ffbdbdbf18dba gel point 960000.0 diffracting @ 4040000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/3d197cde195a941bc75ffbdbdbf18dba/diffract_atframe202_type_2/
asq.txt
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 245885b640323bebb68e024c1d0ab7c3 gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/245885b640323bebb68e024c1d0ab7c3/diffract_atframe62_type_2/a
sq.txt
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp 1b657198ba16741b2ff9d4b04cf4c42e gel point 960000.0 diffracting @ 4120000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/1b657198ba16741b2ff9d4b04cf4c42e/diffract_atframe206_type_2/
asq.txt
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso 08178e8713ab89438e0f775a497c63b1 gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/08178e8713ab89438e0f775a497c63b1/diffract_atframe62_type_2/a
sq.txt
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
lin_ramp daf43e56a43528eb5dfc90c848859ae3 gel point 960000.0 diffracting @ 4140000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/daf43e56a43528eb5dfc90c848859ae3/diffract_atframe207_type_2/
asq.txt
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
iso e62f6074db41878eae6d283e578fdc7a gel point 180000.0 diffracting @ 1240000.0
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/cure_path/workspace/e62f6074db41878eae6d283e578fdc7a/diffract_atframe62_type_2/a
sq.txt


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

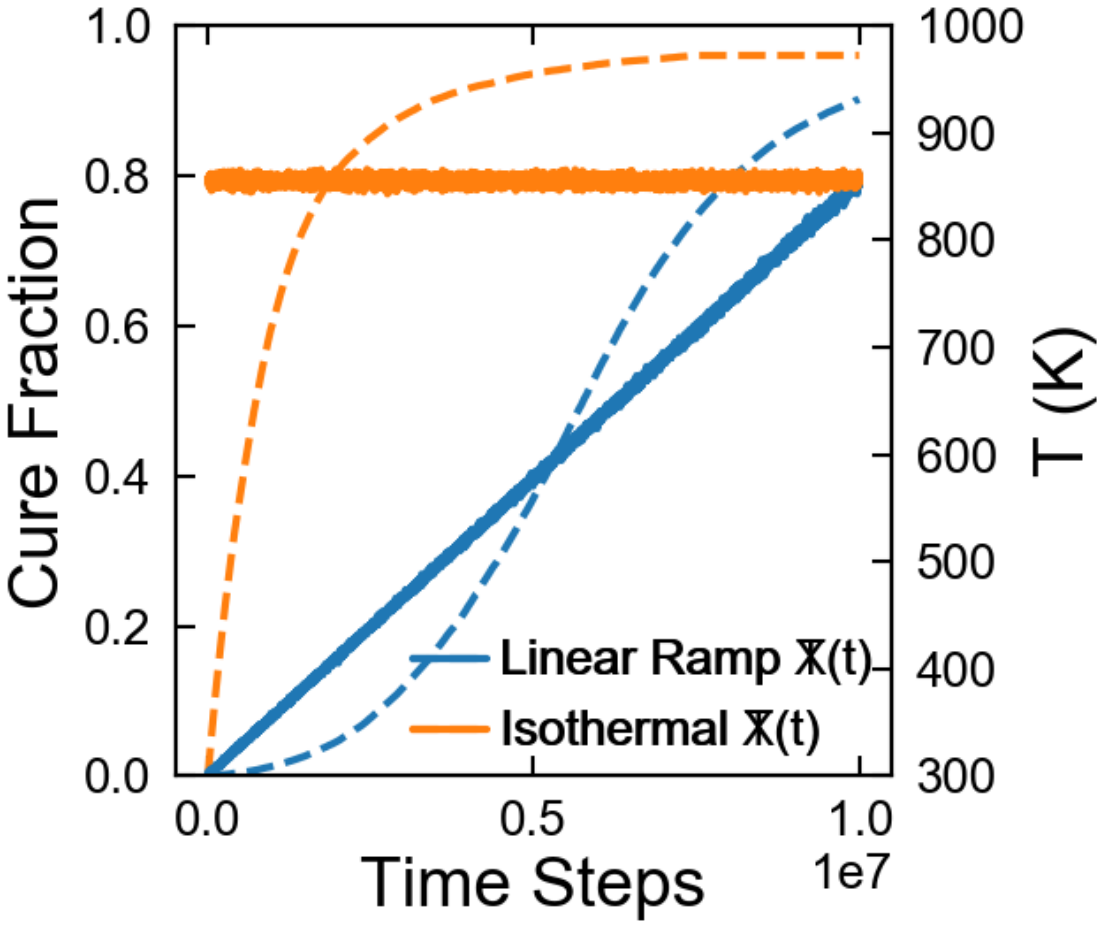

Half Box Length
Isothermal
Linear Ramp
log(Intensity) [Arb]
0
−1
−2
−3
−4
−5
0.0
0.2
0.4
0.6
0.8
1.0
q [nm^−1]

In [3]:
```
### data_path = '/Users/stephenthomas/projects/epoxy_sim/epoxpy/examples/dybond/data/'
#data_path = '/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_pape
r/data/Diffusivity/blend_PES10_CED_molar_vol_50/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_4/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_2_0'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_2_vary_Ea'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/ttt'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/ttt'

#compare_freud_dybond_rdf_sorted_nlist/'
#compare_freud_dybond_rdf_after_dist_check/'#tps_comparison_after_bug_fix/'
import gsd
import gsd.fl
import gsd.hoomd
from freud import box, density
import signac
import matplotlib.pyplot as plt
import numpy as np
#from epoxpy.utils import RDF
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
project = signac.get_project(root=data_path)
markers={'iso':'s','lin_ramp':'P','step':'>'}

project = signac.get_project(root=data_path)

df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_lin_ramp_bonding = df[(df.bond==True)&
                         (df.activation_energy==5.0)&
(df.kT==1.9346163405434587)&#2.2760192241687775)&#1.3656115345012650)&
                         (df.profile!='step')&
                         (df.trial==0)]#(df.profile=='step')]
#print('linear ramp thermal cured bonding jobs')
grpedByCalib = df_lin_ramp_bonding.groupby('calibrationT').apply(lambda x:
x.sort_values('kT'))
#print(grpedByCalib['kT'],grpedByCalib['profile'])
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax2 = ax1.twinx()
for signac_id in grpedByCalib['signac_id']:
    job = project.open_job(id=signac_id)
    print(job.sp.temp_prof)
    if job.sp.temp_prof[-1][0]==9995000.0:
        data = np.genfromtxt(job.fn('out.log'))
        if job.sp.profile=='iso':
            prof = 'Isothermal'
        elif job.sp.profile == 'lin_ramp':
            prof='Linear Ramp'
        else:
            prof='step'
        ax1.plot(data[:,0],data[:,9]/100,linestyle='--',label='{} X(t)'.format(prof))
        ax2.plot(data[:,0],data[:,6]*job.sp.calibrationT,label='{} T(t)'.format(prof))
        ax1.set_xlabel('Time Steps')
        ax1.set_ylabel('Cure Fraction')
        ax1.set_ylim(0,1)
```

```
        ax2.set_ylim(300,1000)
        ax2.set_ylabel('T (K)')
        legend = ax1.legend(loc='best', shadow=False, prop={'size':20},
handlelength=1.5, borderaxespad = 0)
        legend = ax2.legend(loc='best', shadow=False, prop={'size':20},
handlelength=1.5, borderaxespad = 0)
        #ax1.legend(loc='lower right')
        #ax2.legend(loc='center left')
#plt.title('box size: {},{},{},harmonic bond
k:{}'.format(round(fbox.Lx,2),round(fbox.Ly,2),round(fbox.Lz,2),10),fontsize=20)
plt.show()
```

```
[[5000.0, 1.9346163405434587], [4995000.0, 1.9346163405434587]]
[[5000.0, 0.6828057672506325], [4995000.0, 1.9346163405434587]]
[[5000.0, 0.6828057672506325], [9995000.0, 1.9346163405434587]]
[[5000.0, 1.9346163405434587], [9995000.0, 1.9346163405434587]]
```

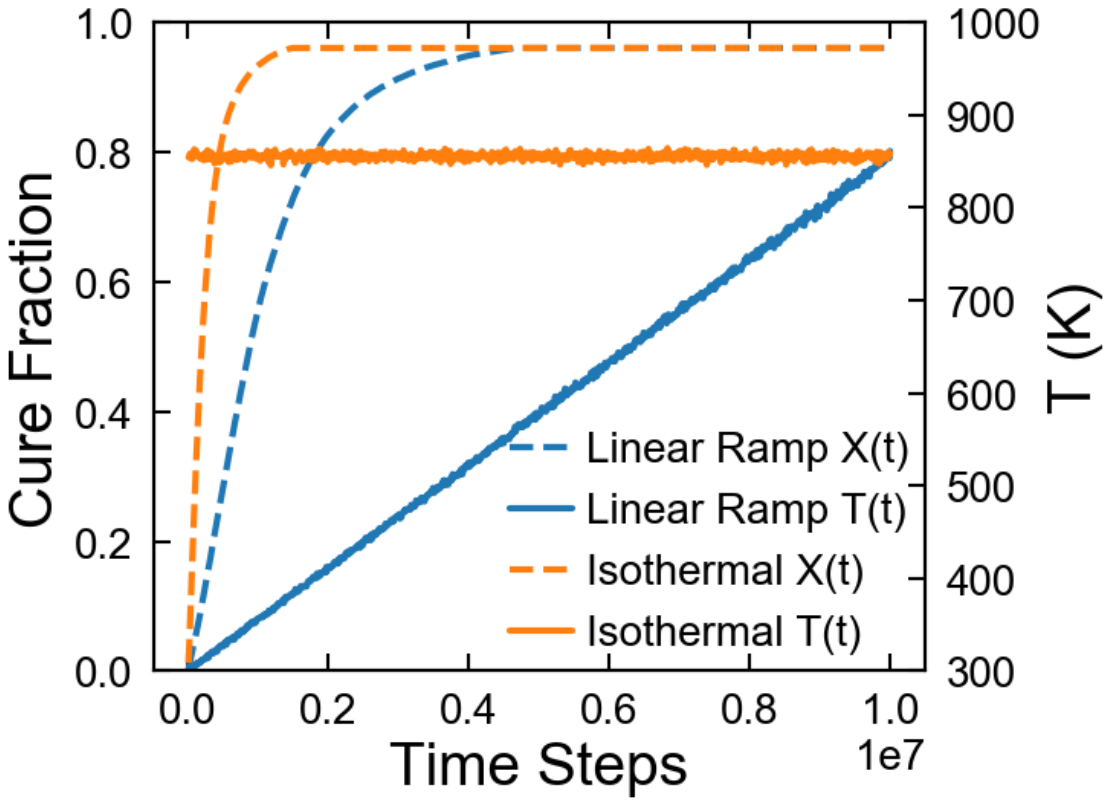
```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [25]:
```
### data_path = '/Users/stephenthomas/projects/epoxy_sim/epoxpy/examples/dybond/data/'
#data_path = '/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_pape
r/data/Diffusivity/blend_PES10_CED_molar_vol_50/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_4/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_2_0'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/av_aij_A_0_2_vary_Ea'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/ttt'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/cure_path'

#compare_freud_dybond_rdf_sorted_nlist/'
#compare_freud_dybond_rdf_after_dist_check/'#tps_comparison_after_bug_fix/'
import gsd
import gsd.fl
import gsd.hoomd
from freud import box, density
import signac
import matplotlib.pyplot as plt
import numpy as np
#from epoxpy.utils import RDF
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
project = signac.get_project(root=data_path)
markers={'iso':'s','lin_ramp':'P','step':'>'}

project = signac.get_project(root=data_path)

df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_lin_ramp_bonding = df[(df.activation_energy==2)&
          (df.kT*df.calibrationT==850.0)&
          (df.dcd_write==20000)&
          (df.t_Final==1e7)&
          (df.log_write==10000.0)&
          (df.trial==0)]
#print('linear ramp thermal cured bonding jobs')
grpedByCalib = df_lin_ramp_bonding.groupby('calibrationT').apply(lambda x:
x.sort_values('kT'))
#print(grpedByCalib['kT'],grpedByCalib['profile'])
fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(111)
ax2 = ax1.twinx()
lines=[]
for signac_id in grpedByCalib['signac_id']:
    job = project.open_job(id=signac_id)
    print(job.sp.temp_prof)
    print(job)
    data = np.genfromtxt(job.fn('out.log'))
    if job.sp.profile=='iso':
        prof = 'Isothermal'
    elif job.sp.profile == 'lin_ramp':
        prof='Linear Ramp'
    else:
        prof='step'
    ln1 = ax1.plot(data[:,0],
                   data[:,9]/100,
                   linestyle='--',
```

```
                    label='{} X(t)'.format(prof))
        ln2 = ax2.plot(data[:,0],
                    data[:,5]*job.sp.calibrationT,
                    #marker=markers[job.sp.profile],
                    label='{} T(t)'.format(prof))
        lines.append(ln1)
        lines.append(ln2)
        ax1.set_xlabel('Time Steps')
        ax1.set_ylabel('Cure Fraction')
        ax1.set_ylim(0,1)
        ax2.set_ylim(300,1000)
        #ax1.set_xlim(0,4e6)
        ax2.set_ylabel('T (K)')

    for i,line in enumerate(lines):
        if i==0:
            lns=line
        else:
            lns=lns+line
    labs = [l.get_label() for l in lns]
    ax1.legend(lns, labs, loc=0)
    legend = ax1.legend(lns, labs,loc='lower right', shadow=False, prop={'size':20},
    handlelength=1.5, borderaxespad = 0)
    #legend = ax2.legend(loc='center left', shadow=False, prop={'size':15},
    handlelength=1.5, borderaxespad = 0)
        #ax1.legend(loc='lower right')
        #ax2.legend(loc='center left')
    plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/cure_pr
    ofile_vs_morphology.png',transparent=True)
    #plt.title('box size: {},{},{},harmonic bond
    k:{}'.format(round(fbox.Lx,2),round(fbox.Ly,2),round(fbox.Lz,2),10),fontsize=20)
    plt.show()
```

```
[[5000.0, 0.6828057672506325], [10005000.0, 1.9346163405434587]]
a9397dd8d86cfebc9edbb4633db41456
[[5000.0, 1.9346163405434587], [10005000.0, 1.9346163405434587]]
e62f6074db41878eae6d283e578fdc7a


/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


Cure Fraction
T (K)
Time Steps
1e7
Linear Ramp X(t)
Linear Ramp T(t)
Isothermal X(t)
Isothermal T(t)
0.0
0.2
0.4
0.6
0.8
1.0
300
400
500
600
700
800
900
1000

In [29]:
```
import numpy as np
import math
import signac
import gsd
import gsd.fl
import gsd.hoomd
import networkx as nx
from freud import box, density
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.cm as mplcm
import matplotlib.colors as colors
import os
from scipy.optimize import curve_fit
%matplotlib inline

import scipy
def fit_sim_data_with_model_plot_wrt_A(data_path,alpha_inf,model,kTs,dt=0.01,fit_availab
le=False, A=1.0,plot=True,save_path=None):
    #kTs = [0.1,0.5,1.0,2.0,4.0,6.0]
    project = signac.init_project('MyProject', data_path)
    if plot:
        plt.figure(0)
    C=8.12
    R2s = []
    dt_to_tau = 0.01
    best_job=None
    Hs = []
    for kT in kTs:
        #print('kT')
        jobs = project.find_jobs()
        best_fit_timesteps = []
        best_fit_cure_percents = []
        worst_fit_timesteps = []
        worst_fit_cure_percents = []
        best_fit_cure=[]
        best_bond_period = -1
        best_bpps = -1
        best_last_frame = -1
        maxr = -100000
        minr = 1000000
        found_jobs = False
        for job in jobs:
            thiskT = job.sp.temp_prof[-1][1]
            thisnB = job.sp.percent_bonds_per_step*(40000./100.)
            thisTauB = job.sp.bond_period
            thisA = thisnB/thisTauB
            #print('kT',thiskT,'nB',thisnB,'tauB',thisTauB,'A',thisA)
            if thiskT == kT and thisA==A and job.sp.bond==True:
                if get_status(job)=='job-computed':
                    #print('job computed')
                    found_jobs = True
                else:
                    #print('job',job,'did not complete')
                    continue
                #print('kT',thiskT,'nB',thisnB,'tauB',thisTauB,'A',thisA)
                data = np.genfromtxt(job.fn('out.log'),skip_header=1)
                n_ts = len(data)
                time_steps = []
                cure_percents = []
                truncated_cure_fractions = []
                truncated_time_steps = []
                #print(data[:,9])
                last_index = next((i for i, v in enumerate(data[:,9]) if v >=alpha_inf),
-1)
                first_index = next((i for i, v in enumerate(data[:,9]) if v >0), -1)
                #print('last_index',last_index,'first_index',first_index)
                #print('system reached {} at {} kT by time step {}'.format(alpha_inf,
```

```
                #                                                       kT,
                #
data[last_index,0]*dt))

                if last_index<=0 and fit_available:
                    last_index=len(data[:,9])
                    #print('modified last index:',last_index,'first index:',first_index)
                #print('last_index',last_index,'first_index',first_index,job)
                if last_index > 0 and first_index >= 0 and (last_index-first_index)>1:
                    #print('conditions met')
truncated_time_steps.extend(data[first_index:last_index,0]*dt)#/0.01)
                    truncated_cure_fractions.extend(data[first_index:last_index,9]/100.)
                    Ea=job.sp.activation_energy#1.0
                    a_inf = truncated_cure_fractions[-1]#0.96
                    import warnings
                    np.seterr(all='raise')
                    plot_fit_fails=True
                    try:
                        popt, pcov = curve_fit(lambda times, H: f_t(times, C,
H,Ea,kT,truncated_cure_fractions[0],
a_inf,model=model),
truncated_time_steps,truncated_cure_fractions,
                                               p0=[1],
                                               bounds=([0],[np.infty]))#,maxfev=200000)
                        #print('found fit')
                    except FloatingPointError:
                        if plot_fit_fails and plot:
                            #print('Curve fitting failed(FloatingPointError) for {} kT
and tauB{}, nB{}, A{}'.format(kT,thisTauB,thisnB, thisA))
                            #print(job)
                            plt.figure(1,figsize=(10,8))
plt.scatter(truncated_time_steps,truncated_cure_fractions,marker='x',label='FAILED:
$kT$: {},$\\tau_B$:{}, $n_B$:{}, A:{}'.format(kT,thisTauB,thisnB,thisA))
                            plt.legend(fontsize=20)
                            plt.ylim(0,1)
                            #plt.xlim(0,5e6)
                            plt.xlabel('time',fontsize=20)
                            plt.ylabel('cure fraction',fontsize=20)
                        continue

                    except RuntimeError:
                        if plot_fit_fails and plot:
                            #print('Curve fitting failed(FloatingPointError) for {} kT
and tauB{}, nB{}, A{}'.format(kT,thisTauB,thisnB, thisA))
                            #print(job)
                            plt.figure(1,figsize=(10,8))
plt.scatter(truncated_time_steps,truncated_cure_fractions,label='FAILED: $kT$:
{},$\\tau_B$:{}, $n_B$:{}, A:{}'.format(kT,thisTauB,thisnB,thisA))
                            plt.legend(fontsize=20)
                            plt.ylim(0,1)
                            #plt.xlim(0,5e6)
                            plt.xlabel('time',fontsize=20)
                            plt.ylabel('cure fraction',fontsize=20)
                        continue

                    except TypeError:
                        if plot_fit_fails and plot:
                            #print('Curve fitting failed(FloatingPointError) for {} kT
and tauB{}, nB{}, A{}'.format(kT,thisTauB,thisnB, thisA))
                            #print(job)
                            plt.figure(1,figsize=(10,8))
plt.scatter(truncated_time_steps,truncated_cure_fractions,label='FAILED: $kT$:
{},$\\tau_B$:{}, $n_B$:{}, A:{}'.format(kT,thisTauB,thisnB,thisA))
                            plt.legend(fontsize=20)
                            plt.ylim(0,1)
                            #plt.xlim(0,5e6)
                            plt.xlabel('time',fontsize=20)
                            plt.ylabel('cure fraction',fontsize=20)
```

```
                    continue
                except ValueError:
                    continue

                ydata = np.asarray(truncated_cure_fractions)
                fit_ydata =
f_t(truncated_time_steps,C,*popt,Ea,kT,truncated_cure_fractions[0],a_inf,model=model)
                residuals = ydata - fit_ydata
                ss_res = np.sum(residuals**2)
                ss_tot = np.sum((ydata-np.mean(ydata))**2)
                #print('ss_res',ss_res,'ss_tot',ss_tot)
                if ss_tot == 0:
                    #print('found ss_tot: 0')
                    r_squared = 0
                else:
                    r_squared = 1 - (ss_res / ss_tot)
                    #print('found least squares fit! r2',r_squared)

                #cs.append(popt[0])

                if r_squared >0:# and (-2<popt[0]<11):
                    #R2s.append(r_squared)
                    #bond_periods.append(job.sp.bond_period)
                    #bppss.append(job.sp.percent_bonds_per_step)
                    #if (kT!=4.0) or (kT == 4.0 and job.sp.bond_period == 1 and
job.sp.percent_bonds_per_step == 0.00250) :
                        if r_squared > maxr:
                            maxr = r_squared
                            best_bond_period = job.sp.bond_period
                            best_kT = kT
                            best_cure = ydata[-1]
                            best_fit_timesteps = truncated_time_steps
                            best_fit_cure_percents = truncated_cure_fractions
                            best_fit_cure = fit_ydata
                            best_bpps = job.sp.percent_bonds_per_step
                            if model == 'FO' or model== 'SO':
                                best_H = popt[0]
                            else:
                                best_C = C#popt[0]
                                best_H = popt[0]
                            best_job = job.get_id()
                            if kT == 0.1:
                                best_last_frame = len(cure_percents)
                        if r_squared < minr:
                            minr = r_squared
                            worst_bond_period = job.sp.bond_period
                            worst_kT = kT
                            #worst_cure = ydata[-1]
                            worst_fit_timesteps = truncated_time_steps
                            worst_fit_cure_percents = truncated_cure_fractions
                            worst_fit_cure = fit_ydata
                            worst_bpps = job.sp.percent_bonds_per_step
                            worst_C = popt[0]
                else:
                    if last_index == -1:
                        insuff_cure = data[-1,9]
                        #print('last index = -1')
                    elif last_index == 0:
                        too_fast_cure = data[-1,9]
                    #print('conditions not met')
                    #continue
            best_bps = best_bpps*40000/100
            #print('thisA',thisA,'A',A)
            ######## UNCOMMENT BELOW LINE TO GET VALUES FOR TABLE IN PAPER##############
            #print('A:',A,'kT',kT,'best tauB',best_bond_period,'best I(nB/NB)',best_bpps)
            #print(best_last_frame)

            #print('maxr',maxr)
```

```
        if found_jobs:
            if last_index == -1:
                label = '$kT$: {} Insufficient cure of {}'.format(kT,insuff_cure/100.)
            elif last_index == 0:
                label = '$kT$: {} Cured to {} at first time
step'.format(kT,too_fast_cure)
            else:
                if maxr == -100000:
                    label='$kT$: {} Least squares fit not found!'.format(kT)
                else:
                    R2s.append(maxr)
                    Hs.append(best_H)
                    common = '$\\tau_B$:{}, $n_B$:{},
A:{}'.format(best_bond_period,best_bps,best_bps/best_bond_period)
                    #label='$kT$: {}, H:{}, ($R^2$={})'.format(kT,round(best_H,4),
round(maxr,4))
                    label='$kT$: {} ($R^2$={})'.format(kT, round(maxr,4))
            #print(label)
            if plot:
                plt.figure(0)
plt.scatter(best_fit_timesteps,best_fit_cure_percents,marker='x')#,label='cure percent')
                #plt.title(common)
                plt.plot(best_fit_timesteps,best_fit_cure,label=label)

#plt.scatter(worst_fit_timesteps,worst_fit_cure_percents,marker='x')#,label='cure
percent')
                #plt.plot(worst_fit_timesteps,worst_fit_cure,label='kT:{}, bond
period,perstep:{},{} ($R^2$={})'.format(worst_kT,worst_bond_period,worst_bpps,
round(minr,3)))

                plt.xlabel('Time')
                plt.ylabel('Cure Fraction')
                plt.xlim(0,6.2e4)
                plt.ylim(0,1)
                legend = plt.legend(loc='best', shadow=False, prop={'size':20},
handlelength=1.5, borderaxespad = 0)
        #print(best_job)

        #print('R2s',R2s)
    if len(R2s)>0:
        #print('minC',np.min(cs), 'maxC',np.max(cs))
        success = True
    else:
        R2s = [0.]
        cs = [0.]
        Hs = [0.]
        success = False
    if plot:
        #plt.title('{}, average
$R^2$:{}'.format(model,round(np.mean(R2s),4)),fontsize=8)
        if save_path is None:
            path='{}_A_{}.png'.format(model,A)
        else:
            path=save_path
        plt.savefig(path, transparent=True)
        print('mean H:',np.mean(Hs))
        #plt.show()
        return success,len(R2s),round(np.mean(R2s),5)
    else:
        return success,len(R2s),round(np.mean(R2s),5),np.mean(Hs)

def get_status(job):
    status = 'init'
    if job.isfile('data.gsd') and job.isfile('out.log'):
        status = 'job-computed'
    elif job.isfile('temperature_profile.png'):
        status = 'temperature-profile-created'
```

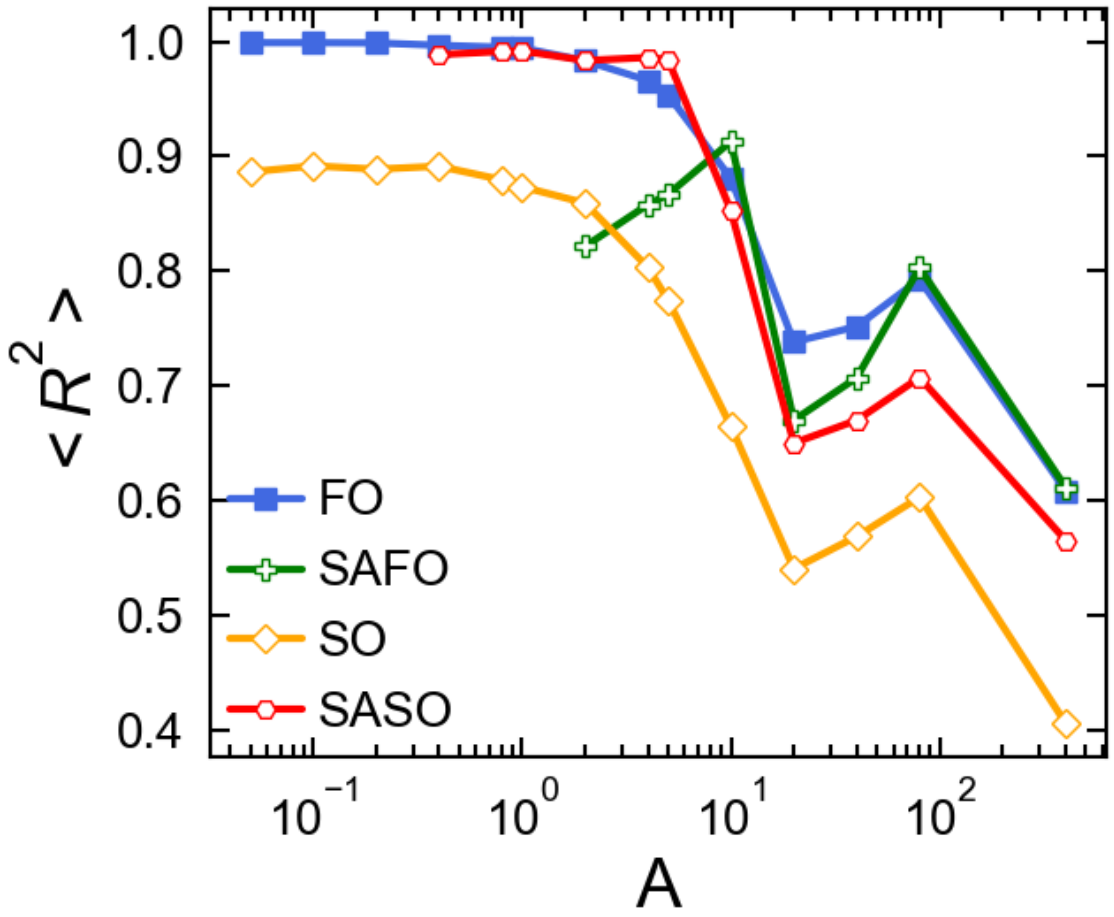
```
        return status

    def f_t(times,C,H,Ea,kT,a_start,a_inf,breakAt_a=None,model='SAFO'):
        #print('C,H',C,H)
        alphas = []
        minutes = []
        alpha=a_start
        #a_inf = 0.96

        for t in times:
            #print(f(alpha))
            #print(k(t,kT))
            k = H*math.exp(-Ea/(kT))
            if model == 'SAFO':
                dadt = k*(a_inf-alpha)*(1+C*alpha)
            elif model == 'FO':
                dadt = k*(a_inf-alpha)
            elif model == 'SO':
                dadt = k*(a_inf-alpha)**2
            elif model == 'SASO':
                dadt = k*(1-alpha)*(a_inf-alpha)*(1+C*alpha)
            #print(dadt)
            alpha += dadt
            alphas.append(alpha)
            minutes.append(t/60.)
            #print(alpha,a_inf)

            if (breakAt_a is not None) and (alpha >= breakAt_a):
                t_minutes = t/60
                print('{} reached @ {} minutes according to the
    model'.format(alpha,t_minutes))
                return alphas,minutes,t_minutes
                #    print('done at',t)
                #    break
        return alphas

In [4]: import itertools
        import matplotlib

        data_path = '/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
        /data/fitting_curing_with_exp_95_percent_fixed_temp_variant/'#fitting_curing_with_exp_95
        _percent_added_cooling_down/'

        names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
        colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
        project = signac.get_project(root=data_path)
        markers={'iso':'s','lin_ramp':'P','step':'>'}

        bonding_periods = [1,5,10,20,40,80,100]
        percent_bonds_per_steps = [0.00250,.005,0.01,1]
        As = []
        for bonding_period in bonding_periods:
            for percent_bonds_per_step in percent_bonds_per_steps:
                As.append(percent_bonds_per_step*(40000./100.)/bonding_period)

        As = np.unique(As)
        #print(As)
        kTs = [0.5,1.0,2.0,4.0,6.0]

        models = ['FO','SAFO','SO','SASO']
        plt.figure()

        colors = itertools.cycle(["royalblue", "g", "orange", "r"])
        markers = itertools.cycle(["s", "P", "D", "H"])
        for model in models:
            Rs = []
            NonZeroAs = []
            ns = []
```

```
            for A in As:
                success, n, r,h = fit_sim_data_with_model_plot_wrt_A(data_path,
                                                        alpha_inf=95.0,
                                                        model=model,
                                                        A=A,
                                                        kTs=kTs,
                                                        plot=False)
                if success:#r > 0:
                    NonZeroAs.append(A)
                    Rs.append(r)
                    ns.append(n)
            color = next(colors)
            marker = next(markers)
            if color is 'royalblue':
                facecolor='royalblue'
            else:
                facecolor='white'
            plt.plot(NonZeroAs,
                     Rs,
                     color=color,
                     label=model,
                     marker=marker,
                     markeredgewidth=1,
                     markerfacecolor=facecolor,
                     markersize=8,
                     #linestyle='--',
                     markeredgecolor=color)
            #plt.figure(0,figsize=(10,8))
            #plt.plot(NonZeroAs,ns,marker='o',color=color,label=model,markersize=20)
            print(model,NonZeroAs)
            print(Rs)


        #plt.figure(0,figsize=(10,8))
        #plt.xlabel('A')
        #plt.ylabel('Number of temperatures that has a fit')
        #plt.xscale('log')
        ##plt.xlim(-1,50.1)
        ##plt.ylim(0.8,1.05)
        #plt.legend()

        #plt.figure(figsize=(15,10))
        plt.xlabel('A')
        plt.ylabel('$< R^2 >$')
        plt.xscale('log')
        #plt.xlim(0,90)
        #plt.ylim(0.8,1.05)
        plt.legend(loc='best')
        legend = plt.legend(loc='best', shadow=False, prop={'size':20}, handlelength=1.5,
        borderaxespad = 0)
        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/compari
        ng_all_models.png', transparent=True)
```

```
FO [0.050000000000000003, 0.10000000000000001, 0.20000000000000001,
0.40000000000000002, 0.80000000000000004, 1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 40.0, 80.0,
400.0]
[0.99895999999999996, 0.99895, 0.99877000000000005, 0.99670000000000003,
0.99500999999999995, 0.99446999999999997, 0.98328000000000004, 0.96613000000000004,
0.95367999999999997, 0.88075000000000003, 0.73828000000000005, 0.75168999999999997,
0.79301999999999995, 0.60863999999999996]
SAFO [2.0, 4.0, 5.0, 10.0, 20.0, 40.0, 80.0, 400.0]
[0.82149000000000005, 0.85785, 0.86682000000000003, 0.91288000000000002, 0.67013,
0.70708000000000004, 0.80291999999999997, 0.61175000000000002]
SO [0.050000000000000003, 0.10000000000000001, 0.20000000000000001,
0.40000000000000002, 0.80000000000000004, 1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 40.0, 80.0,
400.0]
```

```
[0.88641000000000003, 0.89168999999999998, 0.88878999999999997, 0.89137999999999995,
0.87953000000000003, 0.87346999999999997, 0.85933999999999999, 0.80339000000000005,
0.77480000000000004, 0.66490000000000005, 0.54073000000000004, 0.56881999999999999,
0.60311000000000003, 0.40622999999999998]
SASO [0.40000000000000002, 0.80000000000000004, 1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 40.0,
80.0, 400.0]
[0.98860999999999999, 0.99136000000000002, 0.99187000000000003, 0.98343999999999998,
0.98607999999999996, 0.98424, 0.85267000000000004, 0.65024999999999999,
0.67003999999999997, 0.70723999999999998, 0.56560999999999995]
```

```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



1 best_curing_fit

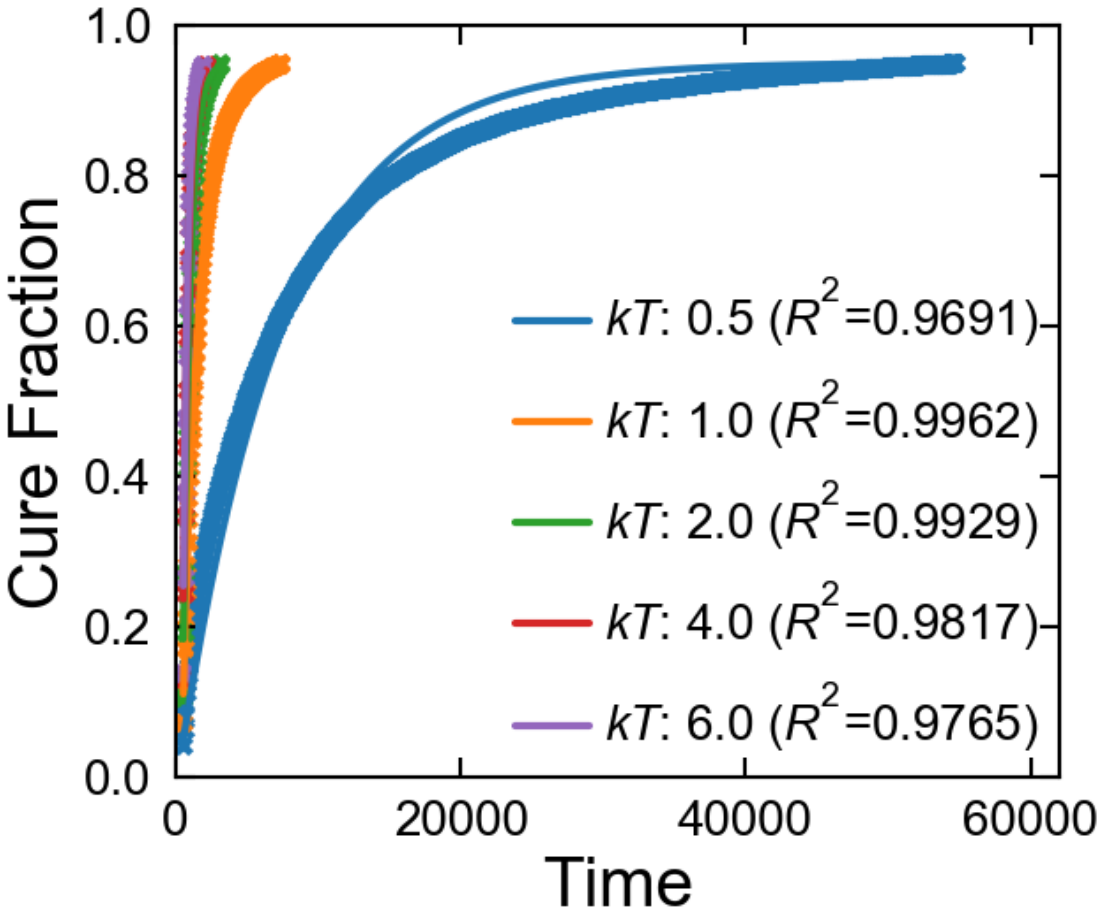
```
In [30]: kTs = [0.5,1.0,2.0,4.0,6.0]
         save_path='/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/best_curi
```

```
        ng_fit.png'
        fit_sim_data_with_model_plot_wrt_A(data_path,model='FO',
                                           A=2.0,
                                           kTs=kTs,
                                           alpha_inf=95.0,
                                           fit_available=True,
                                           save_path=save_path)
```

```
mean H: 0.130831533633
```

Out[30]: (True, 5, 0.98328000000000004)

```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



2 FO_model_lowA

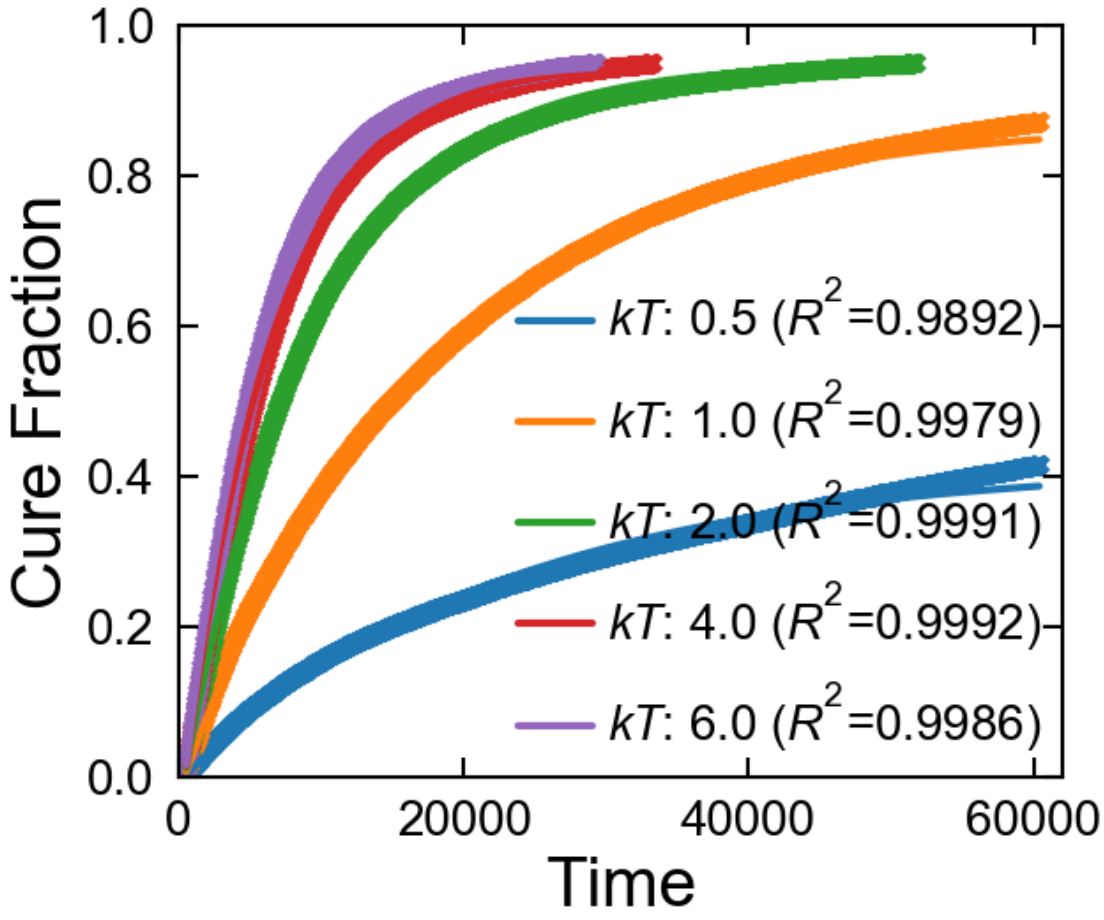
```
In [31]: kTs = [0.5,1.0,2.0,4.0,6.0]
         save_path='/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/FO_model_
```

```
lowA.png'
fit_sim_data_with_model_plot_wrt_A(data_path,model='FO',
                                   A=0.1,
                                   kTs=kTs,
                                   alpha_inf=95.0,
                                   fit_available=True,
                                   save_path=save_path)
```

```
mean H: 0.0107703447572
```

Out[31]: (True, 5, 0.99678999999999995)

```
/Users/stephenthomas/miniconda3/envs/mbuild_0_7_3/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

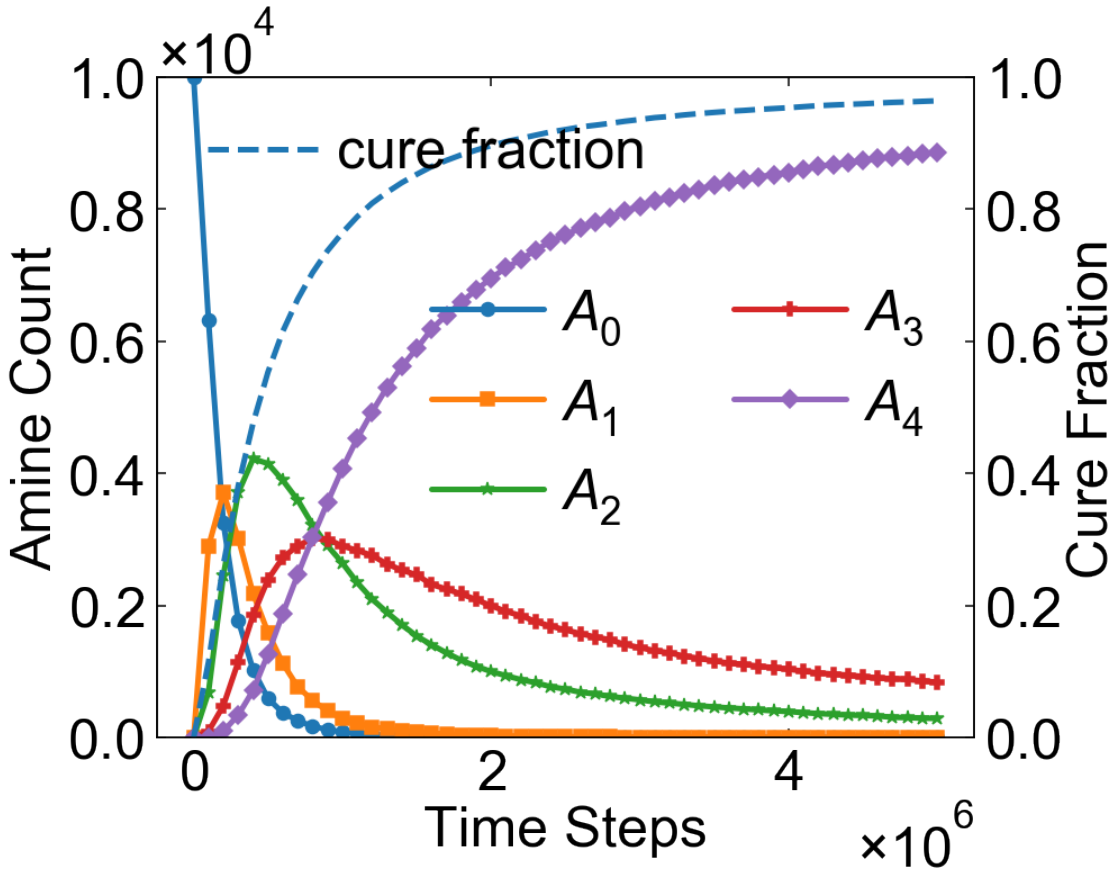


In [5]:
```
import signac
import gsd
import random
import gsd.fl
import gsd.hoomd
import os
import numpy as np
from freud import box, density
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib

def get_status(job):
    status = 'init'
    if job.isfile('data.gsd'):
        status = 'job-computed'
    elif job.isfile('temperature_profile.png'):
        status = 'temperature-profile-created'

    return status

#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/tps_comparison2/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/tps_comparison_after_bug_fix/'


project = signac.init_project('MyProject', os.path.join(data_path))
jobs = project.find_jobs()

matplotlib.rc('xtick', labelsize=50)
matplotlib.rc('ytick', labelsize=50)
matplotlib.rc('axes', labelsize=30)
matplotlib.rc('lines', linewidth=5)

for job in jobs:
    job_status = get_status(job)
    kT = job.sp.temp_prof[-1][1]
    isdybond = job.sp.use_dybond_plugin is True
    isfrued = job.sp.use_dybond_plugin is False and job.sp.legacy_bonding is False
    if (job_status == 'job-computed' and kT==1.0 and (isdybond)):# or isfrued)):
        #plt.figure(figsize=(15,10))


        fig,ax1 = plt.subplots(figsize=(14,11))
        ax2 = ax1.twinx()
        print('final state points: {} time step, {} kT,
status:{}'.format(job.sp.temp_prof[-1][0],kT,job_status))
        print(job)

        f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
        t = gsd.hoomd.HOOMDTrajectory(f)
        n_frames = len(t)
        print('frames: ',n_frames)


        frames = []
        totalA = job.sp.n_mul*10 #A:B:C10 = 10:20:2
        frames.append(0)
        primaryA_cnts = []
        primaryA_cnts.append(totalA)
        secondaryA_cnts = []
        secondaryA_cnts.append(0)
        ternaryA_cnts = []
        ternaryA_cnts.append(0)
        quarternaryA_cnts = []
        quarternaryA_cnts.append(0)
        fivernaryA_cnts = []
```

```
        fivernaryA_cnts.append(0)
        otherA_cnts = []
        otherA_cnts.append(0)
        cure_fractions=[]
        cure_fractions.append(0)

        for i in range(n_frames):
            bond_rank_dict = {}
            snapshot = t[i]
            #find how many repaeting indices are found in the bond table
            #print(snapshot.bonds.group)
            for pairs in snapshot.bonds.group:
                #print(pairs)
                for p in pairs:
                    ptype = snapshot.particles.typeid[p]
                    #print(ptype)
                    #print(p)
                    if ptype == 0:
                        if p in bond_rank_dict:
                            bond_rank_dict[p]+=1
                        else:
                            bond_rank_dict[p]=1
            #print(bond_rank_dict)

            #classify the repeating indices by type
            #count them to get primary Amines and epoxies
            primaryA_cnt=0;
            secondaryA_cnt=0
            ternaryA_cnt=0
            quarternaryA_cnt=0
            fivernaryA_cnt=0
            otherA_cnt=0
            #print(snapshot.particles.__dir__())
            for p in bond_rank_dict:
                if snapshot.particles.typeid[p] == 0: #if A
                    bond_cnt = bond_rank_dict[p]
                    if bond_cnt==1:
                        secondaryA_cnt+=1
                    elif bond_cnt==2:
                        ternaryA_cnt+=1
                    elif bond_cnt==3:
                        quarternaryA_cnt+=1
                    elif bond_cnt==4:
                        fivernaryA_cnt+=1
                    else:#if bond_cnt==5:
                        otherA_cnt+=1


            frames.append(i+1)

            total_bondedA = len(bond_rank_dict)
            primaryA_cnt = totalA-total_bondedA
            #print('totalA:',totalA,'total
bondedA:',total_bondedA,'primaryA:',primaryA_cnt)
            totalA_bonds_made = sum(bond_rank_dict.values())
            total_possible_A_bonds = totalA*4
            cure_fractions.append(totalA_bonds_made/total_possible_A_bonds)
            primaryA_cnts.append(primaryA_cnt)
            secondaryA_cnts.append(secondaryA_cnt)
            ternaryA_cnts.append(ternaryA_cnt)
            quarternaryA_cnts.append(quarternaryA_cnt)
            fivernaryA_cnts.append(fivernaryA_cnt)
            otherA_cnts.append(otherA_cnt)

        bondmethod = 'dybond'
        if isdybond is False:
            bondmethod = 'freud'
```

```
            legend = '{} kT, bond
        weight:{}'.format(job.sp.temp_prof[-1][1],job.sp.sec_bond_weight)
            frames=np.asarray(frames)
            #plt.plot(data[:,0],data[:,1],marker='o',label='curing @{}'.format(legend))
            l1=ax1.plot(frames*job.sp.dcd_write,primaryA_cnts,marker='o',markersize=12,label
        ='$A_0$')#primary amine')
            l2=ax1.plot(frames*job.sp.dcd_write,secondaryA_cnts,marker='s',markersize=12,lab
        el='$A_1$')#'secondary amine')
            l3=ax1.plot(frames*job.sp.dcd_write,ternaryA_cnts,marker='*',markersize=12,label
        ='$A_2$')#'ternary amine')
            l4=ax1.plot(frames*job.sp.dcd_write,quarternaryA_cnts,marker='P',markersize=12,l
        abel='$A_3$')#'quarternary amine')
            l5=ax1.plot(frames*job.sp.dcd_write,fivernaryA_cnts,marker='D',markersize=12,lab
        el='$A_4$')#'quinary amine')
            #ax1.plot(frames,otherA_cnts,marker='o',label='other amine')
            l6=ax2.plot(frames*job.sp.dcd_write,cure_fractions,linestyle='--',label='cure
        fraction')
            #print(bonding_method, 'cure fraction', cure_fractions[-1])

            #plot_lines.append([l1, l2, l3, l4, l5, l6])
            random.seed(12345)
            kTs = [1]
            activation_energy = 1
            weights = [1]
            for weight in weights:
                legend = 'weight:{}'.format(weight)
                #plot_bonding(kTs, activation_energy,no_weights=False,time=5e6,show_all=Fals
        e,weight=weight,show_figure=False,legend=legend)


            ax1.set_xlabel('Time Steps',fontsize=50)
            ax1.set_ylabel('Amine Count',fontsize=50)
            ax2.set_ylabel('Cure Fraction',fontsize=50)
            ax2.set_ylim(0,1)
            #plt.ylim(0,100)

            ax1.legend(fontsize=50,loc='center right',ncol=2)
            ax2.legend(fontsize=50,loc='upper left')
            ax1.xaxis.major.formatter._useMathText = True
            ax1.yaxis.major.formatter._useMathText = True
            ax1.set_ylim(0,1.0e4)
            ax1.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
            ax1.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
            #plt.title(bondmethod,fontsize=20)
            plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images
        /species_concentration.png',transparent=True)
            plt.show()
```

```
final state points: 5040001.0 time step, 1 kT, status:job-computed
c0e7479ed42445d9cc321bce8c9138ce
frames:  50


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


×10^4
1.0
0.8
0.6
0.4
0.2
0.0
Amine Count
cure fraction
A0
A1
A2
A3
A4
1.0
0.8
0.6
0.4
0.2
0.0
Cure Fraction
0
2
4
Time Steps
×10^6

In [2]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

matplotlib.rc('xtick', labelsize=40)
matplotlib.rc('ytick', labelsize=40)
matplotlib.rc('axes', labelsize=20)
matplotlib.rc('lines', linewidth=4)

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_Ea1=df[(df.activation_energy==1)&
          (df.kT*df.calibrationT==500.0)&
          (df.profile=='iso')&
          (df.n_dt == 2995000)]
df_Ea1 = df_Ea1.sort_values('n_particles')
jobIds=np.asarray(df_Ea1.signac_id)
jobIds
```

Out[2]:
```
array(['2434aa0fb3bb465acba90e3b7de26c28',
       'd22fc36554f63ee299afe1c9ec9fc04a',
       '89b2b959bbc964ff5eceb38fce28b71c',
       'a09892aca2c0ad478e7ddb9abd959df6',
       'd77da4e8684580dce74c4172c09515ed'], dtype=object)
```

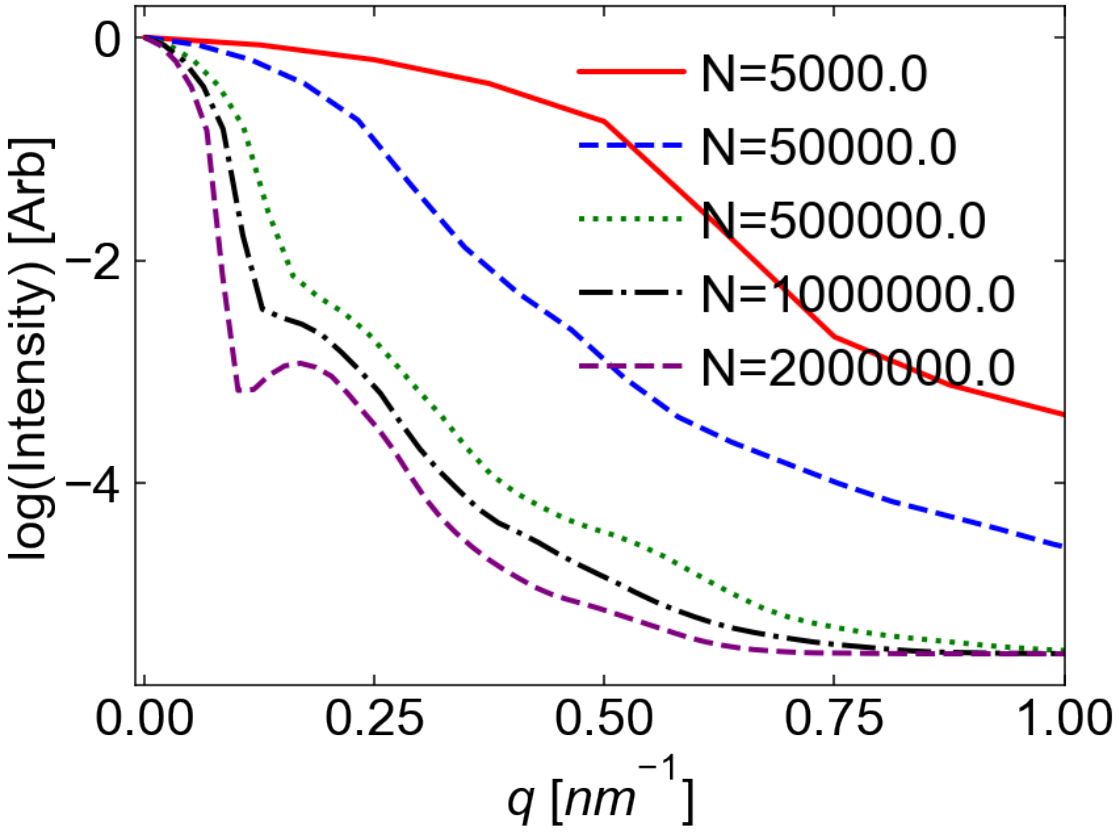
In [21]:
```
from cycler import cycler
plt.figure(figsize=(12,9))
plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
for i,jobId in enumerate(jobIds):
    job = project.open_job(id=jobId)
    print(job.sp.temp_prof)
    if job.sp.temp_prof[-1][0] == 2995000.0:
        if 'gel_point' in job.document:
            print(job.sp.profile,job,'gel point',job.document['gel_point'])
        else:
            print(job.sp.profile,job)
        if job.isfile('diffract_type_2/asq.txt'):
            data=np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
            legend='N={}'.format(job.sp.n_particles)
            q = data[:,0]/1.06
            #print(data[:,0])
            #print(q)
            plt.plot(q,data[:,1],label=legend)
        else:
            print('can\'t find',job.fn('diffract_type_2/asq.txt'))
plt.xlabel(r"$q$ [$nm^{-1}$]",fontsize=40)
plt.ylabel("log(Intensity) [Arb]",fontsize=40)
plt.xlim(-0.01,1)
plt.legend(fontsize=40)
```

```
        #plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/struct
        ure_factor.png',transparent=True)
```

```
[[5000.0, 1.1380096120843874], [2995000.0, 1.1380096120843874]]
iso 2434aa0fb3bb465acba90e3b7de26c28 gel point 0.0
[[5000.0, 1.1380096120843874], [2995000.0, 1.1380096120843874]]
iso d22fc36554f63ee299afe1c9ec9fc04a gel point 150000.0
[[5000.0, 1.1380096120843874], [2995000.0, 1.1380096120843874]]
iso 89b2b959bbc964ff5eceb38fce28b71c gel point 150000.0
[[5000.0, 1.1380096120843874], [2995000.0, 1.1380096120843874]]
iso a09892aca2c0ad478e7ddb9abd959df6 gel point 150000.0
[[5000.0, 1.1380096120843874], [2995000.0, 1.1380096120843874]]
iso d77da4e8684580dce74c4172c09515ed gel point 150000.0
```

Out[21]: <matplotlib.legend.Legend at 0x11e12bf98>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [4]: from cme_utils.job_control import signac_helpers as diff
        diff_keys = diff.diff_jobs(data_path,'1db14da7ad7f9f6dd8ac4f1cd8818afc','d22fc36554f63ee
        299afe1c9ec9fc04a')
```

```
temp_prof
 1db14da7ad7f9f6dd8ac4f1cd8818afc : [[5000.0, 1.1380096120843874], [9995000.0,
1.1380096120843874]]
 d22fc36554f63ee299afe1c9ec9fc04a : [[5000.0, 1.1380096120843874], [2995000.0,
1.1380096120843874]]
n_dt
 1db14da7ad7f9f6dd8ac4f1cd8818afc : None
 d22fc36554f63ee299afe1c9ec9fc04a : 2995000.0
```

In [32]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

matplotlib.rc('xtick', labelsize=40)
matplotlib.rc('ytick', labelsize=40)
matplotlib.rc('axes', labelsize=20)
matplotlib.rc('lines', linewidth=4)

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_Ea1=df[(df.activation_energy==2)&
          (df.kT*df.calibrationT==500.0)&
          #(df.profile=='iso')&
          (df.n_mul==40000)]#&
          #(df.n_dt == 2995000)]
df_Ea1 = df_Ea1.sort_values('n_particles')
jobIds=np.asarray(df_Ea1.signac_id)
jobIds
```

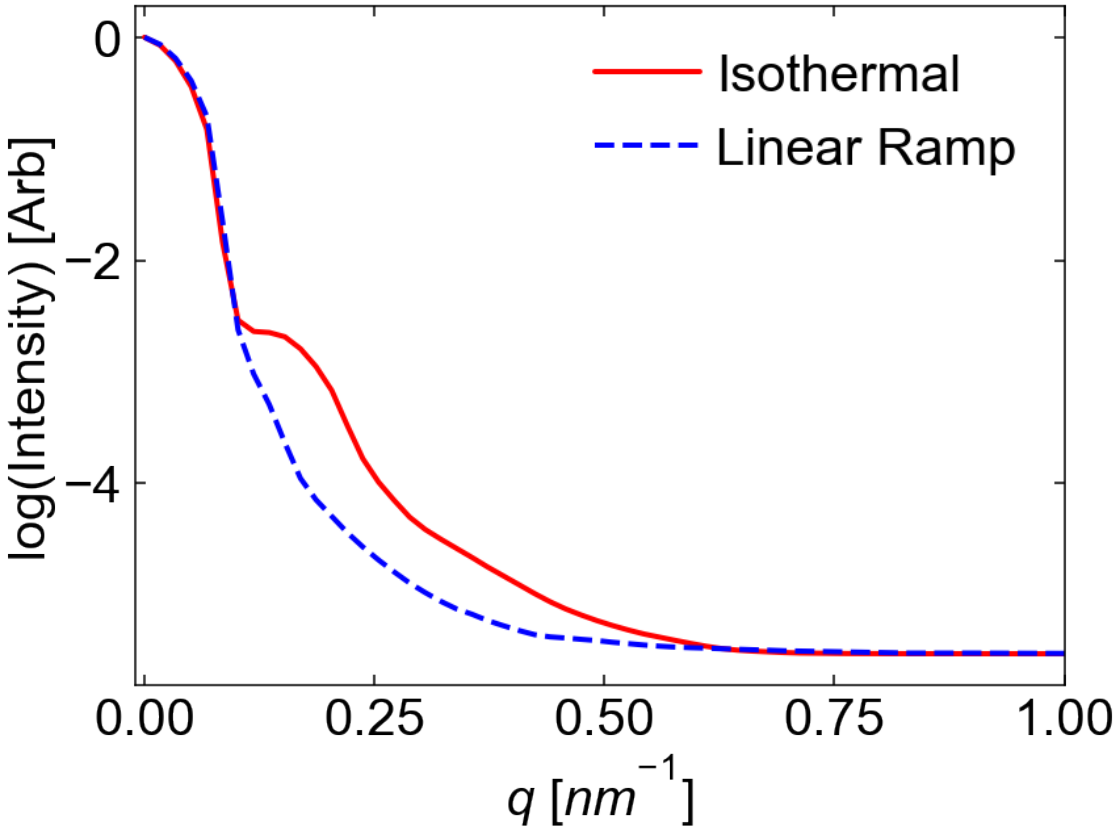
Out[32]:
```
array(['0c05a756d9ba03f975c52d022097d915',
       '30d533afc4e66be8845bb7846e6a2d89',
       '99b96dceb6594706756274423867ae80'], dtype=object)
```

In [34]:
```
from cycler import cycler
plt.figure(figsize=(12,9))
plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
for i,jobId in enumerate(jobIds):
    job = project.open_job(id=jobId)
    print(job.sp.temp_prof)
    #if job.sp.temp_prof[-1][0] == 2995000.0:
    if 'gel_point' in job.document:
        print(job.sp.profile,job,'gel point',job.document['gel_point'])
    else:
        print(job.sp.profile,job)
    if job.isfile('diffract_type_2/asq.txt'):
        data=np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
        legend=names[job.sp.profile]#'N={}'.format(job.sp.n_particles)
        q = data[:,0]/1.06
        #print(data[:,0])
        #print(q)
        plt.plot(q,data[:,1],label=legend)
    else:
        print('can\'t find',job.fn('diffract_type_2/asq.txt'))
plt.xlabel(r"$q$ [$nm^{-1}$]",fontsize=40)
plt.ylabel("log(Intensity) [Arb]",fontsize=40)
plt.xlim(-0.01,1)
plt.legend(fontsize=40)
#plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/struct
ure_factor.png',transparent=True)
```

```
[[5000.0, 1.1380096120843874], [4995000.0, 1.1380096120843874]]
iso 0c05a756d9ba03f975c52d022097d915 gel point 250000.0
[[5000.0, 0.6828057672506325], [4995000.0, 1.1380096120843874]]
lin_ramp 30d533afc4e66be8845bb7846e6a2d89 gel point 1000000.0
[[5000.0, 1.1380096120843874], [7005000.0, 1.1380096120843874]]
iso 99b96dceb65947067562744238 67ae80
can't find /Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper
/data/Diffusivity/large_system/workspace/99b96dceb65947067562744238 67ae80/diffract_typ
e_2/asq.txt
```

Out[34]: <matplotlib.legend.Legend at 0x1180a3710>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [4]:
```
from cme_utils.job_control import signac_helpers as diff
diff_keys = diff.diff_jobs(data_path,'1db14da7ad7f9f6dd8ac4f1cd8818afc','d22fc36554f63ee
299afe1c9ec9fc04a')
```

```
temp_prof
 1db14da7ad7f9f6dd8ac4f1cd8818afc : [[5000.0, 1.1380096120843874], [9995000.0,
1.1380096120843874]]
```

```
 d22fc36554f63ee299afe1c9ec9fc04a : [[5000.0, 1.1380096120843874], [2995000.0,
1.1380096120843874]]
n_dt
 1db14da7ad7f9f6dd8ac4f1cd8818afc : None
 d22fc36554f63ee299afe1c9ec9fc04a : 2995000.0
```

In [36]:
```
import gsd
import gsd.fl
import gsd.hoomd
from freud import box, density
import signac
import matplotlib.pyplot as plt
import numpy as np
from epoxpy.utils import RDF
%matplotlib inline
import hoomd.deprecated
import hoomd.dump
import hoomd
import os
from cme_utils.analyze import diffractometer

def diffract_frame(job,frameid, typeId=2):
    f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
    t = gsd.hoomd.HOOMDTrajectory(f)
    n_frames = len(t)
    if frameid < n_frames and frameid > 0:
        snapshot = t[frameid]
        sim_box = snapshot.configuration.box
        box_dim=(sim_box[0], sim_box[1], sim_box[2])
        box_dim = np.array(box_dim)
        print('box_dim',box_dim)
        l_pos = snapshot.particles.position
        pos = l_pos[np.where(snapshot.particles.typeid == typeId)]
        diffract_dir = job.fn('diffract_atframe{}_type_{}'.format(frameid,typeId))
        if not os.path.exists(diffract_dir):
                os.makedirs(diffract_dir)
        D = diffractometer.diffractometer(working_dir=diffract_dir)
        D.set(grid_size=512,  peak_width=1, zoom=8, n_views=20,
length_scale=1.0,bot=1e-5,top=1)
        D.load(pos,box_dim)
        D.prep_matrices()
        D.average()
    else:
        print('INVALID FRAME ID GIVEN')


def getFrameAtCurePercent(jobid,cure_p=40):
    thisjob = project.open_job(id=jobid)
    print(thisjob,thisjob.sp.profile)
    data = np.genfromtxt(thisjob.fn('out.log'))
    for i,cure_p in enumerate(data[:,9]):
        if cure_p>=40:
            print('time step:',round(data[i,0]/thisjob.sp.dcd_write))
            return int(round(data[i,0]/thisjob.sp.dcd_write))
```

In [37]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/ttt/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
%matplotlib inline

matplotlib.rc('xtick', labelsize=40)
matplotlib.rc('ytick', labelsize=40)
matplotlib.rc('axes', labelsize=20)
matplotlib.rc('lines', linewidth=4)

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
```

```
        linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

        project = signac.get_project(data_path)
        df_index = pd.DataFrame(project.index())
        df_index = df_index.set_index(['_id'])
        statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
        df = pd.DataFrame(statepoints).T.join(df_index)
        df_Ea1=df[(df.activation_energy==5)&
                  (df.kT*df.calibrationT==850.0)&
                  (df.trial==4)]#&
                  #(df.n_mul==10000)]#&
                  #(df.n_dt == 2995000)]
        df_Ea1 = df_Ea1.sort_values('n_particles')
        jobIds=np.asarray(df_Ea1.signac_id)
        jobIds
```

```
Out[37]: array(['3cc39c7909fc46d5db16a7f9286e4f24',
                '416dccf84bce41a29c489a567224e6b7',
                '4f8c0106906f219e44fa45cd734a989c',
                '9e8bffd84a3fa9c262969f00552f4d08',
                'bcad31cdcd84057607210fc142267d4d',
                'e8f383f7fea96439885b68e247a5298e'], dtype=object)
```

```
In [40]: from cme_utils.job_control import signac_helpers as diff
         diff_keys = diff.diff_jobs(data_path,'9e8bffd84a3fa9c262969f00552f4d08','4f8c0106906f219
         e44fa45cd734a989c')
```

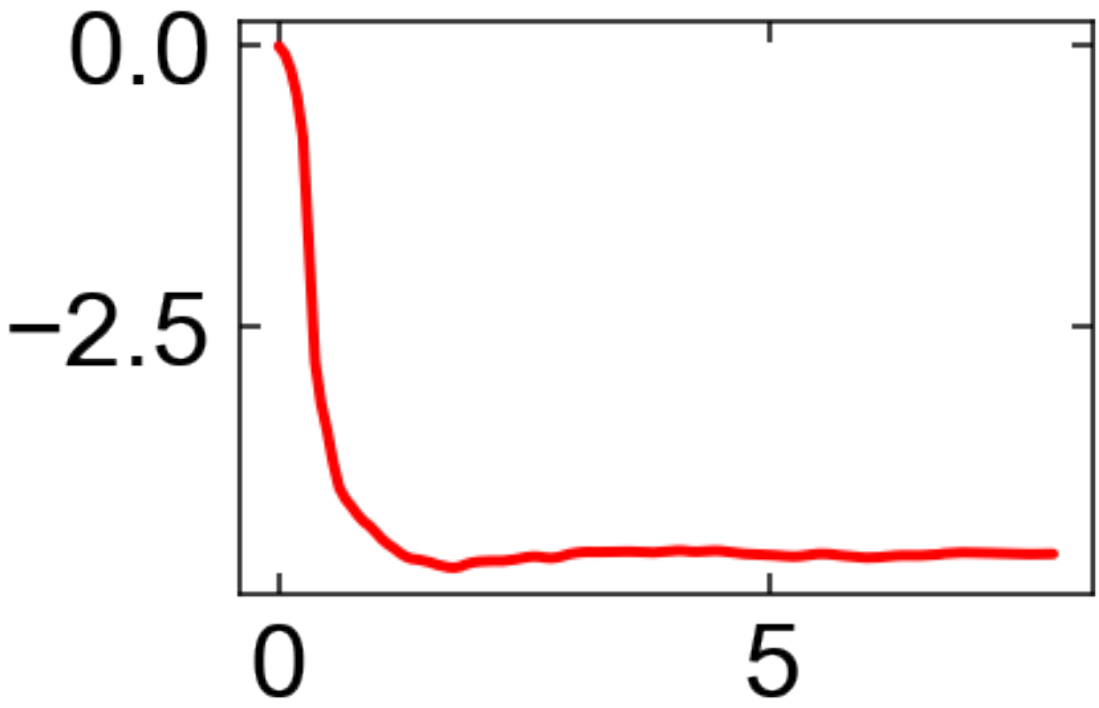
```
sim_name
 9e8bffd84a3fa9c262969f00552f4d08 : blend_lin_ramp_1.9346163405434587kT
 4f8c0106906f219e44fa45cd734a989c : blend_iso_1.9346163405434587kT
profile
 9e8bffd84a3fa9c262969f00552f4d08 : lin_ramp
 4f8c0106906f219e44fa45cd734a989c : iso
temp_prof
 9e8bffd84a3fa9c262969f00552f4d08 : [[5000.0, 0.6828057672506325], [4995000.0,
1.9346163405434587]]
 4f8c0106906f219e44fa45cd734a989c : [[5000.0, 1.9346163405434587], [4995000.0,
1.9346163405434587]]
```

```
In [45]: typeId=2
         for i,jobId in enumerate(jobIds):
             job = project.open_job(id=jobId)
             if job.sp.temp_prof[-1][0] == 4995000.0:
                 print(job,job.sp.temp_prof)
                 hoomd.context.initialize('--mode=cpu')
                 frame = getFrameAtCurePercent(jobId,40)
                 print('cured to 40 percent at frame:',frame)

                 system = hoomd.init.read_gsd(job.fn('data.gsd'),frame=frame)
                 #hoomd.dump.gsd(group=hoomd.group.all(), filename=job.fn('final.gsd'),
         overwrite=True, period=None)
                 group_c = hoomd.group.type(name='c-particles', type='C')
                 hoomd.deprecated.dump.xml(group=group_c,
         filename=job.fn('CsAtFrame{}.hoomdxml'.format(frame)), position=True)
                 diffract_frame(job,frame,typeId=typeId)
```

```
4f8c0106906f219e44fa45cd734a989c [[5000.0, 1.9346163405434587], [4995000.0,
1.9346163405434587]]
4f8c0106906f219e44fa45cd734a989c iso
time step: 11.0
cured to 40 percent at frame: 11
```

```
notice(2): Group "all" created containing 50000 particles
notice(2): Group "c-particles" created containing 20000 particles
box_dim [ 25.54364777  25.54364777  25.54364777]
Diffracting 10000 particles
1/23
2/23
3/23
4/23
5/23
6/23
7/23
8/23
9/23
10/23
11/23
12/23
13/23
14/23
15/23
16/23
17/23
18/23
19/23
20/23
21/23
22/23
23/23
29.19972586631775 seconds for  23 views.
1.2695532985355542 seconds per view
9e8bffd84a3fa9c262969f00552f4d08 [[5000.0, 0.6828057672506325], [4995000.0,
1.9346163405434587]]
9e8bffd84a3fa9c262969f00552f4d08 lin_ramp
time step: 66.0
cured to 40 percent at frame: 66
notice(2): Group "all" created containing 50000 particles
notice(2): Group "c-particles" created containing 20000 particles
box_dim [ 25.54364777  25.54364777  25.54364777]
Diffracting 10000 particles
1/23
2/23
3/23
4/23
5/23
6/23
7/23
8/23
9/23
10/23
11/23
12/23
13/23
14/23
15/23
16/23
17/23
18/23
19/23
20/23
21/23
```

```
22/23
23/23
30.08100390434265 seconds for  23 views.
1.3078697349714197 seconds per view


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



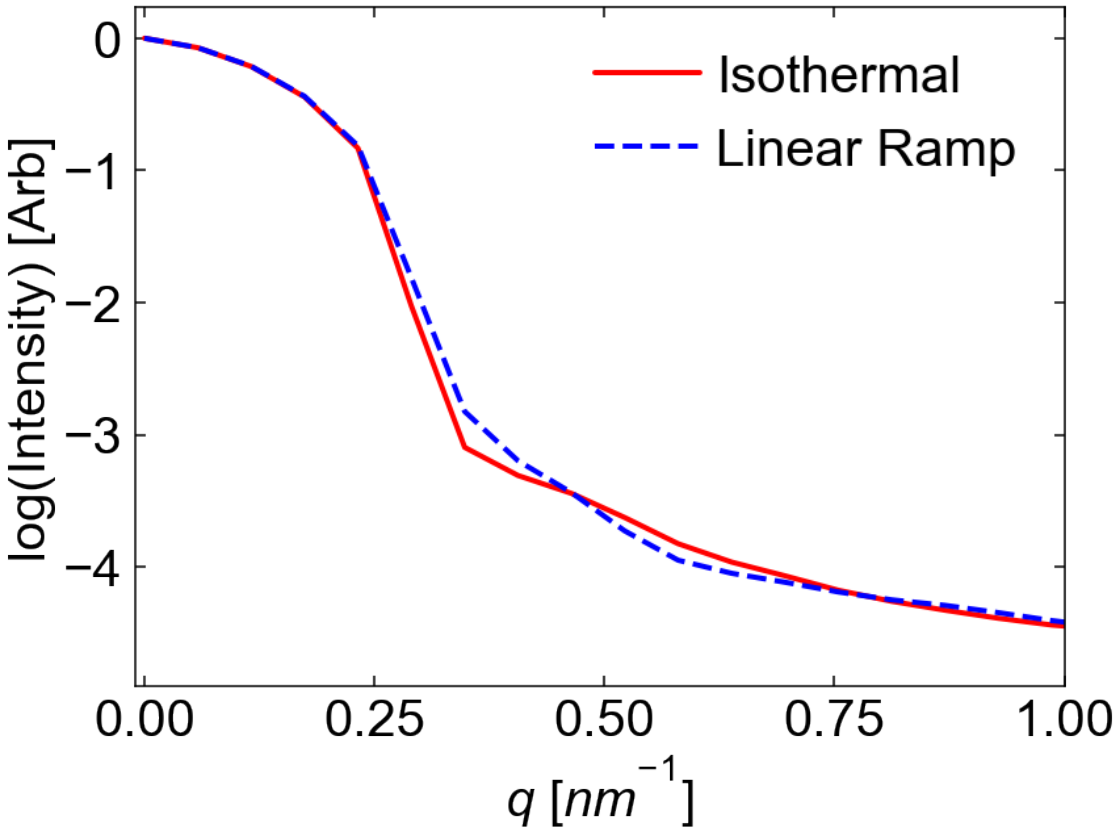

```
In [46]: from cycler import cycler
         plt.figure(figsize=(12,9))
         plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
                                    cycler('linestyle', ['-', '--', ':', '-.','--'])))
         typeId=2
         for i,jobId in enumerate(jobIds):
             job = project.open_job(id=jobId)
             print(job.sp.temp_prof)
             if job.sp.temp_prof[-1][0] == 4995000.0:
                 frame = getFrameAtCurePercent(jobId,40)
                 if 'gel_point' in job.document:
                     print(job.sp.profile,job,'gel point',job.document['gel_point'],'diffracting
         @ frame',frame)
                 else:
                     print(job.sp.profile,job)
                 if job.isfile('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)):
         data=np.genfromtxt(job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
                     legend=names[job.sp.profile]#'N={}'.format(job.sp.n_particles)
                     q = data[:,0]/1.06
                     #print(data[:,0])
```

```
                #print(q)
                plt.plot(q,data[:,1],label=legend)
            else:
                print('can\'t
        find',job.fn('diffract_atframe{}_type_{}/asq.txt'.format(frame,typeId)))
        plt.xlabel(r"$q$ [$nm^{-1}$]",fontsize=40)
        plt.ylabel("log(Intensity) [Arb]",fontsize=40)
        plt.xlim(-0.01,1)
        plt.legend(fontsize=40)
        #plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/struct
        ure_factor.png',transparent=True)
```

```
[[5000.0, 0.6828057672506325], [755000.0, 1.9346163405434587], [5745000.0,
1.9346163405434587]]
[[5000.0, 0.6828057672506325], [9995000.0, 1.9346163405434587]]
[[5000.0, 1.9346163405434587], [4995000.0, 1.9346163405434587]]
4f8c0106906f219e44fa45cd734a989c iso
time step: 11.0
iso 4f8c0106906f219e44fa45cd734a989c gel point 750000.0 diffracting @ frame 11
[[5000.0, 0.6828057672506325], [4995000.0, 1.9346163405434587]]
9e8bffd84a3fa9c262969f00552f4d08 lin_ramp
time step: 66.0
lin_ramp 9e8bffd84a3fa9c262969f00552f4d08 gel point 3600000.0 diffracting @ frame 66
[[5000.0, 1.9346163405434587], [9995000.0, 1.9346163405434587]]
[[5000.0, 0.6828057672506325], [755000.0, 1.9346163405434587], [10745000.0,
1.9346163405434587]]
```

Out[46]: <matplotlib.legend.Legend at 0x11b5cdf98>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


0
−1
−2
−3
−4
log(Intensity) [Arb]
Isothermal
Linear Ramp
0.00
0.25
0.50
0.75
1.00
q [nm^−1]

In [1]:
```
## import os

import math

import gsd
import gsd.fl
import gsd.hoomd
from cme_utils.analyze import autocorr

def get_all_maximas(job,q,intensities):
    half_box_length = job.document['Lx']*0.5
    q_half_length = 2*math.pi/(half_box_length)
    peaks_q = []
    peaks_I = []
 #   print(q)
    maxima_i = argex(intensities,np.greater)[0]
    for i in maxima_i:
        if q[i] > q_half_length:
  #          print(i)
            peaks_q.append(q[i])
            peaks_I.append(intensities[i])
  #  print(peaks_q)
    return peaks_q,peaks_I

def get_highest_maxima(job,q,intensities):
    peaks_q,peaks_I = get_all_maximas(job,q,intensities)
    largest_peak_I = np.max(peaks_I)
    index_largest_I = peaks_I.index(largest_peak_I)
    largest_peak_q = peaks_q[index_largest_I]
    return largest_peak_q,largest_peak_I

def get_nth_maxima(job,q,intensities,n=1):
    '''
    Use 'n'=1 for first maxima and 'n'=2 for second maxima etc..
    '''
    peaks_q,peaks_I = get_all_maximas(job,q,intensities)
    sorted_peaks_I = np.sort(peaks_I)
    #print(peaks_q,peaks_I,sorted_peaks_I)
    nth_largest_peak_I = sorted_peaks_I[n*-1]
    index_nth_largest_I = peaks_I.index(nth_largest_peak_I)
    nth_largest_peak_q = peaks_q[index_nth_largest_I]
    return nth_largest_peak_q,nth_largest_peak_I

def save_frame(job,frame):
    with gsd.hoomd.open('Frame{}.gsd'.format(frame), 'wb') as t_new:
        f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
        t = gsd.hoomd.HOOMDTrajectory(f)
        snap = t[frame]
        t_new.append(snap)
    #hoomd.deprecated.dump.xml(group=hoomd.group.all(),
filename=job.fn('Frame{}.hoomdxml'.format(frame)), position=True)

from mpl_toolkits.mplot3d import axes3d

class MyAxes3D(axes3d.Axes3D):

    def __init__(self, baseObject, sides_to_draw):
        self.__class__ = type(baseObject.__class__.__name__,
                              (self.__class__, baseObject.__class__),
                              {})
        self.__dict__ = baseObject.__dict__
        self.sides_to_draw = list(sides_to_draw)
        self.mouse_init()

    def set_some_features_visibility(self, visible):
        for t in self.w_zaxis.get_ticklines() + self.w_zaxis.get_ticklabels():
            t.set_visible(visible)
        self.w_zaxis.line.set_visible(visible)
```

```
        self.w_zaxis.pane.set_visible(visible)
        self.w_zaxis.label.set_visible(visible)

    def draw(self, renderer):
        # set visibility of some features False
        self.set_some_features_visibility(False)
        # draw the axes
        super(MyAxes3D, self).draw(renderer)
        # set visibility of some features True.
        # This could be adapted to set your features to desired visibility,
        # e.g. storing the previous values and restoring the values
        self.set_some_features_visibility(True)

        zaxis = self.zaxis
        draw_grid_old = zaxis.axes._draw_grid
        # disable draw grid
        zaxis.axes._draw_grid = False

        tmp_planes = zaxis._PLANES

        if 'l' in self.sides_to_draw :
            # draw zaxis on the left side
            zaxis._PLANES = (tmp_planes[2], tmp_planes[3],
                             tmp_planes[0], tmp_planes[1],
                             tmp_planes[4], tmp_planes[5])
            zaxis.draw(renderer)
        if 'r' in self.sides_to_draw :
            # draw zaxis on the right side
            zaxis._PLANES = (tmp_planes[3], tmp_planes[2],
                             tmp_planes[1], tmp_planes[0],
                             tmp_planes[4], tmp_planes[5])
            zaxis.draw(renderer)

        zaxis._PLANES = tmp_planes

        # disable draw grid
        zaxis.axes._draw_grid = draw_grid_old

def _get_decorrelation_time(prop_values,
                            time_steps):
    t = time_steps - time_steps[0]
    dt = t[1] - t[0]
    acorr = autocorr.autocorr1D(prop_values)
    for acorr_i in range(len(acorr)):
        if acorr[acorr_i]<0:
            break
    lags = [i*dt for i in range(len(acorr))]

    decorrelation_time = int(lags[acorr_i])
    if decorrelation_time == 0:
        decorrelation_time = 1
    decorrelation_stride = int(decorrelation_time/dt)
    nsamples = (int(t[-1])-t[0])/decorrelation_time
    temps = "There are %.5e steps, (" % t[-1]
    temps = temps + "%d" % int(t[-1])
    temps = temps + " frames)\n"
    temps = temps + "You can start sampling at t=%.5e" % t[0]
    temps = temps + " (frame %d)" % int(t[0] )
    temps = temps + " for %d samples\n" % nsamples
    temps = temps + "Because the autocorrelation time is %.5e" % lags[acorr_i]
    temps = temps + " (%d frames)\n" % int(lags[acorr_i])
    #print(temps)
    return decorrelation_time, decorrelation_stride

def plot_equilibriation(df_filtered,prop_name):
    df_sorted = df_filtered.sort_values(by=['T'])
    df_sorted.index
```

```
    quenchTs=[]
    mean_vols=[]
    vol_stds=[]

    for job_id in df_sorted.index:
        job = project.open_job(id=job_id)
        #print(job)
        if job.isfile('out.log'):
            log_path = job.fn('out.log')
            data = np.genfromtxt(log_path, names=True)
            PROP_NAME =prop_name
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            time_steps = data['timestep']
            start_i, start_t = autocorr.find_equilibrated_window(time_steps,
data['potential_energy'])
            decorrelation_time, decorrelation_stride =
_get_decorrelation_time(data['potential_energy'][start_i:], time_steps[start_i:])
            print('decorrelation_stride:',decorrelation_stride)
            print('decorrelation_time:',decorrelation_time)
            print('start_i:',start_i)
            print('start_t:',start_t)
            independent_vals_i = np.arange(start_i, len(prop_values)-1,
decorrelation_stride)
            independent_vals = time_steps[independent_vals_i]
            #starttime_steps.index(start_t)
            for xval in independent_vals:
                plt.axvline(x=xval,linestyle='--',linewidth=0.2)
            if 'quench_T' in job.sp:
                label =
'q_T:{},cure:{}'.format(job.sp.quench_T,job.sp.stop_after_percent)
            else:
                label = 'kT:{},cure:{}'.format(job.sp.kT,job.sp.stop_after_percent)
            plt.plot(time_steps,prop_values,'o',label=label)
plt.plot(time_steps[start_i],prop_values[start_i],marker='*',color='r',markersize=20)
            #print(time_steps)
            #decorr_i = np.where(time_steps >= decor_time)[0][0]
            #print(decorr_
```

In [4]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/system_size_dependence_of_sq/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df_Ea1=df[(df.activation_energy==1)&
          (df['T']==500.0)&
          (df.n_particles==3e6)&
          (df.t_Final==1e7)]
df_Ea1 = df_Ea1.sort_values('n_particles')
jobIds=np.asarray(df_Ea1.signac_id)
```

```
jobIds
```

Out[4]: array(['9200af25cc3cbf69a58ea3438e66c271'], dtype=object)

1 2D

```
In [9]: import os
        #import structure_factor as sf
        import math

        import gsd
        import gsd.fl
        import gsd.hoomd

        from matplotlib import cm
        from mpl_toolkits.mplot3d import Axes3D
        fig = plt.figure()
        ax = fig.gca()
        #ax3 = ax1.twinx()
        typeId=2
        n_views=40
        grid_size=512

        colors = itertools.cycle(cm.rainbow(np.linspace(0, 1, 10)))

        df_filtered = df[(df.activation_energy==1)&
                         (df['T']==500.0)&
                         (df.n_particles==3.0e6)&
                         (df.t_Final==1e7)]


        #print(df_filtered)
        #grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
        times_for_all_trials=[]
        qs_for_all_trials=[]
        Is_for_all_trials=[]

        qms_all=[] #qmax
        Is_all=[]
        Qs_all=[]
        times_all=[]
        cure_all=[]
        color=next(colors)
        for signac_id in df_filtered['signac_id']:
            job = project.open_job(id=signac_id)
            print(job.workspace())
            if 'Lx' in job.document:
                half_box_length = job.document['Lx']/2
            else:
                print('Lx not found in',job)
            q_half_length = 2*math.pi/(half_box_length/1.06)
            diffract_dir_pattern
        ='diffract_type_{}_n_views_{}_grid_size_{}_frame'.format(typeId,
        n_views,
        grid_size)
            directories = os.listdir(job.workspace())
            directories = [d for d in os.listdir(job.workspace()) if
        d.startswith(diffract_dir_pattern)]
            directories.sort(key = lambda x: int(x.split('_')[-1]))
            print(len(directories))
            #print(directories)
            num_frames = len(directories)
            if num_frames > 0:
                qs_for_all_times=[]
                Is_for_all_times=[]
                times_for_all_times=[]
```

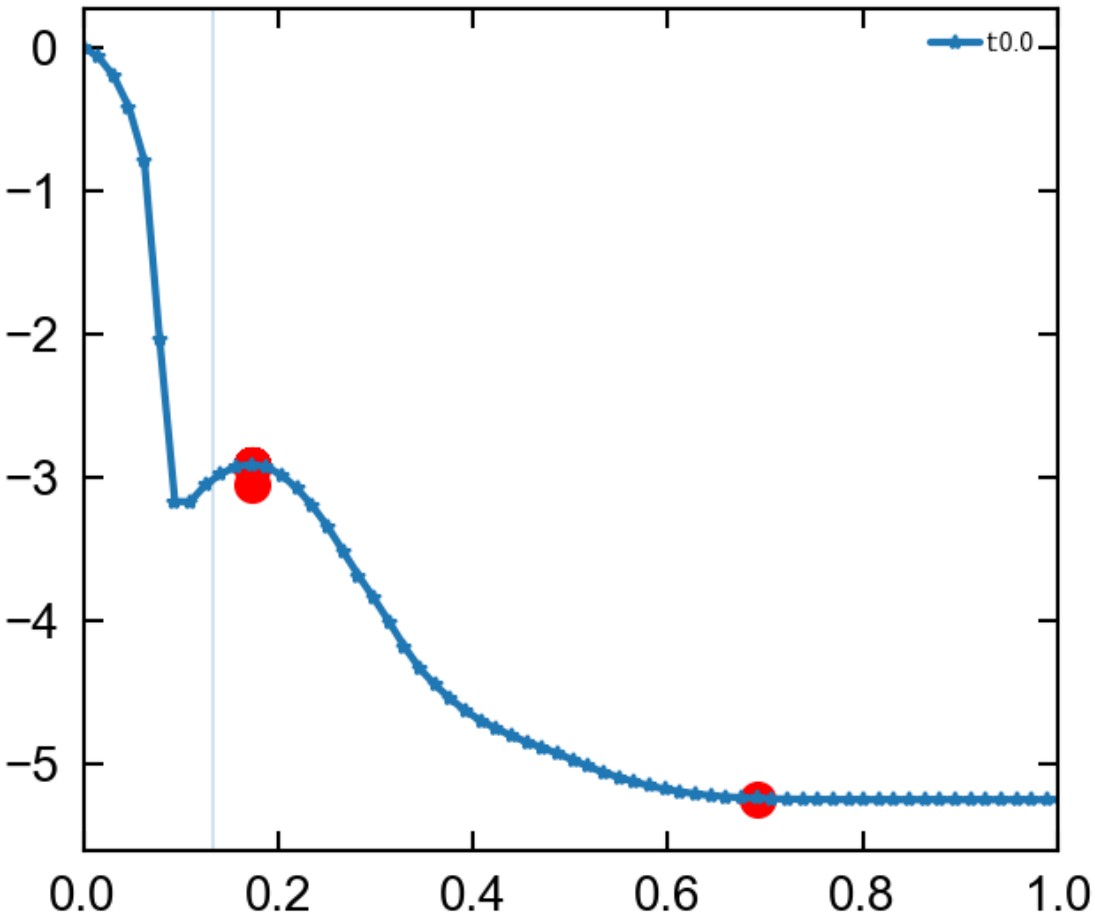
```
        qs_list = []
        times_list = []
        Is_list = []
        Qs_list=[]
        for i,diffract_dir in enumerate(directories):
            print("Progress {:2.1%}".format(i / num_frames), end="\r")

            #print(diffract_dir)

            if diffract_dir.startswith(diffract_dir_pattern):
                frame = int(diffract_dir.split('_')[-1])
                if frame%1==0:# and frame <3e6/job.sp.dcd_write:#==119 or
frame==123:#%100 == 0:#num_frames/30:
                    if job.isfile('{}/asq.txt'.format(diffract_dir)):
                        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                        time = round(frame*job.sp.dcd_write)/1e6
                        legend = '{} $\\Delta t(\Gamma:{})$'.format(time,job.sp.gamma)
                        qs = data[:,0]
                        Is = data[:,1]
                        #print(qs.shape)
                        qs_for_all_times.append(qs)
                        Is_for_all_times.append(Is)
                        times_for_all_times.append(time)

                        dq=qs[1]-qs[0]
                        Is_exp = np.exp(Is)
                        q_sq = qs**2
                        Q = np.sum(Is_exp*qs*dq)
                        Qs_list.append(Q)
                        first_peak_q,first_peak_i = get_highest_maxima(job,qs,Is)
                        #if first_peak_q >0.8 and time > 2.0e5:
                        #    first_peak_q=q_half_length

                        qs_list.append(first_peak_q)
                        times_list.append(time)
                        Is_list.append(first_peak_i)
                    else:
                        print(job,'did not contain diffraction data in ',diffract_dir)
                else:
                    print(job,'directory {} is not as
expected:{}'.format(diffract_dir,diffract_dir_pattern))
        else:
            print(job,'did not contain diffraction data for time evolution')

        qs_for_all_trials.append(qs_for_all_times)
        Is_for_all_trials.append(Is_for_all_times)
        times_for_all_trials.append(times_for_all_times)

        qms_all.append(np.asarray(qs_list))
        Is_all.append(np.asarray(Is_list))
        Qs_all.append(np.asarray(Qs_list))
        log_data = np.genfromtxt(job.fn('out.log'))
        times = log_data[:,0]#/(job.sp.dt*job.sp.dcd_write)
        cure = log_data[:,9]
        times_all.append(times)
        cure_all.append(cure)

    #print(qs_all)
    qs_for_all_trials=np.asarray(qs_for_all_trials)
    Is_for_all_trials=np.asarray(Is_for_all_trials)
    times_for_all_trials=np.asarray(times_for_all_trials)
    q_mean = np.mean(qs_for_all_trials,axis=0)
    I_mean = np.mean(Is_for_all_trials,axis=0)
    time_mean= np.mean(times_for_all_trials,axis=0)

    qms_all = np.asarray(qms_all)
    Is_all = np.asarray(Is_all)
    Qs_all = np.asarray(Qs_all)
```

```
        times_all = np.asarray(times_all)
        cure_all = np.asarray(cure_all)
        Qs_av = np.mean(Qs_all,axis=0)
        Qs_std = np.std(Qs_all,axis=0)
        qs_av = np.mean(qms_all,axis=0)
        qs_std = np.std(qms_all,axis=0)
        Is_av = np.mean(Is_all,axis=0)
        times_av = np.mean(times_all,axis=0)
        cure_av = np.mean(cure_all,axis=0)
        cure_std = np.std(cure_all,axis=0)
            #print(len(times_list),len(qs_av))
        for i,q in enumerate(q_mean):
            if i==len(q_mean)-1:
                I=I_mean[i]
                time=time_mean[i]
                legend='t:{}'.format(round(time/1e6,2))
                ax.plot(q,I,label=legend,marker='*')
                ax.axvline(x=q_half_length,linewidth=0.3)
        ax.scatter(qs_av, Is_av,color='r',s=200)
        ax.set_xlim(0,1.0)
        ax.legend(fontsize=10)
```

```
/Users/stephenthomas/Google Drive/Research/2017/Papers/Epoxpy_methods_paper/data/syste
m_size_dependence_of_sq/workspace/9200af25cc3cbf69a58ea3438e66c271
30
Progress 96.7%
```

Out[9]: <matplotlib.legend.Legend at 0x1151caba8>

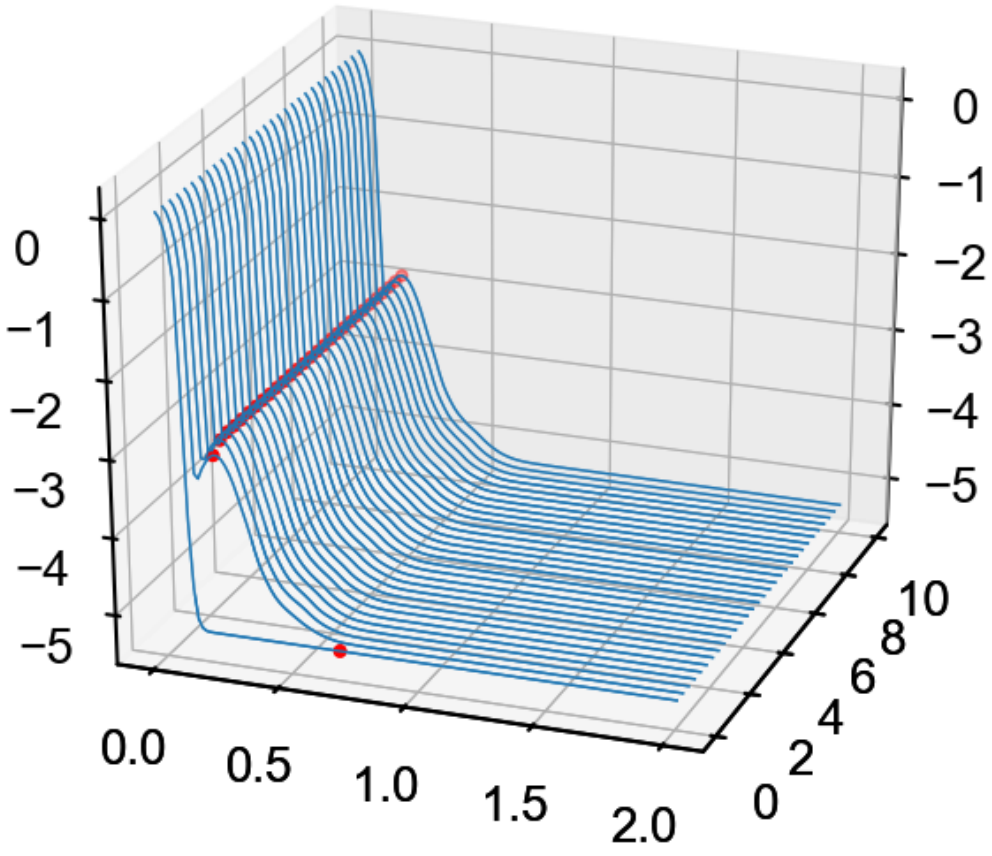
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [8]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#ax.set_title('z-axis left side')
ax = fig.add_axes(MyAxes3D(ax, 'l'))


q = q_mean[0]#[:50]
I=I_mean#[:,:50]
#print(Is_av)
#print(I)
#X,Y = np.meshgrid(q_mean[0],time_mean)
X,Y = np.meshgrid(q,time_mean)
#print(X.shape)
#Z=I_mean
Z=I
#matplotlib.rcParams['xtick.labelsize'] = 15
#matplotlib.rcParams['ytick.labelsize'] = 15
#matplotlib.rcParams['xtick.major.pad'] = 2
#matplotlib.rcParams['ytick.major.pad'] = 2
#matplotlib.rcParams['ztick.major.pad'] = 1
#surf = ax.plot_surface(X, Y, Z, cmap=cm.plasma,)#,rstride=1, cstride=1,linewidth=1,
antialiased=True)
surf = ax.plot_wireframe(X, Y, Z,
                         linewidth=1.0,
                         zorder=0.2,
                         antialiased=True,
```

```
                                #rstride=0,
                                cstride=0)

        view_1 = (0, 180)#back
        view_2 = (25, -70)#angled
        view_3 = (25, 0)#front from top
        view_4 = (-5, 90)#right
        init_view = view_2
        ax.view_init(*init_view)
        #ax.set_xlabel(r"$q$ [$nm^{-1}$]", rotation=150)
        #ax.set_ylabel(r"$Time[\tau]$",fontsize=10, rotation=150)
        #ax.set_zlabel(r"$log(I)$ [Arb]", rotation=90)
        #ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0),
        useMathText=True,rotation=0)
        ax.xaxis._axinfo['label']['space_factor'] = 1.0
        ax.yaxis._axinfo['label']['space_factor'] = 1.0
        ax.zaxis._axinfo['label']['space_factor'] = 1.0
        ax.scatter3D(qs_av, time_mean, Is_av,color='r',zorder=0.5,marker='o',antialiased=True)

        plt.savefig('3D_q_plot',transparent=True)

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [46]:
```
import os
#import structure_factor as sf
import math

import gsd
import gsd.fl
import gsd.hoomd

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.gca()
#ax3 = ax1.twinx()
typeId=2
n_views=40
grid_size=512

colors = itertools.cycle(cm.rainbow(np.linspace(0, 1, len(gammas))))

df_filtered = df[(df.activation_energy==1)&
              (df['T']==500.0)&
              (df.n_particles==2.0e6)&
              (df.t_Final==1e7)]


#print(df_filtered)
#grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
times_for_all_trials=[]
qs_for_all_trials=[]
Is_for_all_trials=[]

qms_all=[] #qmax
Is_all=[]
Qs_all=[]
times_all=[]
cure_all=[]
color=next(colors)
mean_qs=[]
mean_Is=[]
std_qs=[]
std_Is=[]

for signac_id in df_filtered['signac_id']:
    job = project.open_job(id=signac_id)
    #print(job.workspace())
    if 'Lx' in job.document:
        half_box_length = job.document['Lx']/2
    else:
        print('Lx not found in',job)
    q_half_length = 2*math.pi/(half_box_length/1.06)
    diffract_dir_pattern
='diffract_type_{}_n_views_{}_grid_size_{}_frame'.format(typeId,
n_views,
grid_size)
    directories = os.listdir(job.workspace())
    directories = [d for d in os.listdir(job.workspace()) if
d.startswith(diffract_dir_pattern)]
    directories.sort(key = lambda x: int(x.split('_')[-1]))
    print(len(directories))
    #print(directories)
    num_frames = len(directories)

    log_path = job.fn('out.log')
    data = np.genfromtxt(log_path, names=True)
    time_steps = data['timestep']
```

```
    start_i, start_t = autocorr.find_equilibrated_window(time_steps,
data['potential_energy'])
    decorrelation_time, decorrelation_stride =
_get_decorrelation_time(data['potential_energy'][start_i:], time_steps[start_i:])
    print('decorrelation_stride:',decorrelation_stride)
    print('decorrelation_time:',decorrelation_time)
    print('start_i:',start_i)
    print('start_t:',start_t)

    if num_frames > 0:
        qs_for_all_times=[]
        Is_for_all_times=[]
        times_for_all_times=[]
        qs_list = []
        times_list = []
        Is_list = []
        Qs_list=[]
        for i,diffract_dir in enumerate(directories):
            print("Progress {:2.1%}".format(i / num_frames), end="\r")

            #print(diffract_dir)

            if diffract_dir.startswith(diffract_dir_pattern):
                frame = int(diffract_dir.split('_')[-1])

                decorrelated_frame_stride = int(decorrelation_time/job.sp.dcd_write)
                decorrelated_frame_stride = max(decorrelated_frame_stride,1)
                time = round(frame*job.sp.dcd_write)
                print('decorrelated frame stride is:',decorrelated_frame_stride)
                print('Equilibriated after time:',start_t)
                if time >= start_t and frame%decorrelated_frame_stride==0:# and frame
<3e6/job.sp.dcd_write:#==119 or frame==123:#%100 == 0:#num_frames/30:
                    if job.isfile('{}/asq.txt'.format(diffract_dir)):
                        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))

                        legend = '{} $\\Delta t(\Gamma:{})$'.format(time,job.sp.gamma)
                        qs = data[:,0]
                        Is = data[:,1]
                        #print(qs.shape)
                        qs_for_all_times.append(qs)
                        Is_for_all_times.append(Is)
                        times_for_all_times.append(time)

                        dq=qs[1]-qs[0]
                        Is_exp = np.exp(Is)
                        q_sq = qs**2
                        Q = np.sum(Is_exp*qs*dq)
                        Qs_list.append(Q)
                        first_peak_q,first_peak_i = get_highest_maxima(job,qs,Is)
                        #if first_peak_q >0.8 and time > 2.0e5:
                        #    first_peak_q=q_half_length

                        qs_list.append(first_peak_q)
                        times_list.append(time)
                        Is_list.append(first_peak_i)
                    else:
                        print(job,'did not contain diffraction data in ',diffract_dir)
                else:
                    print(job,'directory {} is not as
expected:{}'.format(diffract_dir,diffract_dir_pattern))
    else:
        print(job,'did not contain diffraction data for time evolution')
    mean_qs.append(np.mean(qs_for_all_times,axis=0))
    std_qs.append(np.std(qs_for_all_times,axis=0))
    mean_Is.append(np.mean(Is_for_all_times,axis=0))
    std_Is.append(np.std(Is_for_all_times,axis=0))

    qs_for_all_trials.append(qs_for_all_times)
```

```
            Is_for_all_trials.append(Is_for_all_times)
            times_for_all_trials.append(times_for_all_times)

            qms_all.append(np.asarray(qs_list))
            Is_all.append(np.asarray(Is_list))
            Qs_all.append(np.asarray(Qs_list))
            log_data = np.genfromtxt(job.fn('out.log'))
            times = log_data[:,0]#/(job.sp.dt*job.sp.dcd_write)
            cure = log_data[:,9]
            times_all.append(times)
            cure_all.append(cure)

        #print(qs_all)
        qs_for_all_trials=np.asarray(qs_for_all_trials)
        Is_for_all_trials=np.asarray(Is_for_all_trials)
        times_for_all_trials=np.asarray(times_for_all_trials)
        q_mean = np.mean(qs_for_all_trials,axis=0)
        I_mean = np.mean(Is_for_all_trials,axis=0)
        time_mean= np.mean(times_for_all_trials,axis=0)

        qms_all = np.asarray(qms_all)
        Is_all = np.asarray(Is_all)
        Qs_all = np.asarray(Qs_all)
        times_all = np.asarray(times_all)
        cure_all = np.asarray(cure_all)
        Qs_av = np.mean(Qs_all,axis=0)
        Qs_std = np.std(Qs_all,axis=0)
        qs_av = np.mean(qms_all,axis=0)
        qs_std = np.std(qms_all,axis=0)
        Is_av = np.mean(Is_all,axis=0)
        times_av = np.mean(times_all,axis=0)
        cure_av = np.mean(cure_all,axis=0)
        cure_std = np.std(cure_all,axis=0)
            #print(len(times_list),len(qs_av))
        for i,q in enumerate(mean_qs):
            I=mean_Is[i]
            I_std=std_Is[i]

            #time=time_mean[i]
            legend='N:{:.1e}'.format(2e6)
            ax.errorbar(q,I,I_std,
                        markersize=5,
                        capsize=4,
                        label=legend)
        ax.scatter(qs_av, Is_av,color='r',s=200)
        ax.set_xlim(0,1.20)
        ax.legend(fontsize=10)
```

```
34
decorrelation_stride: 6
decorrelation_time: 60000
start_i: 124
start_t: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_1 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_11 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
```

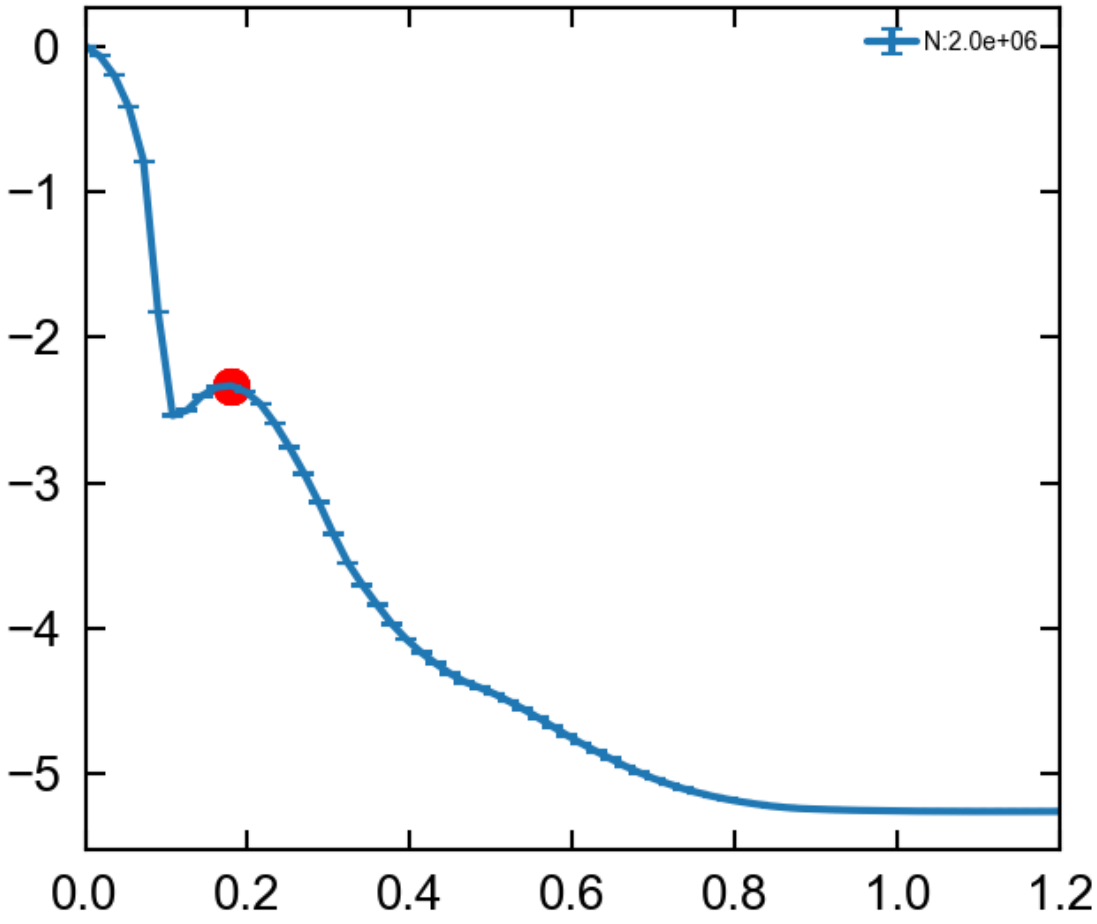
```
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_18 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_21 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_32 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_35 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_42 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_52 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_86 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_104 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_121 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_155 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_172 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
```

```
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_224 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_241 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_275 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_292 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_310 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_344 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_361 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_395 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_413 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_430 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
```

```
Equilibriated after time: 1250000.0
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_464 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_481 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
decorrelated frame stride is: 3
Equilibriated after time: 1250000.0
3d30a5818b965dd878a0f0059f3acd2a directory
diffract_type_2_n_views_40_grid_size_512_frame_499 is not as
expected:diffract_type_2_n_views_40_grid_size_512_frame
```

Out[46]: <matplotlib.legend.Legend at 0x121b1eba8>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

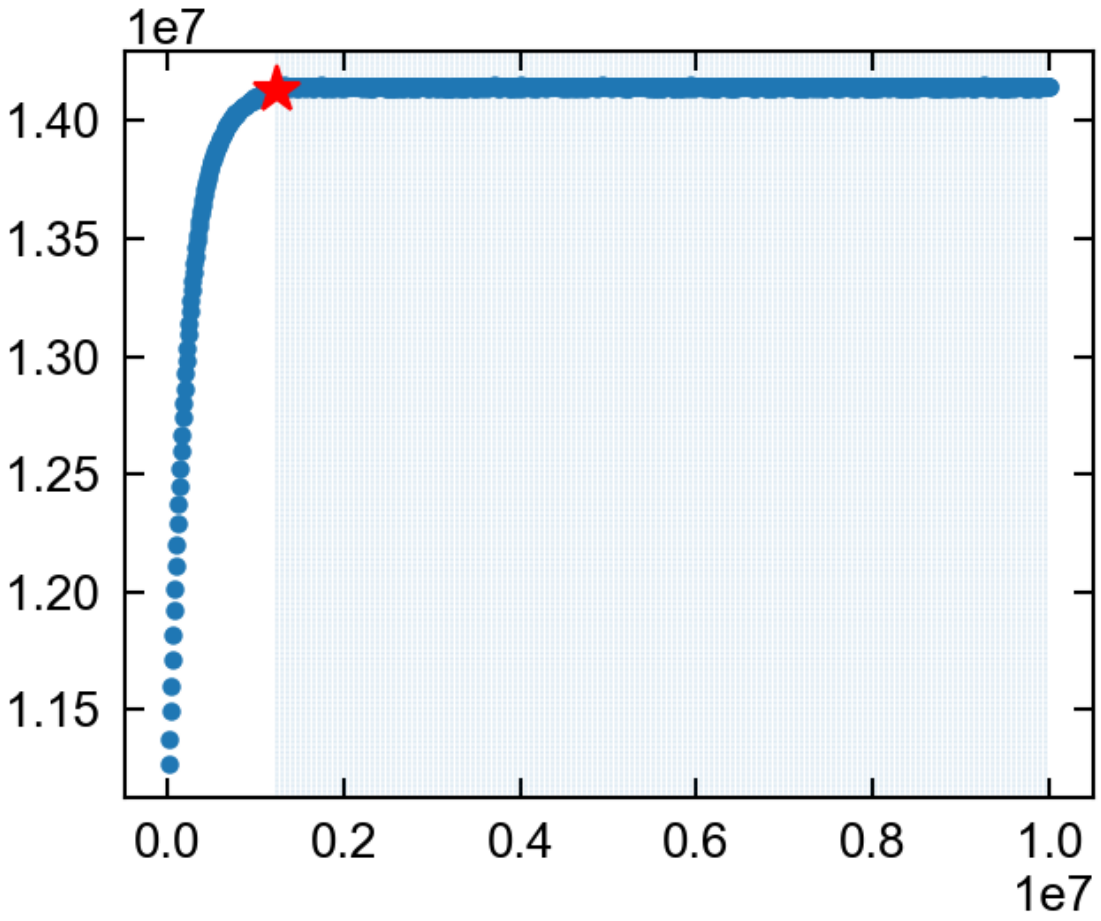


```
In [47]: df_filtered = df[(df.activation_energy==1)&
                          (df['T']==500.0)&
                          (df.n_particles==2.0e6)&
                          (df.t_Final==1e7)]
         plot_equilibriation(df_filtered,'potential_energy')
         df_filtered.dcd_write
```

```
decorrelation_stride: 6
decorrelation_time: 60000
start_i: 124
start_t: 1250000.0
```

```
Out[47]: 3d30a5818b965dd878a0f0059f3acd2a    20000
         Name: dcd_write, dtype: object
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [1]:
```
## import os

import math

import gsd
import gsd.fl
import gsd.hoomd
from scipy.signal import argrelextrema as argex
from cme_utils.analyze import autocorr

def get_all_maximas(job,q,intensities):
    half_box_length = job.document['Lx']*0.6
    q_half_length = 2*math.pi/(half_box_length)
    peaks_q = []
    peaks_I = []
 #   print(q)
    maxima_i = argex(intensities,np.greater)[0]
    if len(maxima_i)>0:
        for i in maxima_i:
            if q[i] >= q_half_length:
      #          print(i)
                peaks_q.append(q[i])
                peaks_I.append(intensities[i])
      #  print(peaks_q)
    else:
        print('WARNING: no maximas found for the structure factor. returning a default
intensity of 0 at 0')
        peaks_q.append(0.)
        peaks_I.append(0.)
    return peaks_q,peaks_I

def get_highest_maxima(job,q,intensities):
    peaks_q,peaks_I = get_all_maximas(job,q,intensities)
    largest_peak_I = np.max(peaks_I)
    index_largest_I = peaks_I.index(largest_peak_I)
    largest_peak_q = peaks_q[index_largest_I]
    return largest_peak_q,largest_peak_I

def get_nth_maxima(job,q,intensities,n=1):
    '''
    Use 'n'=1 for first maxima and 'n'=2 for second maxima etc..
    '''
    peaks_q,peaks_I = get_all_maximas(job,q,intensities)
    sorted_peaks_I = np.sort(peaks_I)
    #print(peaks_q,peaks_I,sorted_peaks_I)
    nth_largest_peak_I = sorted_peaks_I[n*-1]
    index_nth_largest_I = peaks_I.index(nth_largest_peak_I)
    nth_largest_peak_q = peaks_q[index_nth_largest_I]
    return nth_largest_peak_q,nth_largest_peak_I

def save_frame(job,frame):
    with gsd.hoomd.open('Frame{}.gsd'.format(frame), 'wb') as t_new:
        f = gsd.fl.GSDFile(job.fn('data.gsd'), 'rb')
        t = gsd.hoomd.HOOMDTrajectory(f)
        snap = t[frame]
        t_new.append(snap)
    #hoomd.deprecated.dump.xml(group=hoomd.group.all(),
filename=job.fn('Frame{}.hoomdxml'.format(frame)), position=True)

from mpl_toolkits.mplot3d import axes3d

class MyAxes3D(axes3d.Axes3D):

    def __init__(self, baseObject, sides_to_draw):
        self.__class__ = type(baseObject.__class__.__name__,
                              (self.__class__, baseObject.__class__),
                              {})
        self.__dict__ = baseObject.__dict__
```

```
        self.sides_to_draw = list(sides_to_draw)
        self.mouse_init()

    def set_some_features_visibility(self, visible):
        for t in self.w_zaxis.get_ticklines() + self.w_zaxis.get_ticklabels():
            t.set_visible(visible)
        self.w_zaxis.line.set_visible(visible)
        self.w_zaxis.pane.set_visible(visible)
        self.w_zaxis.label.set_visible(visible)

    def draw(self, renderer):
        # set visibility of some features False
        self.set_some_features_visibility(False)
        # draw the axes
        super(MyAxes3D, self).draw(renderer)
        # set visibility of some features True.
        # This could be adapted to set your features to desired visibility,
        # e.g. storing the previous values and restoring the values
        self.set_some_features_visibility(True)

        zaxis = self.zaxis
        draw_grid_old = zaxis.axes._draw_grid
        # disable draw grid
        zaxis.axes._draw_grid = False

        tmp_planes = zaxis._PLANES

        if 'l' in self.sides_to_draw :
            # draw zaxis on the left side
            zaxis._PLANES = (tmp_planes[2], tmp_planes[3],
                             tmp_planes[0], tmp_planes[1],
                             tmp_planes[4], tmp_planes[5])
            zaxis.draw(renderer)
        if 'r' in self.sides_to_draw :
            # draw zaxis on the right side
            zaxis._PLANES = (tmp_planes[3], tmp_planes[2],
                             tmp_planes[1], tmp_planes[0],
                             tmp_planes[4], tmp_planes[5])
            zaxis.draw(renderer)

        zaxis._PLANES = tmp_planes

        # disable draw grid
        zaxis.axes._draw_grid = draw_grid_old

def _get_decorrelation_time(prop_values,
                            time_steps):
    t = time_steps - time_steps[0]
    dt = t[1] - t[0]
    acorr = autocorr.autocorr1D(prop_values)
    for acorr_i in range(len(acorr)):
        if acorr[acorr_i]<0:
            break
    lags = [i*dt for i in range(len(acorr))]

    decorrelation_time = int(lags[acorr_i])
    if decorrelation_time == 0:
        decorrelation_time = 1
    decorrelation_stride = int(decorrelation_time/dt)
    nsamples = (int(t[-1])-t[0])/decorrelation_time
    temps = "There are %.5e steps, (" % t[-1]
    temps = temps + "%d" % int(t[-1])
    temps = temps + " frames)\n"
    temps = temps + "You can start sampling at t=%.5e" % t[0]
    temps = temps + " (frame %d)" % int(t[0] )
    temps = temps + " for %d samples\n" % nsamples
    temps = temps + "Because the autocorrelation time is %.5e" % lags[acorr_i]
    temps = temps + " (%d frames)\n" % int(lags[acorr_i])
```

```
            #print(temps)
            return decorrelation_time, decorrelation_stride
```

```
In [2]: data_path='/Users/stephenthomas/Google
        Drive/Research/2017/Papers/Epoxpy_methods_paper/data/system_size_dependence_of_sq/'

        import signac
        import matplotlib.pyplot as plt
        import numpy as np
        import matplotlib.gridspec as gridspec
        import pandas as pd
        import matplotlib
        %matplotlib inline


        names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
        colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
        markers={'iso':'s','lin_ramp':'P','step':'>'}
        linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

        project = signac.get_project(data_path)
        df_index = pd.DataFrame(project.index())
        df_index = df_index.set_index(['_id'])
        statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
        df = pd.DataFrame(statepoints).T.join(df_index)
        df_Ea1=df[(df.activation_energy==1)&
                  (df['T']==500.0)&
                  (df.trial==0)&
                  #(df.n_particles==2e6)&
                  (df.t_Final==1e7)]
        df_Ea1 = df_Ea1.sort_values('n_particles')
        jobIds=np.asarray(df_Ea1.signac_id)
```

```
In [4]: df_filtered = df[(df.activation_energy==1)&
                       (df['T']==500.0)&
                        #  (df.trial==0)&
                       (df.n_particles==3e6)&
                       (df.t_Final==1e7)]
        df_filtered = df_filtered.sort_values('n_particles')
        print(df_filtered.n_particles,df_filtered.Lx,df_filtered.trial)
```

```
575726af6f220c1321526f49c1536c19    3e+06
9200af25cc3cbf69a58ea3438e66c271    3e+06
98ae4e87fa0ab183e10468ee205e39dc    3e+06
9c848bf9e12b8ad8275717b8002d6547    3e+06
b5af97f7b979a7365834ca233eec1f4f    3e+06
Name: n_particles, dtype: object 575726af6f220c1321526f49c1536c19    100.0
9200af25cc3cbf69a58ea3438e66c271    100.0
98ae4e87fa0ab183e10468ee205e39dc    100.0
9c848bf9e12b8ad8275717b8002d6547    100.0
b5af97f7b979a7365834ca233eec1f4f    100.0
Name: Lx, dtype: float64 575726af6f220c1321526f49c1536c19    4
9200af25cc3cbf69a58ea3438e66c271    0
98ae4e87fa0ab183e10468ee205e39dc    3
9c848bf9e12b8ad8275717b8002d6547    1
b5af97f7b979a7365834ca233eec1f4f    2
Name: trial, dtype: object
```

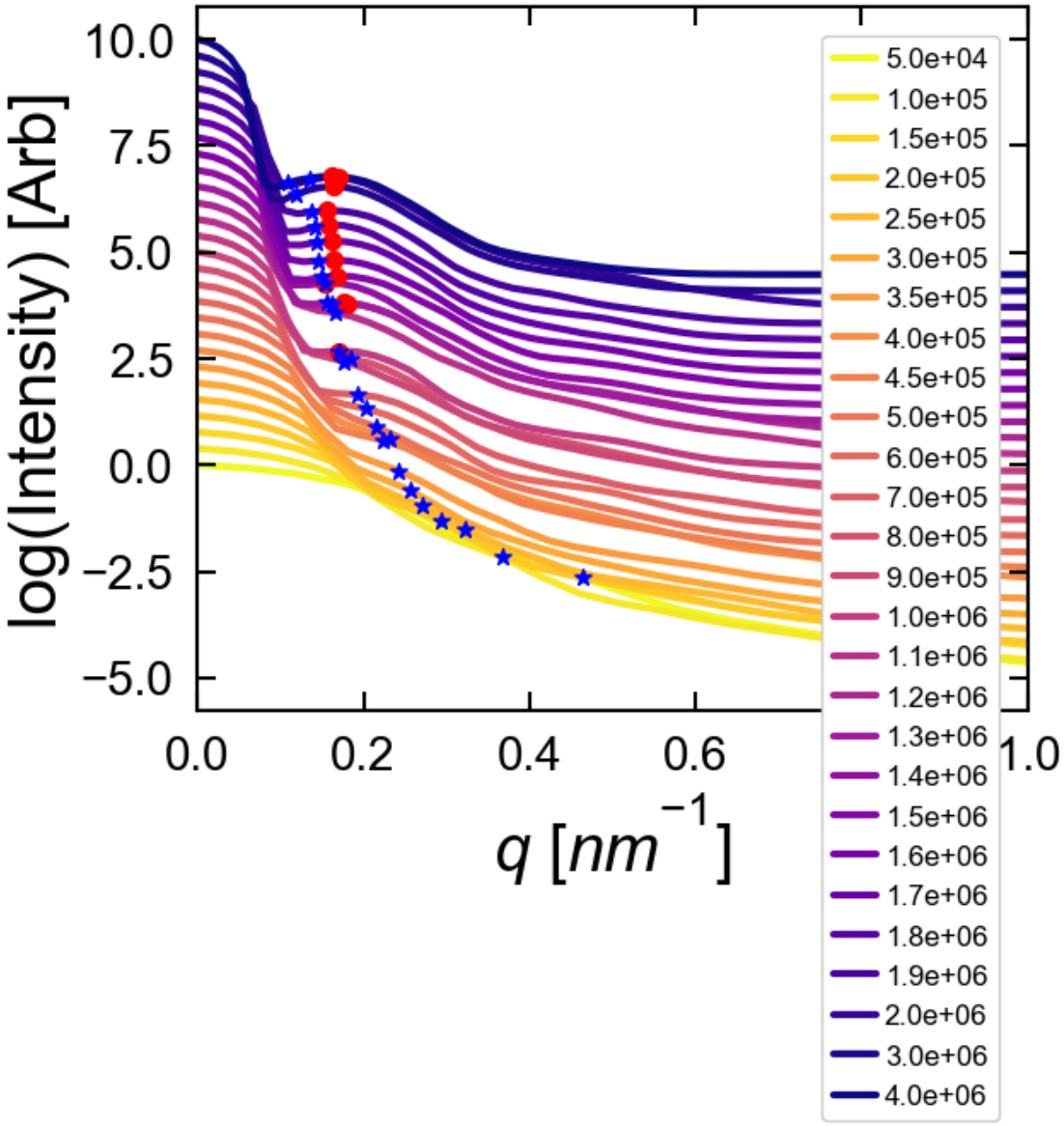
```
In [5]: from cycler import cycler
        from scipy import interpolate

        fig = plt.figure()
        ax = fig.gca()
```

```
#plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
#                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
Is_of_qm=[]
qs_of_qm=[]
df_filtered = df[(df.activation_energy==1)&
              (df['T']==500.0)&
                 (df.trial==0)&
              #(df.n_particles==1.4e6)&
              (df.t_Final==1e7)]
df_filtered = df_filtered.sort_values('n_particles',ascending=True)
offsets = np.linspace(0,10,num=len(df_filtered))
print(offsets)
ax.set_color_cycle([plt.cm.plasma(i) for i in np.linspace(1, 0, len(df_filtered))])
for i,signac_id in enumerate(df_filtered['signac_id']):
    job = project.open_job(id=signac_id)
    #print(job.sp.temp_prof)
    #if job.sp.n_particles != 5e5 and job.sp.n_particles != 1e6 and job.sp.n_particles
!= 2e6:
    #        continue
    if 'gel_point' in job.document:
        print(job.sp.profile,job,'gel point',job.document['gel_point'])
    else:
        print(job.sp.profile,job)
    if job.isfile('diffract_type_2/asq.txt'):
        data=np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
        legend='{:.1e}'.format(job.sp.n_particles)
        q = data[:,0]/1.06
        I = data[:,1]
        if 'Lx' in job.document:
            box_length = float(job.document['Lx'])
            #print(type(box_length),box_length)
        else:
            print('Lx not found in',job)
        first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
        #print(data[:,0])
        #print(q)
        fn = interpolate.interp1d(q,I,kind='cubic')
        offset = offsets[i]
        I=I+offset
        #print(I)
        ax.plot(q,
                I,
                #marker='*',
                #linewidth=1,
                label=legend,
                zorder=1)
        #plt.plot(q,I+1,label=legend,linewidth=1)
        Is_of_qm.append(first_peak_i)
        qs_of_qm.append(first_peak_q)

        ax.scatter(first_peak_q,
                   first_peak_i+offset,
                   marker='o',
                   s=50,
                   color='r',
                   zorder=2)#,s=10)

        q_hbl=2*math.pi/(box_length*1.06/2)
        ax.scatter(q_hbl,
                   fn(q_hbl)+offset,
                   marker='*',
                   s=50,
                   color='b',
                   zorder=3)#,s=10)

    else:
        print('can\'t find',job.fn('diffract_type_2/asq.txt'))
```

```
        legend = ax.legend(loc='upper right',
                           shadow=False,
                           frameon=True,
                           prop={'size':12},
                           handlelength=1.5,
                           borderaxespad = 1)
        legend.get_frame().set_facecolor('white')
        legend.get_frame().set_alpha(1.0)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")
        ax.set_xlim(0.0,1.0)
        print(Is_of_qm)
        plt.show()
```

```
[  0.           0.38461538   0.76923077   1.15384615   1.53846154
   1.92307692   2.30769231   2.69230769   3.07692308   3.46153846
   3.84615385   4.23076923   4.61538462   5.           5.38461538
   5.76923077   6.15384615   6.53846154   6.92307692   7.30769231
   7.69230769   8.07692308   8.46153846   8.84615385   9.23076923
   9.61538462  10.        ]
iso d1012ec6925fb67d7a4283087cdb52ce gel point 0.0
iso 9f7d16db580d6b92ec456c1d930fc70c gel point 0.0
iso 58f779389453a611aeed590791c2fa6d gel point 0.0
iso 9d06aeadea0152829fc9abd7604579b3 gel point 0.0
iso 5ba3c48c0aba8e488c252533f45ed7ca gel point 0.0
iso 454f5d805e71ea6e174c8ae98aa60bc9 gel point 0.0
iso 9621a96411e1e55dddd2640cb7e58083
iso 896939a80117c20fb3a400e898089e9c
iso cf13e0bd18bb44658b471b05b8fe7f77
iso 21d2ed27b62459dd8a9b08a30fb67d44 gel point 0.0
iso faf664b4295ecd046152a5009790ac5f gel point 0.0
iso 735f8945d682a34c0bacfeea7508742e gel point 0.0
iso 59154b818c4ee59feb5bdfdd71f7a0d0 gel point 0.0
iso 15484fc70e1d1fa5fb5c5f5581ce2309 gel point 0.0
iso a94074b20791f20da55499865ecf6c80 gel point 0.0


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)


iso 6856cccb3865573f667ff8ced1896bbb gel point 0.0
iso 87d38a17b6114c73a06aa73b9b5ef60b gel point 0.0
iso 539673d82da04f2c140ecffe52797446 gel point 0.0
iso a19f9b89a6006515b0c91459be1b71d0 gel point 0.0
iso c82d60b3f677a0cb4e39033f519dc7d1 gel point 0.0
iso f93e34d45542f1d4d90c8faebeb43529 gel point 0.0
iso 3a5f1b7fb1764df53d678fdd03a5d842 gel point 0.0
iso 3bf282961ef1f311d97b76681935b08d gel point 0.0
iso 16efcafdee27999a99deae378d318037 gel point 0.0
iso 3d30a5818b965dd878a0f0059f3acd2a gel point 0.0
iso 9200af25cc3cbf69a58ea3438e66c271 gel point 0.0
iso d6292c69c26fa5be88202214c7edf5fc
[-4.5355646022829861, -4.8729711324842961, -5.443684375729001, -5.4609162134511253,
-5.2706147482436476, -5.5089118630150598, -5.3939106174858944, -5.5250787244770256,
-5.4571978732881119, -5.5250018201467865, -5.5057410546038659, -5.5254559411047763,
-5.5261261559577308, -5.5223895553915918, -2.7084124075994152, -5.5244480669763449,
-2.3526773022433871, -2.724823274590491, -2.6612166598887463, -2.8664542962473321,
```

```
-2.8782889875180393, -2.8175135915126575, -2.8218260453354311, -2.8744580313448922,
-2.4742400284712618, -3.0770788694692488, -3.2212741520170236]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

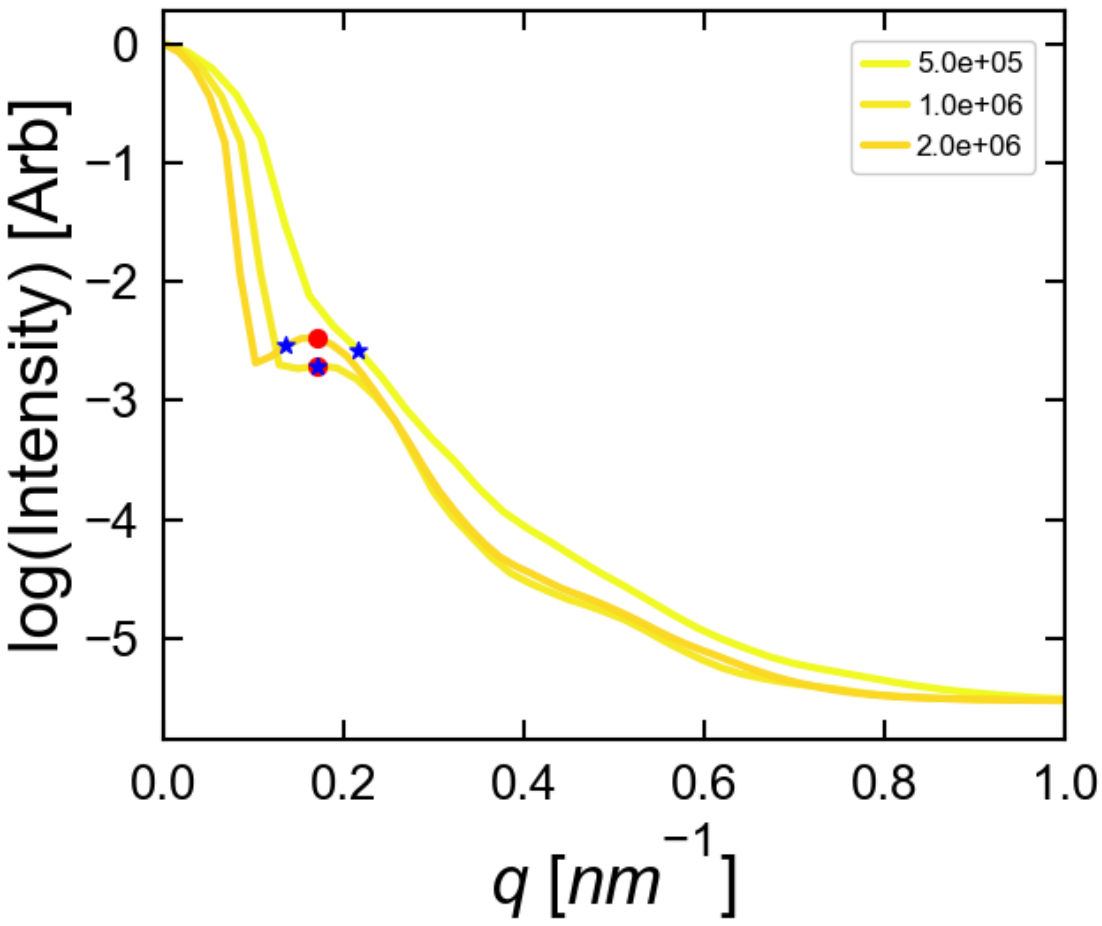



```
In [6]: from cycler import cycler
        from scipy import interpolate

        fig = plt.figure()
```

```
ax = fig.gca()


#plt.rc('axes', prop_cycle=(cycler('color', ['r','b','g','k','purple']) +
#                           cycler('linestyle', ['-', '--', ':', '-.','--'])))
Is_of_qm=[]
qs_of_qm=[]
df_filtered = df[(df.activation_energy==1)&
              (df['T']==500.0)&
                 (df.trial==0)&
              #(df.n_particles==1.4e6)&
              (df.t_Final==1e7)]
df_filtered = df_filtered.sort_values('n_particles',ascending=True)
offsets = np.linspace(0,10,num=len(df_filtered))
print(offsets)
ax.set_color_cycle([plt.cm.plasma(i) for i in np.linspace(1, 0, len(df_filtered))])
for i,signac_id in enumerate(df_filtered['signac_id']):
    job = project.open_job(id=signac_id)
    #print(job.sp.temp_prof)
    if job.sp.n_particles != 5e5 and\
       job.sp.n_particles != 1e6 and\
        job.sp.n_particles != 2e6:
            continue
    if 'gel_point' in job.document:
        print(job.sp.profile,job,'gel point',job.document['gel_point'])
    else:
        print(job.sp.profile,job)
    if job.isfile('diffract_type_2/asq.txt'):
        data=np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
        legend='{:.1e}'.format(job.sp.n_particles)
        q = data[:,0]/1.06
        I = data[:,1]
        if 'Lx' in job.document:
            box_length = float(job.document['Lx'])
            #print(type(box_length),box_length)
        else:
            print('Lx not found in',job)
        first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
        #print(data[:,0])
        #print(q)
        fn = interpolate.interp1d(q,I,kind='cubic')
        offset = 0#offsets[i]
        I=I+offset
        #print(I)
        ax.plot(q,
                I,
                #marker='*',
                #linewidth=1,
                label=legend,
                zorder=1)
        #plt.plot(q,I+1,label=legend,linewidth=1)
        Is_of_qm.append(first_peak_i)
        qs_of_qm.append(first_peak_q)

        ax.scatter(first_peak_q,
                    first_peak_i+offset,
                    marker='o',
                    s=50,
                    color='r',
                    zorder=2)#,s=10)

        q_hbl=2*math.pi/(box_length*1.06/2)
        ax.scatter(q_hbl,
                    fn(q_hbl)+offset,
                    marker='*',
                    s=50,
                    color='b',
                    zorder=3)#,s=10)
```

```
            else:
                print('can\'t find',job.fn('diffract_type_2/asq.txt'))

        legend = ax.legend(loc='upper right',
                           shadow=False,
                           frameon=True,
                           prop={'size':12},
                           handlelength=1.5,
                           borderaxespad = 1)
        legend.get_frame().set_facecolor('white')
        legend.get_frame().set_alpha(1.0)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")
        ax.set_xlim(0.0,1.0)
        print(Is_of_qm)
        plt.show()
```

```
[  0.           0.38461538   0.76923077   1.15384615   1.53846154
   1.92307692   2.30769231   2.69230769   3.07692308   3.46153846
   3.84615385   4.23076923   4.61538462   5.           5.38461538
   5.76923077   6.15384615   6.53846154   6.92307692   7.30769231
   7.69230769   8.07692308   8.46153846   8.84615385   9.23076923
   9.61538462  10.        ]
iso 21d2ed27b62459dd8a9b08a30fb67d44 gel point 0.0
iso a94074b20791f20da55499865ecf6c80 gel point 0.0
iso 3d30a5818b965dd878a0f0059f3acd2a gel point 0.0
[-5.5250018201467865, -2.7084124075994152, -2.4742400284712618]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



1 3D

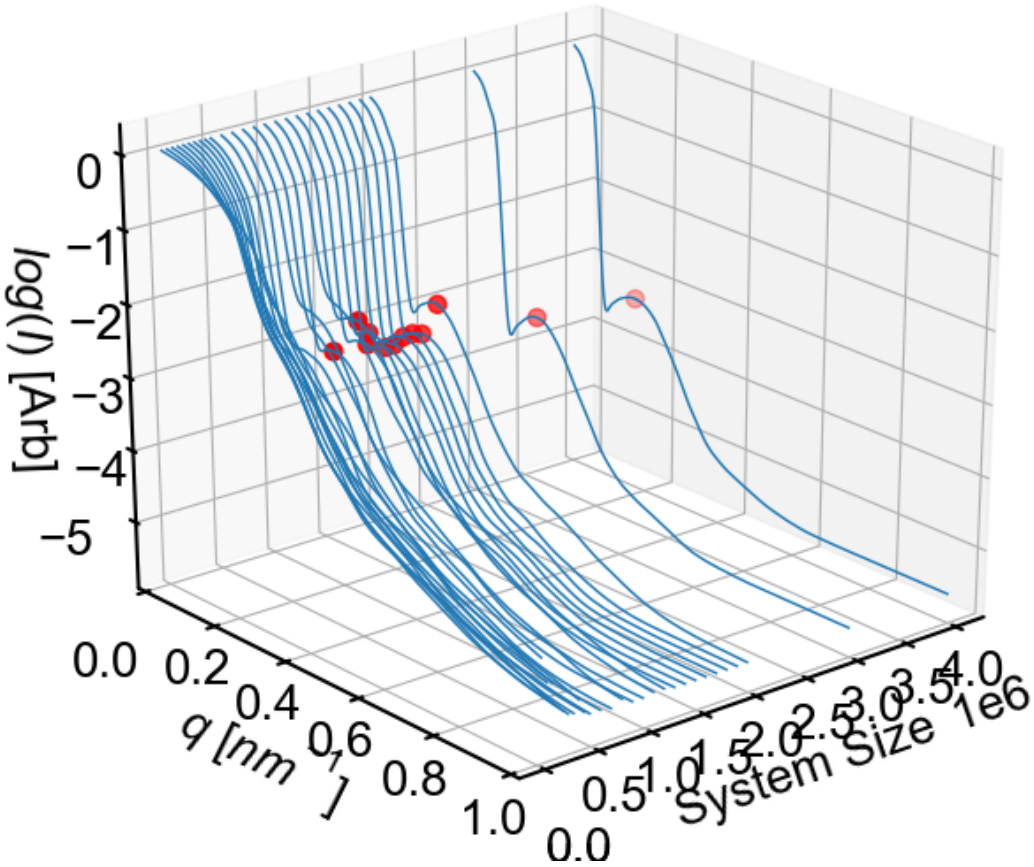
```
In [7]: import os
        #import structure_factor as sf
        import math

        import gsd
        import gsd.fl
        import gsd.hoomd
        import itertools
        import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib import cm
        from mpl_toolkits.mplot3d import Axes3D
        from scipy import interpolate

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        #ax.set_title('z-axis left side')
        ax = fig.add_axes(MyAxes3D(ax, 'l'))

        #ax3 = ax1.twinx()
        typeId=2
        n_views=40
```

```
grid_size=512

n_particles=0


df_filtered = df[(df.activation_energy==1)&
              (df['T']==500.0)&
                 (df.trial==0)&
              (df.t_Final==1e7)]
df_filtered = df_filtered.sort_values('n_particles')
Ns=[]
qs=[]
Is=[]
box_lengths=[]
Is_of_qm=[]
qs_of_qm=[]
Ns_of_qm=[]
cure_all=[]

for signac_id in df_filtered['signac_id']:
    job = project.open_job(id=signac_id)
    #print(job.workspace())
    if 'Lx' in job.document:
        box_length = job.document['Lx']
        half_box_length = box_length/2
        box_lengths.append(box_length)
    else:
        print('Lx not found in',job)
    q_half_length = 2*math.pi/(half_box_length/1.06)
    diffract_dir ='diffract_type_2'
    n_particles=job.sp.n_particles

    if job.isfile('{}/asq.txt'.format(diffract_dir)):
        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
        #print(np.shape(data[:,0]))
        q=data[:,0]
        I=data[:,1]
        if True:
            fn = interpolate.interp1d(q,I,kind='cubic')
            x = np.linspace(0.0,1.0,num=500)
            y=fn(x)
            qs.append(x)#(q)
            Is.append(y)#(I)
            Ns.append([n_particles]*len(x))#(data[:,0]))
        else:
            qs.append(q)
            Is.append(I)
            Ns.append([n_particles]*len(q))

        first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
        Is_of_qm.append(first_peak_i)
        qs_of_qm.append(first_peak_q)
        Ns_of_qm.append(n_particles)
    else:
        print(job,'did not contain diffraction data in ',diffract_dir)


matplotlib.rcParams['xtick.labelsize'] = 15
matplotlib.rcParams['ytick.labelsize'] = 15
#matplotlib.rcParams['ztick.labelsize'] = 15
matplotlib.rcParams['xtick.major.pad'] = 0
matplotlib.rcParams['ytick.major.pad'] = 0
#matplotlib.rcParams['ztick.major.pad'] = 1
qs=np.asarray(qs)
Ns=np.asarray(Ns)
Is=np.asarray(Is)
X=qs#[:,:55]
Y=Ns#[:,:55]
```

```
        Z=Is#[:,:55]
        #print(X)
        print(np.shape(X))
        print(np.shape(Y))
        print(np.shape(Z))

        ax.xaxis._axinfo['label']['space_factor'] = 1.0
        ax.yaxis._axinfo['label']['space_factor'] = 1.0
        ax.zaxis._axinfo['label']['space_factor'] = 1.0
        #ax.xaxis._axinfo['xtick']['labelsize'] = 1.0
        #ax.yaxis._axinfo['ytick']['labelsize'] = 1.0
        #ax.zaxis._axinfo['ztick']['labelsize'] = 1.0


        surf = ax.plot_wireframe(X, Y, Z,
                                 linewidth=1.0,
                                 zorder=1,
                                 antialiased=True,
                                 cstride=0,
                                 rstride=1)

        view_1 = (0, 180)#back
        view_2 = (25, -40)#angled
        view_3 = (25, 0)#front from top
        view_4 = (-5, 90)#right
        init_view = view_2
        ax.view_init(*init_view)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]",fontsize=20, rotation=150)
        ax.set_ylabel("System Size",fontsize=20, rotation=0)
        ax.set_zlabel(r"$log(I)$ [Arb]",fontsize=20, rotation=90)
        ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True,rotation=0)

        cut_off=38#-1#28#45
        qs=np.asarray(qs_of_qm[1:])
        Ns=np.asarray(Ns_of_qm[1:])
        Is=np.asarray(Is_of_qm[1:])
        indexes = np.where(qs<=1.0)
        ax.scatter3D(qs[indexes],#q_maxs,
                     Ns[indexes],
                     Is[indexes],#I_of_q_maxs,
                     color='r',
                     zorder=2,
                     marker='o',
                     s=50,
                     antialiased=True)
        ax.set_xlim(0,1.0)
        #plt.savefig('3D_q_plot_gamma_{}_n_{}.png'.format(gammas[0],n_particles),transparent=Tru
        e)

        plt.show()
```

```
(27, 500)
(27, 500)
(27, 500)
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



2 Using average over trials and independent frames

```
In [8]: df_filtered = df[(df.activation_energy==1)&
                         (df['T']==500.0)&
                         (df.n_particles==3e6)&
                        (df.trial==0)&
                         (df.t_Final==1e7)]
        sid = df_filtered.signac_id
        print(sid[0])#index[0])
```

```
9200af25cc3cbf69a58ea3438e66c271
```

```
In [9]: from scipy import stats

        def get_mean_sf_from_independent_frames(job):
            typeId=2
            n_views=40
            grid_size=512
            diffract_dir_pattern
        ='diffract_type_{}_n_views_{}_grid_size_{}_frame'.format(typeId,
        n_views,
        grid_size)
```

```
    directories = os.listdir(job.workspace())
    directories = [d for d in os.listdir(job.workspace()) if
d.startswith(diffract_dir_pattern)]
    directories.sort(key = lambda x: int(x.split('_')[-1]))
    #print(len(directories))
    #print(directories)
    num_frames = len(directories)
    log_path = job.fn('out.log')
    data = np.genfromtxt(log_path, names=True)
    time_steps = data['timestep']
    #print('time steps',time_steps)
    start_i, start_t = autocorr.find_equilibrated_window(time_steps,
data['potential_energy'])
    decorrelation_time, decorrelation_stride =
_get_decorrelation_time(data['potential_energy'][start_i:], time_steps[start_i:])
    #print('decorrelation_stride:',decorrelation_stride)
    #print('decorrelation_time:',decorrelation_time)
    #print('start_i:',start_i)
    #print('start_t:',start_t)
    if num_frames > 0:
        qs_for_all_times=[]
        Is_for_all_times=[]
        times_for_all_times=[]
        qs_list = []
        times_list = []
        Is_list = []
        Qs_list=[]

        for i,diffract_dir in enumerate(directories):
            #print("Progress {:2.1%}".format(i / num_frames), end="\r")

            #print(diffract_dir)

            if diffract_dir.startswith(diffract_dir_pattern):
                frame = int(diffract_dir.split('_')[-1])

                decorrelated_frame_stride = int(decorrelation_time/job.sp.dcd_write)
                decorrelated_frame_stride = max(decorrelated_frame_stride,1)
                time = round(frame*job.sp.dcd_write)
                #print('decorrelated frame stride is:',decorrelated_frame_stride)
                #print('Equilibriated after time:',start_t)
                if time >= start_t and frame%decorrelated_frame_stride==0:# and frame
<3e6/job.sp.dcd_write:#==119 or frame==123:#%100 == 0:#num_frames/30:
                    if job.isfile('{}/asq.txt'.format(diffract_dir)):
                        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))

                        legend = '{} $\\Delta t(\Gamma:{})$'.format(time,job.sp.gamma)
                        qs = data[:,0]
                        Is = data[:,1]
                        #print(qs.shape)
                        qs_for_all_times.append(qs)
                        Is_for_all_times.append(Is)
                        times_for_all_times.append(time)

                        dq=qs[1]-qs[0]
                        Is_exp = np.exp(Is)
                        q_sq = qs**2
                        Q = np.sum(Is_exp*qs*dq)
                        Qs_list.append(Q)
                        #first_peak_q,first_peak_i = get_highest_maxima(job,qs,Is)
                        #if first_peak_q >0.8 and time > 2.0e5:
                        #    first_peak_q=q_half_length

                        #qs_list.append(first_peak_q)
                        #times_list.append(time)
                        #Is_list.append(first_peak_i)
                    else:
                        print(job,'did not contain diffraction data in ',diffract_dir)
```

```
                    #else:
                    #    print(job,'directory {} is not as
        expected:{}'.format(diffract_dir,diffract_dir_pattern))
            else:
                print(job,'did not contain diffraction data for time evolution')
            print('Number of independent frames for average:',len(qs_for_all_times))
            m_q = np.mean(qs_for_all_times,axis=0)
            std_q = stats.sem(qs_for_all_times,axis=0)
            m_I = np.mean(Is_for_all_times,axis=0)
            std_I = stats.sem(Is_for_all_times,axis=0)
            return m_q,std_q,m_I,std_I

In [10]: import os
         #import structure_factor as sf
         import math

         import gsd
         import gsd.fl
         import gsd.hoomd


         #ax3 = ax1.twinx()


         Ns = np.arange(5e4,5e5,5e4,dtype=int)
         Ns_1 = np.arange(5e5,2.1e6,1e5,dtype=int)
         Ns=np.append(Ns,Ns_1)
         Ns=np.append(Ns,3e6)
         #Ns=[2e6,3e6]
         #Ns=[5e4,1e5,3e5,5e5,1e6,2e6,3e6]

         mean_qs_for_Ns = []
         mean_Is_for_Ns = []
         mean_times_for_Ns = []
         std_qs_for_Ns = []
         std_Is_for_Ns = []
         q_hbls=[]
         USE_INDE_FRAMES=True
         for N in Ns:
             df_filtered = df[(df.activation_energy==1)&
                              (df['T']==500.0)&
                                 #(df.trial==0)&
                              (df.n_particles==N)&
                              (df.t_Final==1e7)]

             df_filtered = df_filtered.sort_values('n_particles')
             #print(df_filtered)
             #grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
             times_for_all_trials=[]
             qs_for_all_trials=[]
             Is_for_all_trials=[]

             qms_all=[] #qmax
             Is_all=[]
             Qs_all=[]
             times_all=[]
             cure_all=[]
             mean_qs=[]
             mean_Is=[]
             std_qs=[]
             std_Is=[]
             mean_qms=[]
             mean_Is_of_qm=[]

             print('Analyzing strcture for',N,'ntrial:',len(df_filtered))


             for signac_id in df_filtered['signac_id']:
```

```
                job = project.open_job(id=signac_id)
                if 'Lx' in job.document:#checking if the job completed
                    if USE_INDE_FRAMES:
                        mqfif,stdqfif,mifif,stdifif = get_mean_sf_from_independent_frames(job)
                        mean_qs.append(mqfif)
                        std_qs.append(stdqfif)
                        mean_Is.append(mifif)
                        std_Is.append(stdifif)
                    else:
                        diffract_dir ='diffract_type_2'
                        n_particles=job.sp.n_particles
                        if job.isfile('{}/asq.txt'.format(diffract_dir)):
                            data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                            #print(np.shape(data[:,0]))
                            q=data[:,0]
                            I=data[:,1]
                            mean_qs.append(q)
                            #std_qs.append(stdqfif)
                            mean_Is.append(I)
                            #std_Is.append(stdifif)
                        else:
                            print('Final frame is not diffracted')

            mean_q_for_N = np.mean(mean_qs,axis=0)
            mean_qs_for_Ns.append(mean_q_for_N)
            std_q_for_N = stats.sem(mean_qs,axis=0)
            std_qs_for_Ns.append(std_q_for_N)
            mean_I_for_N = np.mean(mean_Is,axis=0)
            mean_Is_for_Ns.append(mean_I_for_N)
            std_I_for_N = stats.sem(mean_Is,axis=0)
            std_Is_for_Ns.append(std_I_for_N)
```

```
Analyzing strcture for 50000.0 ntrial: 5
Number of independent frames for average: 3
Number of independent frames for average: 23
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Analyzing strcture for 100000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 4
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 150000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Analyzing strcture for 200000.0 ntrial: 5
Number of independent frames for average: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 250000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 300000.0 ntrial: 5
Number of independent frames for average: 26
```

```
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 350000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 400000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 450000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 500000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 26
Analyzing strcture for 600000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 21
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 700000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 800000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 900000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 1000000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 8
Analyzing strcture for 1100000.0 ntrial: 5
```

```
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 1200000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 6
Analyzing strcture for 1300000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 1400000.0 ntrial: 5
Number of independent frames for average: 23
Number of independent frames for average: 10
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 6
Analyzing strcture for 1500000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 6
Number of independent frames for average: 26
Number of independent frames for average: 8
Analyzing strcture for 1600000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 12
Number of independent frames for average: 6
Analyzing strcture for 1700000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 15
Number of independent frames for average: 8
Analyzing strcture for 1800000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 12
Number of independent frames for average: 8
Number of independent frames for average: 6
Analyzing strcture for 1900000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 6
Analyzing strcture for 2000000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 8
Number of independent frames for average: 8
Number of independent frames for average: 26
```

```
Analyzing strcture for 3000000.0 ntrial: 5
Number of independent frames for average: 8
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 6
Number of independent frames for average: 8
```

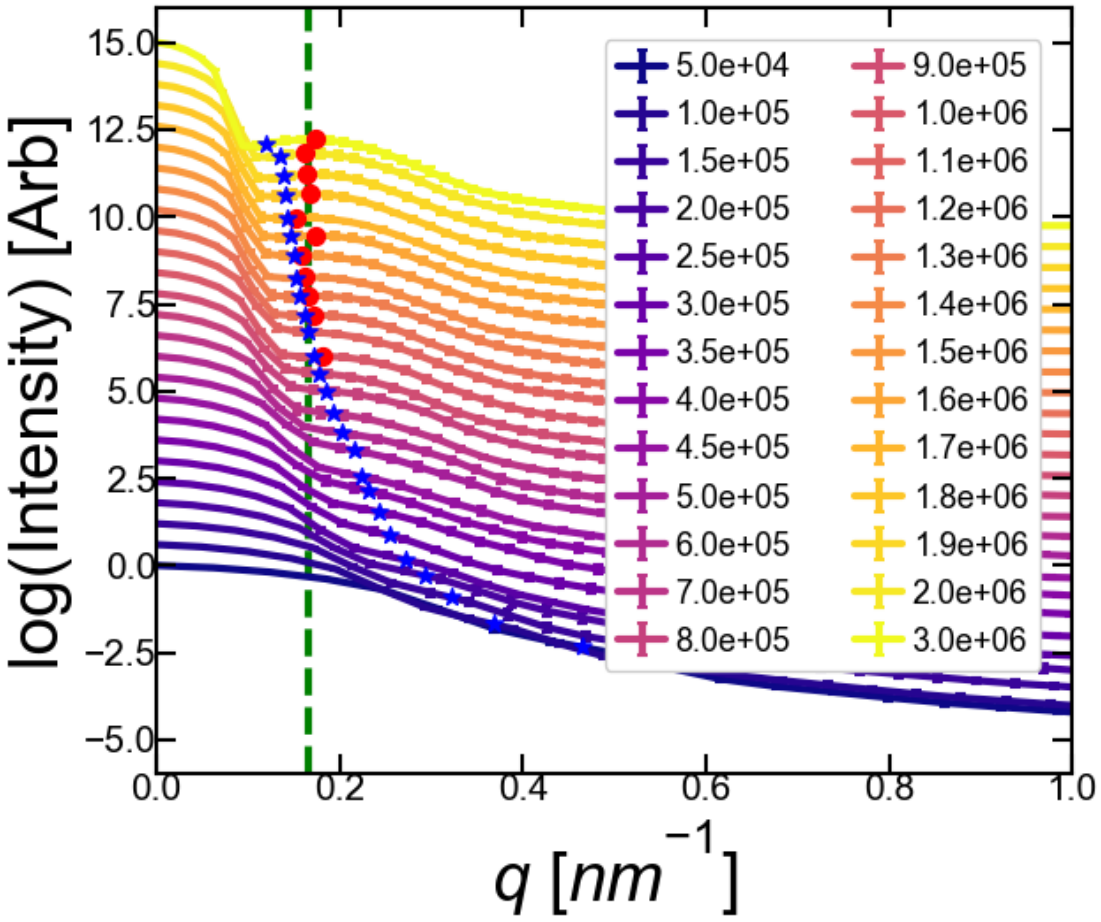
```
In [11]: from matplotlib import cm
         from mpl_toolkits.mplot3d import Axes3D
         from scipy import interpolate
         fig = plt.figure()
         ax = fig.gca()
         print(Ns)
         cmap = [plt.cm.plasma(i) for i in np.linspace(0, 1, len(Ns))]
         ax.set_color_cycle(cmap)
         offsets = np.linspace(0,15,num=len(Ns))
         first_peak_qs=[]
         for i,N in enumerate(Ns):
             print('N',N)
             #i=l-i_temp-1
             q=mean_qs_for_Ns[i]
             I=mean_Is_for_Ns[i]
             I_std=std_Is_for_Ns[i]

             offset = offsets[i]
             #time=time_mean[i]
             legend='{:.1e}'.format(N)
             fn = interpolate.interp1d(q,I,kind='cubic')
             ax.errorbar(q,
                         I+offset,
                         I_std,
                         #markersize=5,
                         capsize=2,
                         label=legend,
                         zorder=1)
             df_filtered = df[(df.activation_energy==1)&
                              (df['T']==500.0)&
                              (df.n_particles==N)&
                              (df.trial==0)&
                              (df.t_Final==1e7)]
             sids = df_filtered.signac_id
             job = project.open_job(id=sids[0])
             if 'Lx' not in job.document:
                 print(job,'does not contain Lx')
                 continue
             #print(I)
             first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
             if first_peak_i != 0. and first_peak_q<1.0:
                 first_peak_qs.append(first_peak_q)
                 ax.scatter(first_peak_q,
                            first_peak_i+offset,
                            color='r',s=50,zorder=2)
             half_box_length = job.document['Lx']/2
             q_hbl = 2*math.pi/(half_box_length*1.06)
             ax.scatter(q_hbl,
                        fn(q_hbl)+offset,
                        marker='*',
                        s=50,
                        color='b',
                        zorder=2)#,s=10)

         #ax.axvline(x=2*math.pi/(35))
         mean_q=np.mean(first_peak_qs)
         print('mean first peak',mean_q)
         ax.axvline(x=mean_q,linestyle='--',color='g',zorder=0)#,label='$\\langle
         q_{max}\\rangle$')
```

```
        legend = ax.legend(loc='upper right',
                           shadow=False,
                           frameon=True,
                           prop={'size':13},
                           ncol=2,
                           handlelength=1.5,
                           borderaxespad = 1.0)
        legend.get_frame().set_alpha(1.0)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")

        ax.set_xlim(0.0,1.0)
        #plt.colorbar()

        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/2D_q_pl
        ot_wrt_N.png',transparent=True)
        plt.show()
```

```
[   50000.   100000.   150000.   200000.   250000.   300000.   350000.
   400000.   450000.   500000.   600000.   700000.   800000.   900000.
  1000000.  1100000.  1200000.  1300000.  1400000.  1500000.  1600000.
  1700000.  1800000.  1900000.  2000000.  3000000.]
N 50000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 100000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 150000.0
N 200000.0
N 250000.0
N 300000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 350000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 400000.0
N 450000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 500000.0
N 600000.0


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)


N 700000.0
N 800000.0
N 900000.0
N 1000000.0
N 1100000.0
N 1200000.0
N 1300000.0
N 1400000.0
N 1500000.0
N 1600000.0
```

```
N 1700000.0
N 1800000.0
N 1900000.0
N 2000000.0
N 3000000.0
mean first peak 0.16647492293
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [12]:
```
from matplotlib import cm
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import interpolate
fig = plt.figure()
ax = fig.gca()
ax1 = fig.add_axes([ 0.95, 0.25,0.03, 0.65])

#Ns = np.arange(5e4,5e5,1e5,dtype=int)
```

```
Ns_ticks = np.arange(5e4,2e6,5e5,dtype=int)
Ns_ticks=np.append(Ns_ticks,2e6)
#Ns=np.append(Ns,Ns_1)
ticks=np.append(Ns_ticks,3e6)

cmap = mpl.cm.plasma
norm = mpl.colors.Normalize(vmin=5e4, vmax=3e6)
cb1 = mpl.colorbar.ColorbarBase(ax1, cmap=cmap,
                                norm=norm,
                                ticks=ticks,
                                format = "%.1e",
                                spacing='uniform',
                                orientation='vertical')

for l in ax1.yaxis.get_ticklabels():
    #l.set_weight("bold")
    l.set_fontsize(15)
print(Ns)
cmap_indices = []
maxN = np.max(Ns)
minN = np.min(Ns)
for N in Ns:
    cmap_i = (N-minN)/(maxN-minN)
    cmap_indices.append(cmap_i)
#cmap = [plt.cm.plasma(i) for i in np.linspace(0, 1, len(Ns))]
cmap = [plt.cm.plasma(i) for i in cmap_indices]
ax.set_color_cycle(cmap)
offsets = np.linspace(0,15,num=len(Ns))
first_peak_qs=[]
for i,N in enumerate(Ns):
    print('N',N)
    #i=l-i_temp-1
    q=mean_qs_for_Ns[i]
    I=mean_Is_for_Ns[i]
    I_std=std_Is_for_Ns[i]

    offset = offsets[i]
    #time=time_mean[i]
    legend='{:.1e}'.format(N)
    fn = interpolate.interp1d(q,I,kind='cubic')
    ax.errorbar(q,
                I+offset,
                I_std,
                #markersize=5,
                capsize=2,
                label=legend,
                zorder=1)
    df_filtered = df[(df.activation_energy==1)&
                  (df['T']==500.0)&
                  (df.n_particles==N)&
                  (df.trial==0)&
                  (df.t_Final==1e7)]
    sids = df_filtered.signac_id
    job = project.open_job(id=sids[0])
    if 'Lx' not in job.document:
        print(job,'does not contain Lx')
        continue
    #print(I)
    first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
    if first_peak_i != 0. and first_peak_q<1.0:
        first_peak_qs.append(first_peak_q)
        ax.scatter(first_peak_q,
                   first_peak_i+offset,
                   color='r',s=50,zorder=2)
    half_box_length = job.document['Lx']/2
    q_hbl = 2*math.pi/(half_box_length*1.06)
    ax.scatter(q_hbl,
                fn(q_hbl)+offset,
```

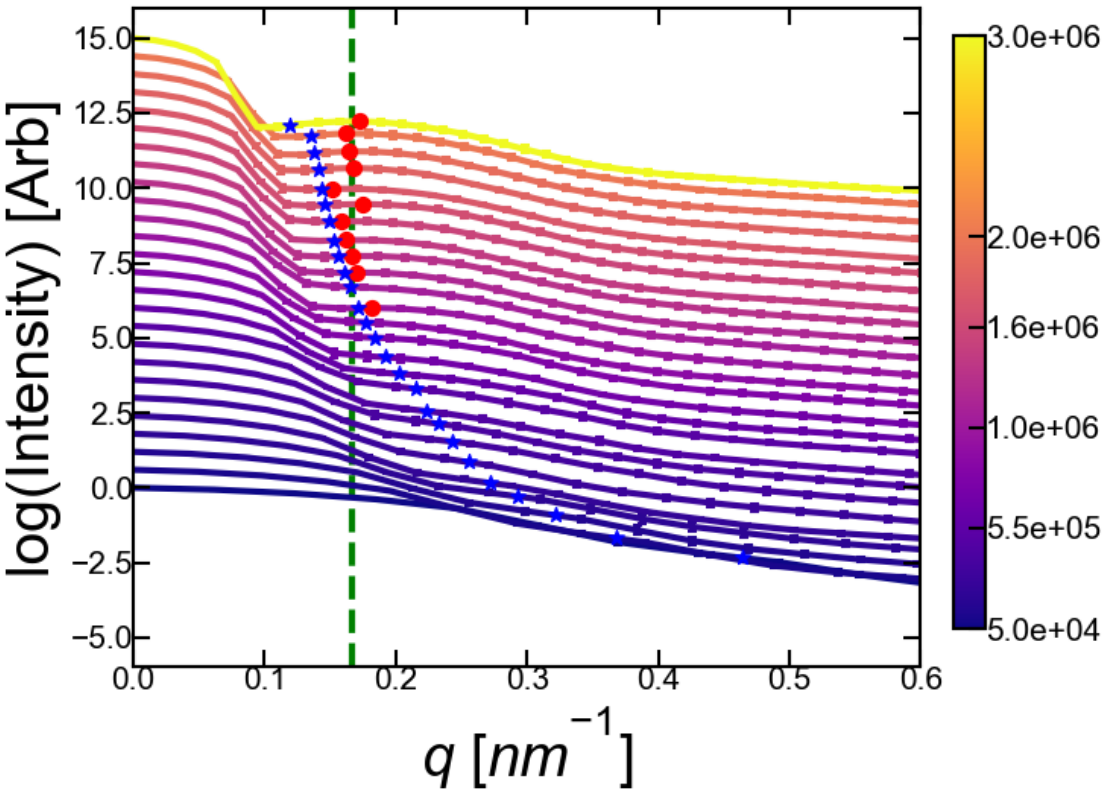
```
                    marker='*',
                    s=50,
                    color='b',
                    zorder=2)#,s=10)

        #ax.axvline(x=2*math.pi/(35))
        mean_q=np.mean(first_peak_qs)
        print('mean first peak',mean_q)
        ax.axvline(x=mean_q,linestyle='--',color='g',zorder=0)#,label='$\\langle
        q_{max}\\rangle$')
        #legend = ax.legend(loc='upper right',
        #                   shadow=False,
        #                   frameon=True,
        #                   prop={'size':13},
        #                   ncol=2,
        #                   handlelength=1.5,
        #                   borderaxespad = 1.0)
        #legend.get_frame().set_alpha(1.0)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")

        ax.set_xlim(0.0,0.6)
        #plt.colorbar()

        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/2D_q_pl
        ot_wrt_N_1.png',transparent=True)
        plt.show()
```

```
[   50000.   100000.   150000.   200000.   250000.   300000.   350000.
   400000.   450000.   500000.   600000.   700000.   800000.   900000.
  1000000.  1100000.  1200000.  1300000.  1400000.  1500000.  1600000.
  1700000.  1800000.  1900000.  2000000.  3000000.]
N 50000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 100000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 150000.0
N 200000.0
N 250000.0
N 300000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 350000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 400000.0
N 450000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 500000.0
N 600000.0
N 700000.0
N 800000.0
N 900000.0
N 1000000.0


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
```

```
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
N 1100000.0
N 1200000.0
N 1300000.0
N 1400000.0
N 1500000.0
N 1600000.0
N 1700000.0
N 1800000.0
N 1900000.0
N 2000000.0
N 3000000.0
mean first peak 0.16647492293
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

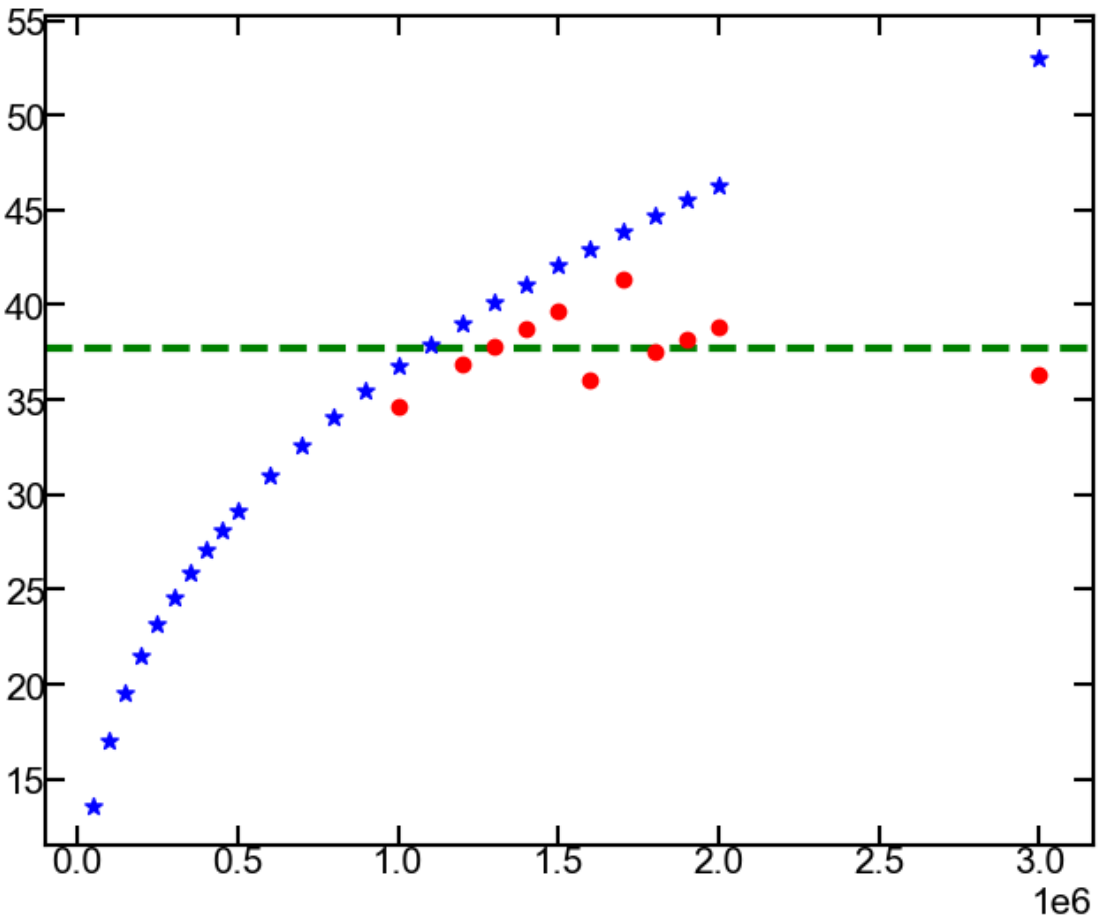


In [13]:
```
fig = plt.figure()
ax = fig.gca()
first_peaks = []
half_box_qs = []
```

```
        for i,N in enumerate(Ns):
            print('N',N)
            #i=l-i_temp-1
            q=mean_qs_for_Ns[i]
            I=mean_Is_for_Ns[i]
            I_std=std_Is_for_Ns[i]

            offset = offsets[i]
            #time=time_mean[i]
            legend='{:.1e}'.format(N)
            fn = interpolate.interp1d(q,I,kind='cubic')

            df_filtered = df[(df.activation_energy==1)&
                         (df['T']==500.0)&
                         (df.n_particles==N)&
                         (df.trial==0)&
                         (df.t_Final==1e7)]
            sids = df_filtered.signac_id
            job = project.open_job(id=sids[0])
            if 'Lx' not in job.document:
                print(job,'does not contain Lx')
                continue
            #print(I)
            first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
            if first_peak_i != 0. and first_peak_q<1.0:
                print(N,first_peak_q)

                ax.scatter(N,
                        2*math.pi/first_peak_q,
                          # first_peak_i+offset,
                           color='r',zorder=2)
            half_box_length = job.document['Lx']/2
            q_hbl = 2*math.pi/(half_box_length*1.06)
            ax.scatter(N,
                       half_box_length*1.06,#2*math.pi/q_hbl,
                        #fn(q_hbl)+offset,
                        marker='*',
                        s=50,
                        color='b',
                        zorder=2)#,s=10)
        av_feat_size = 2*math.pi/mean_q
        print(av_feat_size)
        ax.axhline(y=av_feat_size,linestyle='--',color='g',zorder=0)
        legend = ax.legend(loc='upper right',
                        shadow=False,
                        frameon=True,
                        prop={'size':7},
                        handlelength=1.5,
                        borderaxespad = 1)
        ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useMathText=True)
        plt.show()
```

```
N 50000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 100000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 150000.0
N 200000.0
N 250000.0
N 300000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 350000.0
```

```
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 400000.0
N 450000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 500000.0
N 600000.0
N 700000.0
N 800000.0
N 900000.0
N 1000000.0
1000000.0 0.181238424098
N 1100000.0
N 1200000.0
1200000.0 0.170551880593
N 1300000.0
1300000.0 0.166061574496
N 1400000.0
1400000.0 0.162009669158
N 1500000.0
1500000.0 0.158326341489
N 1600000.0
1600000.0 0.174326265588
N 1700000.0
1700000.0 0.15185670544
N 1800000.0
1800000.0 0.167614642525
N 1900000.0
1900000.0 0.164620891306
N 2000000.0
2000000.0 0.161830161595
N 3000000.0
3000000.0 0.172787595947
37.7425332091


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [14]:
```
import os
#import structure_factor as sf
import math

import gsd
import gsd.fl
import gsd.hoomd


#ax3 = ax1.twinx()


Ns = np.arange(5e4,5e5,5e4,dtype=int)
Ns_1 = np.arange(5e5,2.1e6,1e5,dtype=int)
Ns=np.append(Ns,Ns_1)
Ns=np.append(Ns,3e6)
#Ns=[2e6,3e6]
Ns=[5e4,1e5,3e5,5e5,1e6,2e6,3e6]

mean_qs_for_Ns = []
mean_Is_for_Ns = []
mean_times_for_Ns = []
std_qs_for_Ns = []
std_Is_for_Ns = []
q_hbls=[]
USE_INDE_FRAMES=True
```

```
for N in Ns:
    df_filtered = df[(df.activation_energy==1)&
                     (df['T']==500.0)&
                        #(df.trial==0)&
                     (df.n_particles==N)&
                     (df.t_Final==1e7)]

    df_filtered = df_filtered.sort_values('n_particles')
    #print(df_filtered)
    #grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
    times_for_all_trials=[]
    qs_for_all_trials=[]
    Is_for_all_trials=[]

    qms_all=[] #qmax
    Is_all=[]
    Qs_all=[]
    times_all=[]
    cure_all=[]
    mean_qs=[]
    mean_Is=[]
    std_qs=[]
    std_Is=[]
    mean_qms=[]
    mean_Is_of_qm=[]

    print('Analyzing strcture for',N,'ntrial:',len(df_filtered))


    for signac_id in df_filtered['signac_id']:
        job = project.open_job(id=signac_id)
        if 'Lx' in job.document:#checking if the job completed
            if USE_INDE_FRAMES:
                mqfif,stdqfif,mifif,stdifif = get_mean_sf_from_independent_frames(job)
                mean_qs.append(mqfif)
                std_qs.append(stdqfif)
                mean_Is.append(mifif)
                std_Is.append(stdifif)
            else:
                diffract_dir ='diffract_type_2'
                n_particles=job.sp.n_particles
                if job.isfile('{}/asq.txt'.format(diffract_dir)):
                    data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                    #print(np.shape(data[:,0]))
                    q=data[:,0]
                    I=data[:,1]
                    mean_qs.append(q)
                    #std_qs.append(stdqfif)
                    mean_Is.append(I)
                    #std_Is.append(stdifif)
                else:
                    print('Final frame is not diffracted')

    mean_q_for_N = np.mean(mean_qs,axis=0)
    mean_qs_for_Ns.append(mean_q_for_N)
    std_q_for_N = stats.sem(mean_qs,axis=0)
    std_qs_for_Ns.append(std_q_for_N)
    mean_I_for_N = np.mean(mean_Is,axis=0)
    mean_Is_for_Ns.append(mean_I_for_N)
    std_I_for_N = stats.sem(mean_Is,axis=0)
    std_Is_for_Ns.append(std_I_for_N)
```

```
Analyzing strcture for 50000.0 ntrial: 5
Number of independent frames for average: 3
Number of independent frames for average: 23
Number of independent frames for average: 26
Number of independent frames for average: 8
```

```
Number of independent frames for average: 26
Analyzing strcture for 100000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 4
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Analyzing strcture for 300000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Analyzing strcture for 500000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 26
Analyzing strcture for 1000000.0 ntrial: 5
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 8
Number of independent frames for average: 26
Number of independent frames for average: 8
Analyzing strcture for 2000000.0 ntrial: 5
Number of independent frames for average: 26
Number of independent frames for average: 12
Number of independent frames for average: 8
Number of independent frames for average: 8
Number of independent frames for average: 26
Analyzing strcture for 3000000.0 ntrial: 5
Number of independent frames for average: 8
Number of independent frames for average: 12
Number of independent frames for average: 26
Number of independent frames for average: 6
Number of independent frames for average: 8
```

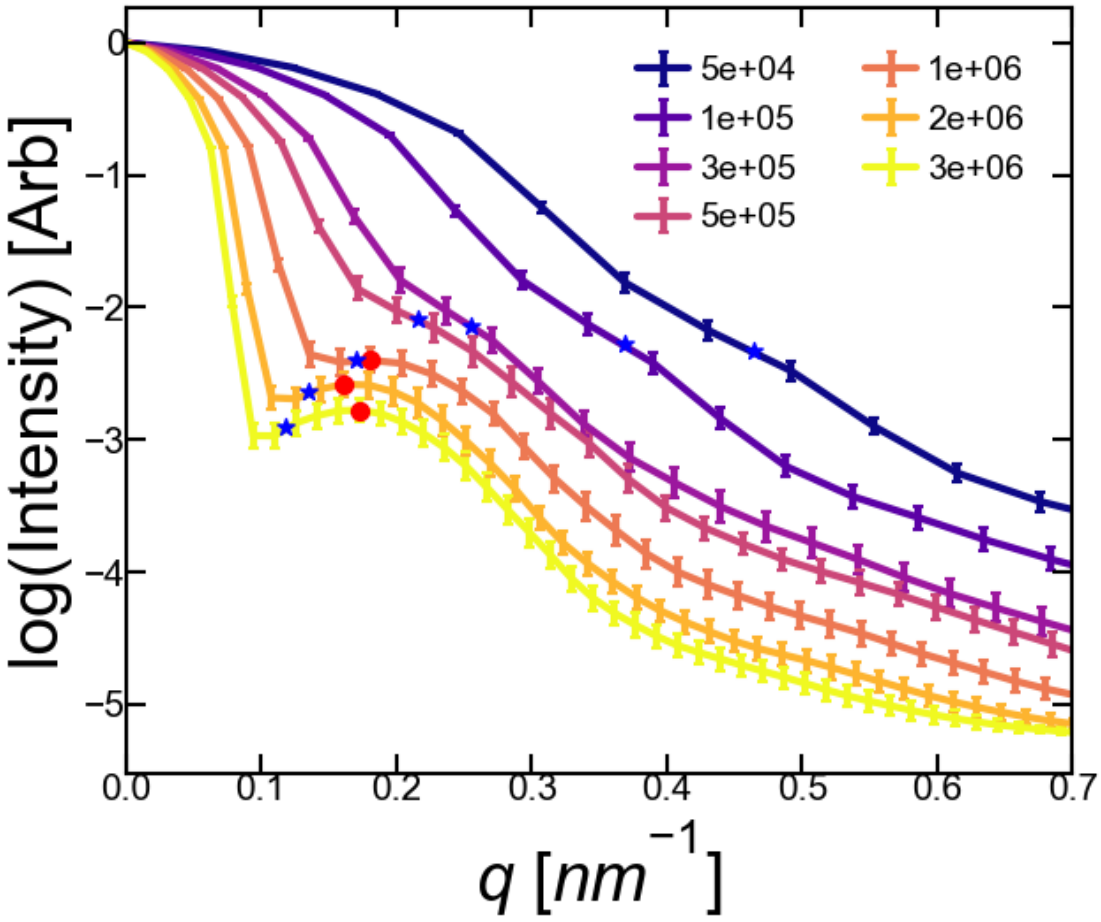
In [15]:
```
#### from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from scipy import interpolate
fig = plt.figure()
ax = fig.gca()

print(Ns)
cmap = [plt.cm.plasma(i) for i in np.linspace(0, 1, len(Ns))]
ax.set_color_cycle(cmap)
offsets = np.linspace(0,15,num=len(Ns))
first_peak_qs=[]

for i,N in enumerate(Ns):
    print('N',N)
    #i=l-i_temp-1
    q=mean_qs_for_Ns[i]
    I=mean_Is_for_Ns[i]
    I_std=std_Is_for_Ns[i]

    offset = 0#offsets[i]
    #time=time_mean[i]
    legend='{:.0e}'.format(N)
    fn = interpolate.interp1d(q,I,kind='cubic')
```

```
            ax.errorbar(q,
                        I+offset,
                        I_std,
                        markersize=5,
                        capsize=2,
                        label=legend,
                        zorder=1)
            df_filtered = df[(df.activation_energy==1)&
                             (df['T']==500.0)&
                             (df.n_particles==N)&
                             (df.trial==0)&
                             (df.t_Final==1e7)]
            sids = df_filtered.signac_id
            job = project.open_job(id=sids[0])
            if 'Lx' not in job.document:
                print(job,'does not contain Lx')
                continue
            #print(I)
            first_peak_q,first_peak_i = get_highest_maxima(job,q,I)
            if first_peak_i != 0. and first_peak_q<1.0:
                first_peak_qs.append(first_peak_q)
                ax.scatter(first_peak_q,
                           first_peak_i+offset,
                           color='r',s=50,zorder=2)
            half_box_length = job.document['Lx']/2
            q_hbl = 2*math.pi/(half_box_length*1.06)
            ax.scatter(q_hbl,
                        fn(q_hbl)+offset,
                        marker='*',
                        s=50,
                        color='b',
                        zorder=2)#,s=10)

        #ax.axvline(x=2*math.pi/(35))
        mean_q=np.mean(first_peak_qs)
        print('mean first peak',mean_q)
        #ax.axvline(x=mean_q,linestyle='--',color='g')
        legend = ax.legend(loc='upper right',
                        shadow=False,
                        frameon=False,
                        prop={'size':14},
                        ncol=2,
                        handlelength=1.5,
                        borderaxespad = 1)
        legend.get_frame().set_alpha(1.0)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")

        ax.set_xlim(0.0,0.7)
        #plt.colorbar()

        plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/2D_q_pl
        ot_wrt_N_subset.png',transparent=True)
        plt.show()
```

```
[50000.0, 100000.0, 300000.0, 500000.0, 1000000.0, 2000000.0, 3000000.0]
N 50000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 100000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 300000.0
WARNING: no maximas found for the structure factor. returning a default intensity of 0
at 0
N 500000.0
```

```
N 1000000.0
N 2000000.0
N 3000000.0
mean first peak 0.171952060547
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

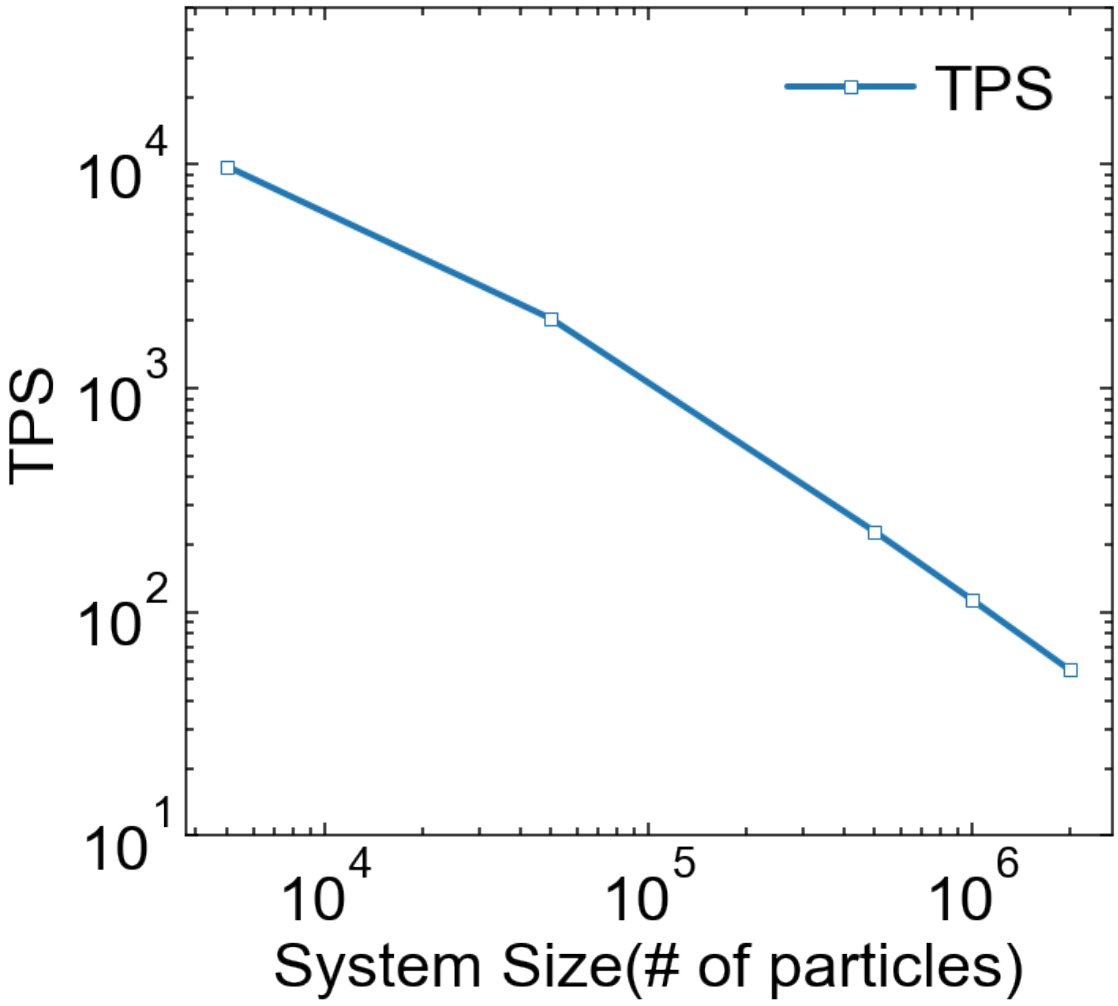


In [9]:
```
import signac
import gsd
import random
import gsd.fl
import gsd.hoomd
import os
from freud import box, density
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
import numpy as np

matplotlib.rc('xtick', labelsize=40)
matplotlib.rc('ytick', labelsize=40)
matplotlib.rc('axes', labelsize=20)
matplotlib.rc('lines', linewidth=4)

searchStr = 'Average TPS'
legacy_tps = []
legacy_temps = []
freud_tps = []
freud_temps = []
dybond_tps = []
dybond_n = []
cure_percents = []
num_bps = []

data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
#tps_wrt_system_size_after_percent_bonds/'#tps_compare_wrt_system_size'
project = signac.get_project(data_path)
jobs = project.find_jobs({'activation_energy':1,'T':500,'profile':'iso','n_dt':2995000})

fig,ax1 = plt.subplots(figsize=(10,9))
#ax2=ax1.twinx()
for job in jobs:
    #print(str(job), 'computed, kT:',kT)
    data = np.genfromtxt(job.fn('out.log'))
    fiIn = open(os.path.join(data_path,str(job)+'-process.o')).readlines()
    avg_tpss = []
    for line in fiIn:
        if line.startswith(searchStr):
            #print(line)
            data = line.split(":")
            if len(data) == 2:
                avg_tpss.append(data[1])
            else:
                print('Wow wow.. wrong TPS string!')

    dybond_tps.append(avg_tpss[-1])
    dybond_n.append(job.sp.n_particles)

print(dybond_tps)
if len(dybond_tps) > 0:
    dybond_n1, dybond_tps = zip(*sorted(zip(dybond_n, dybond_tps)))
    ax1.plot(dybond_n1,dybond_tps,label='TPS',marker = 's',
                    markeredgewidth=1, markerfacecolor="white" ,markersize=8)
    #dybond_n2, cure_percents = zip(*sorted(zip(dybond_n, cure_percents)))
    #ax2.plot(dybond_n2,cure_percents,marker='o', linestyle='--',label='cure percent')
ax1.set_xlabel('System Size(# of particles)',fontsize=40)
ax1.set_ylabel('TPS',fontsize=40)
ax1.set_yscale('log')
ax1.set_xscale('log')
ax1.set_ylim(1.0e1,5e4)
ax1.legend(fontsize=40)
#plt.savefig('/Users/stephenthomas/projects/epoxy_methods_manuscript/paper/Images/TPS_vs
_N.png',transparent=True)
```

```
[' 55.0486\n', ' 228.613\n', ' 2045.5\n', ' 9787.98\n', ' 113.232\n']
```

Out[9]: <matplotlib.legend.Legend at 0x1187de048>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



F.2 Code for Chapter 4

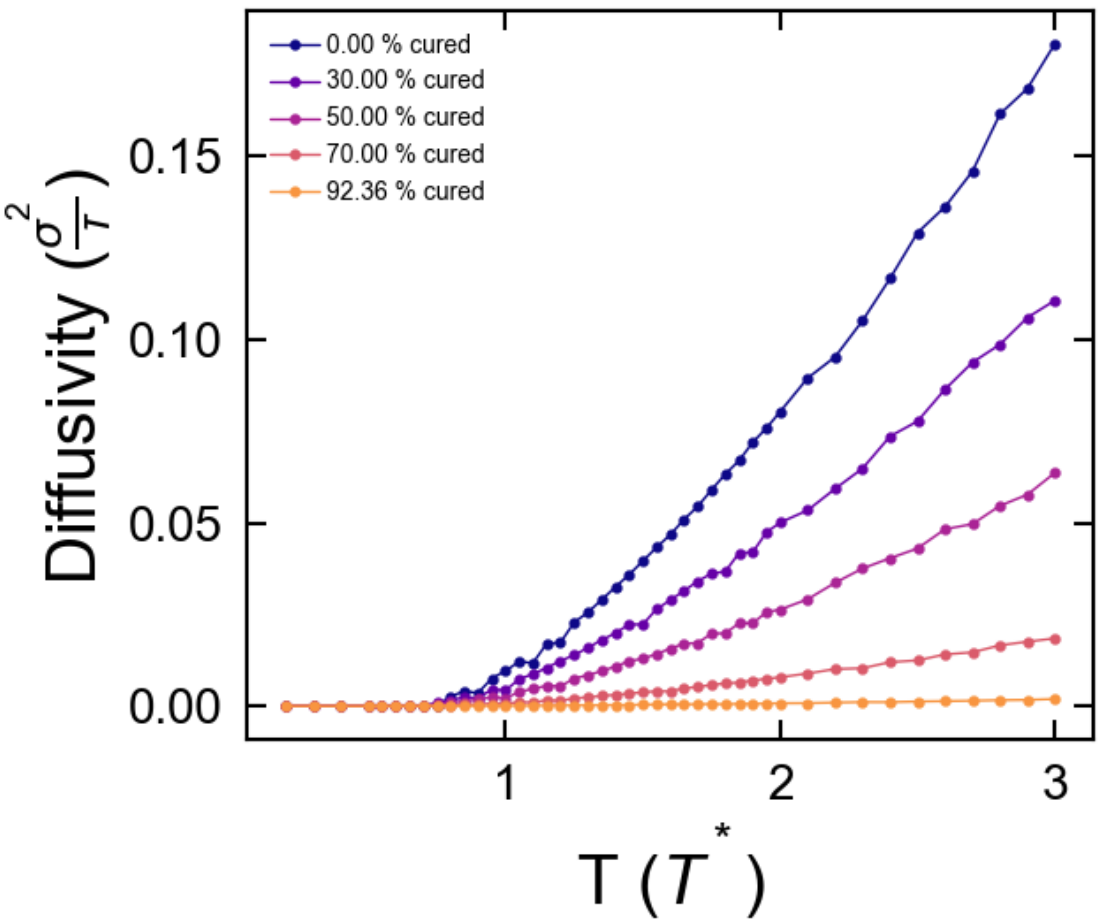
In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *

def H0(x,x0,c):
        return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

def hyper(x,x0,y0,a,b,c):
    #a=0.0
    return y0+a*(x-x0)+b*H0(x,x0,c)

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,x0,a,b,c):
    #a=0.0
    return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

def get_tangent(x,x0,y0,a,b,c):
    m=slope_of_tangent(x,x0,a,b,c)
```

```
            y=hyper(x,x0,y0,a,b,c)
            b=y-(m*x)
            return m,b
```

In [3]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0,30.,50.,70.,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)


        plt.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=1.0,
                 label='{:.2f} % cured'.format(cure_percent))


plt.legend(fontsize=10)
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
plt.xlabel('T ($T^*$)')
plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
savefig(plt,'diffusivity_rawdata','all_alphas.pdf')
#savefig(plt,'hyperbola_fitting_unbounded','all_alphas.pdf')
plt.show()
```

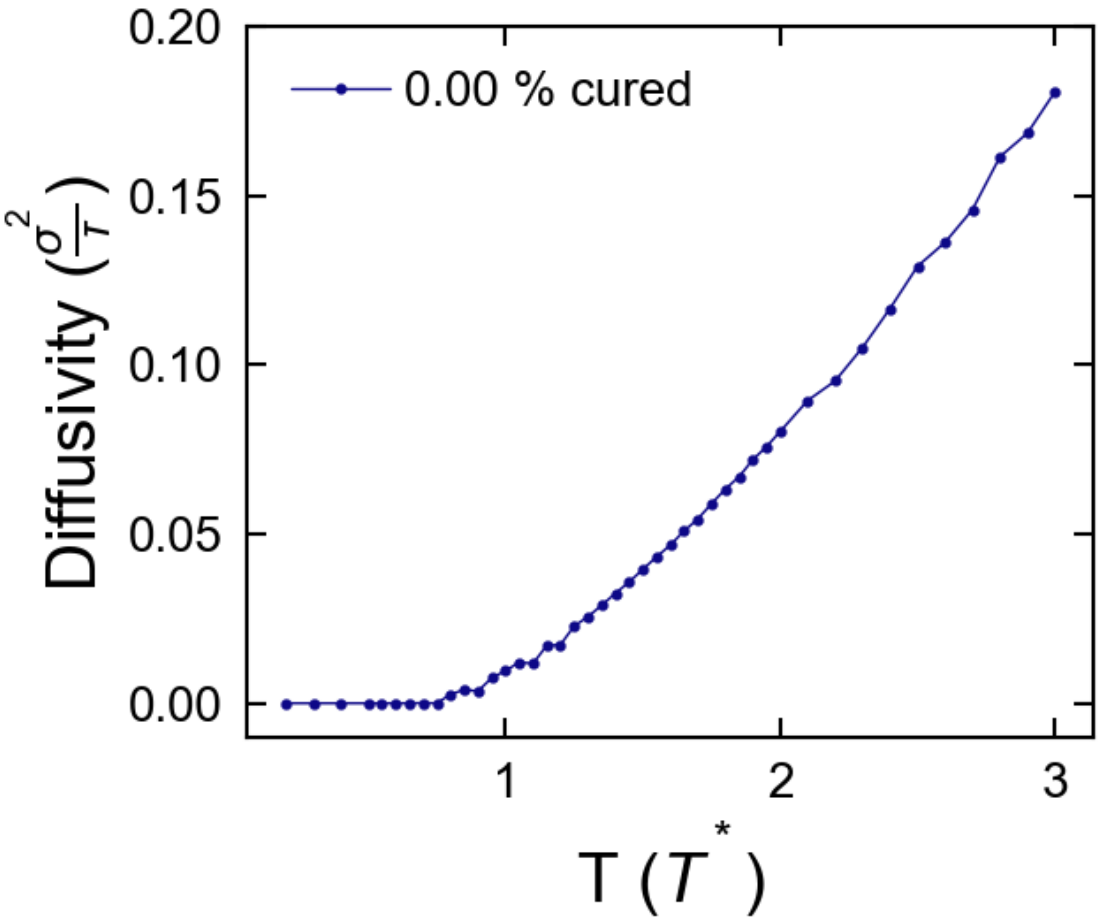
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [18]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.0]#,30.,50.,70.,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=3.0)&
                        (df.quench_T>=0.2)&
         (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
         for i,sap in enumerate(filter_saps):
             cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
             for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                 df_curing = df_grp[(df_grp.bond==False)&
```

```
                    (df_grp.calibrationT==305)&
                    (df_grp.cooling_method==cooling_method)&
                    (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)


            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=1.0,
                     label='{:.2f} % cured'.format(cure_percent))


    plt.legend(fontsize=20)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    plt.xlabel('T ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    plt.ylim(-0.01,0.2)
    savefig(plt,'diffusivity_rawdata','0_alpha.pdf')
    #savefig(plt,'hyperbola_fitting_unbounded','all_alphas.pdf')
    plt.show()
```

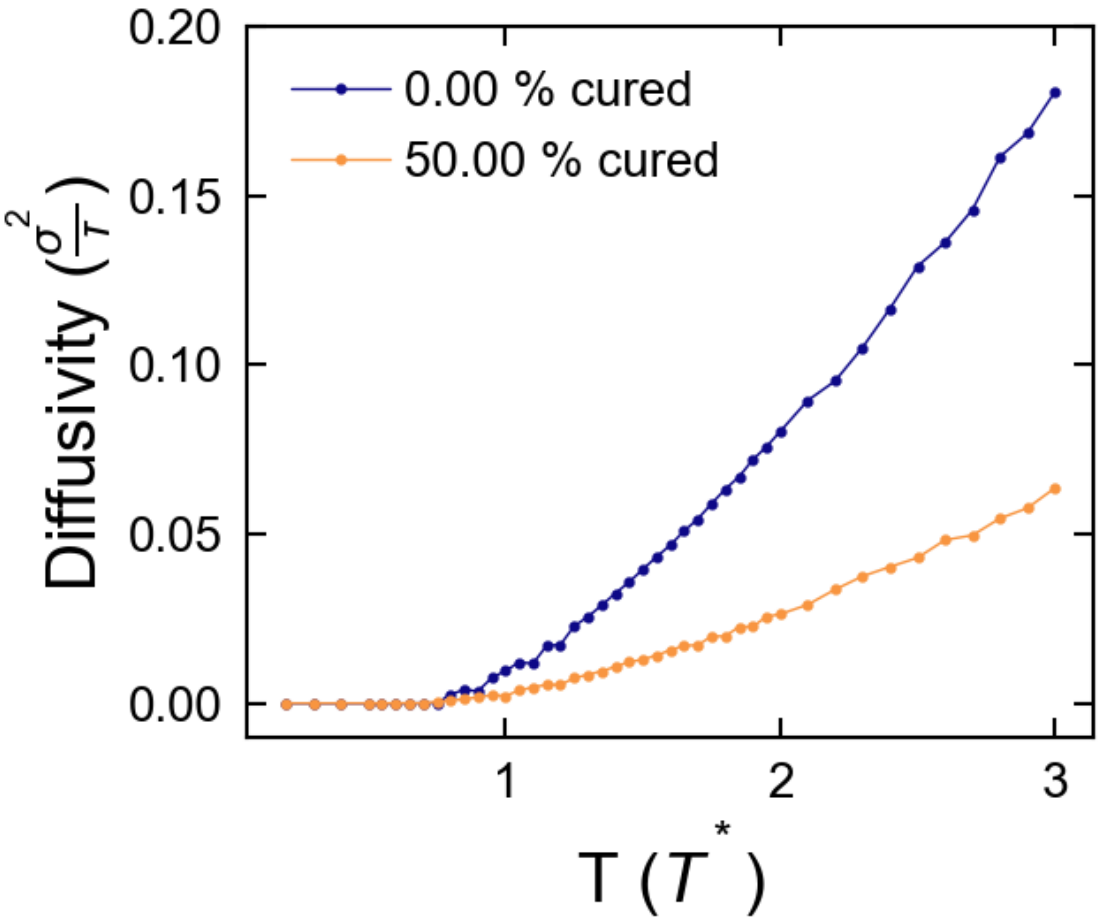
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [20]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.0,50.0]#,30.,50.,70.,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=3.0)&
                        (df.quench_T>=0.2)&
         (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
         for i,sap in enumerate(filter_saps):
             cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
             for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                 df_curing = df_grp[(df_grp.bond==False)&
```

```
                    (df_grp.calibrationT==305)&
                    (df_grp.cooling_method==cooling_method)&
                    (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)


            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=1.0,
                     label='{:.2f} % cured'.format(cure_percent))


    plt.legend(fontsize=20)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    plt.xlabel('T ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    plt.ylim(-0.01,0.2)
    savefig(plt,'diffusivity_rawdata','50_alpha.pdf')
    #savefig(plt,'hyperbola_fitting_unbounded','all_alphas.pdf')
    plt.show()
```

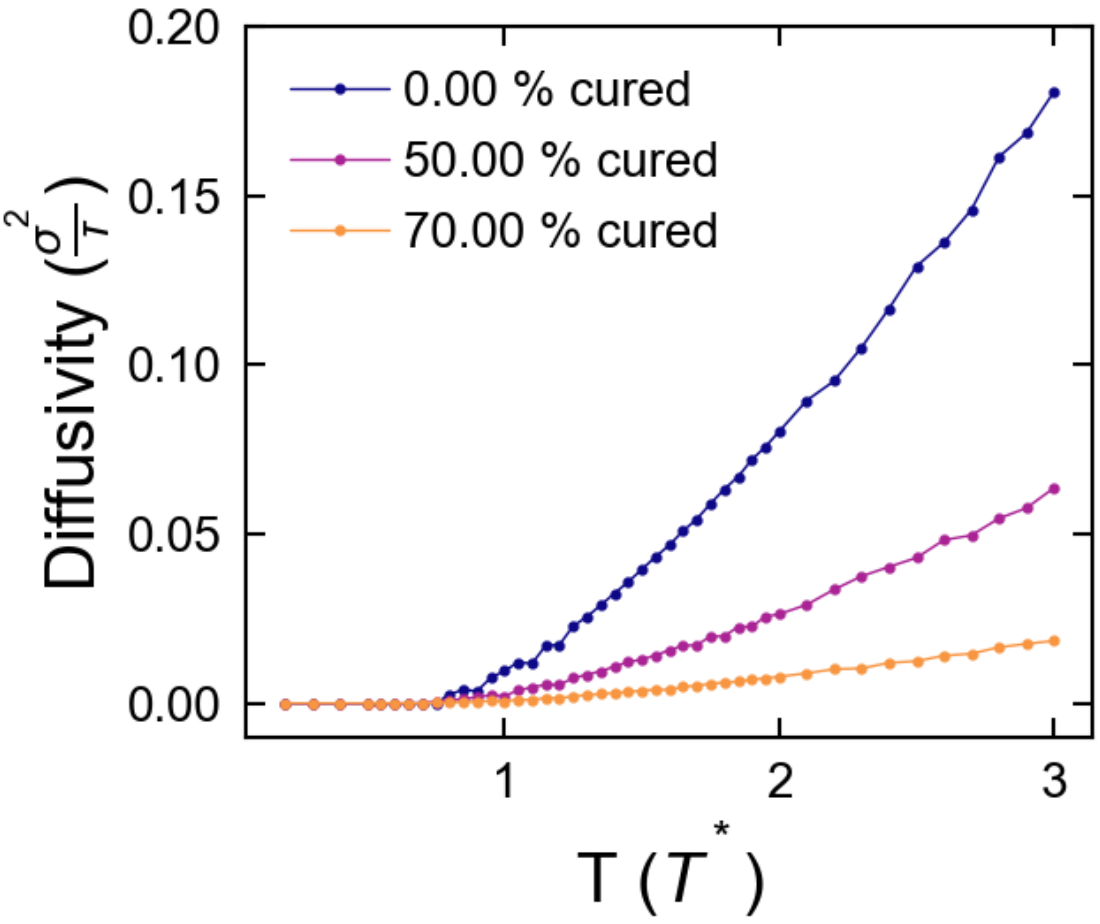
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [21]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.0,50.0,70]#,30.,50.,70.,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=3.0)&
                        (df.quench_T>=0.2)&
         (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
         for i,sap in enumerate(filter_saps):
             cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
             for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                 df_curing = df_grp[(df_grp.bond==False)&
```

```
                    (df_grp.calibrationT==305)&
                    (df_grp.cooling_method==cooling_method)&
                    (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)


            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=1.0,
                     label='{:.2f} % cured'.format(cure_percent))


    plt.legend(fontsize=20)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    plt.xlabel('T ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    plt.ylim(-0.01,0.2)
    savefig(plt,'diffusivity_rawdata','100_alpha.pdf')
    #savefig(plt,'hyperbola_fitting_unbounded','all_alphas.pdf')
    plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


0.20
0.15
0.10
0.05
0.00
Diffusivity ($\frac{\sigma^2}{T}$)
0.00 % cured
50.00 % cured
70.00 % cured
1
2
3
T (T^*)

In [8]:
```
from common import *
import numpy as np
```

1 Parameters for neat DGEBA/DDS

In [9]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 1.7
N=50001
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
activation_energy=2.0
density=1.0
quench_time=5e6
P = 6.0#[4.5, 6, 8]
stop_after_percents = np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.,40.,50.]#[60.,70.,
80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
```

In [10]:
```
#data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_kestrel/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_fry/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_glass_transition/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_step_quench/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Tg/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

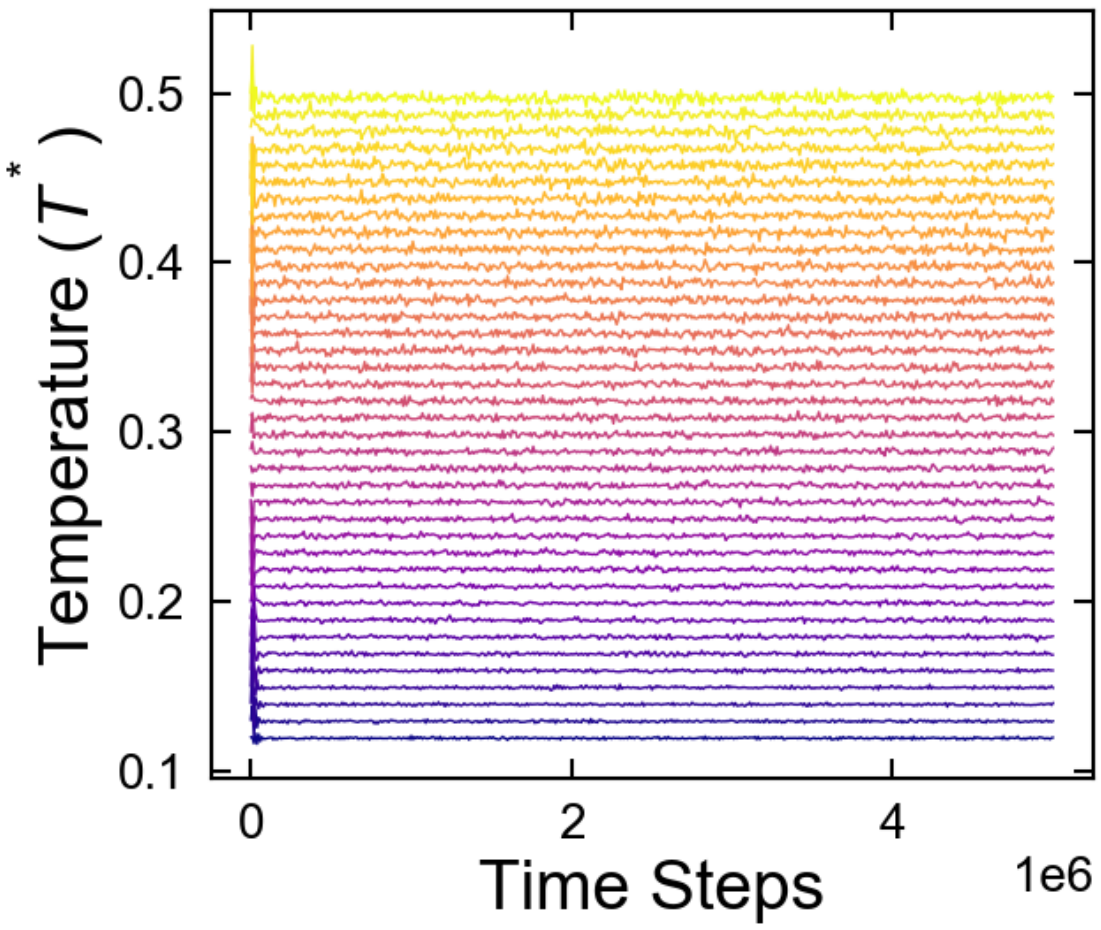
In [11]:
```
df_filtered = df[(df.bond==True)&
                 (df.cure_percent>80)]
```

```
        print('FO_model_R2',df_filtered.FO_model_R2.mean())
        print('SAFO_model_R2',df_filtered.SAFO_model_R2.mean())
        print('SO_model_R2',df_filtered.SO_model_R2.mean())
        print('SASO_model_R2',df_filtered.SASO_model_R2.mean())
        print('SASO_model_R2',df_filtered.SASO_model_R2.mean())
        #df_filtered.temp_prof
```

```
FO_model_R2 0.991424533763
SAFO_model_R2 -1.57121114976
SO_model_R2 0.894395915679
SASO_model_R2 0.992131186264
SASO_model_R2 0.992131186264
```

```
In [12]: PROP_NAME ='temperature'#'potential_energy'#'volume'


        df_filtered = df[(df.integrator==integrator) &
                         (df.kT==kT) &
                         (df.quench_T<=0.5)&
                         (df.quench_T>=0.1)&
                         (df.quench_time ==quench_time)&
                         (df.tauP == tauP)&
                         (df.n_particles==N)&
                         (df.density==density)&
                         (df.activation_energy==activation_energy)&
                         (df.P==P)&
                         (df.cooling_method==cooling_method)&
                         (df.pot=='LangH')&
                         (df.stop_after_percent==30.0)]

        plot_equilibriation(df_filtered,
                            project,
                            PROP_NAME,
                            draw_equilibrium_window=False,
                            mean_from_second_half=True)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        plt.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        plt.xlabel('Time Steps')
        plt.ylabel('Temperature ($T^*$)')
        savefig(plt,
                'toy_lj_Tg',
               'quench_temperature.png')
        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



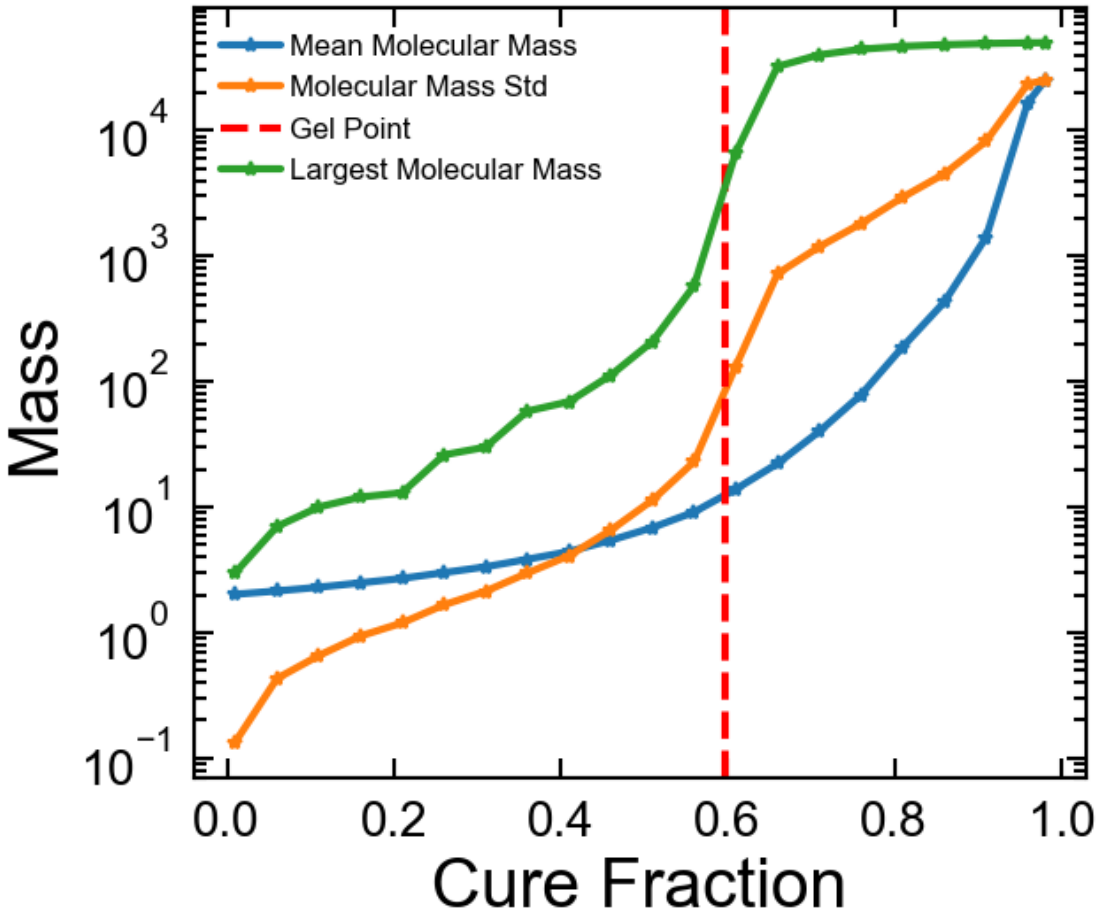
In [13]:
```
### import numpy as np
import gsd
import gsd.fl
import gsd.hoomd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib
#plt.style.use('ggplot')
#%matplotlib inline


def find_indices(lst, condition):
    return [i for i, elem in enumerate(lst) if condition(elem)]

def save_network_data(job,nSamples=20):
    '''
        Performs network analysis on the final frame and saves important info in job
document.
    '''
    gsd_path=job.fn('data.gsd')
    #print(job)
    f = gsd.fl.GSDFile(gsd_path, 'rb')
    t = gsd.hoomd.HOOMDTrajectory(f)
    n_frames = len(t)
    print('total frames ',n_frames)
    last_frame = n_frames-1
```

```
        cluster_distributions = []
        fig = plt.figure(figsize=(12,9))
        time_conv = job.sp.dcd_write
        snapshot = t[int(last_frame)]
        bonds = snapshot.bonds.group
        G = nx.MultiGraph(ts='time_step:{}, kT:{}'.format(last_frame*time_conv,job.sp.kT))
        bond_types = [snapshot.bonds.types[i] for i in snapshot.bonds.typeid]
        #print(bond_types)
        ab_indices = find_indices(bond_types, lambda e: e == 'A-B')
        ab_bonds = [bonds[i] for i in ab_indices]
        for bond in ab_bonds:
            G.add_edge(bond[0],bond[1])
        sorted_cc = [len(c) for c in sorted(nx.connected_components(G), key=len,
    reverse=False)]
        num_clusters=len(sorted_cc)
        if len(sorted_cc) > 0:
            largest_network=sorted_cc[-1]
            if len(sorted_cc)==1:
                second_largest_network=sorted_cc[-1]
            else:
                second_largest_network=sorted_cc[-2]
            average_network_len=np.mean(sorted_cc)
        else:
            largest_networks.append(0)
            second_largest_networks.append(0)
        n, bins, patches = plt.hist(sorted_cc, 50, normed=1, facecolor='green', alpha=0.75)
        plt.xlabel('Cluster Length',fontsize=40)
        plt.ylabel('Probability',fontsize=40)
        #plt.plot(time_steps,
    average_network_lens,color='m',linestyle='-',marker='o',label='average cluster')
        plt.savefig(job.fn('clusters_hist.png'),transparent=True)
        job.document['largest_network']=largest_network
        job.document['second_largest_network']=second_largest_network
        job.document['average_network']=average_network_len
        job.document['num_clusters']=num_clusters

In [14]: df_filtered = df[(df.bond==True)]
    df_sorted = df_filtered.sort_values('cure_percent')
    #df_sorted.plot(x='cure_percent',y='average_network')
    df_100 = df_filtered[df_filtered.stop_after_percent==100]
    print(df_100['curing_at_gel_point'][0])
    plt.plot(df_sorted['cure_percent']/100,
                 df_sorted['average_network'],
                 #df_sorted['network_size_std'],
             marker='*',
             label='Mean Molecular Mass')
    plt.plot(df_sorted['cure_percent']/100,
                 df_sorted['network_size_std'],
             marker='*',
             label='Molecular Mass Std')
    plt.axvline(x=df_100['curing_at_gel_point'][0]/100,
                color='r',
                linestyle='--',
               label='Gel Point')
    plt.plot(df_sorted['cure_percent']/100,
             df_sorted['largest_network'],
             marker='*',
             label='Largest Molecular Mass')
    #plt.plot(df_sorted['cure_percent'],
    #         df_sorted['second_largest_network'],
    #         marker='*',
    #         label='Second Largest Molecular Mass')
    plt.yscale('log')
    plt.xlabel('Cure Fraction')
    plt.ylabel('Mass')
    plt.legend(fontsize=12)
    savefig(plt,
           'toy_lj_Tg',
```

```
            'gel_point_and_molecular_mass.pdf')
        #df_sorted.plot(x='cure_percent',y='largest_network')
```

```
59.73030472
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

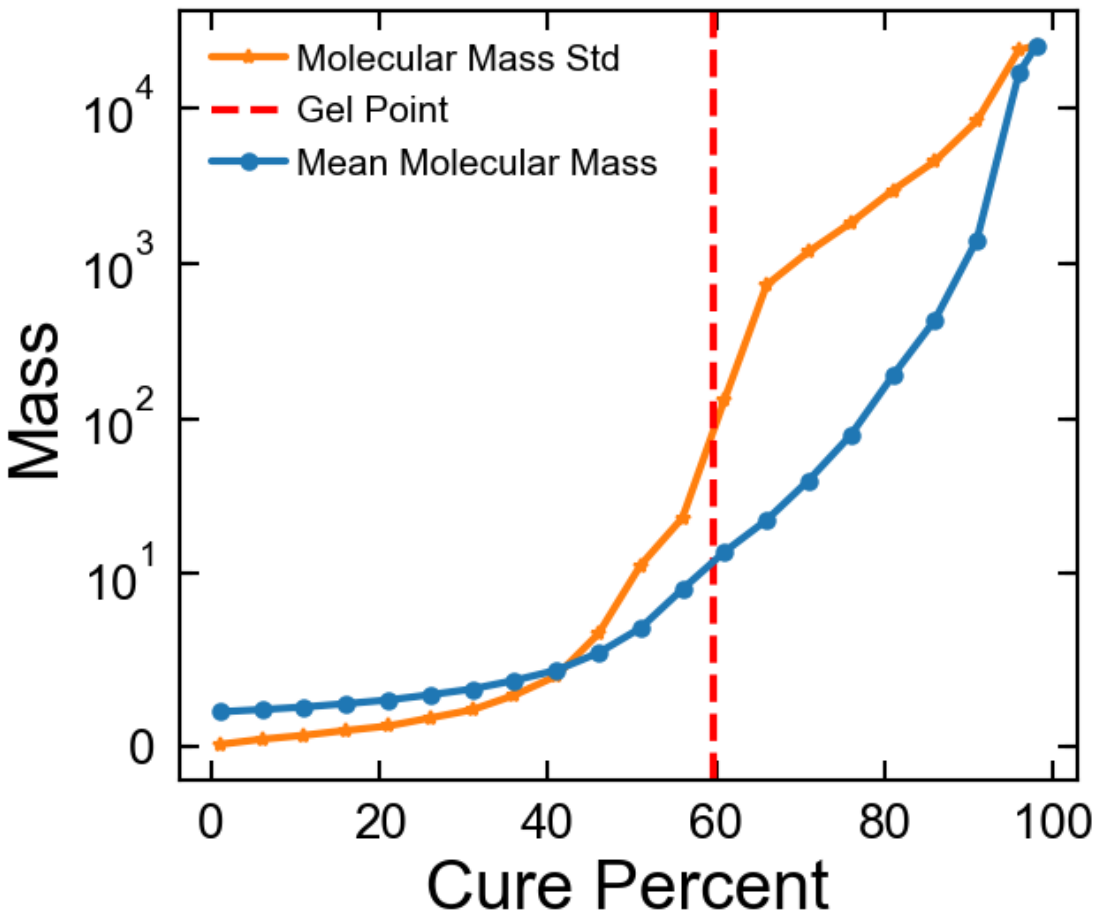


In [15]:
```
df_filtered = df[(df.bond==True)]
df_sorted = df_filtered.sort_values('cure_percent')
#df_sorted.plot(x='cure_percent',y='average_network')
df_100 = df_filtered[df_filtered.stop_after_percent==100]
print(df_100['curing_at_gel_point'][0])
#print(df_sorted['average_network'])
#print(df_sorted['network_size_std'])
plt.errorbar(df_sorted['cure_percent'],
             df_sorted['average_network'],
             #np.round(df_sorted['network_size_std']),
             marker='o',
```

```
                #fmt='o',
                #capthick=2,
                capsize=5,
                label='Mean Molecular Mass')
        plt.plot(df_sorted['cure_percent'],
                 df_sorted['network_size_std'],
                marker='*',
                label='Molecular Mass Std')
        plt.axvline(x=df_100['curing_at_gel_point'][0],
                    color='r',
                    linestyle='--',
                   label='Gel Point')
        #plt.plot(df_sorted['cure_percent'],
        #         df_sorted['largest_network'],
        #         marker='*',
        #         label='Largest Molecular Mass')
        #plt.plot(df_sorted['cure_percent'],
        #         df_sorted['second_largest_network'],
        #         marker='*',
        #         label='Second Largest Molecular Mass')
        plt.yscale('symlog',linthreshy=10)#,nonposy="mask")
        plt.xlabel('Cure Percent')
        plt.ylabel('Mass')
        #plt.ylim(1e-10,1e6)
        plt.legend(fontsize=15)
        savefig(plt,
               'toy_lj_Tg',
               'gel_point_and_molecular_mass_only.png')
        #df_sorted.plot(x='cure_percent',y='largest_network')
```

```
59.73030472
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

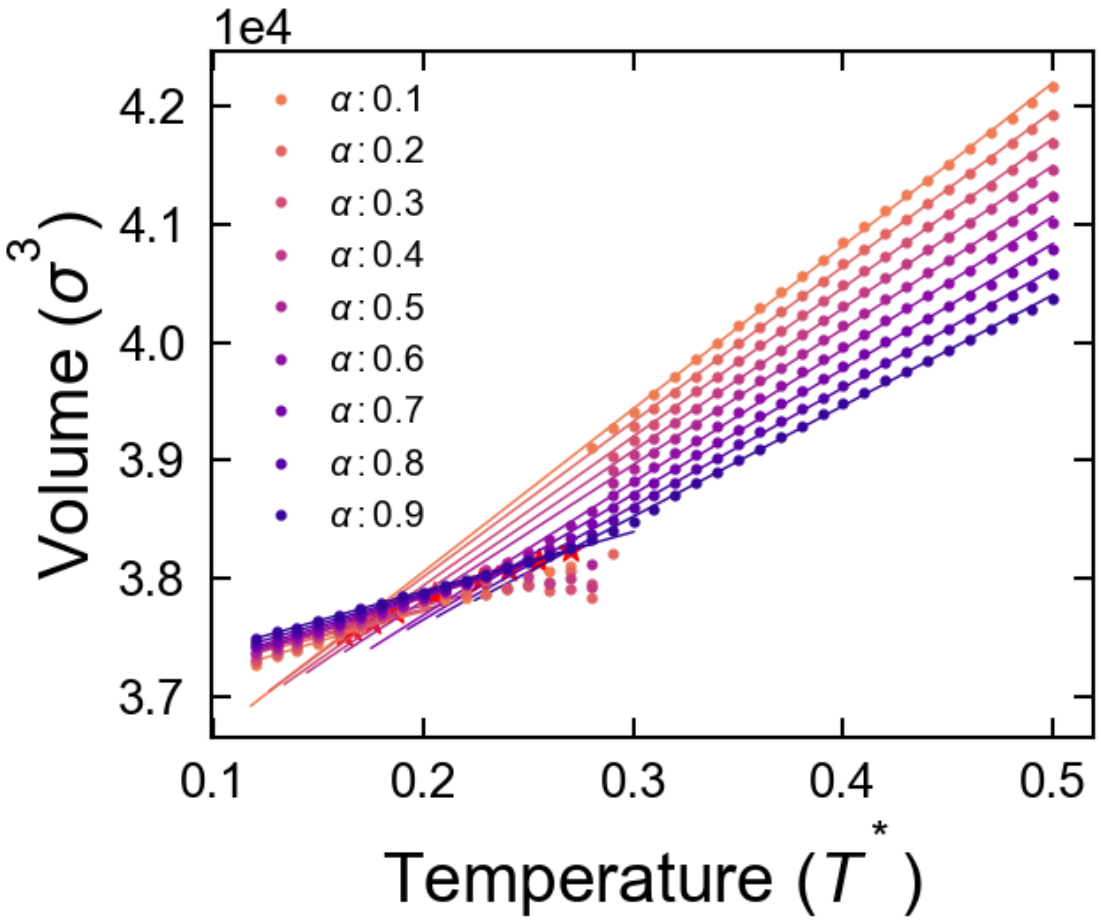


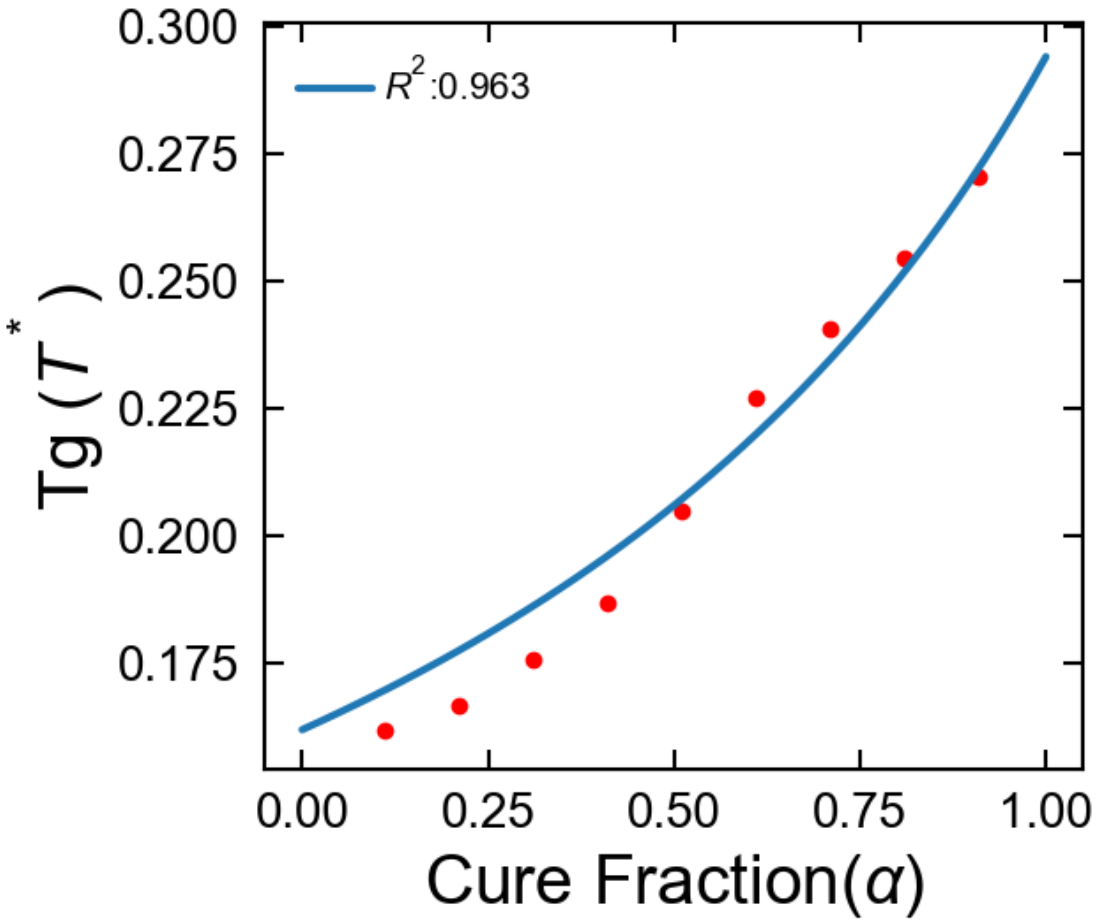

```
In [268]: from piecewise.regressor import piecewise
          #https://www.datadoghq.com/blog/engineering/piecewise-regression/
          from piecewise.plotter import plot_data_with_regression
          #stop_after_percents = [10,100.]
          plt.figure(0)
          colors = plt.cm.plasma(np.linspace(0.75,0,len(stop_after_percents)))
          Tgs=[]
          cure_percents = []
          for i,sap in enumerate(stop_after_percents):
              #plt.figure()
              filter_saps =
          [10.,20.,30.,40.,50.,60.,70.,80.,90.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
              if sap not in filter_saps:
                  continue
              df_filtered = df[(df.integrator==integrator) &
                               (df.kT==kT) &
                               (df.quench_T<=0.5)&
                               (df.quench_T>=0.1)&
                               (df.quench_time ==quench_time)&
                               (df.tauP == tauP)&
                               (df.n_particles==N)&
                               (df.density==density)&
                               (df.activation_energy==activation_energy)&
                               (df.P==P)&
                               (df.num_c10==num_c10)&
                               (df.cooling_method==cooling_method)&
```

```
                    (df.pot=='LangH')&
                    (df.stop_after_percent==sap)]
    cure_percents.append(df_filtered.cure_percent.values[0])
    prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
    quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                          project,
                                                          prop,
                                                          mean_from_second_half=True)

    #plot_data_with_regression(quenchTs, mean_vals)
    if True:
        model = piecewise(quenchTs, mean_vals)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in regression!')
    plt.errorbar(quenchTs,
                 mean_vals,
                 val_stds,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha: {}$'.format(sap/100),
                 linewidth=0.,
                zorder=1)#1.5)
    Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
    #print(Tg)
    Tgs.append(Tg)
    show_transition = True
    if show_transition:
        #plt.axvline(x=Tg,
        #            linewidth=1.0,
        #            color=colors[i])
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=100,
                    color='r',
                   zorder=0)#colors[i])
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

plt.xlabel('Temperature ($T^*$)')
plt.ylabel('Volume ($\\sigma^3$)')
#plt.xlim(0.11,0.51)
#plt.ylim(34000,43000)
plt.legend(fontsize=15)
savefig(plt,
```

```
                'toy_lj_Tg',
               'V_qT.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
        plt.figure(1)
        cure_percents = np.asarray(cure_percents)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction($\\alpha$)')
        plt.ylabel('Tg ($T^*$)')
        plt.legend(fontsize=15)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        np.savetxt('DGEBA_DDS_Tg_quench.txt',np.transpose(Tg_data))
        #plt.xlim(0.25,0.35)
        #plt.ylim(-0.005,0.01)
        savefig(plt,
               'toy_lj_Tg',
               'dibeneditto_from_V_all_alphas.pdf')
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
T1 0.294042430255 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e4
4.2
4.1
4.0
3.9
3.8
3.7
Volume (σ^3)
α:0.1
α:0.2
α:0.3
α:0.4
α:0.5
α:0.6
α:0.7
α:0.8
α:0.9
0.1
0.2
0.3
0.4
0.5
Temperature (T^*)



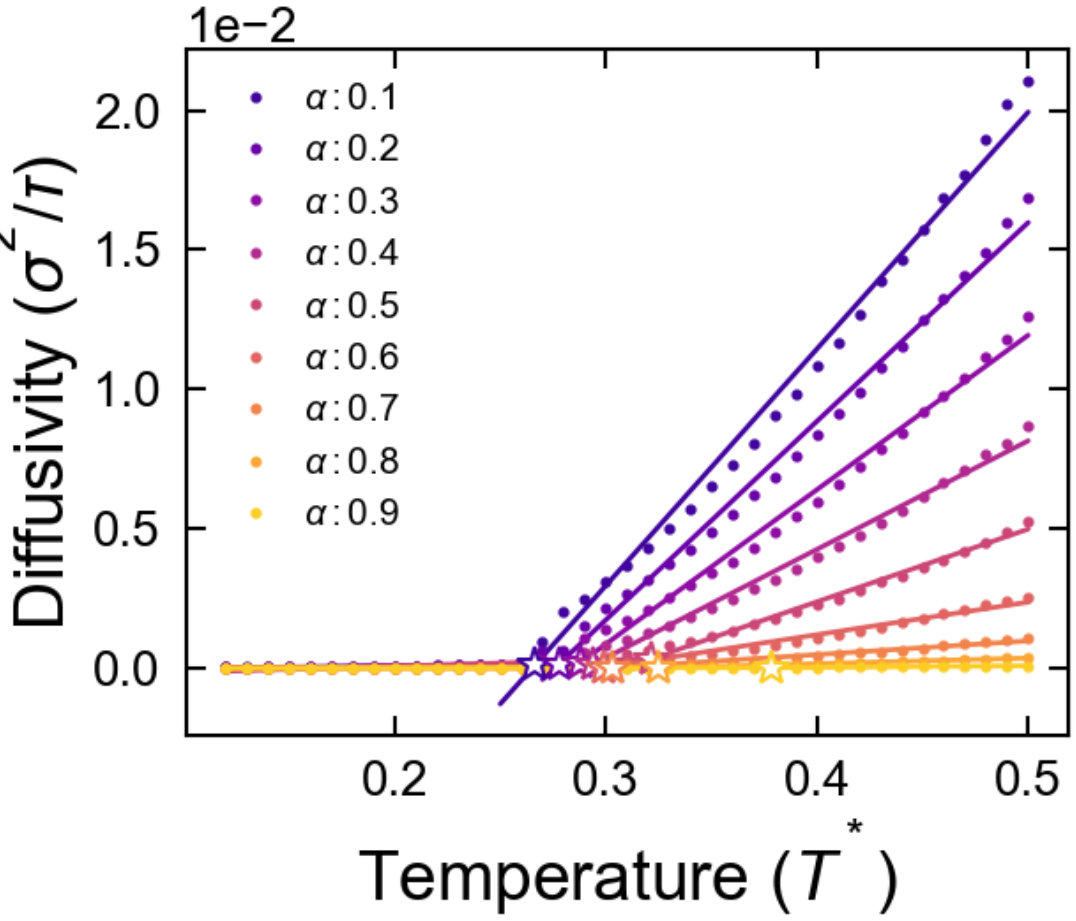

```
In [270]: colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
          Tgs=[]
          cure_percents = []
          PROP_NAME='bparticles'
          plt.figure(0)
          for i,sap in enumerate(stop_after_percents):
              filter_saps =[10,20,30.,40,50,60.,70,80,90]#
          [10.,20.,30.,40.,50.,60.,70.,80.,90.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
              if sap not in filter_saps:
                  continue
              df_filtered = df[(df.integrator==integrator) &
                               (df.kT==kT) &
                               (df.quench_T<=0.5)&
                               (df.quench_T>=0.1)&
                               (df.quench_time ==quench_time)&
                               (df.tauP == tauP)&
                               (df.n_particles==N)&
                               (df.density==density)&
                               (df.activation_energy==activation_energy)&
                               (df.P==P)&
                               (df.cooling_method==cooling_method)&
                               (df.pot=='LangH')&
                               (df.stop_after_percent==sap)]
              df_filtered.quench_T
              Ts,Ds=getDiffusivities(project,
                                     df_filtered,
```

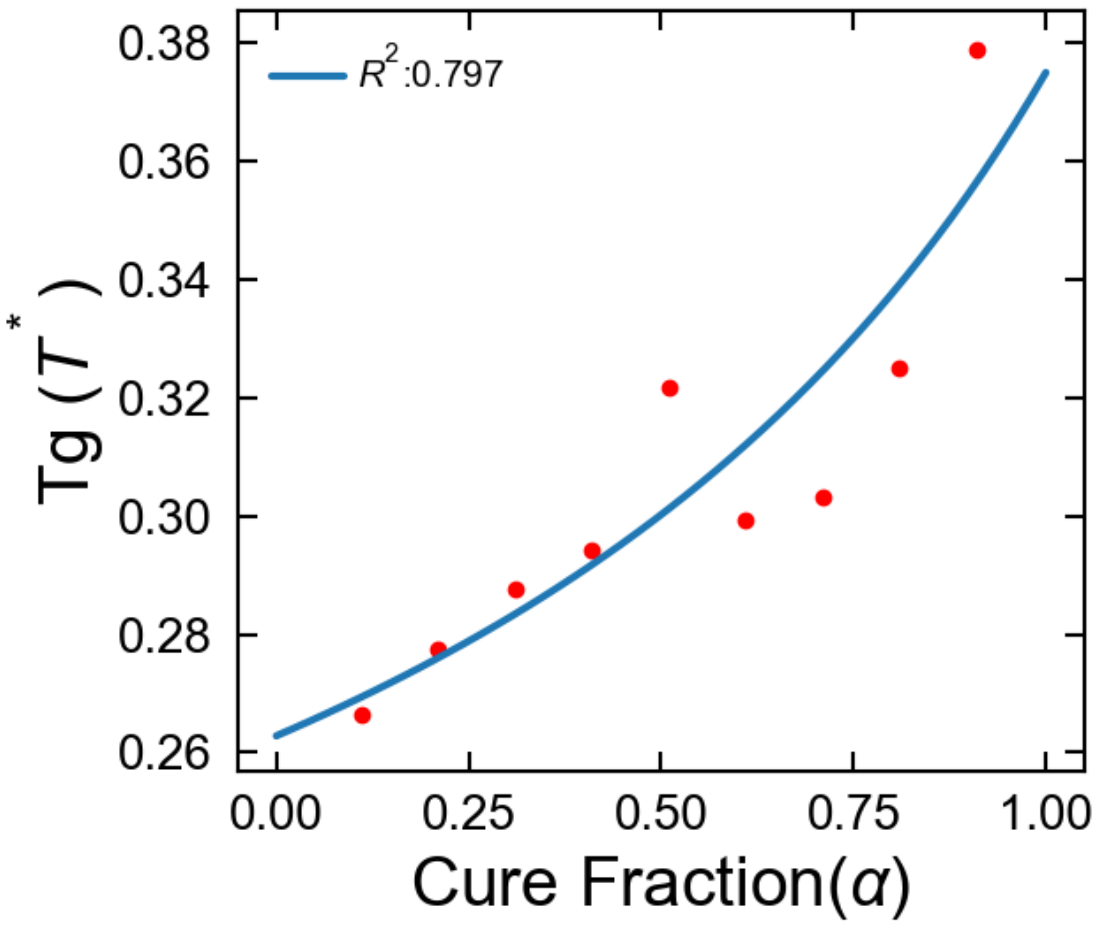
```
                        name=PROP_NAME,
                        quench_time=quench_time,
                       equilibriated_ts_percentage=0.0)
    plt.plot(Ts,
             Ds,
             marker='.',
             color=colors[i],
             label='$\\alpha: {}$'.format(sap/100),
             linewidth=0.,
             zorder=1)#1.5)
    try:
        mul_fact=1.0
        Ds_scaled = Ds*mul_fact
        line_vals=[]

        Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                  Ds_scaled,
                                  method='intersection',
                                  min_D=0,
                                  ver=1)#,
                                  #viscous_line_index=0,
                                  #l1_T_bounds=custom_ranges_l1[sap],
                                  #l2_T_bounds=custom_ranges_l2[sap])
        line_vals.append((x1,y1))
        line_vals.append((x2,y2))
        xs = Ts#np.linspace(0.1,4)

        plt.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 markerfacecolor='w',
                 color=colors[i],#cooling_colors[j],
                 markersize=15)#,
        l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            plt.plot(xs,
                     ys,
                     color=colors[i],#cooling_colors[j],
                     zorder=1,
                     linewidth=2)
        Tgs.append(Tg)
        cure_percents.append(df_filtered.cure_percent.values[0])

    except Exception as e:
        print(e)

plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
plt.xlabel('Temperature ($T^*$)')
plt.ylabel('Diffusivity ($\\sigma^2/\\tau$)')
#plt.xscale('log')
#plt.yscale('log')
plt.legend(fontsize=15)
savefig(plt,
       'toy_lj_Tg',
       'D_vs_qT_all_alphas.pdf')
plt.figure(1)
cure_percents = np.asarray(cure_percents)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
R2,fit_Tgs,T1,inter_parm,T0 =
fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=None)
print('T1',T1,'lambda',inter_parm)
#plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
print(cure_percents_ss)
alphas = np.linspace(0,1)
```

```
        fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction($\\alpha$)')
        plt.ylabel('Tg ($T^*$)')
        plt.legend(fontsize=15)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        np.savetxt('DGEBA_DDS_Tg_quench.txt',np.transpose(Tg_data))
        #plt.xlim(0.25,0.35)
        #plt.ylim(-0.005,0.01)
        savefig(plt,
               'toy_lj_Tg',
               'dibeneditto_from_D_all_alphas.pdf')
        plt.show()
```

```
T1 0.374930396559 lambda 0.5
[ 11.00078011  21.00107956  31.00138092  41.00017929  51.00048065
  61.0007782   71.00108337  81.00138092  91.00018311]
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```






```
In [271]: from scipy.interpolate import InterpolatedUnivariateSpline
          from piecewise.regressor import piecewise

          plt.figure()
          fig,ax1 = plt.subplots()
          #ax2 = ax1.twinx()
          colors = plt.cm.plasma(np.linspace(0,0.75,len(stop_after_percents)))
          for i,sap in enumerate(stop_after_percents):
              filter_saps = [40.,60.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
              if sap not in filter_saps:
                  continue
              df_filtered = df[(df.integrator==integrator) &
                               (df.kT==kT) &
                               (df.quench_T<=0.5)&
                               (df.quench_T>=0.1)&
                               (df.quench_time ==quench_time)&
                               (df.tauP == tauP)&
                               (df.n_particles==N)&
                               (df.density==density)&
                               (df.activation_energy==activation_energy)&
                               (df.P==P)&
                               (df.cooling_method==cooling_method)&
```

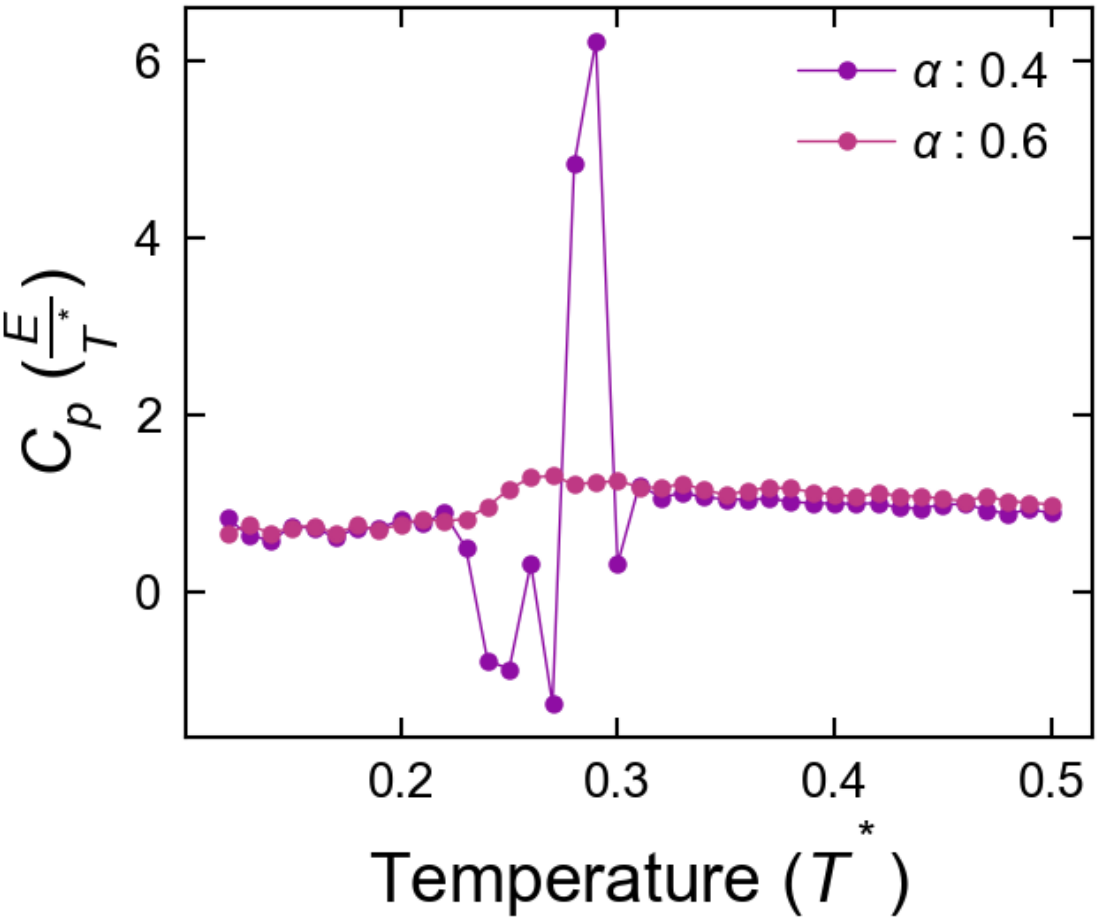
```
                    (df.pot=='LangH')&
                    (df.stop_after_percent==sap)]
    prop='potential_energy'#'potential_energy'#'pair_lj_energy'#'
    quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                          project,
                                                          prop,
                                                          mean_from_second_half=True)
    mean_vals = np.asarray(mean_vals)/N
    quenchTs = np.asarray(quenchTs)
    # Get a function that evaluates the linear spline at any quench T
    f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
    dUdT = f.derivative(n=1)
    d2UdT2 = f.derivative(n=2)
    dU_dTs = dUdT(quenchTs)
    d2U_dT2s = d2UdT2(quenchTs)
    regression = False
    if regression:
        model = piecewise(quenchTs, dU_dTs)
        #print(model)
        if len(model.segments) >= 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            extend_lines = False
            if extend_lines:
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,ys,color=colors[i],zorder=0,linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                ax1.plot(xs,ys,color=colors[i],zorder=0,linewidth=1)
            else:
                xs = np.linspace(l1.start_t,l1.end_t)
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,ys,color=colors[i],zorder=0,linewidth=1)
                xs = np.linspace(l2.start_t,l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                ax1.plot(xs,ys,color=colors[i],zorder=0,linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in regression!')

    Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
    print(Tg)
    #ax1.axvline(x=Tg,linewidth=0.1)
    if True:
        ax1.plot(quenchTs,
                 dU_dTs,
                 color=colors[i],
                 marker='o',
                 label='$\\alpha$ : {}'.format(sap/100),
                 linewidth=1.0,
                 zorder=1)#1.5)
    if False:
        ax2.plot(quenchTs,
                 d2U_dT2s,
                 color=colors[i],
                 #marker='.',
                 label='{}% second derivative'.format(sap),
                 linestyle='--',
                 linewidth=1.0,
                 zorder=4)#1.5)
ax1.set_xlabel('Temperature ($T^*$)')
ax1.set_ylabel('$C_p\ (\\frac{E}{T^*})$')
```

```
        #ax2.set_ylabel(prop+' (2nd der)')
        #ax1.set_ylim(0.2,1.1)
        #ax1.set_ylim(-1e5,1.0e6)
        #plt.ylim(36000,42000)
        ax1.legend(fontsize=20,loc='best')
        #ax2.legend(fontsize=15)
        savefig(plt,
                'toy_lj_Tg',
                'Cp_asymm.pdf')
        plt.show()
```

```
using line iftting
0.166422210935
using line iftting
0.222495256256
```

```
<matplotlib.figure.Figure at 0x117d99358>
```

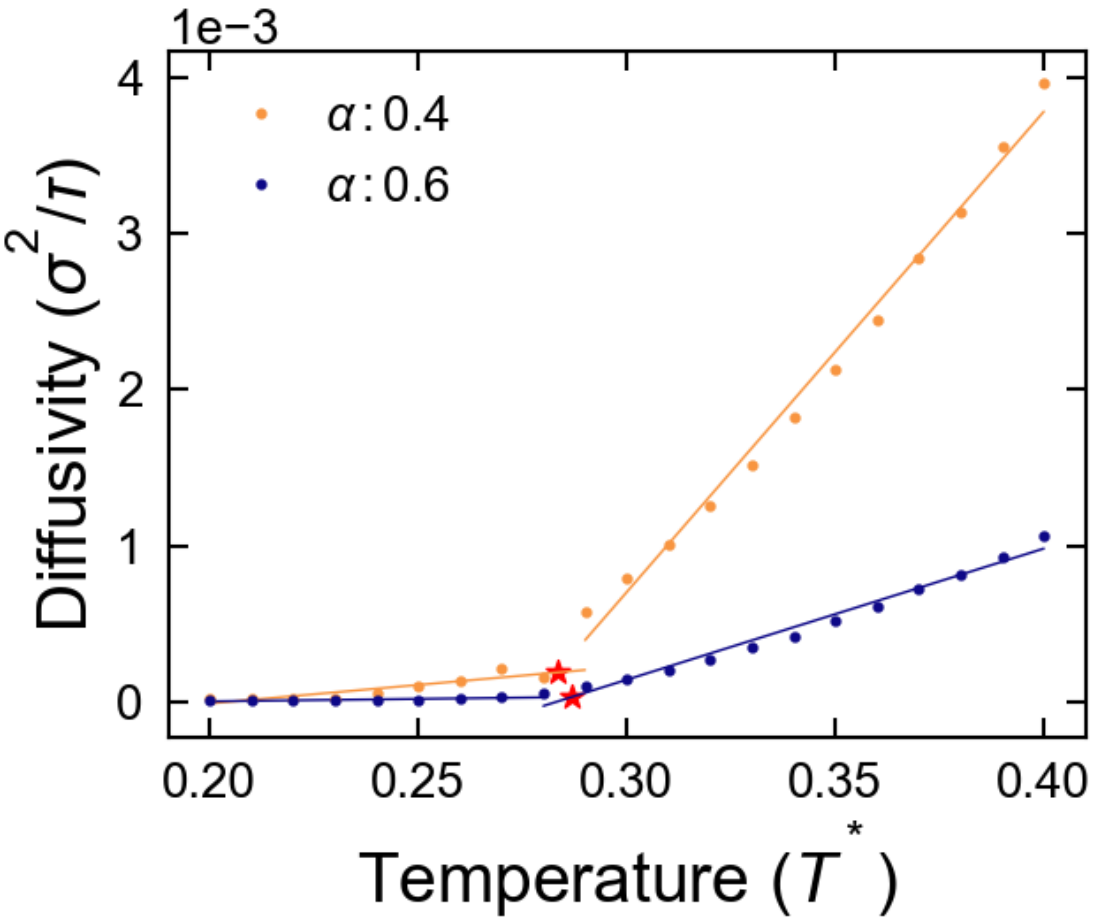
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [12]:
```
Tgs=[]
cure_percents = []
PROP_NAME='bparticles'
plt.figure(0)
filter_saps =[40,60.]#
[10.,20.,30.,40.,50.,60.,70.,80.,90.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
colors = plt.cm.plasma(np.linspace(0.75,0.,len(filter_saps)))

for i,sap in enumerate(filter_saps):
    if sap not in filter_saps:
        continue
    df_filtered = df[(df.integrator==integrator) &
                     (df.kT==kT) &
                     (df.quench_T<=0.4)&
                     (df.quench_T>=0.2)&
                     (df.quench_time ==quench_time)&
                     (df.tauP == tauP)&
                     (df.n_particles==N)&
                     (df.density==density)&
                     (df.activation_energy==activation_energy)&
                     (df.P==P)&
                     (df.cooling_method==cooling_method)&
                     (df.pot=='LangH')&
                     (df.stop_after_percent==sap)]
    df_filtered.quench_T
    Ts,Ds=getDiffusivities(project,
                           df_filtered,
                           name=PROP_NAME,
                           quench_time=quench_time,
                          equilibriated_ts_percentage=0.0)
    plt.plot(Ts,
             Ds,
             marker='.',
             color=colors[i],
             label='$\\alpha: {}$'.format(sap/100),
             linewidth=0.,
             zorder=2)#1.5)
    try:
        mul_fact=1.0
        Ds_scaled = Ds*mul_fact
        line_vals=[]

        Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                  Ds_scaled,
                                  method='intersection',
                                  min_D=0,
                                  ver=1)#,
                                  #viscous_line_index=0,
                                  #l1_T_bounds=custom_ranges_l1[sap],
                                  #l2_T_bounds=custom_ranges_l2[sap])
        line_vals.append((x1,y1))
        line_vals.append((x2,y2))
        xs = Ts#np.linspace(0.1,4)

        plt.scatter(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 #markerfacecolor='w',
                 zorder=0,
                 color='r',#colors[i],#cooling_colors[j],
                 s=100)#,
        l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
```

```
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=colors[i],#cooling_colors[j],
                         zorder=1,
                         linewidth=1)
            Tgs.append(Tg)
            cure_percents.append(df_filtered.cure_percent.values[0])

        except Exception as e:
            print(e)

    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    plt.xlabel('Temperature ($T^*$)')
    plt.ylabel('Diffusivity ($\\sigma^2/\\tau$)')

    #plt.xscale('log')
    #plt.yscale('log')
    plt.legend(fontsize=20)
    savefig(plt,
           'toy_lj_Tg',
           'D_vs_qT_subset.pdf')

    plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

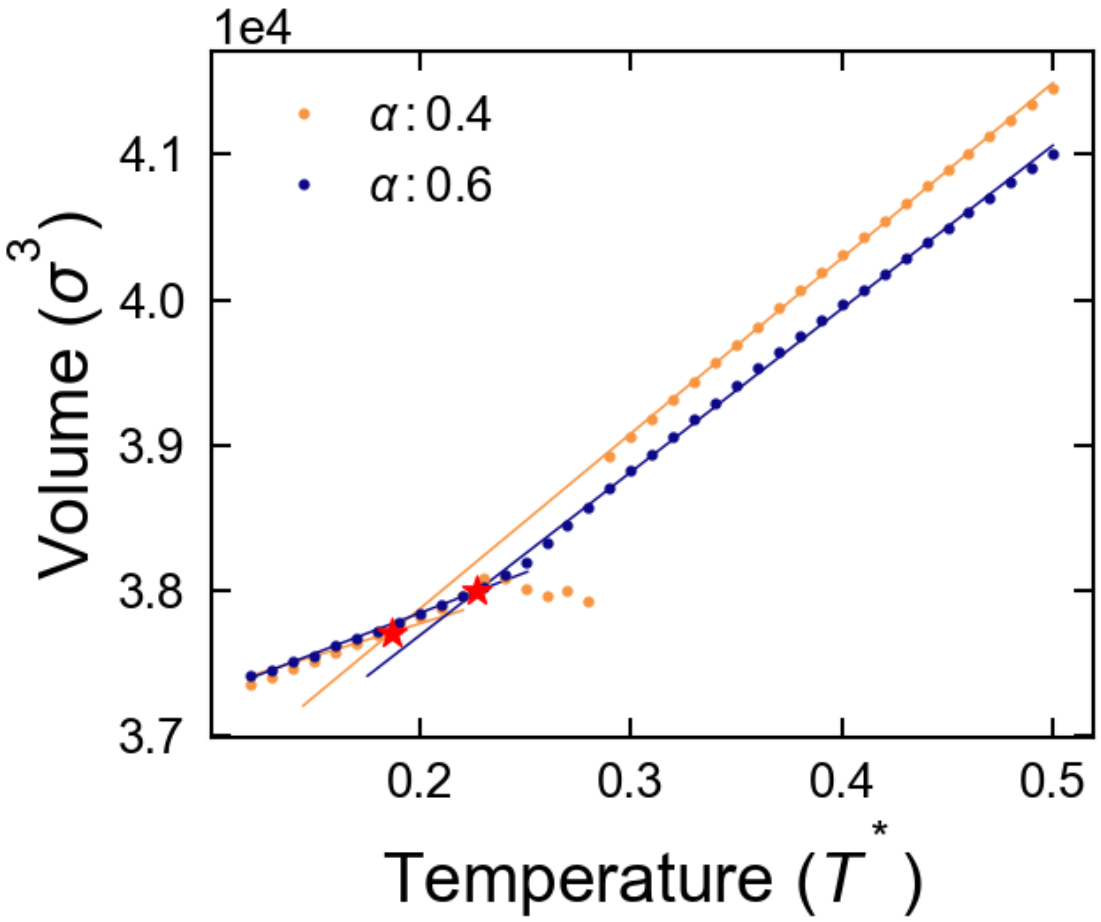



```
In [25]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = [10,100.]
         filter_saps = [40.,60.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         plt.figure()
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.5)&
                              (df.quench_T>=0.1)&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.num_c10==num_c10)&
                              (df.cooling_method==cooling_method)&
                              (df.pot=='LangH')&
                              (df.stop_after_percent==sap)]
             prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
```

```
    quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                          project,
                                                          prop,
                                                          mean_from_second_half=True)

    #plot_data_with_regression(quenchTs, mean_vals)
    if True:
        model = piecewise(quenchTs, mean_vals)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in regression!')
    plt.errorbar(quenchTs,
                 mean_vals,
                 val_stds,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha: {}$'.format(sap/100),
                 linewidth=0.,
                zorder=1)#1.5)
    Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
    print(Tg)
    show_transition = True
    if show_transition:
        #plt.axvline(x=Tg,
        #            linewidth=1.0,
        #            color=colors[i])
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=150,
                    color='r',
                   zorder=2)#colors[i])
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

plt.xlabel('Temperature ($T^*$)')
plt.ylabel('Volume ($\\sigma^3$)')
#plt.xlim(0.11,0.51)
#plt.ylim(34000,43000)
plt.legend(fontsize=20)
savefig(plt,
        'toy_lj_Tg',
       'V_qT_subset.pdf')
#plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
#plt.show()
```

```
using line iftting
0.186699085968
using line iftting
0.227163285334
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

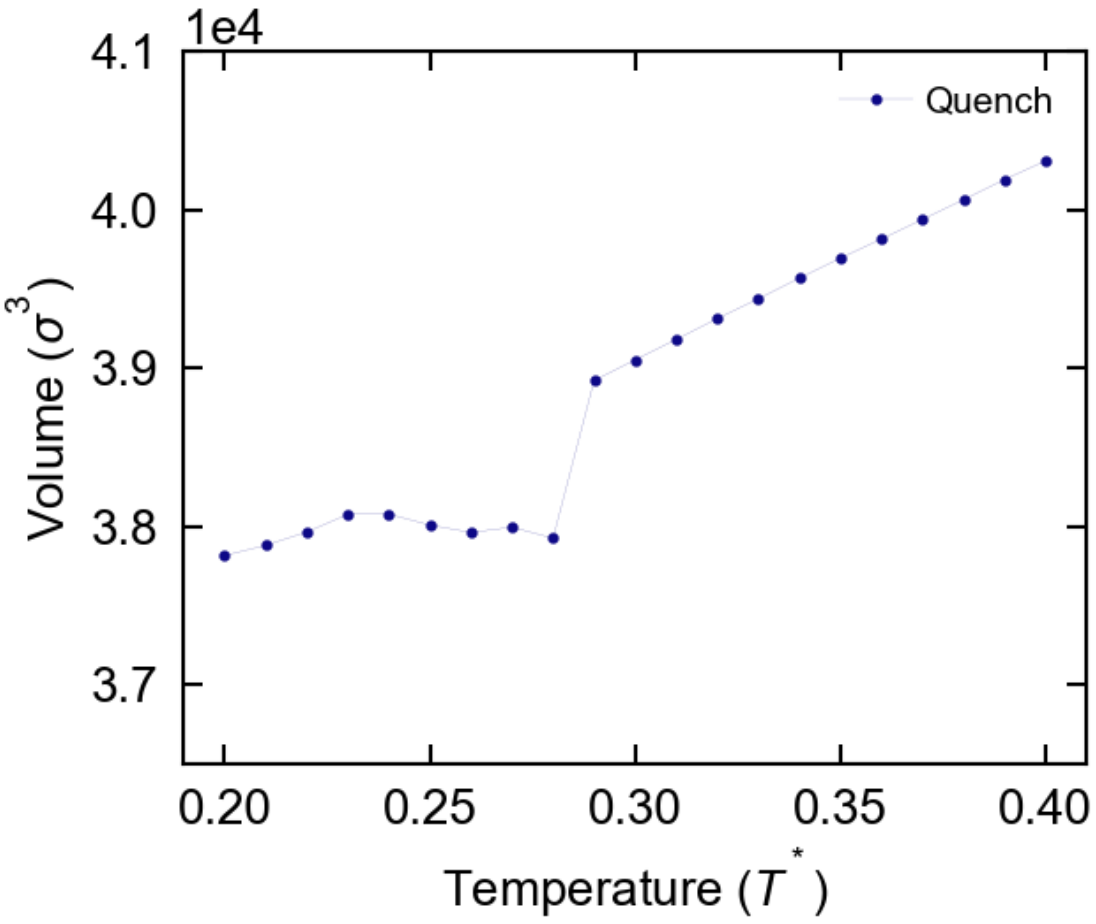


In [18]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = [10,100.]
filter_saps = [40]#[30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]

colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
for i,sap in enumerate(filter_saps):
    #plt.figure()
    plt.figure()
    if sap not in filter_saps:
```

```
    continue
df_filtered = df[(df.integrator==integrator) &
                 (df.kT==kT) &
                 (df.quench_T<=0.4)&
                 (df.quench_T>=0.2)&
                 (df.quench_time ==quench_time)&
                 (df.tauP == tauP)&
                 (df.n_particles==N)&
                 (df.density==density)&
                 (df.activation_energy==activation_energy)&
                 (df.P==P)&
                 (df.num_c10==num_c10)&
                 (df.cooling_method==cooling_method)&
                 (df.pot=='LangH')&
                 (df.stop_after_percent==sap)]
prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                      project,
                                                      prop,
                                                      mean_from_second_half=True)

#plot_data_with_regression(quenchTs, mean_vals)
print(df_filtered.quench_T)
if False:
    model = piecewise(quenchTs, mean_vals)
    #print(model)
    if len(model.segments) == 2:
        lines = []
        l1 = model.segments[0]
        m1 = l1.coeffs[1]
        b1 = l1.coeffs[0]
        l2 = model.segments[1]
        m2 = l2.coeffs[1]
        b2 = l2.coeffs[0]
        x,y = line_intersect(m1,b1,m2,b2)
        xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
        ys = l1.coeffs[1]*xs+l1.coeffs[0]
        plt.plot(xs,
                 ys,
                 color=colors[i],
                 zorder=0,
                 linewidth=1)
        xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
        ys = l2.coeffs[1]*xs+l2.coeffs[0]
        plt.plot(xs,
                 ys,
                 color=colors[i],
                 zorder=0,
                 linewidth=1)
    else:
        print('WARNING: found more or less than 2 line segments in regression!')
plt.plot(quenchTs,
             mean_vals,
             #val_stds,
             marker='.',
             color=colors[i],
             label='Quench',
             linewidth=0.1,
            zorder=1)#1.5)
Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
print(Tg)
show_transition = False
if show_transition:
    #plt.axvline(x=Tg,
    #            linewidth=1.0,
    #            color=colors[i])
    plt.scatter(Tg,
                Tg_prop,
```

```
                        marker='*',
                        s=150,
                        color='r',
                       zorder=2)#colors[i])
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

            plt.xlabel('Temperature ($T^*$)',fontsize=20)
            plt.ylabel('Volume ($\\sigma^3$)',fontsize=20)
            #plt.xlim(0.11,0.51)
            plt.ylim(36500,41000)
            plt.legend(fontsize=15,loc='upper right')
            savefig(plt,
                    'toy_lj_Tg',
                   'V_qT_alpha_{}.pdf'.format(sap))
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
006e689136046ee927a95f672d7b45d1    0.28
aafcd1103162e708caa93fe98dc214ad     0.2
a19cda4ec60c6b5c10984c389a9f3490    0.35
be53972dca8d86c72ddbc2a09b7d6eeb    0.24
8ef13c434b7059afad003e08401b1492    0.31
8892f927e06640cf3a64d11a9f6be15a    0.36
939db733770e3df3b280d8ddaaa519a9    0.21
94458a937c44f673ac7882f659a3889c    0.34
96dcc69ca4335e16d03c30f7fab716f9    0.25
e55801d6b25dddce5c5399b1be84074c    0.23
e9b82071200a1a5a34b22d5468086c87    0.32
f57bc4f4831e789fd62358656f77cf3d    0.26
ca3815ee80c37c97a7b2ce509e1260fc    0.29
e01e586c166191c37eef0c02d8bd50c2    0.33
3ebe1fa65ad9c2dcb041ebb716eb9bd2     0.3
211f73439e427d74ed4e94d7024478ef     0.4
05aa9fd31e4d683781ddbb766a48fe51    0.39
805eb99fd0d138691ec4a37177749ecb    0.38
4cd64c680c148ffa54e0ea37dda217bf    0.22
47f36539477cfd80197f8cb5af1fd9b5    0.27
48405bf03656c5ac27672f72c9392bdd    0.37
Name: quench_T, dtype: object
using line iftting
0.211023836647


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



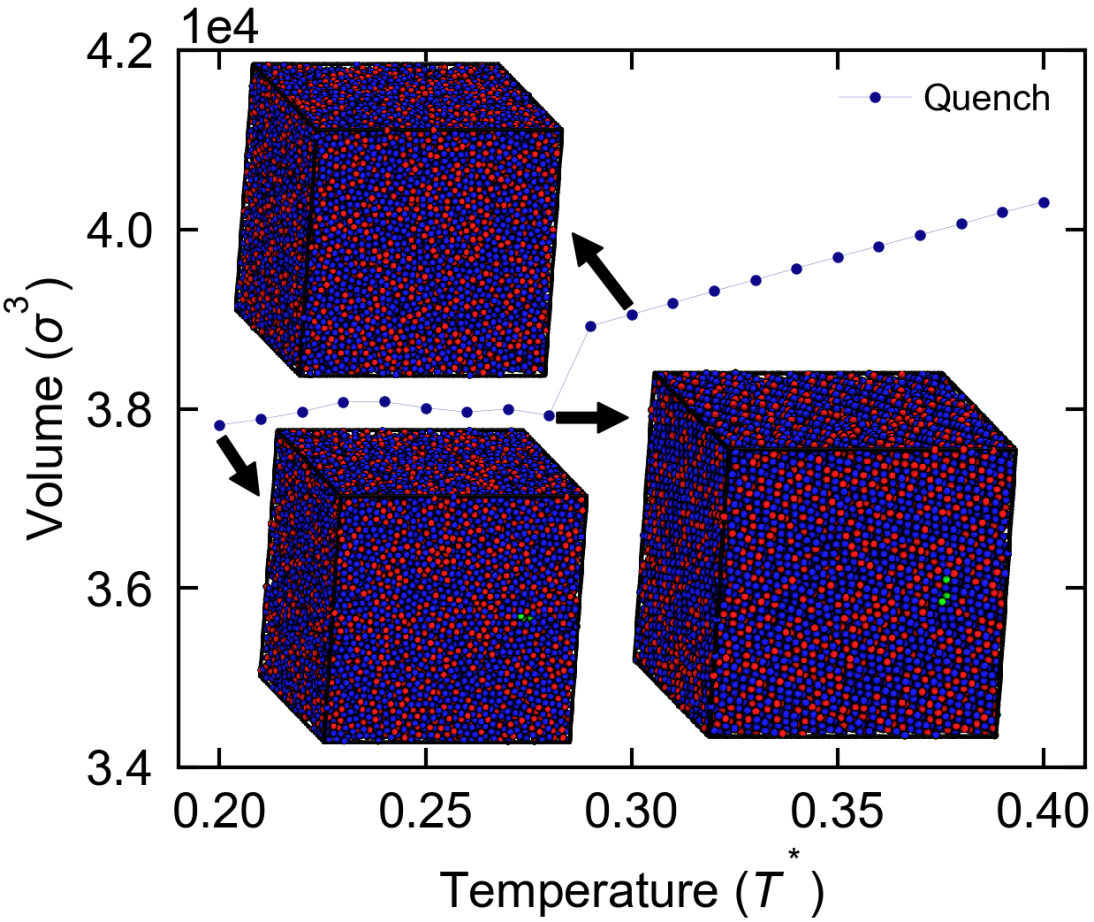

```
In [88]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = [10,100.]
         filter_saps = [40]#[30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]

         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             #plt.figure()
             fig = plt.figure(dpi=200)
             ax1 = fig.add_subplot(111)
             left, bottom, width, height = [0.25, 0.235, 0.3, 0.3]#max t2
             ax2 = fig.add_axes([left, bottom, width, height])
             ax2.axis('off')
             left, bottom, width, height = [0.55, 0.24, 0.35, 0.35]#max t2
             ax3 = fig.add_axes([left, bottom, width, height])
             ax3.axis('off')
             left, bottom, width, height = [0.23, 0.58, 0.3, 0.3]#max t2
             ax4 = fig.add_axes([left, bottom, width, height])
             ax4.axis('off')
             figure_axes=[ax2,ax3,ax4]
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.4)&
```

```
                        (df.quench_T>=0.2)&
                        (df.quench_time ==quench_time)&
                        (df.tauP == tauP)&
                        (df.n_particles==N)&
                        (df.density==density)&
                        (df.activation_energy==activation_energy)&
                        (df.P==P)&
                        (df.num_c10==num_c10)&
                        (df.cooling_method==cooling_method)&
                        (df.pot=='LangH')&
                        (df.stop_after_percent==sap)]
            prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
            quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                                  project,
                                                                  prop,
                                                                  mean_from_second_half=True)

            #plot_data_with_regression(quenchTs, mean_vals)
            print(df_filtered.quench_T)
            ax1.plot(quenchTs,
                     mean_vals,
                     #val_stds,
                     marker='.',
                     color=colors[i],
                     label='Quench',
                     linewidth=0.1,
                    zorder=1)#1.5)
            Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
            print(Tg)

            qTs=[0.2,0.28,0.30]
            for j,qT in reversed(list(enumerate(qTs))):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    im = plt.imread(job.fn('final_snapshot.png'))
                    #figure_axes[j].set_title('T: {:.2f}
        $T^*$'.format(job.sp.quench_T),fontsize=9)
                    figure_axes[j].imshow(im)
            ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            ax1.set_xlabel('Temperature ($T^*$)',fontsize=20)
            ax1.set_ylabel('Volume ($\\sigma^3$)',fontsize=20)
            #plt.xlim(0.11,0.51)
            ax1.set_ylim(34000,42000)
            ax1.legend(fontsize=15,loc='upper right')
            #ax1.arrow(0.2, 3.78e4, 0.01, -5e2,shape='full', lw=0.3, length_includes_head=True)
            ax1.annotate('', xy=(0.21, 3.7e4), xytext=(0.2,
        3.77e4),arrowprops=dict(facecolor='black', shrink=0.05))
            ax1.annotate('', xy=(0.3, 3.79e4), xytext=(0.281,
        3.79e4),arrowprops=dict(facecolor='black', shrink=0.05))
            ax1.annotate('', xy=(0.285, 4e4), xytext=(0.30,
        3.91e4),arrowprops=dict(facecolor='black', shrink=0.05))
            savefig(plt,
                    'toy_lj_Tg',
                   'V_qT_img_alpha_{}.pdf'.format(sap))
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
006e689136046ee927a95f672d7b45d1    0.28
aafcd1103162e708caa93fe98dc214ad     0.2
a19cda4ec60c6b5c10984c389a9f3490    0.35
be53972dca8d86c72ddbc2a09b7d6eeb    0.24
8ef13c434b7059afad003e08401b1492    0.31
8892f927e06640cf3a64d11a9f6be15a    0.36
939db733770e3df3b280d8ddaaa519a9    0.21
94458a937c44f673ac7882f659a3889c    0.34
96dcc69ca4335e16d03c30f7fab716f9    0.25
```

```
e55801d6b25dddce5c5399b1be84074c    0.23
e9b82071200a1a5a34b22d5468086c87    0.32
f57bc4f4831e789fd62358656f77cf3d    0.26
ca3815ee80c37c97a7b2ce509e1260fc    0.29
e01e586c166191c37eef0c02d8bd50c2    0.33
3ebe1fa65ad9c2dcb041ebb716eb9bd2     0.3
211f73439e427d74ed4e94d7024478ef     0.4
05aa9fd31e4d683781ddbb766a48fe51    0.39
805eb99fd0d138691ec4a37177749ecb    0.38
4cd64c680c148ffa54e0ea37dda217bf    0.22
47f36539477cfd80197f8cb5af1fd9b5    0.27
48405bf03656c5ac27672f72c9392bdd    0.37
Name: quench_T, dtype: object
using line iftting
0.211023836647
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



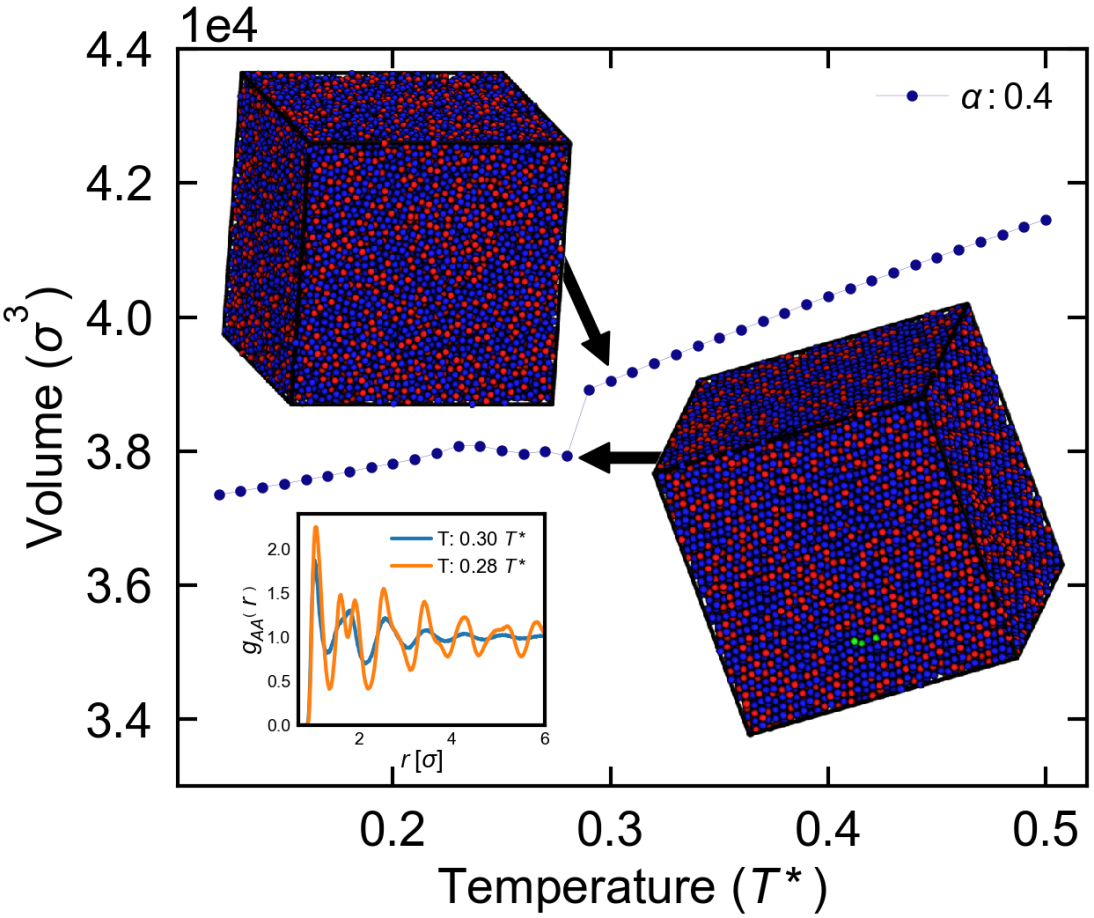
In [285]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
from scipy import ndimage
#stop_after_percents = [10,100.]
filter_saps = [40]#[30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
#plt.figure()
fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.53, 0.22, 0.45, 0.45]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.22, 0.55, 0.32, 0.32]#max t2
ax3 = fig.add_axes([left, bottom, width, height])
ax3.axis('off')

left, bottom, width, height = [0.3, 0.25, 0.2, 0.2]#max t2
ax4 = fig.add_axes([left, bottom, width, height])
#ax4.axis('off')
figure_axes=[ax2,ax3]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
for i,sap in enumerate(filter_saps):

    if sap not in filter_saps:
        continue
    df_filtered = df[(df.integrator==integrator) &
                     (df.kT==kT) &
                     (df.quench_T<=0.5)&
                     (df.quench_T>=0.12)&
                     (df.quench_time ==quench_time)&
                     (df.tauP == tauP)&
                     (df.n_particles==N)&
                     (df.density==density)&
                     (df.activation_energy==activation_energy)&
                     (df.P==P)&
                     (df.num_c10==num_c10)&
                     (df.cooling_method==cooling_method)&
                     (df.pot=='LangH')&
                     (df.stop_after_percent==sap)]
    prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
    quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                          project,
                                                          prop,
                                                          mean_from_second_half=True)

    #plot_data_with_regression(quenchTs, mean_vals)
    #print(df_filtered.quench_T)
    ax1.plot(quenchTs,
                 mean_vals,
                 #val_stds,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha: {}$'.format(sap/100),#label='Quench',
                 linewidth=0.1,
                zorder=1)#1.5)

    if False:
        model = piecewise(quenchTs, mean_vals)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
```

```
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            ax1.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            ax1.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in regression!')

    Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
    print(Tg)
    show_transition = False
    if show_transition:
        #plt.axvline(x=Tg,
        #            linewidth=1.0,
        #            color=colors[i])
        ax1.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=150,
                    color='r',
                   zorder=2)#colors[i])

    qTs=[0.28,0.30]
    rotateby=[-70,0]
    df_sorted = df_filtered.sort_values('quench_T')
    for j,qT in reversed(list(enumerate(qTs))):
        df_xstal = df_sorted[(df_sorted.quench_T==qT)]
        for jobid in df_xstal.index:
            job=project.open_job(id=jobid)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, rotateby[j],prefilter=False)
            #figure_axes[j].set_title('T: {:.2f}
$T^*$'.format(job.sp.quench_T),fontsize=9)
            figure_axes[j].imshow(rotated_img)

            job=project.open_job(id=jobid)
            print(job.isfile('final.hoomdxml'),job)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax4.plot(r,
                     gr,
                     label='T: {:.2f} $T*$'.format(job.sp.quench_T),
                    linewidth=1.5)
    ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    ax1.set_xlabel('Temperature ($T*$)',fontsize=20)
    ax1.set_ylabel('Volume ($\\sigma^3$)',fontsize=20)
    #plt.xlim(0.11,0.51)
    ax1.set_ylim(33000,44000)
    ax1.legend(fontsize=15,loc='upper right')
    #ax1.arrow(0.2, 3.78e4, 0.01, -5e2,shape='full', lw=0.3, length_includes_head=True)
    #ax1.annotate('', xy=(0.21, 3.7e4), xytext=(0.2,
3.77e4),arrowprops=dict(facecolor='black', shrink=0.05))
    ax1.annotate('', xy=(0.283, 3.79e4), xytext=(0.34,
3.79e4),arrowprops=dict(facecolor='black', shrink=0.05))
    ax1.annotate('', xy=(0.30, 3.92e4), xytext=(0.275,
```

```
4.1e4),arrowprops=dict(facecolor='black', shrink=0.05))

    ax4.set_xlabel(r"$r\ [\sigma]$",fontsize=10,labelpad=0)
    ax4.set_ylabel(r"$g_{AA}\left(\ r\ \right)$",fontsize=10,labelpad=0)
    ax4.set_ylim(0,2.4)
    ax4.set_xlim(0.7,6)

    ax4.tick_params(axis='both', labelsize=7,length=0,pad=3)
    ax4.legend(fontsize=8,loc='upper right')
        #ax.tick_params(direction='out', length=6, width=2, colors='r',
          #     grid_color='r', grid_alpha=0.5)

    savefig(plt,
            'toy_lj_Tg',
           'V_qT_img_alpha_{}.pdf'.format(sap))
#plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
plt.show()
```

```
using line iftting
0.186699085968
False 3ebe1fa65ad9c2dcb041ebb716eb9bd2
False 006e689136046ee927a95f672d7b45d1
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

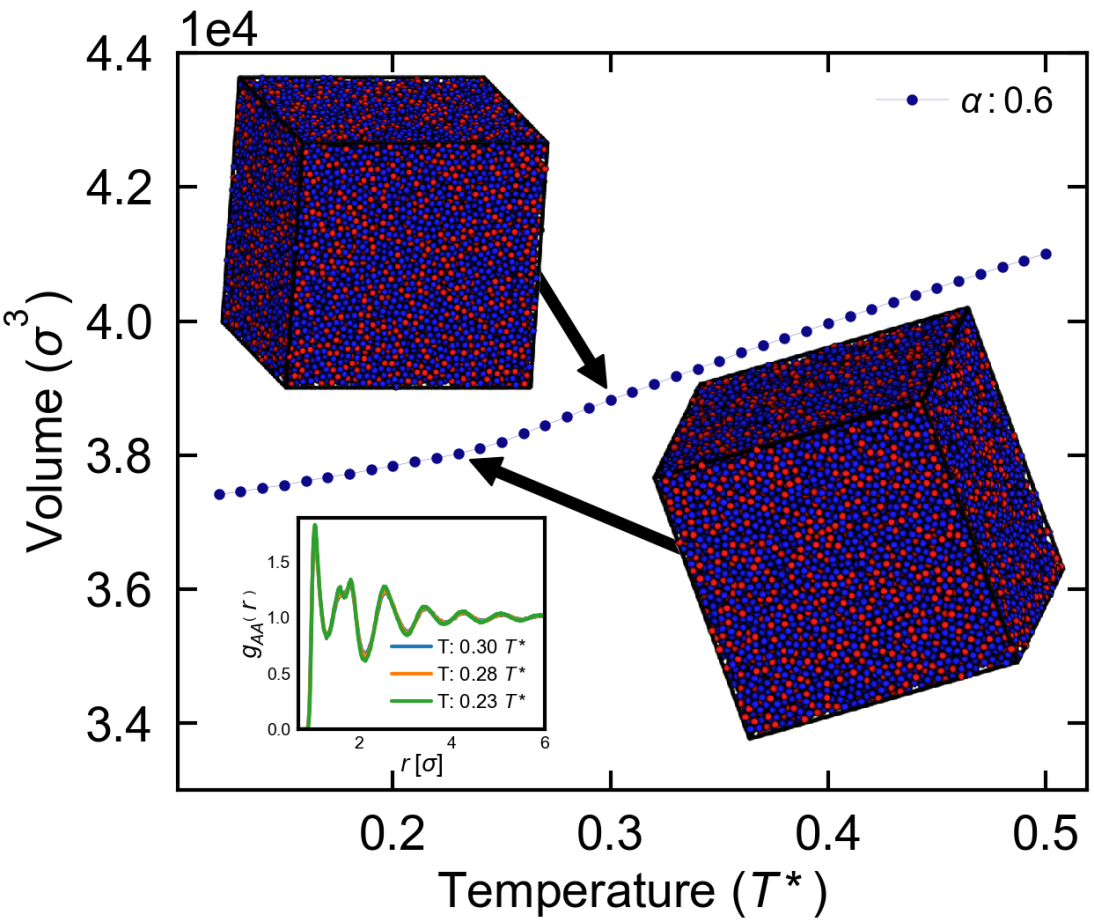


In [282]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
from scipy import ndimage
#stop_after_percents = [10,100.]
filter_saps = [60]#[30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
#plt.figure()
fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.53, 0.22, 0.45, 0.45]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.22, 0.57, 0.3, 0.3]#max t2
ax3 = fig.add_axes([left, bottom, width, height])
ax3.axis('off')

left, bottom, width, height = [0.3, 0.25, 0.2, 0.2]#max t2
ax4 = fig.add_axes([left, bottom, width, height])
#ax4.axis('off')
figure_axes=[ax2,ax3]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
for i,sap in enumerate(filter_saps):

    if sap not in filter_saps:
        continue
    df_filtered = df[(df.integrator==integrator) &
                     (df.kT==kT) &
                     (df.quench_T<=0.5)&
                     (df.quench_T>=0.12)&
                     (df.quench_time ==quench_time)&
                     (df.tauP == tauP)&
                     (df.n_particles==N)&
                     (df.density==density)&
                     (df.activation_energy==activation_energy)&
                     (df.P==P)&
                     (df.num_c10==num_c10)&
                     (df.cooling_method==cooling_method)&
                     (df.pot=='LangH')&
                     (df.stop_after_percent==sap)]
    prop='volume'#'volume'#'potential_energy'#'pair_lj_energy'#'
    quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                          project,
                                                          prop,
                                                          mean_from_second_half=True)

    #plot_data_with_regression(quenchTs, mean_vals)
    #print(df_filtered.quench_T)
    ax1.plot(quenchTs,
                 mean_vals,
                 #val_stds,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha: {}$'.format(sap/100),#label='Quench',
                 linewidth=0.1,
                zorder=1)#1.5)

    if False:
        model = piecewise(quenchTs, mean_vals)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
```

```
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            ax1.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            ax1.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in regression!')

    Tg,Tg_prop = find_Tg(quenchTs,mean_vals)
    print(Tg)
    show_transition = False
    if show_transition:
        #plt.axvline(x=Tg,
        #            linewidth=1.0,
        #            color=colors[i])
        ax1.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=150,
                    color='r',
                   zorder=2)#colors[i])

    qTs=[0.23,0.30]
    rotateby=[-70,0]
    df_sorted = df_filtered.sort_values('quench_T')
    for j,qT in reversed(list(enumerate(qTs))):
        df_xstal = df_sorted[(df_sorted.quench_T==qT)]
        for jobid in df_xstal.index:
            job=project.open_job(id=jobid)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, rotateby[j],prefilter=False)
            #figure_axes[j].set_title('T: {:.2f}
$T^*$'.format(job.sp.quench_T),fontsize=9)
            figure_axes[j].imshow(rotated_img)
    qTs=[0.23,0.28,0.30]
    for j,qT in reversed(list(enumerate(qTs))):
        df_xstal = df_sorted[(df_sorted.quench_T==qT)]
        for jobid in df_xstal.index:
            job=project.open_job(id=jobid)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax4.plot(r,
                     gr,
                     label='T: {:.2f} $T*$'.format(job.sp.quench_T),
                    linewidth=1.5)
    ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    ax1.set_xlabel('Temperature ($T*$)',fontsize=20)
    ax1.set_ylabel('Volume ($\\sigma^3$)',fontsize=20)
    #plt.xlim(0.11,0.51)
    ax1.set_ylim(33000,44000)
    ax1.legend(fontsize=15,loc='upper right')
    #ax1.arrow(0.2, 3.78e4, 0.01, -5e2,shape='full', lw=0.3, length_includes_head=True)
```

```
            #ax1.annotate('', xy=(0.21, 3.7e4), xytext=(0.2,
        3.77e4),arrowprops=dict(facecolor='black', shrink=0.05))
            ax1.annotate('', xy=(0.23, 3.8e4), xytext=(0.34,
        3.65e4),arrowprops=dict(facecolor='black', shrink=0.05))
            ax1.annotate('', xy=(0.30, 3.89e4), xytext=(0.26,
        4.1e4),arrowprops=dict(facecolor='black', shrink=0.05))

            ax4.set_xlabel(r"$r\ [\sigma]$",fontsize=10,labelpad=0)
            ax4.set_ylabel(r"$g_{AA}\left(\ r\ \right)$",fontsize=10,labelpad=0)
            ax4.set_ylim(0,1.89)
            ax4.set_xlim(0.7,6)

            ax4.tick_params(axis='both', labelsize=7,length=0,pad=3)
            ax4.legend(fontsize=8,loc='lower right')
                #ax.tick_params(direction='out', length=6, width=2, colors='r',
                  #     grid_color='r', grid_alpha=0.5)

            savefig(plt,
                    'toy_lj_Tg',
                   'V_qT_img_alpha_{}.pdf'.format(sap))
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        plt.show()
```

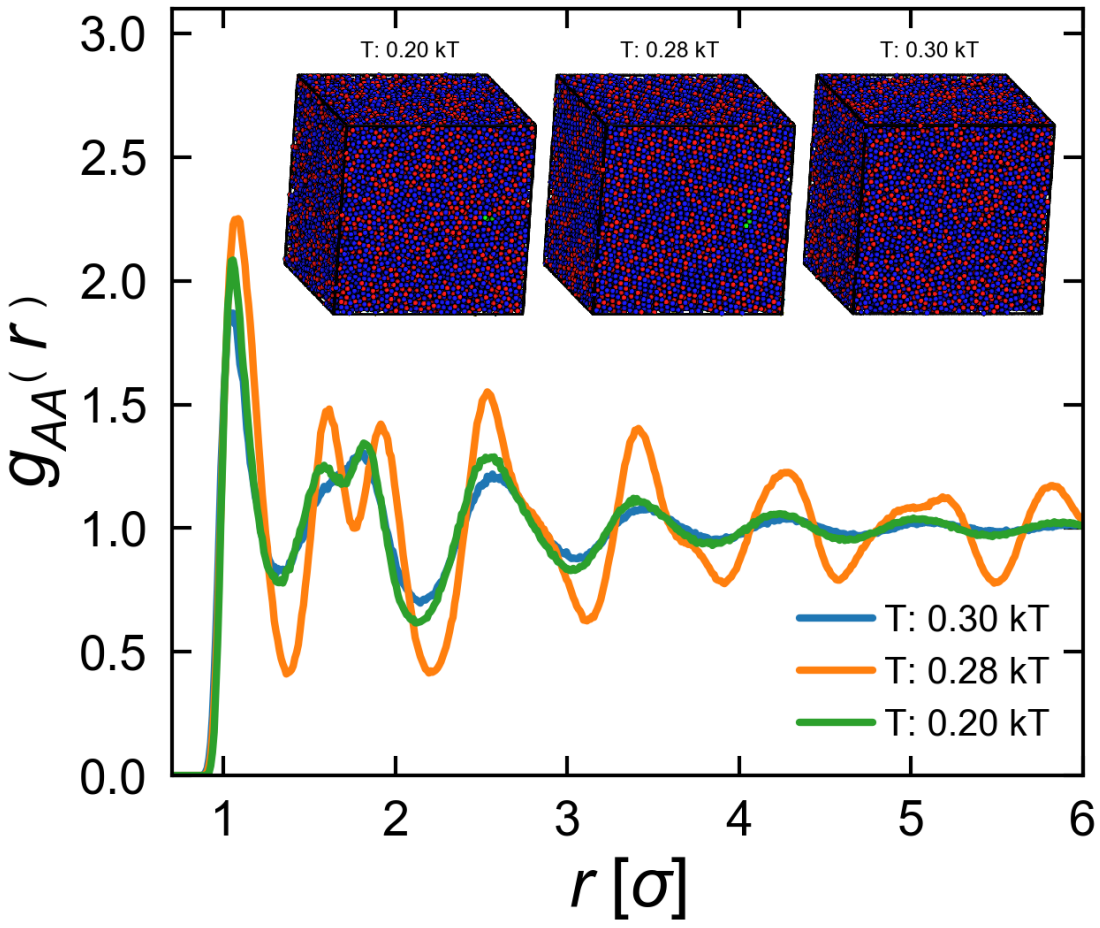
```
using line iftting
0.227163285334


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [17]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.27, 0.64, 0.23, 0.23]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.48, 0.64, 0.23, 0.23]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.69, 0.64, 0.23, 0.23]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')
         figure_axes=[ax2,ax3,ax4]

         filter_saps = [40.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         qTs=[0.2,0.28,0.30]
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(stop_after_percents):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
```

```
                        (df.quench_T<=0.5)&
                        (df.quench_T>=0.1)&
                        (df.quench_time ==quench_time)&
                        (df.tauP == tauP)&
                        (df.n_particles==N)&
                        (df.density==density)&
                        (df.activation_energy==activation_energy)&
                        (df.P==P)&
                        (df.num_c10==num_c10)&
                        (df.cooling_method==cooling_method)&
                        (df.pot=='LangH')&
                        (df.stop_after_percent==sap)]
            df_sorted = df_filtered.sort_values('quench_T')
            for j,qT in reversed(list(enumerate(qTs))):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    print(job.isfile('final.hoomdxml'),job)
                    if job.isfile('rdf_DDS_DDS.txt'):
                        data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr,
                             label='T: {:.2f} kT'.format(job.sp.quench_T))
                    im = plt.imread(job.fn('final_snapshot.png'))
                    figure_axes[j].set_title('T: {:.2f} kT'.format(job.sp.quench_T),fontsize=9)
                    figure_axes[j].imshow(im)

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

            ax1.set_xlabel(r"$r\ [\sigma]$")
            ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
            ax1.set_ylim(0,3.1)
            ax1.set_xlim(0.7,6)
            ax1.legend(fontsize=15,loc='lower right')
            savefig(plt,
                    'toy_lj_Tg',
                    'xstal_near_transition_40_percent_qT_{}.png'.format(qT))
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False 3ebe1fa65ad9c2dcb041ebb716eb9bd2


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "


False 006e689136046ee927a95f672d7b45d1
False aafcd1103162e708caa93fe98dc214ad
```

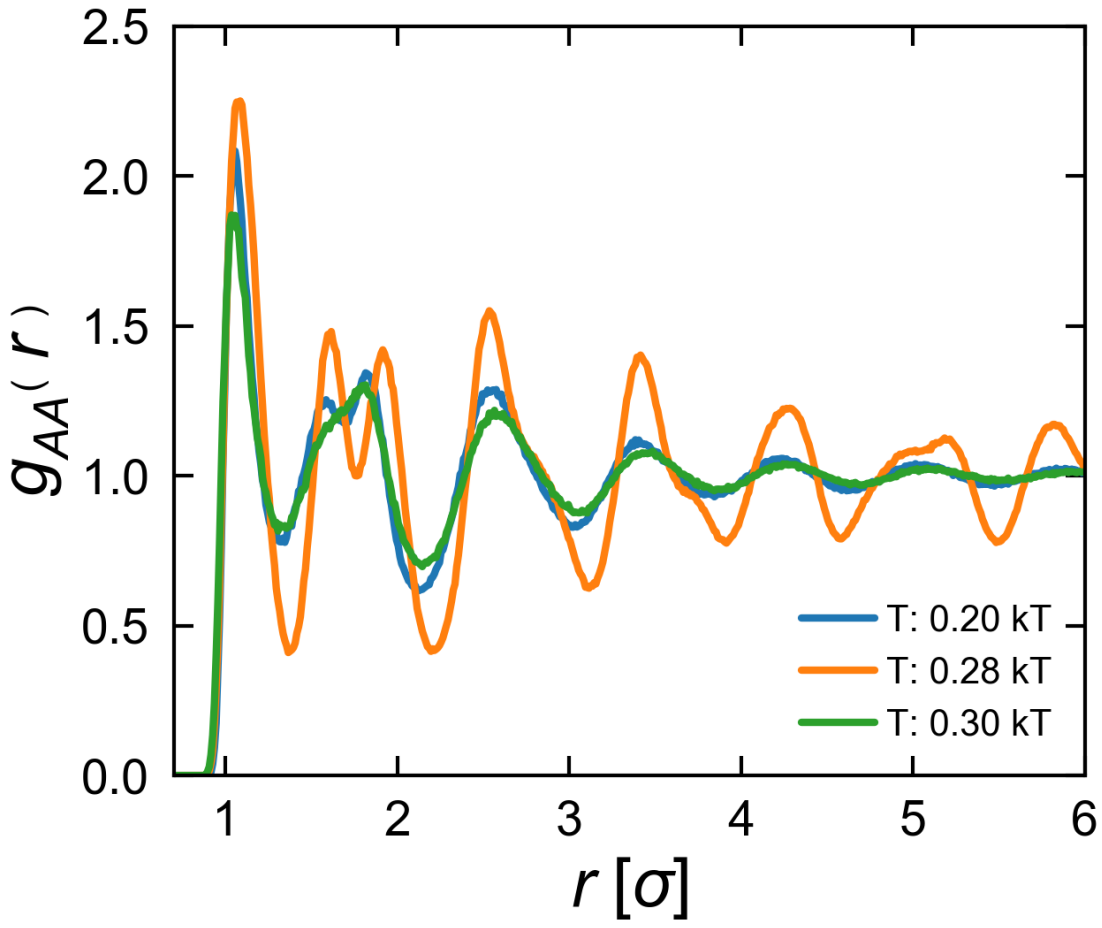



```
In [95]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.27, 0.64, 0.23, 0.23]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.48, 0.64, 0.23, 0.23]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.69, 0.64, 0.23, 0.23]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')
         figure_axes=[ax2,ax3,ax4]

         filter_saps = [40.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         qTs=[0.2,0.28,0.30]
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(stop_after_percents):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
```

```
                            (df.quench_T<=0.5)&
                            (df.quench_T>=0.1)&
                            (df.quench_time ==quench_time)&
                            (df.tauP == tauP)&
                            (df.n_particles==N)&
                            (df.density==density)&
                            (df.activation_energy==activation_energy)&
                            (df.P==P)&
                            (df.num_c10==num_c10)&
                            (df.cooling_method==cooling_method)&
                            (df.pot=='LangH')&
                            (df.stop_after_percent==sap)]
                df_sorted = df_filtered.sort_values('quench_T')
                for j,qT in enumerate(qTs):
                    df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                    for jobid in df_xstal.index:
                        job=project.open_job(id=jobid)
                        print(job.isfile('final.hoomdxml'),job)
                        if job.isfile('rdf_DDS_DDS.txt'):
                            data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                            r=data[:,0]
                            gr=data[:,1]
                        else:
                            raise FileNotFoundError('Dont have the rdf file for',job)
                        ax1.plot(r,
                                 gr,
                                 label='T: {:.2f} kT'.format(job.sp.quench_T))
                        im = plt.imread(job.fn('final_snapshot.png'))
                        figure_axes[j].set_title('T: {:.2f} kT'.format(job.sp.quench_T),fontsize=9)
                        figure_axes[j].imshow(im)

            #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

            ax1.set_xlabel(r"$r\ [\sigma]$")
            ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
            ax1.set_ylim(0,2.7)
            ax1.set_xlim(0.7,6)
            ax1.legend(fontsize=15,loc='lower right')
            savefig(plt,
                    'toy_lj_Tg',
                    'xstal_near_transition_40_percent.pdf')
            #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
            #plt.show()
```

```
False aafcd1103162e708caa93fe98dc214ad
False 006e689136046ee927a95f672d7b45d1
False 3ebe1fa65ad9c2dcb041ebb716eb9bd2
```

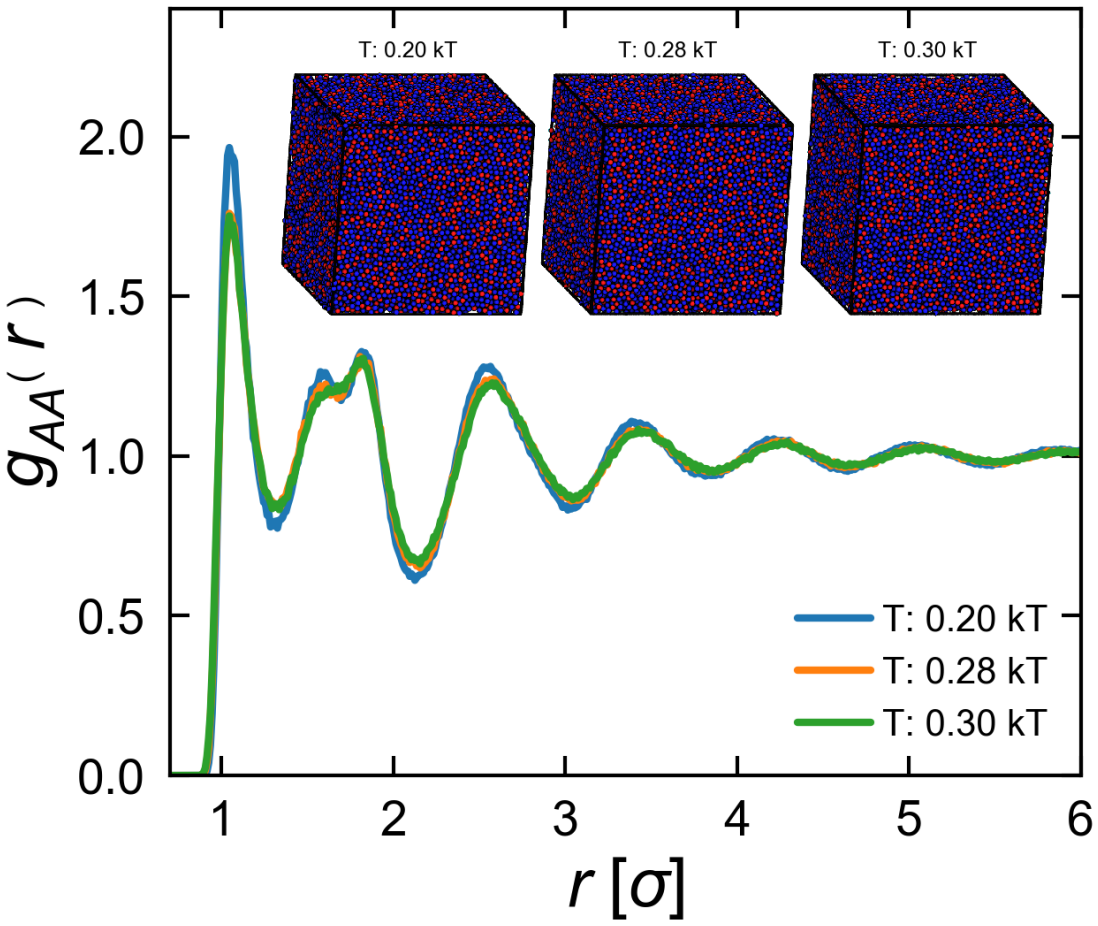
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [44]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.27, 0.64, 0.23, 0.23]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.48, 0.64, 0.23, 0.23]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.69, 0.64, 0.23, 0.23]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')
         figure_axes=[ax2,ax3,ax4]

         filter_saps = [60.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         qTs=[0.2,0.28,0.30]
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(stop_after_percents):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
```

```
                        (df.quench_T<=0.5)&
                        (df.quench_T>=0.1)&
                        (df.quench_time ==quench_time)&
                        (df.tauP == tauP)&
                        (df.n_particles==N)&
                        (df.density==density)&
                        (df.activation_energy==activation_energy)&
                        (df.P==P)&
                        (df.num_c10==num_c10)&
                        (df.cooling_method==cooling_method)&
                        (df.pot=='LangH')&
                        (df.stop_after_percent==sap)]
            df_sorted = df_filtered.sort_values('quench_T')
            for j,qT in enumerate(qTs):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    print(job.isfile('final.hoomdxml'),job)
                    if job.isfile('rdf_DDS_DDS.txt'):
                        data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr,
                             label='T: {:.2f} kT'.format(job.sp.quench_T))
                    im = plt.imread(job.fn('final_snapshot.png'))
                    figure_axes[j].set_title('T: {:.2f} kT'.format(job.sp.quench_T),fontsize=9)
                    figure_axes[j].imshow(im)

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
        ax1.set_ylim(0,2.4)
        ax1.set_xlim(0.7,6)
        ax1.legend(fontsize=15,loc='lower right')
        savefig(plt,
                'toy_lj_Tg',
                'xstal_near_transition_60_percent.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False 4ba52d2952dca20a399c5d0b0490ba1f
False ff810254e3561c17ea8d099e4837fc14
False 358f732d6f1de369cd848f5d1acf6482


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

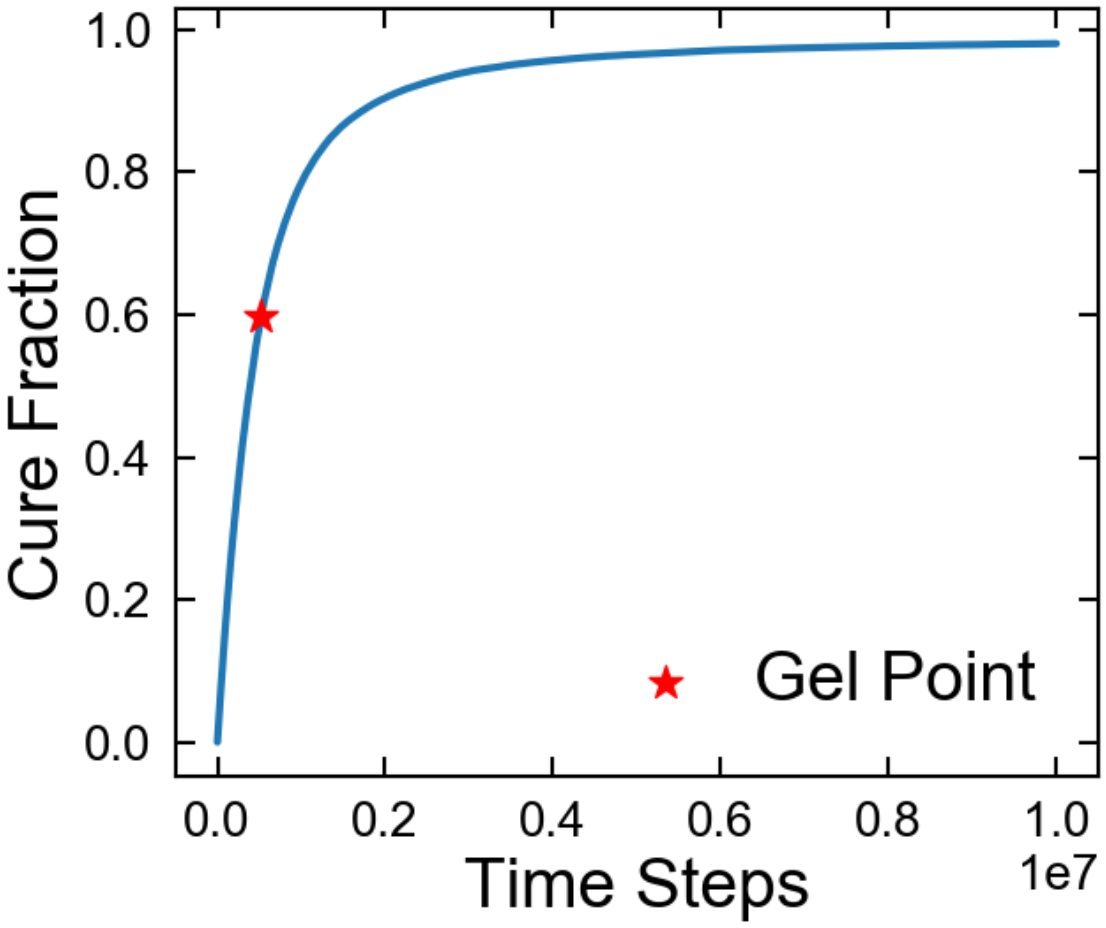



```
In [16]: df_filtered = df[(df.integrator==integrator) &
                          (df.kT==kT) &
                          (df.quench_time ==6e6)&
                          (df.tauP == tauP)&
                          (df.n_particles==N)&
                          (df.density==density)&
                          (df.activation_energy==activation_energy)&
                          (df.P==P)&
                          (df.num_c10==num_c10)&
                          (df.cooling_method=='anneal')&
                          (df.pot=='LangH')&
                          (df.stop_after_percent==0)]
         df_filtered.quench_T

Out[16]: 646df3239f44bb1705e0ff4c9a58106e    0.2
         Name: quench_T, dtype: object

In [17]: df_filtered = df[#(df.integrator==integrator) &
                          #(df.kT==kT) &
                          #(df.quench_T<=0.5)&
                          #(df.quench_T>=0.1)&
                          #(df.quench_time ==quench_time)&
                          #(df.tauP == tauP)&
                          (df.bond == True)&
```

```
                    (df.n_particles==N)&
                    (df.density==density)&
                    (df.activation_energy==activation_energy)&
                    #(df.P==P)&
                    #(df.cooling_method==cooling_method)&
                    #(df.pot=='LangH')&
                    (df.stop_after_percent==100.)]
plt.figure()
for job_id in df_filtered.index:
    #job_id = group.signac_id
    #print(job_id)
    job = project.open_job(id=job_id)
    if job.isfile('out.log'):
        log_path = job.fn('out.log')
        data = np.genfromtxt(log_path, names=True)
        time_steps = data['timestep']
        doc = data['bond_percentAB']/100
        plt.plot(time_steps,
                 doc,
                 zorder=0)
        plt.scatter(job.document['gel_point'],
                    job.document['curing_at_gel_point']/100,
                   marker='*',
                    s=200,
                   color='r',
                   zorder=1,
                   label='Gel Point')
plt.xlabel('Time Steps')
plt.ylabel('Cure Fraction')
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x11c3397b8>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



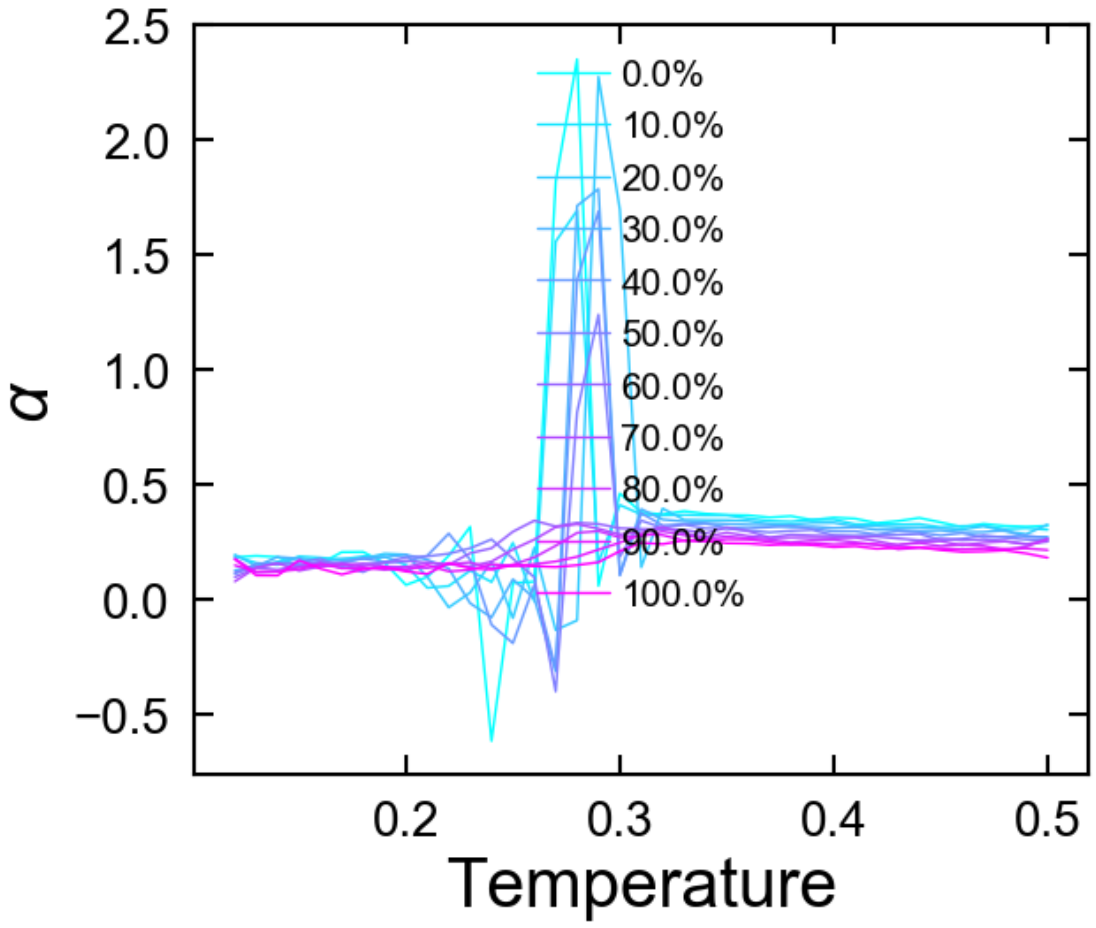

```
In [18]: from scipy.interpolate import InterpolatedUnivariateSpline
         from piecewise.regressor import piecewise

         plt.figure()
         fig,ax1 = plt.subplots()
         #ax2 = ax1.twinx()
         colors = plt.cm.cool(np.linspace(0,1,len(stop_after_percents)))

         for i,sap in enumerate(stop_after_percents):
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.5)&
                              (df.quench_T>=0.1)&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot=='LangH')&
                              (df.stop_after_percent==sap)]
             prop='volume'#'potential_energy'#'pair_lj_energy'#'
             quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                                   project,
                                                                   prop,
```

```
                                                        mean_from_second_half=True)
            mean_vals = np.asarray(mean_vals)
            quenchTs = np.asarray(quenchTs)
            # Get a function that evaluates the linear spline at any quench T
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dVdT = f.derivative(n=1)
            d2VdT2 = f.derivative(n=2)
            dV_dTs = dVdT(quenchTs)
            d2V_dT2s = d2VdT2(quenchTs)
            alpha = dV_dTs/mean_vals #coefficient of volumetric thermal expansion
            Tg,V_Tg = find_Tg(quenchTs,mean_vals)
            show_Tg = False
            if show_Tg:
                ax1.axvline(x=Tg,
                            color=colors[i],
                            linewidth=1.0)
            if True:
                ax1.plot(quenchTs,
                         alpha,
                         color=colors[i],
                         #marker='.',
                         label='{}%'.format(sap),
                         linewidth=1.0,
                         zorder=1)#1.5)
            if False:
                ax2.plot(quenchTs,
                         d2V_dT2s,
                         color=colors[i],
                         #marker='.',
                         label='{}% second derivative'.format(sap),
                         linestyle='--',
                         linewidth=1.0,
                         zorder=4)#1.5)
        ax1.set_xlabel('Temperature')
        ax1.set_ylabel('$\\alpha$')
        #ax2.set_ylabel(prop+' (2nd der)')
        #ax2.set_ylim(-3e8,1e8)
        #ax1.set_ylim(-1e5,1.0e6)
        #plt.ylim(36000,42000)
        ax1.legend(fontsize=15,loc='best')
        #ax2.legend(fontsize=15)
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
```

```
<matplotlib.figure.Figure at 0x11efd7390>
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



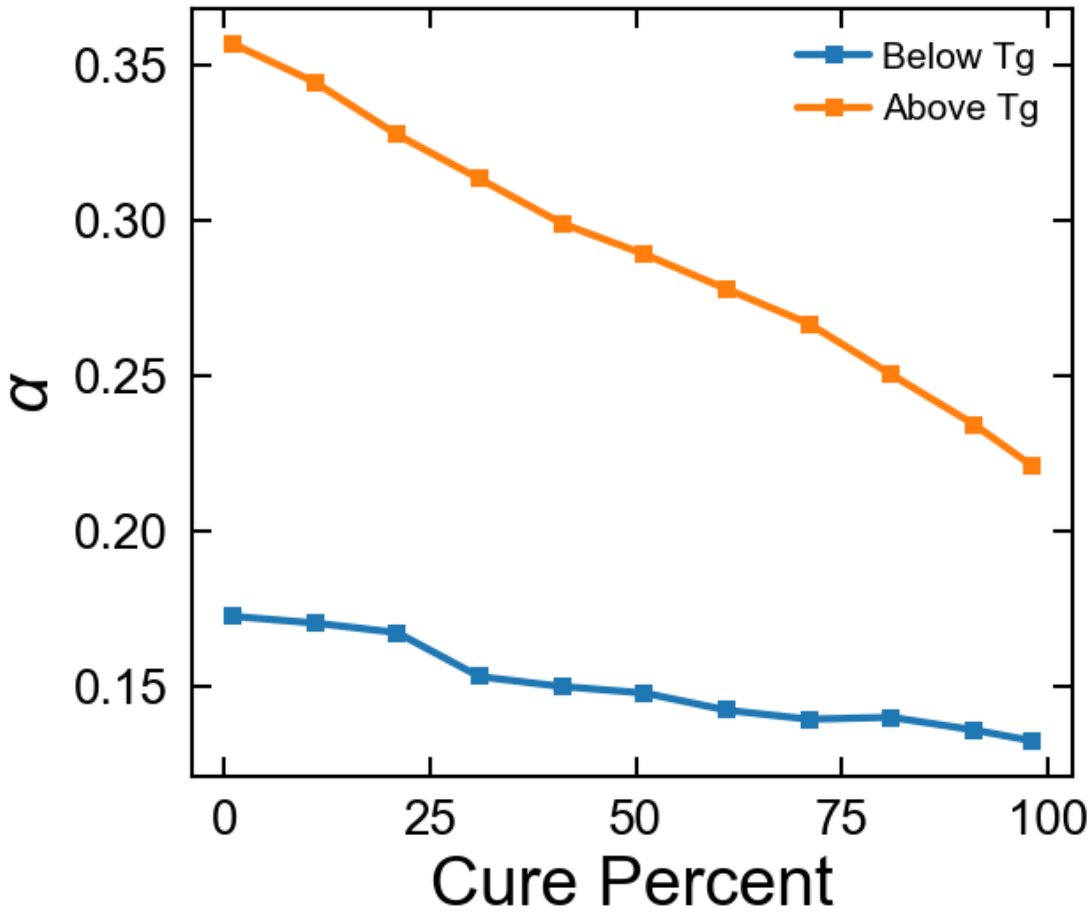

```
In [19]: from scipy.interpolate import InterpolatedUnivariateSpline
         from piecewise.regressor import piecewise

         plt.figure()
         fig,ax1 = plt.subplots()
         #ax2 = ax1.twinx()
         colors = plt.cm.cool(np.linspace(0,1,len(stop_after_percents)))

         conditions = ['Below Tg','Above Tg']
         for condition in conditions:
             alphas=[]
             cure_percents = []
             for i,sap in enumerate(stop_after_percents):
                 df_filtered = df[(df.integrator==integrator) &
                                  (df.kT==kT) &
                                  (df.quench_T<=0.5)&
                                  (df.quench_T>=0.1)&
                                  (df.quench_time ==quench_time)&
                                  (df.tauP == tauP)&
                                  (df.n_particles==N)&
                                  (df.density==density)&
                                  (df.activation_energy==activation_energy)&
                                  (df.P==P)&
                                  (df.cooling_method==cooling_method)&
                                  (df.pot=='LangH')&
                                  (df.stop_after_percent==sap)]
```

```
            prop='volume'#'potential_energy'#'pair_lj_energy'#'
            quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                                  project,
                                                                  prop,
        mean_from_second_half=True)
            mean_vals = np.asarray(mean_vals)
            quenchTs = np.asarray(quenchTs)
            # Get a function that evaluates the linear spline at any quench T
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dVdT = f.derivative(n=1)
            d2VdT2 = f.derivative(n=2)
            dV_dTs = dVdT(quenchTs)
            d2V_dT2s = d2VdT2(quenchTs)
            alpha = dV_dTs/mean_vals #coefficient of volumetric thermal expansion

            Tg,V_Tg = find_Tg(quenchTs,mean_vals)
            Tg_i = np.where(quenchTs>=Tg)[0][0]
            alpha_after_Tg = alpha[Tg_i:]
            alpha_before_Tg = alpha[:Tg_i]
            cure_percents.append(df_filtered.groupby(['cure_percent']).mean().index[0])
            if condition=='Below Tg':
                mean_alpha = np.mean(alpha_before_Tg)
            else:
                mean_alpha = np.mean(alpha_after_Tg)
            alphas.append(mean_alpha)

        ax1.plot(cure_percents,
                 alphas,
                 #color=colors[i],
                 marker='s',
                 label=condition,
                 #linestyle='--',
                 #linewidth=1.0,
                 zorder=4)#1.5)
        ax1.set_xlabel('Cure Percent')
        ax1.set_ylabel('$\\alpha$')
        #ax2.set_ylabel(prop+' (2nd der)')
        #ax2.set_ylim(-3e8,1e8)
        #ax1.set_ylim(-1e5,1.0e6)
        #plt.ylim(36000,42000)
        ax1.legend(fontsize=15,loc='best')
        #ax2.legend(fontsize=15)
        plt.show()

using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
```

```
using line iftting
```

```
<matplotlib.figure.Figure at 0x11f29aa58>
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

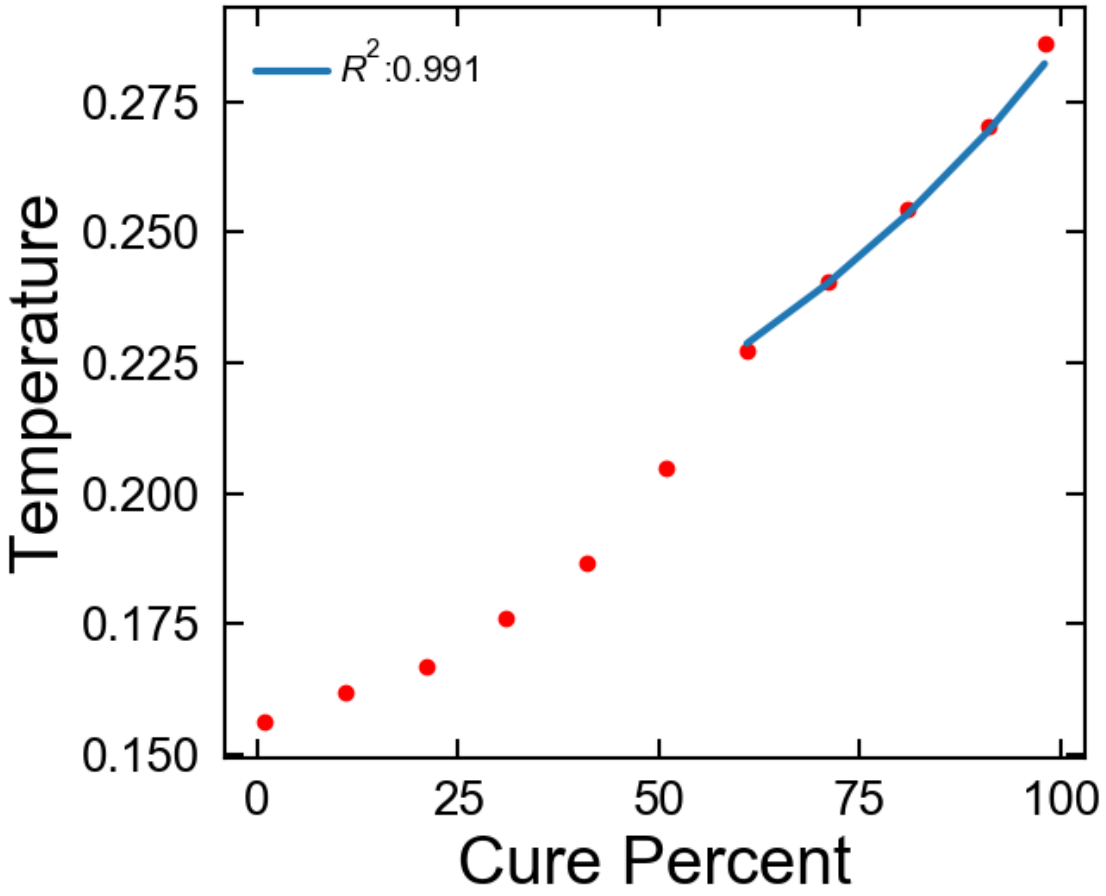



```
In [20]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         plt.figure()
         Eas = [activation_energy]#[1.0,2.0]#
         colors = plt.cm.cool(np.linspace(0,1,len(Eas)))

         for i,activation_energy in enumerate(Eas):
             Tgs=[]
             cure_percents = []
```

```
            for sap in stop_after_percents:
                df_filtered = df[(df.integrator==integrator) &
                            (df.kT==kT) &
                            (df.quench_T<=0.5)&
                            (df.quench_T>=0.1)&
                            (df.quench_time ==quench_time)&
                            (df.tauP == tauP)&
                            (df.n_particles==N)&
                            (df.density==density)&
                            (df.activation_energy==activation_energy)&
                            (df.P==P)&
                            (df.cooling_method==cooling_method)&
                            (df.pot=='LangH')&
                            (df.stop_after_percent==sap)]
                prop='volume'#'potential_energy'#'pair_lj_energy'#'
                quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                                      project,
                                                                      prop,
        mean_from_second_half=True)
                cure_percents.append(df_filtered.groupby(['cure_percent']).mean().index[0])
                Tg = find_Tg(quenchTs, mean_vals)
                Tgs.append(Tg)
            cure_percents = np.asarray(cure_percents)
            Tgs = np.asarray(Tgs)[:,0]
            print(cure_percents,Tgs)
            cure_percents_subset = cure_percents[-5:]
            Tg_subset = Tgs[-5:]
            #R2,fit_Tgs,T0,inter_parm = fit_Tg_to_DiBenedetto(cure_percents/100.,Tgs,Tgs[-1])
            R2,fit_Tgs,T0,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_subset/100.,Tg_subset,Tg_subset[-1])
            print('T0',T0,'lambda',inter_parm)
            #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
            print(cure_percents_subset)
            plt.plot(cure_percents_subset,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
            plt.scatter(cure_percents,
                        Tgs,
                        #label='$E_a$:{}'.format(activation_energy),
                        color='r')#colors[i])
        plt.xlabel('Cure Percent')
        plt.ylabel('Temperature')
        #plt.xlim(0.1,0.5)
        #plt.ylim(36000,42000)
        plt.legend(fontsize=15)
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
[  1.00047994  11.00078011  21.00107956  31.00138092  41.00017929
  51.00048065  61.0007782   71.00108337  81.00138092  91.00018311
  98.05754089] [ 0.1562854   0.16193598  0.16674156  0.17586491  0.18669909
0.20488737
  0.22716329  0.24050735  0.25456462  0.27045237  0.2862303 ]
T0 0.183685001259 lambda 0.5
[ 61.0007782   71.00108337  81.00138092  91.00018311  98.05754089]
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



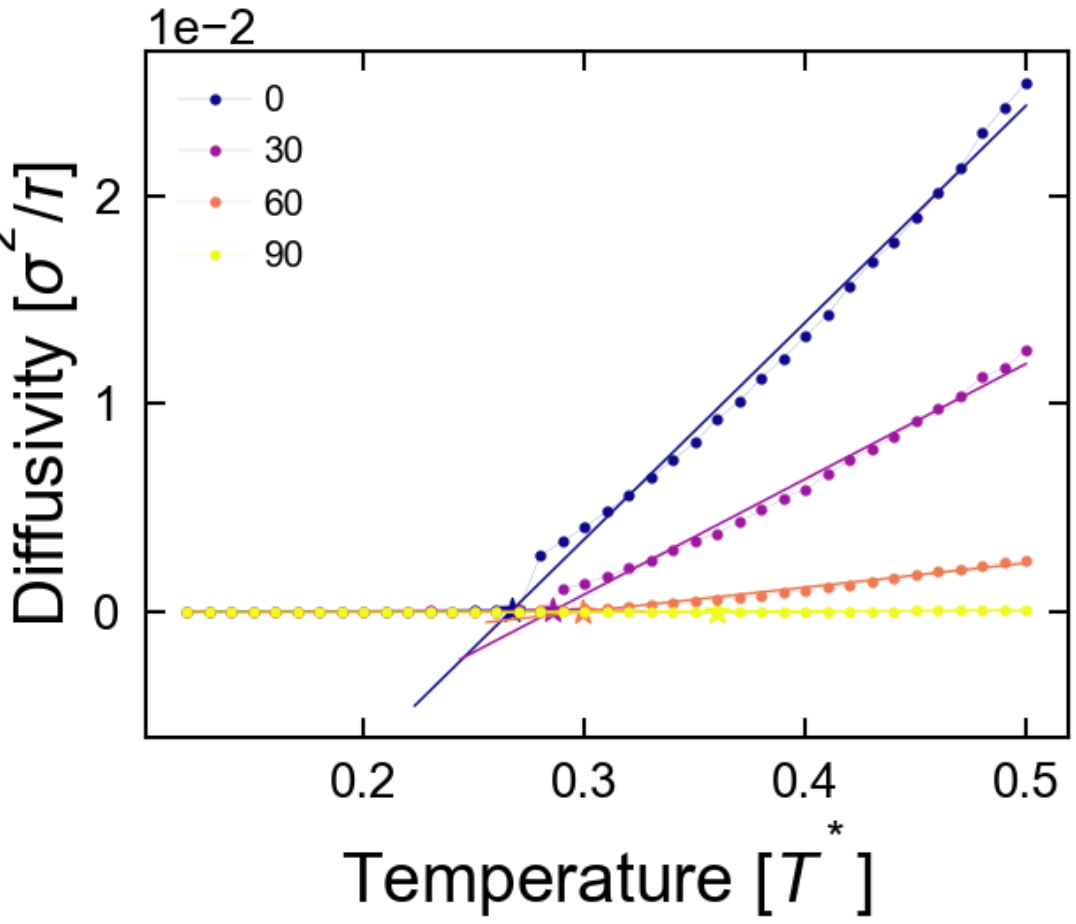

```
In [21]: stop_after_percents = [0,30,60,90]#[0,10,20,40,80,100]#np.arange(10,105,15,dtype=float)
         colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
         Tgs=[]
         cure_percents = []
         for i,sap in enumerate(stop_after_percents):

             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.5)&
                              (df.quench_T>=0.1)&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot=='LangH')&
                              (df.stop_after_percent==sap)]
```

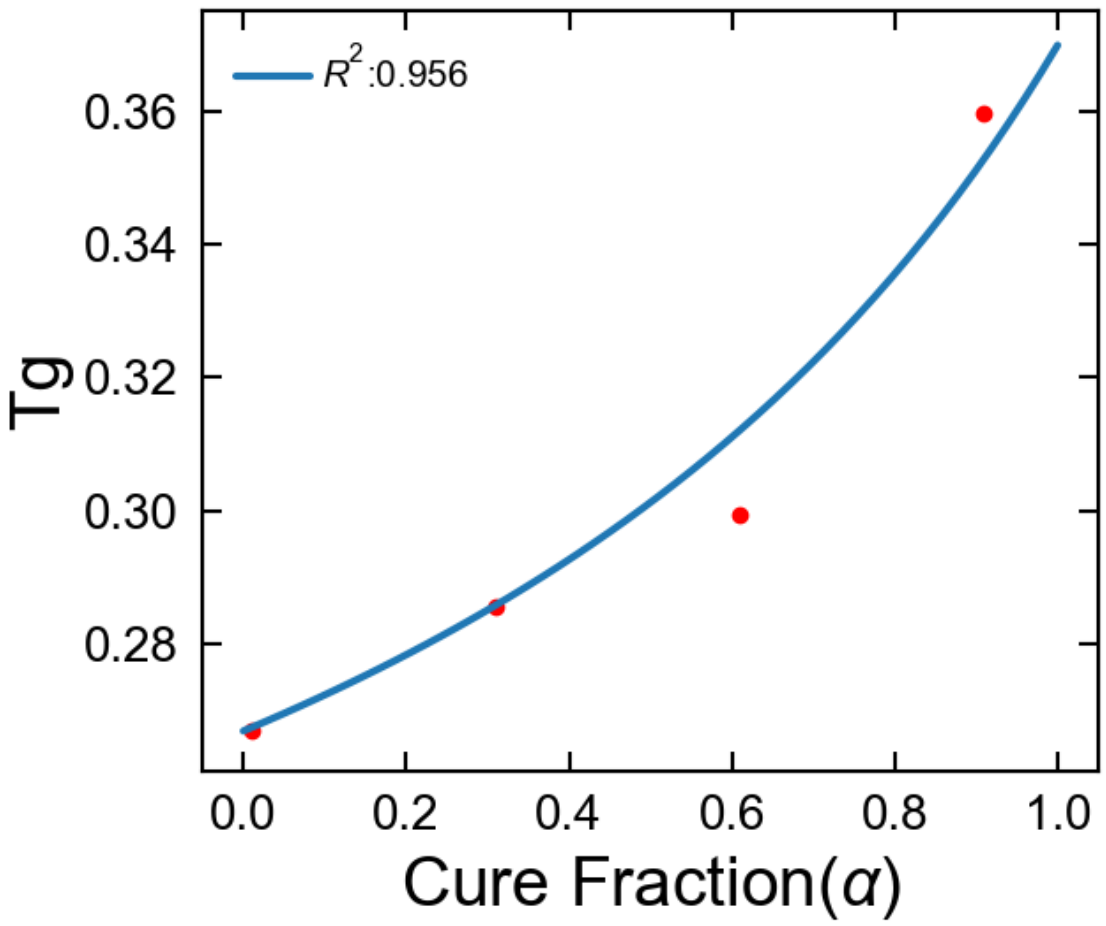
```
    df_filtered.quench_T
    cure_percents.append(df_filtered.cure_percent.values[0])
    df_sorted = df_filtered.sort_values(by=['quench_T'])
    #print(df_sorted.index)
    plt.figure(0)
    plt.plot(df_sorted['quench_T'],
             df_sorted['D_B'],
             color=colors[i],
             marker='.',
             linewidth=0.1,
             label=sap)

    if True:
        #print(df.columns.get_loc("quench_T"))
        quenchTs = list(df_sorted.iloc[:,df_sorted.columns.get_loc("quench_T")].values)
        D_As = list(df_sorted.iloc[:,df_sorted.columns.get_loc("D_B")].values)
        #print(D_As)
        model = piecewise(quenchTs, D_As)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        Tg,Tg_prop = find_Tg(quenchTs,D_As)
        print(Tg)
        Tgs.append(Tg)
        show_transition = True
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop,
                        marker='*',
                        s=100,
                        color=colors[i],
                       zorder=0)#colors[i])
        else:
            print('WARNING: found {} line segments in
regression!'.format(len(model.segments)))

plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
plt.xlabel('Temperature [$T^*$]')
plt.ylabel('Diffusivity [$\\sigma^2/\\tau$]')
#plt.xscale('log')
#plt.yscale('log')
plt.legend(fontsize=15)
savefig(plt,
       'toy_lj_Tg',
```

```
            'D_vs_qT.pdf')
        plt.figure(1)
        cure_percents = np.asarray(cure_percents)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction($\\alpha$)')
        plt.ylabel('Tg')
        plt.legend(fontsize=15)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        np.savetxt('DGEBA_DDS_Tg_quench.txt',np.transpose(Tg_data))
        #plt.xlim(0.25,0.35)
        #plt.ylim(-0.005,0.01)
        savefig(plt,
               'toy_lj_Tg',
               'dibeneditto.pdf')
        plt.show()
```

```
using line iftting
0.267022833755
using line iftting
0.285695967181
using line iftting
0.299347201817
using line iftting
0.359602082836
T1 0.369812079689 lambda 0.5
[  1.00047994  31.00138092  61.0007782   91.00018311]
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−2
0
30
60
90
2
1
0
Diffusivity [σ^2/τ]
0.2
0.3
0.4
0.5
Temperature [T^*]

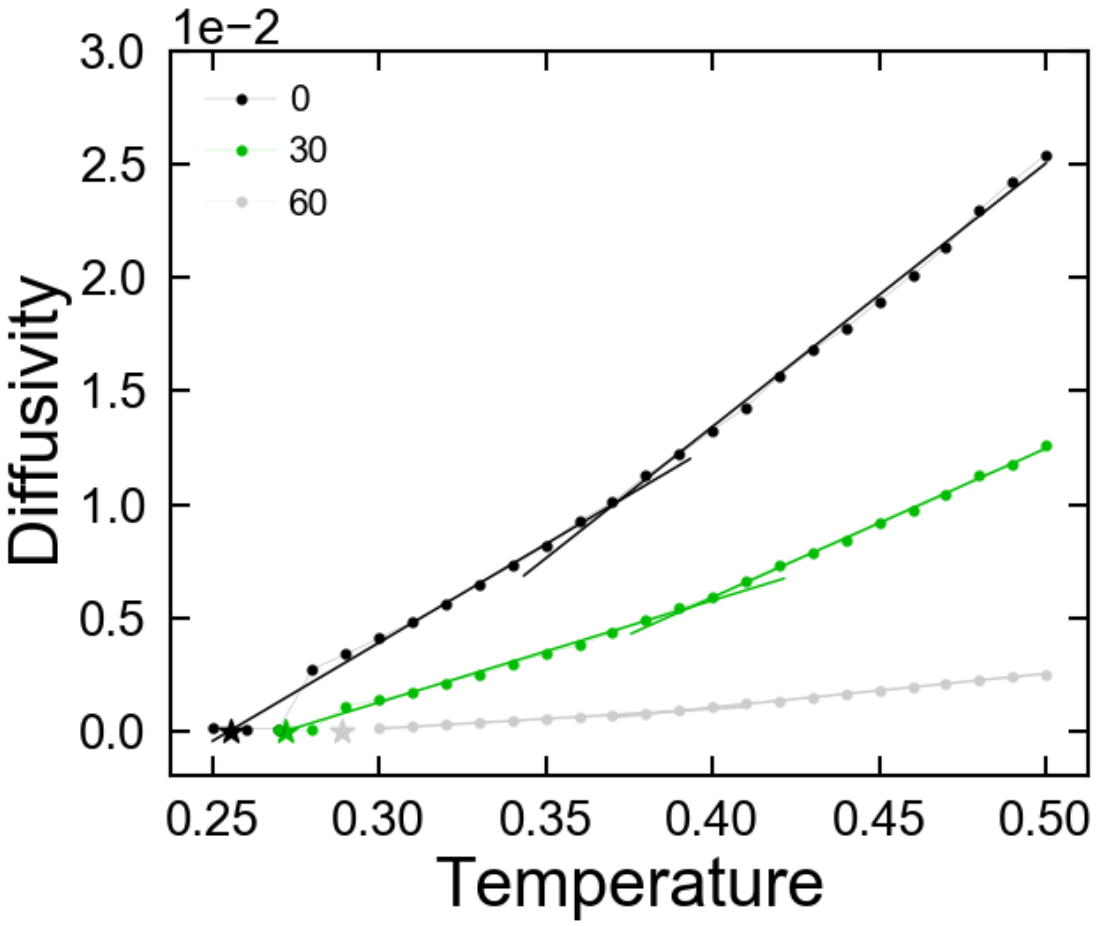


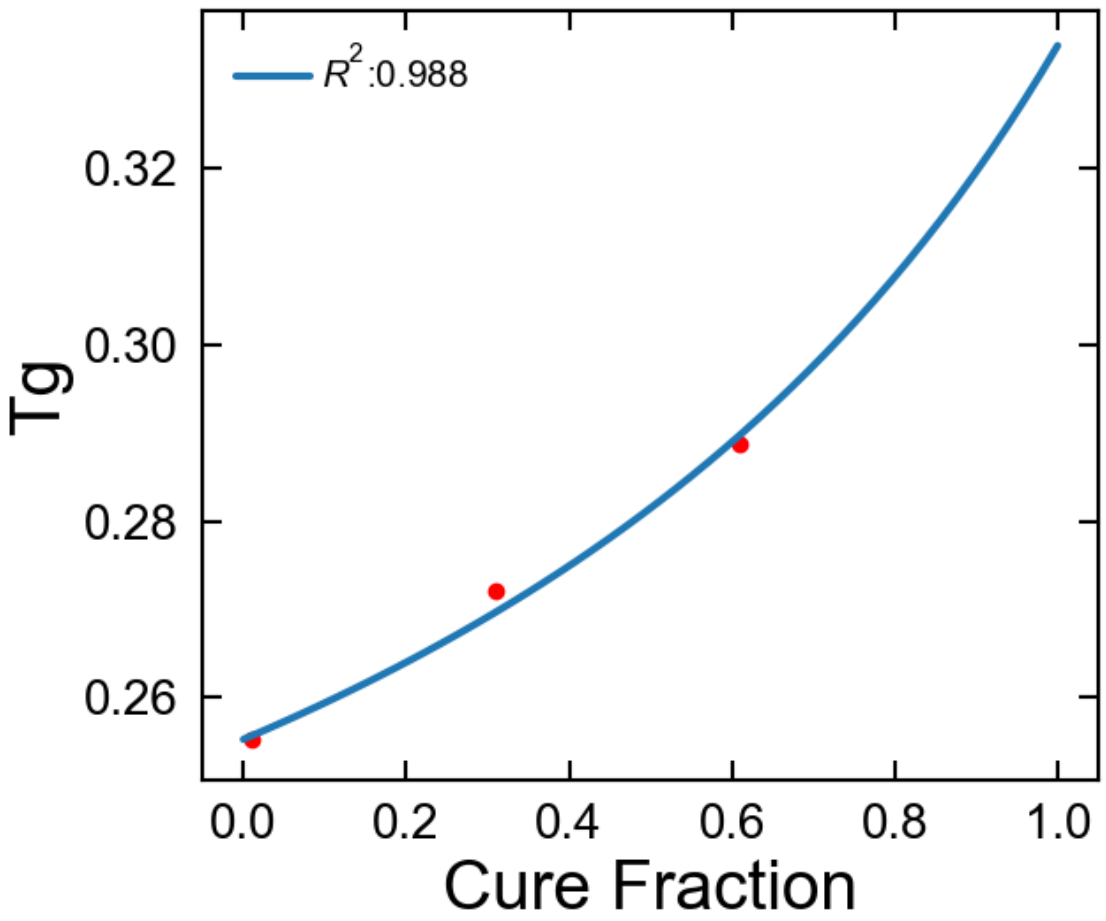

```
In [22]: stop_after_percents =
         [0,30,60]#,90]#[0,10,20,40,80,100]#np.arange(10,105,15,dtype=float)
         colors = plt.cm.spectral(np.linspace(0,1,len(stop_after_percents)))
         Tgs=[]
         cure_percents = []
         for i,sap in enumerate(stop_after_percents):

             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.5)&
                              (df.quench_T>=0.1)&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot=='LangH')&
                              (df.stop_after_percent==sap)]
             df_filtered.quench_T
             cure_percents.append(df_filtered.cure_percent.values[0])
             df_sorted = df_filtered.sort_values(by=['quench_T'])
             #print(df_sorted.index)
             plt.figure(0)
             quenchTs = list(df_sorted.iloc[:,df_sorted.columns.get_loc("quench_T")].values)
```

```
    D_As = np.asarray(df_sorted.iloc[:,df_sorted.columns.get_loc("D_B")].values)
    #print(D_As)
    indices = np.where(D_As>0.000095)
    start_index = indices[0][0]
    print(start_index)
    D_As=D_As[start_index:]
    quenchTs=quenchTs[start_index:]
    plt.plot(quenchTs,#df_sorted['quench_T'],
             D_As,#df_sorted['D_B'],
             color=colors[i],
             marker='.',
             linewidth=0.1,
             label=sap)

    if True:
        model = piecewise(quenchTs, D_As)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            Tg = -b1/m1
            Tg_prop = 0.
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        #Tg,Tg_prop = find_Tg(quenchTs,D_As)
        print(Tg)
        Tgs.append(Tg)
        show_transition = True
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop,
                        marker='*',
                        s=100,
                        color=colors[i],
                       zorder=0)#colors[i])
        else:
            print('WARNING: found {} line segments in
regression!'.format(len(model.segments)))

plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
plt.xlabel('Temperature')
plt.ylabel('Diffusivity')
#plt.xscale('log')
#plt.yscale('log')
plt.ylim(-0.002,0.03)
plt.legend(fontsize=15)
```

```
        plt.figure(1)
        cure_percents = np.asarray(cure_percents)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction')
        plt.ylabel('Tg')
        plt.legend(fontsize=15)

        #plt.ylim(-0.005,0.01)
        plt.show()
```

```
13
0.255269531588
15
0.272112200646
18
0.288773026325
T1 0.33390464442 lambda 0.5
[  1.00047994  31.00138092  61.0007782 ]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

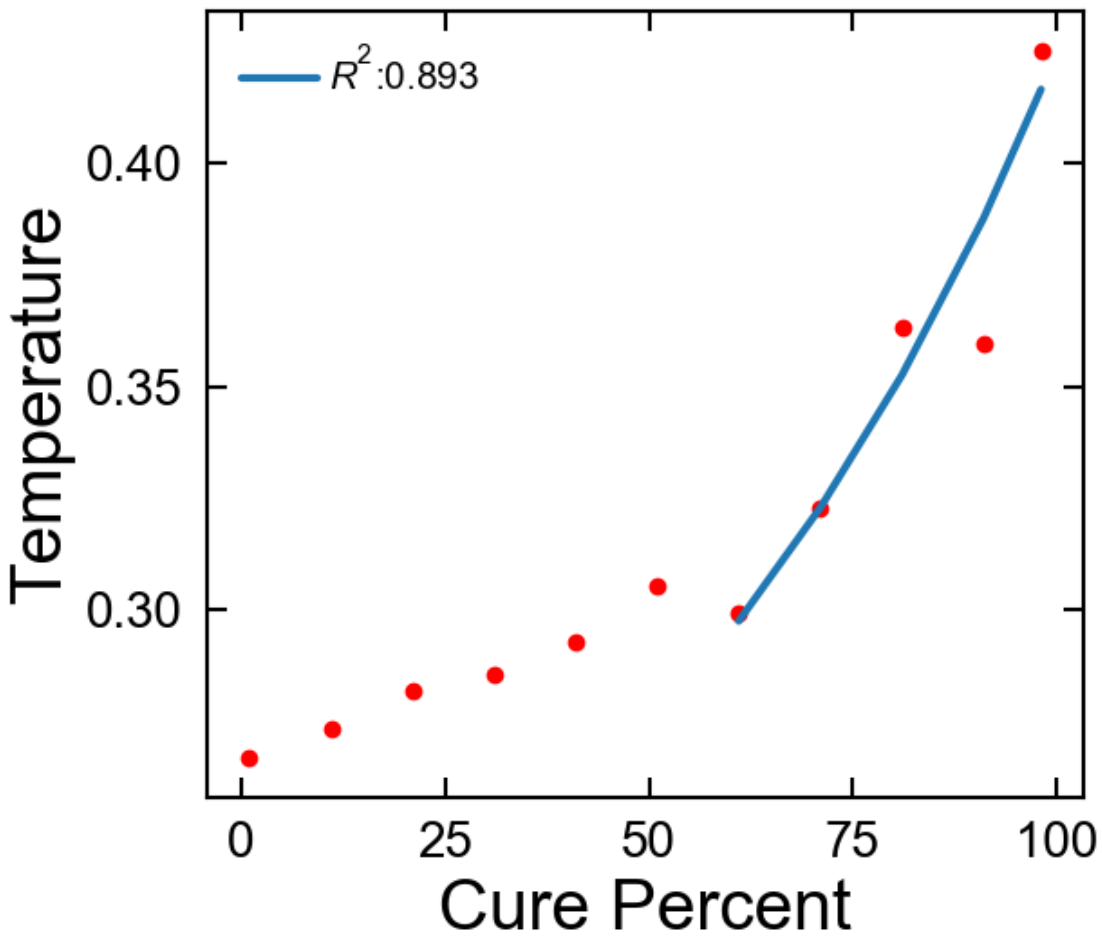

1e−2
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Diffusivity
0
30
60
0.25
0.30
0.35
0.40
0.45
0.50
Temperature




```
In [23]: stop_after_percents = np.arange(0,110,10,dtype=float)
         colors = plt.cm.spectral(np.linspace(0,1,len(stop_after_percents)))
         Tgs=[]
         cure_percents = []
         for i,sap in enumerate(stop_after_percents):

             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_T<=0.5)&
                              (df.quench_T>=0.1)&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot=='LangH')&
                              (df.stop_after_percent==sap)]
             df_sorted = df_filtered.sort_values(by=['quench_T'])
             quenchTs = list(df_sorted.iloc[:,df_sorted.columns.get_loc("quench_T")].values)
             D_As = list(df_sorted.iloc[:,df_sorted.columns.get_loc("D_B")].values)
             cure_percents.append(df_sorted.groupby(['cure_percent']).mean().index[0])
             Tg,Tg_prop = find_Tg(quenchTs,D_As)
             Tgs.append(Tg)
```

```
        cure_percents = np.asarray(cure_percents)
        Tgs = np.asarray(Tgs)
        print(cure_percents[:-1],Tgs)
        cure_percents_subset = cure_percents[-5:]
        Tg_subset = Tgs[-5:]
        #R2,fit_Tgs,T0,inter_parm = fit_Tg_to_DiBenedetto(cure_percents/100.,Tgs,Tgs[-1])
        R2,fit_Tgs,T0,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_subset/100.,Tg_subset,Tg_subset[-1])
        print('T0',T0,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        print(cure_percents_subset)
        plt.plot(cure_percents_subset,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.xlabel('Cure Percent')
        plt.ylabel('Temperature')
        #plt.xlim(0.1,0.5)
        #plt.ylim(36000,42000)
        plt.legend(fontsize=15)
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
[  1.00047994  11.00078011  21.00107956  31.00138092  41.00017929
  51.00048065  61.0007782   71.00108337  81.00138092  91.00018311] [ 0.26702283
0.27323856  0.28204007  0.28569597  0.29288659  0.30547744
  0.2993472   0.32266741  0.3632403   0.35960208  0.42533591]
T0 0.197545555376 lambda 0.5
[ 61.0007782   71.00108337  81.00138092  91.00018311  98.05754089]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


R^2:0.893
0.40
0.35
0.30
Temperature
0
25
50
75
100
Cure Percent

In [12]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [13]:
```
df_filtered=df[df.cooling_method=='anneal']
df_filtered.stop_after_percent
```

Out[13]:
```
90b99ca468d650a13ade4d55a27ce7e1    30
ffd54826a80437a447973bbf73be3f68    90
644d27b8f740cc6df4a628957245df0c    60
646df3239f44bb1705e0ff4c9a58106e     0
Name: stop_after_percent, dtype: object
```

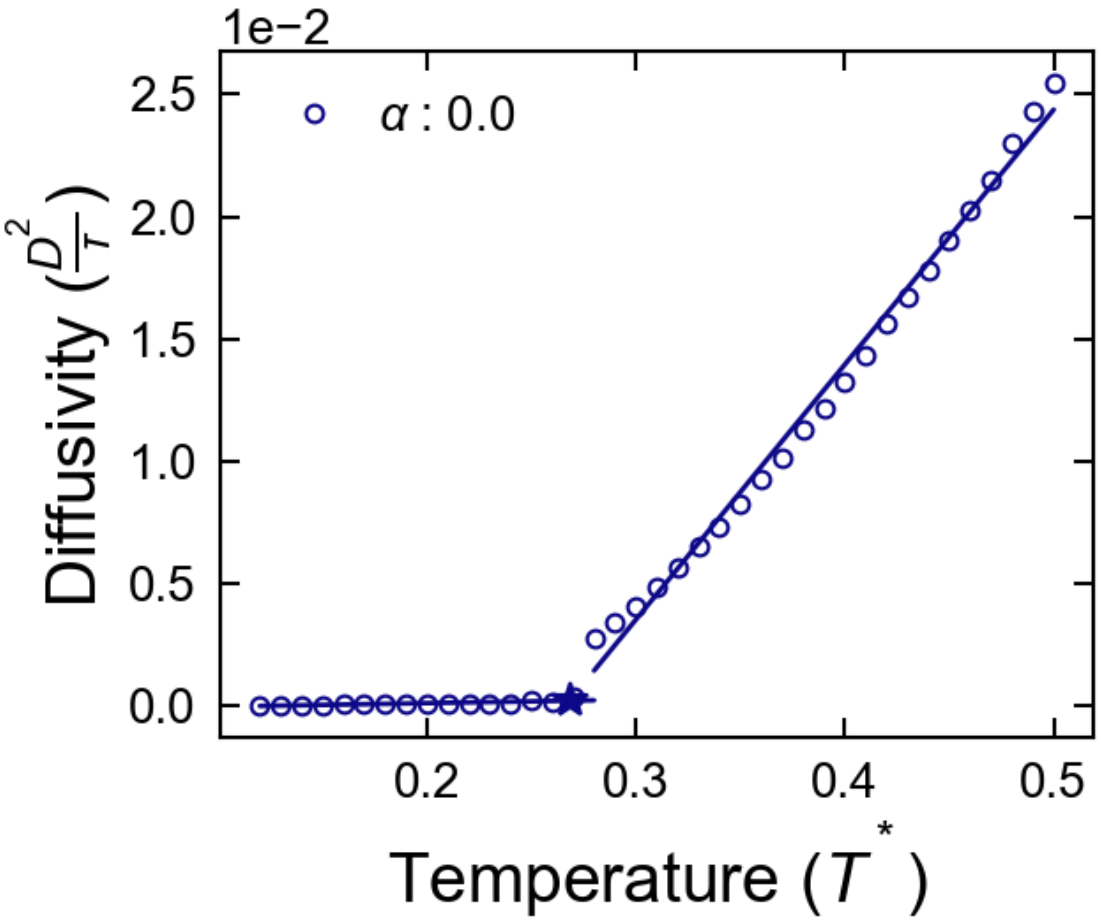
In [15]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#
,90]
```

```
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                       Ds_scaled,
                                       method='intersection',
                                       min_D=0,
                                       ver=1)#,
                                       #viscous_line_index=0,
                                       #l1_T_bounds=custom_ranges_l1[sap],
                                       #l2_T_bounds=custom_ranges_l2[sap])
```

```
                    line_vals.append((x1,y1))
                    line_vals.append((x2,y2))
                    xs = Ts#np.linspace(0.1,4)
                    plt.plot(Tg,
                             Tg_prop/mul_fact,
                             marker=Tg_marker,
                             #markerfacecolor='w',
                             color=colors[i],#cooling_colors[j],
                             markersize=15)#,
                    l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                    l_colors=['r','g']
                    for li,line_val in enumerate(line_vals):
                        xs=line_val[0]
                        ys=line_val[1]/mul_fact
                        plt.plot(xs,
                                 ys,
                                 color=colors[i],#cooling_colors[j],
                                 zorder=1,
                                 linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=20)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_0_percent.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 1.000479937
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

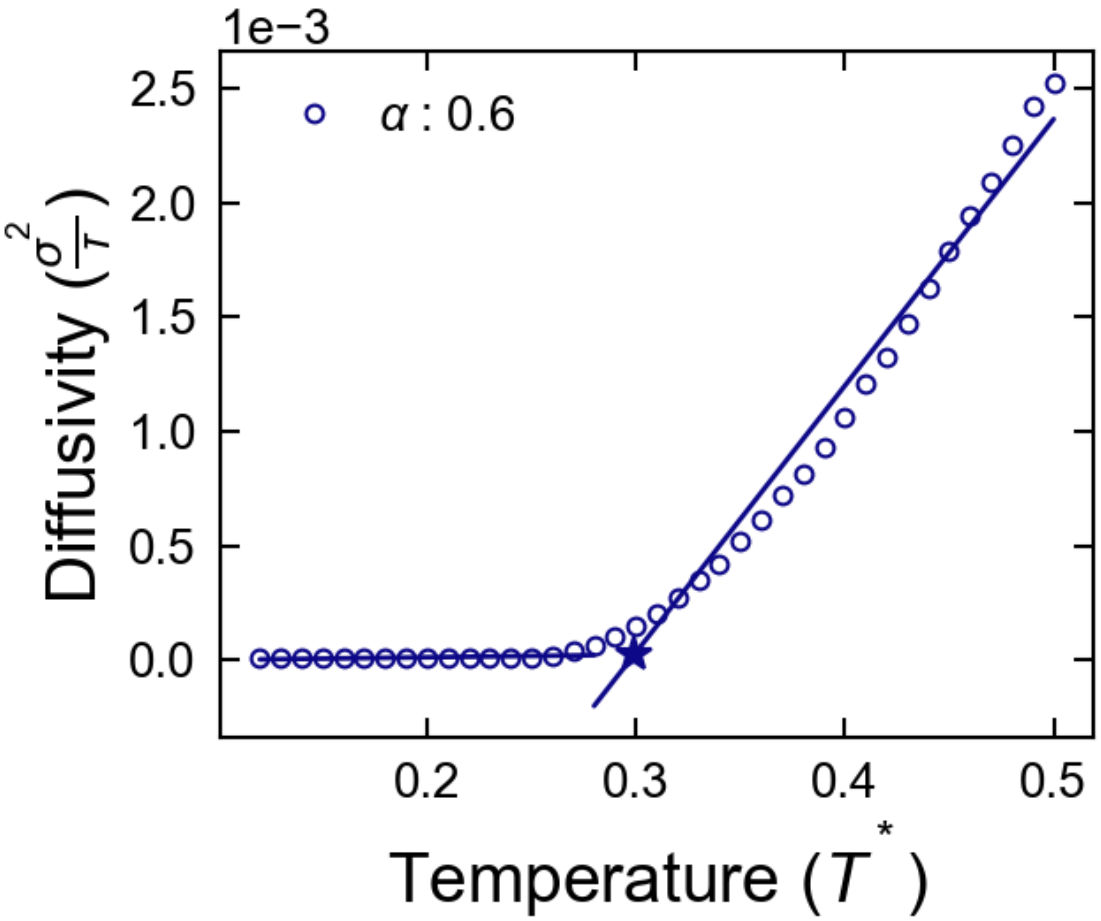



```
In [16]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[60.]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]
         #,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}
```

```
Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='intersection',
                                        min_D=0,
                                        ver=1)#,
                                        #viscous_line_index=0,
                                        #l1_T_bounds=custom_ranges_l1[sap],
                                        #l2_T_bounds=custom_ranges_l2[sap])
            line_vals.append((x1,y1))
            line_vals.append((x2,y2))
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
```

```
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_60_percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 61.0007782


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

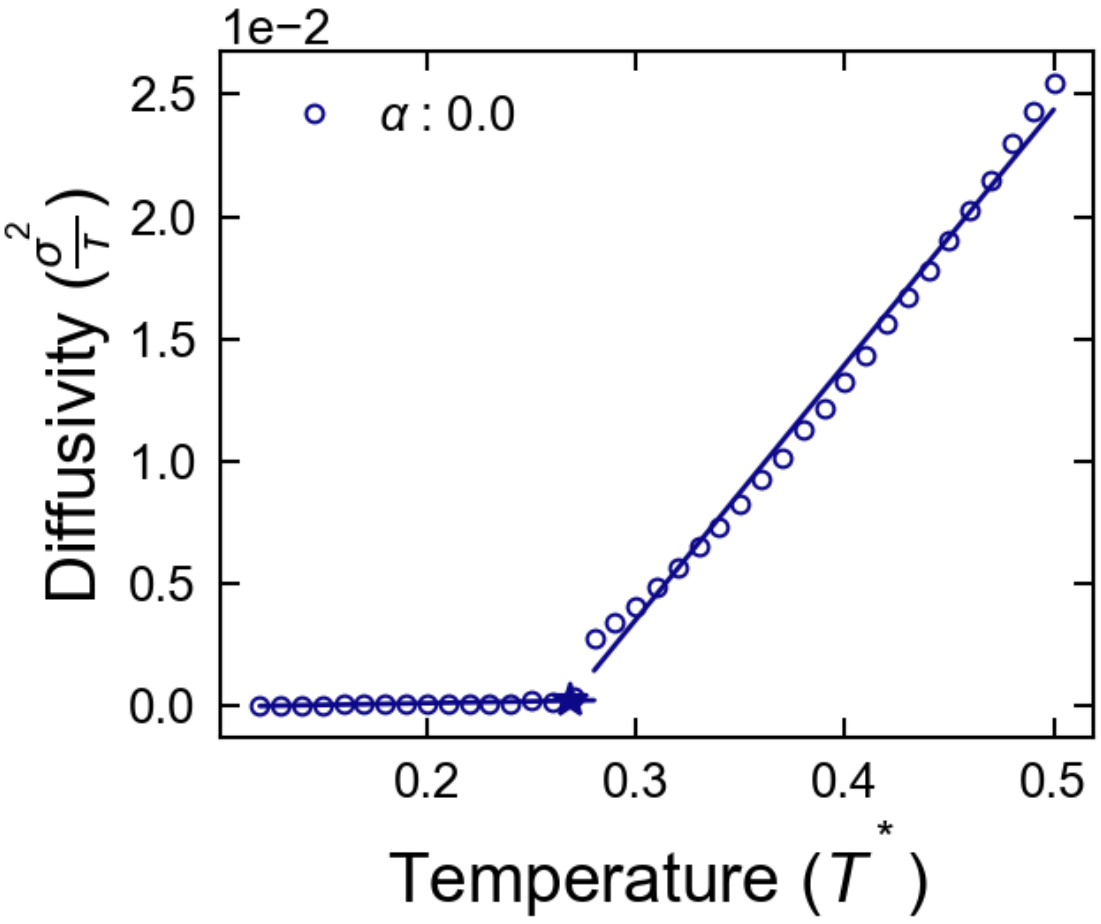



```
In [17]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#
         ,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}
```

```
Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='intersection',
                                        min_D=0,
                                        ver=1)#,
                                        #viscous_line_index=0,
                                        #l1_T_bounds=custom_ranges_l1[sap],
                                        #l2_T_bounds=custom_ranges_l2[sap])
            line_vals.append((x1,y1))
            line_vals.append((x2,y2))
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
```

```
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_all_Ts_0_percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 1.000479937


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



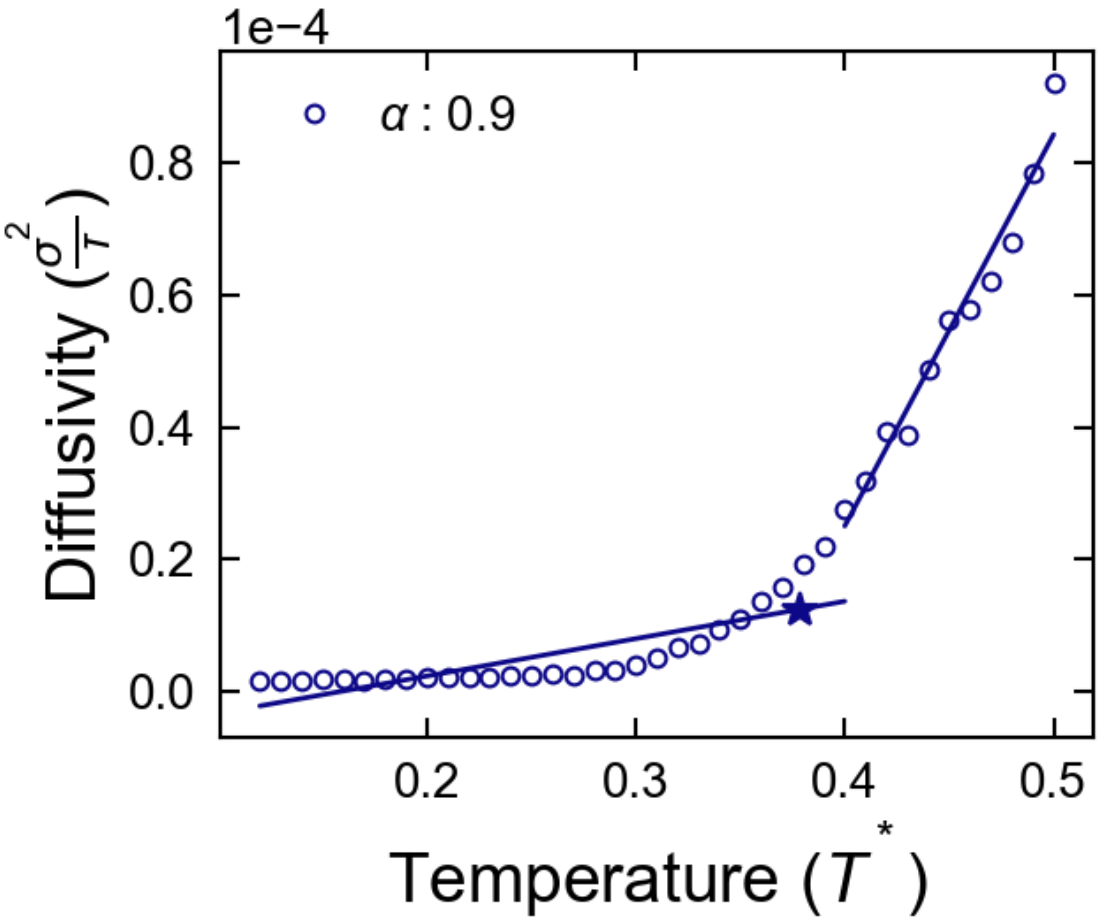

```
In [18]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[90.]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]
         #,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}
```

```
Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='intersection',
                                        min_D=0,
                                        ver=1)#,
                                        #viscous_line_index=0,
                                        #l1_T_bounds=custom_ranges_l1[sap],
                                        #l2_T_bounds=custom_ranges_l2[sap])
            line_vals.append((x1,y1))
            line_vals.append((x2,y2))
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
```

```
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_all_Ts_90_percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 91.00018311
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

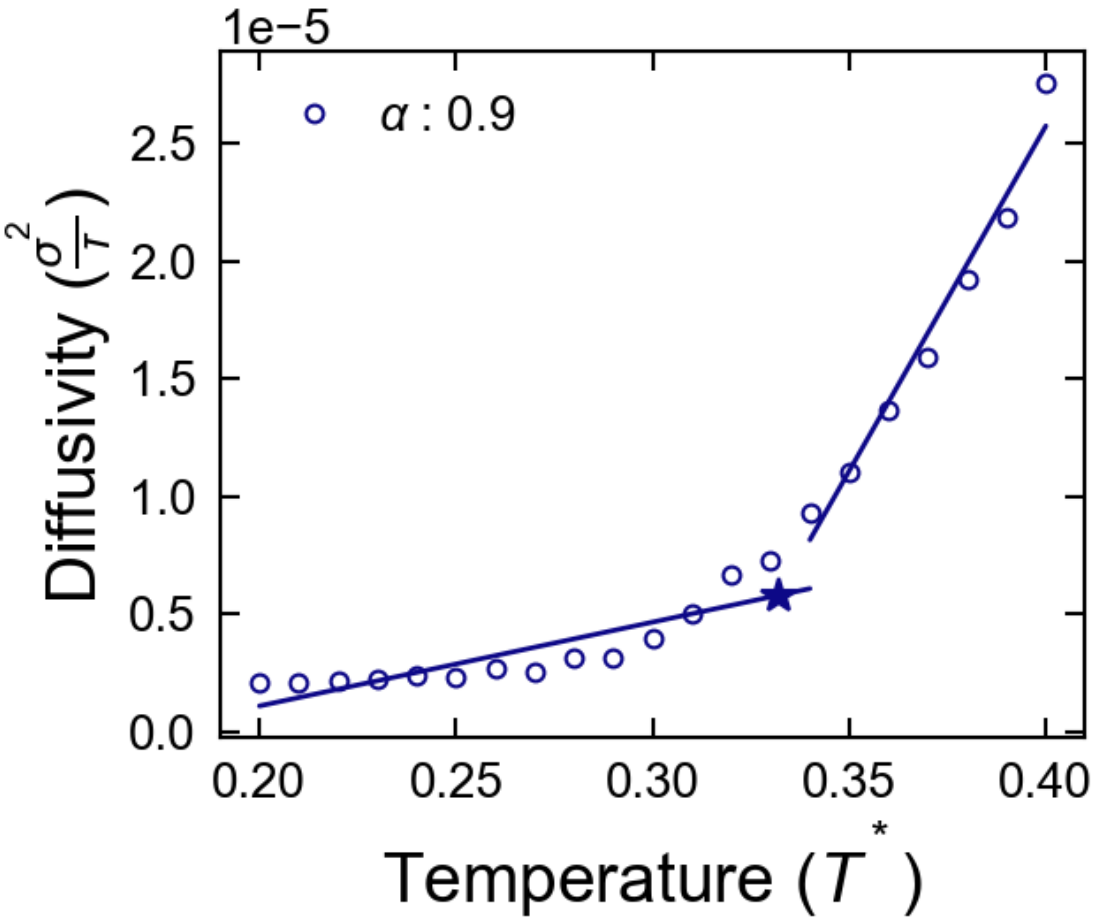



```
In [19]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[90.]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]
         #,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}
```

```
Tgs_dict={}
df_filtered=df[(df.quench_T<=0.4)&(df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='intersection',
                                        min_D=0,
                                        ver=1)#,
                                        #viscous_line_index=0,
                                        #l1_T_bounds=custom_ranges_l1[sap],
                                        #l2_T_bounds=custom_ranges_l2[sap])
            line_vals.append((x1,y1))
            line_vals.append((x2,y2))
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
```

```
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_90_percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 91.00018311


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

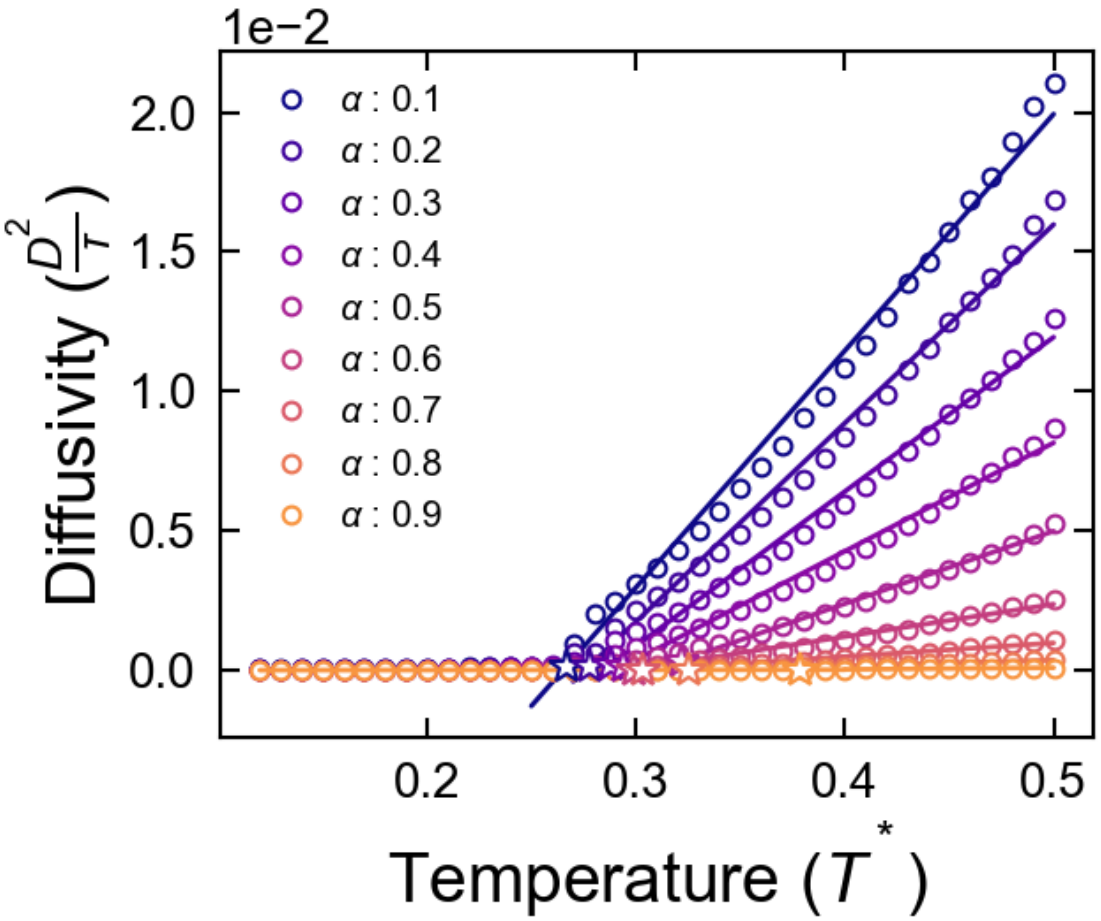



```
In [8]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[10,20,30,40,50,60,70,80,90]#,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0
        ,50.0,100.0]#,30,50,70]#,90]

        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'


        Tgs_dict={}
        df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
        for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
            Tgs=[]
```

```
cure_percents = []
for i,sap in enumerate(filter_saps):
    #print(cooling_method,sap)
    df_curing = df_grp[(df_grp.bond==False)&
                   (df_grp.cooling_method==cooling_method)&
                   (df_grp.stop_after_percent==sap)]
    cure_percent = df_curing.cure_percent.mean()
    print('cure_percent',cure_percent)
    cure_percents.append(cure_percent)
    if cooling_method == 'anneal':
        quench_time = None
        marker='o'
        zorder=1
        markersize=7
        Tg_marker='x'
    else:
        quench_time = 5e6
        marker='o'
        zorder=1
        markersize=7
        Tg_marker='*'
    Ts,Ds=getDiffusivities(project,
                           df_curing,
                           name=PROP_NAME,
                           quench_time=quench_time,
                           equilibriated_ts_percentage=0.0)
    Cure_Ts.append(Ts)
    #print('Ds',Ds)
    mul_fact=1
    Ds_scaled=Ds*mul_fact
    #print('custom_ranges_l2',custom_ranges_l2[i])

    plt.plot(Ts,
             Ds,
             marker=marker,
             markersize=markersize,
             markerfacecolor='w',
             zorder=zorder,
             color=colors[i],#cooling_colors[j],
             linewidth=0.0,
             label='$\\alpha$ : {:.1f}'.format(sap/100))
    if True:
        line_vals=[]
        Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                  Ds_scaled,
                                  method='intersection',
                                  min_D=0,
                                  ver=1)#,
                                  #viscous_line_index=0,
                                  #l1_T_bounds=custom_ranges_l1[sap],
                                  #l2_T_bounds=custom_ranges_l2[sap])
        line_vals.append((x1,y1))
        line_vals.append((x2,y2))
        xs = Ts#np.linspace(0.1,4)
        plt.plot(Tg,
                 Tg_prop/mul_fact,
                 marker=Tg_marker,
                 markerfacecolor='w',
                 color=colors[i],#cooling_colors[j],
                 markersize=15)#,
        l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            plt.plot(xs,
                     ys,
                     color=colors[i],#cooling_colors[j],
```

```
                                zorder=1,
                                linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=15)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            savefig(plt,'piecewise_regression_symmetric_LJ','quench_lj_asym_all_alphas.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

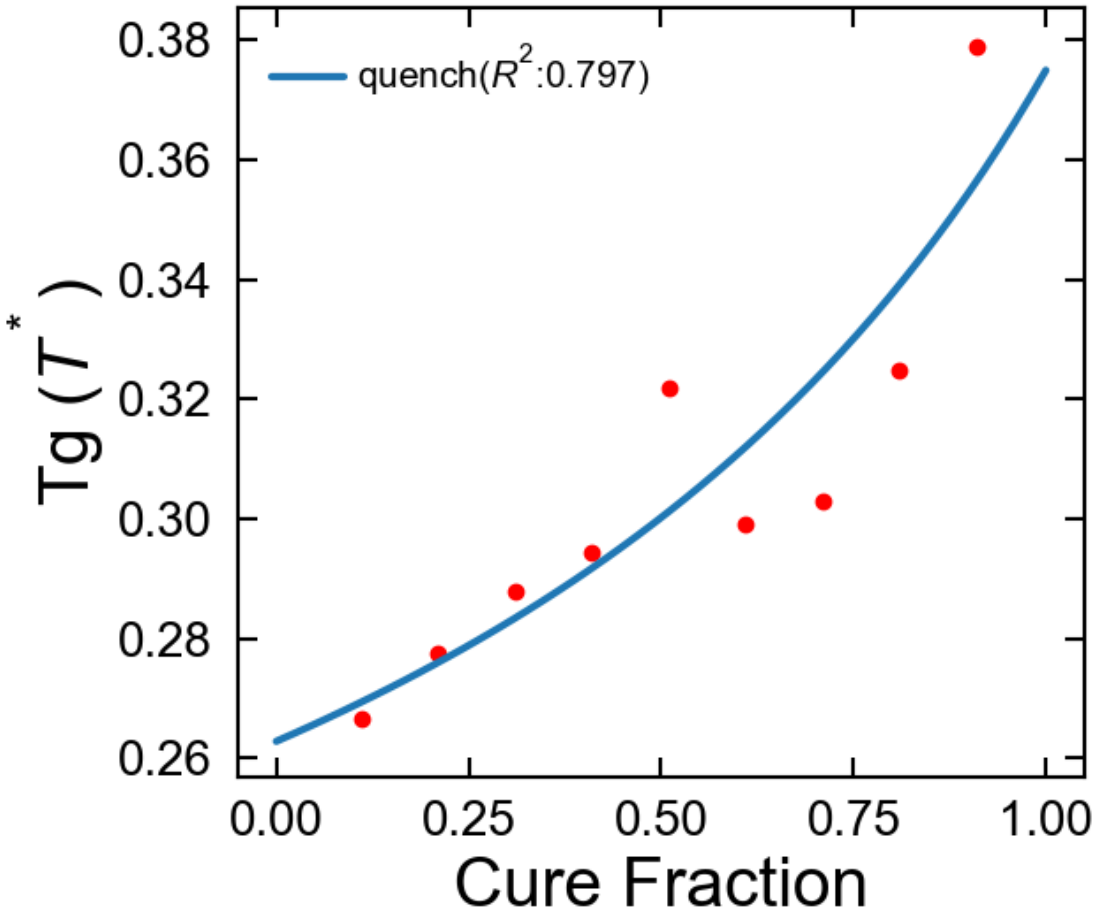
```
cure_percent 11.00078011
cure_percent 21.00107956
cure_percent 31.00138092
cure_percent 41.00017929
cure_percent 51.00048065
cure_percent 61.0007782
cure_percent 71.00108337
cure_percent 81.00138092
cure_percent 91.00018311


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [9]:
```
fig, ax1 = plt.subplots()


cure_percents,Tgs = Tgs_dict['quench']
cure_percents = np.asarray(cure_percents)
Tgs = np.asarray(Tgs)
Tg_data = np.asarray([cure_percents/100.,Tgs])
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                    Tgs_ss,
                                                    T1=None,
                                                    T0=None)
alphas = np.linspace(0,1)
fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
ax1.plot(alphas,
         fit_ydata,
         label='{}($R^2$:{})'.format(cooling_method,round(R2,3)))
ax1.scatter(cure_percents/100.,
            Tgs,
            #label='$E_a$:{}'.format(activation_energy),
            color='r')#colors[i])


ax1.set_xlabel('Cure Fraction')
ax1.set_ylabel('Tg ($T^*$)')
```

```
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_symmetric_LJ','dibeneditto_quench_PRM.pdf')
        #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



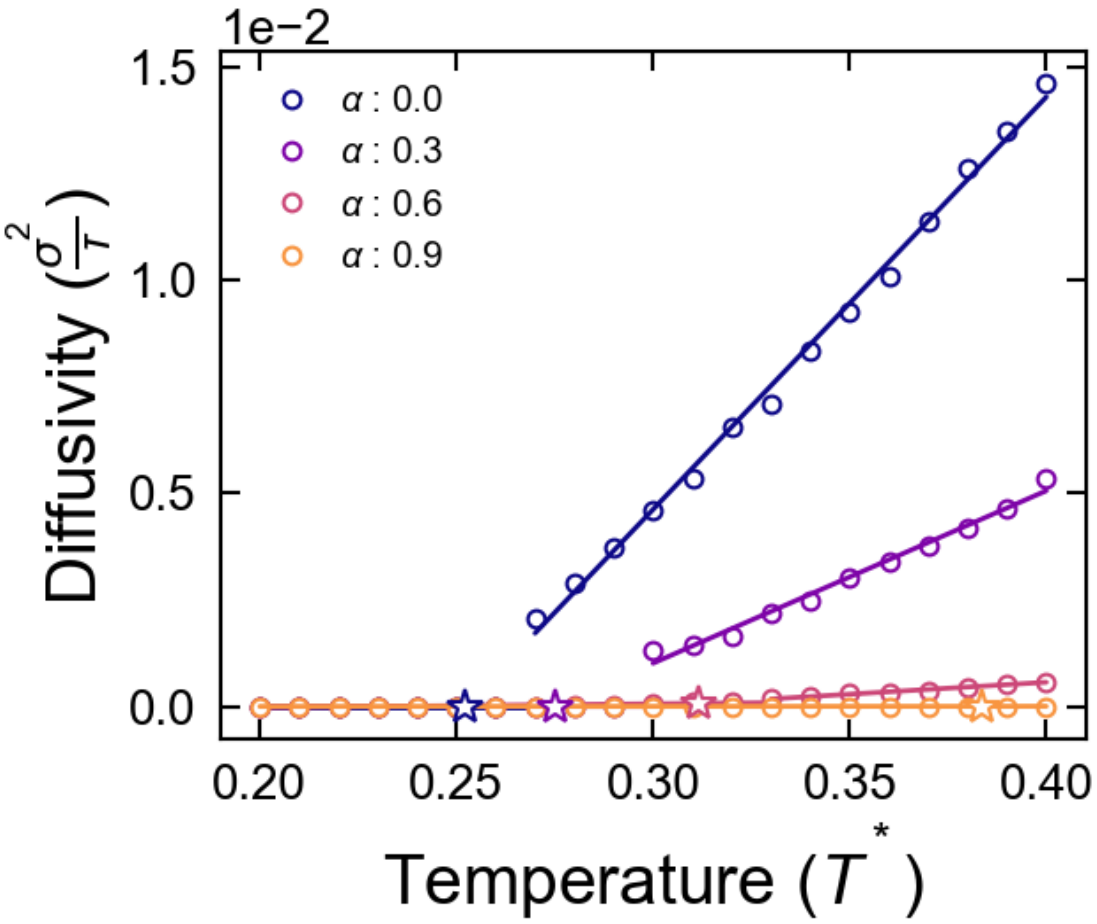
In [10]:
```
        import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,9
        0]
```

```
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='anneal'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            line_vals=[]
            Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                       Ds_scaled,
                                       method='intersection',
                                       min_D=0,
                                       ver=1,
                                       viscous_line_index=0,
                                       l1_T_bounds=custom_ranges_l1[sap],
                                       l2_T_bounds=custom_ranges_l2[sap])
```

```
                    line_vals.append((x1,y1))
                    line_vals.append((x2,y2))
                    xs = Ts#np.linspace(0.1,4)
                    plt.plot(Tg,
                             Tg_prop/mul_fact,
                             marker=Tg_marker,
                             markerfacecolor='w',
                             color=colors[i],#cooling_colors[j],
                             markersize=15)#,
                    l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                    l_colors=['r','g']
                    for li,line_val in enumerate(line_vals):
                        xs=line_val[0]
                        ys=line_val[1]/mul_fact
                        plt.plot(xs,
                                 ys,
                                 color=colors[i],#cooling_colors[j],
                                 zorder=1,
                                 linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=15)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            savefig(plt,'piecewise_regression_symmetric_LJ','quench.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 1.000479937
cure_percent 31.00138092
cure_percent 61.0007782
cure_percent 91.00018311
```

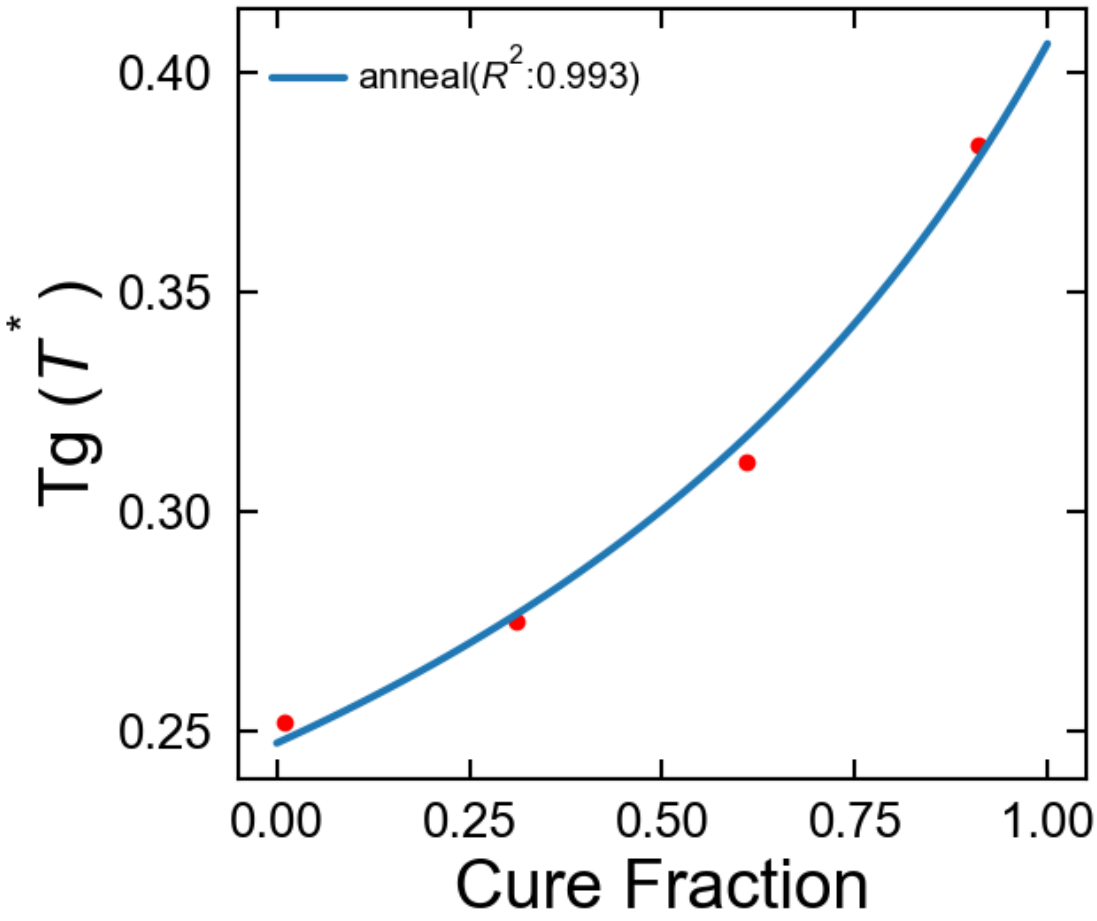
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [11]:
```
fig, ax1 = plt.subplots()

for cooling_method,[cure_percents,Tgs] in Tgs_dict.items():
    cure_percents = np.asarray(cure_percents)
    Tgs = np.asarray(Tgs)
    Tg_data = np.asarray([cure_percents/100.,Tgs])
    cure_percents_ss = cure_percents#[:-1]
    Tgs_ss = Tgs#[:-1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                        Tgs_ss,
                                                        T1=None,
                                                        T0=None)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    ax1.plot(alphas,
             fit_ydata,
             label='{}($R^2$:{})'.format(cooling_method,round(R2,3)))
    ax1.scatter(cure_percents/100.,
                Tgs,
                #label='$E_a$:{}'.format(activation_energy),
                color='r')#colors[i])


ax1.set_xlabel('Cure Fraction')
ax1.set_ylabel('Tg ($T^*$)')
ax1.legend(fontsize=15,loc='upper left')
```

```
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_symmetric_LJ','dibeneditto_anneal_quench.pdf')
        #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



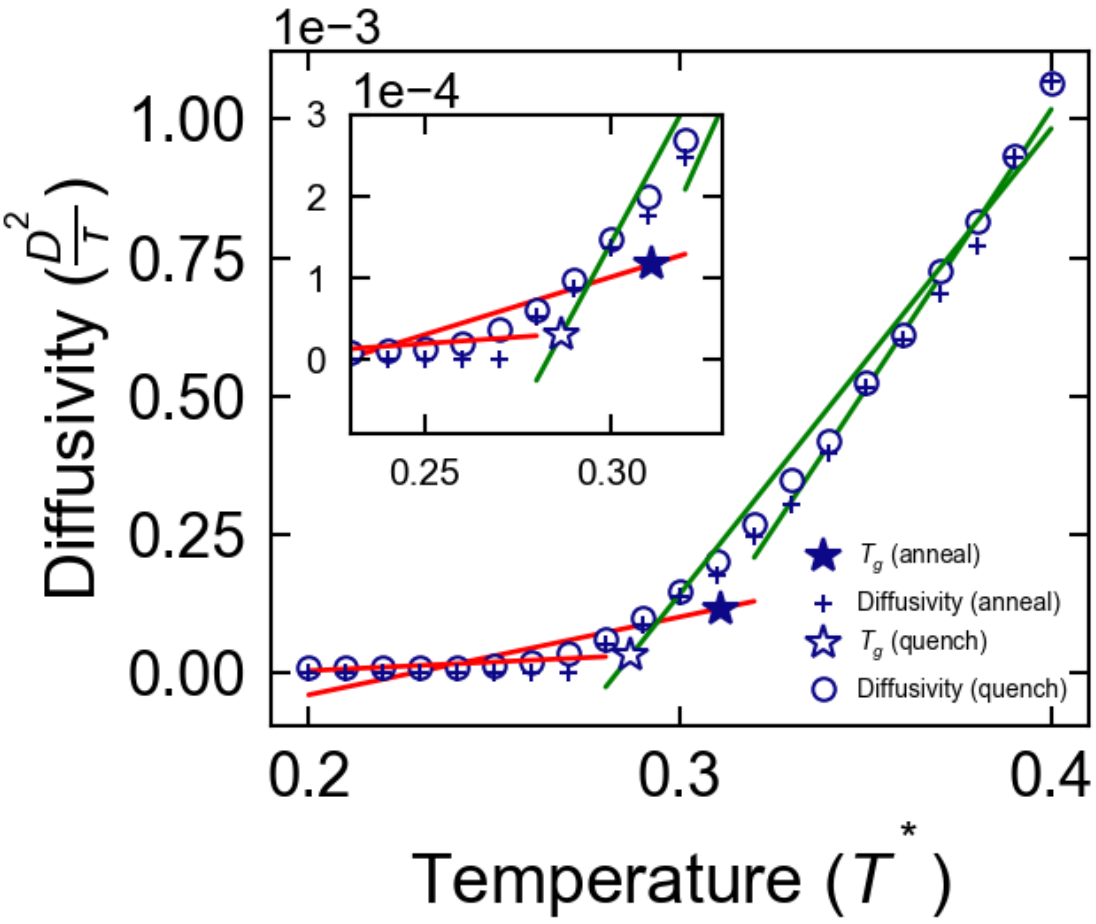
In [11]:
```
        import matplotlib
        #from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.34, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
```

```
#plt.figure()
filter_saps=[60.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.25],
                  90.0:[0.1,0.25]}
custom_ranges_l2={00.0:[0.29,0.35],
                  30.0:[0.30,0.4],
                  60.0:[0.25,0.34],
                  90.0:[0.27,0.35]}

df_filtered=df[(df.quench_T<=0.4)&(df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0
.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        if len(Ds)==0:
            raise ValueError('getDifusivities did not find Ds')
        Cure_Ts.append(Ts)

        mul_fact=1
        Ds_scaled=Ds*mul_fact
        line_vals=[]
        Tg,Tg_prop,x1,y1,x2,y2 = Fit_Diffusivity1(Ts,
                                  Ds_scaled,
                                  method='intersection',
                                  min_D=0,
                                  ver=1,
                                  viscous_line_index=0,
                                  l1_T_bounds=custom_ranges_l1[sap],
                                  l2_T_bounds=custom_ranges_l2[sap])
        line_vals.append((x1,y1))
        line_vals.append((x2,y2))
        xs = Ts#np.linspace(0.1,4)
        plot_lines += ax1.plot(Tg,
```

```
                            Tg_prop/mul_fact,
                            marker='*',
                            markerfacecolor=Tg_markerfacecolor,
                            color=colors[i],
                            zorder=3,
                            markersize=15,
                            linewidth=0,
                            label='$T_g$ ({})'.format(cooling_method))#,
        ax2.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 markerfacecolor=Tg_markerfacecolor,
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,
        plot_lines += ax1.plot(Ts,
                            Ds,
                            marker=marker,
                            markerfacecolor='w',
                            markersize=markersize,
                            color=colors[i],#cooling_colors[j],
                            linewidth=0.0,
                            zorder=zorder,
                            label='Diffusivity ({})'.format(cooling_method))
        ax2.plot(Ts,
                 Ds,
                 marker=marker,
                 markerfacecolor='w',
                 markersize=markersize,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=zorder)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            ax1.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=1,
                     linewidth=2)
            ax2.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=1,
                     linewidth=2)
        Tgs.append(Tg)
    #break
labels = [l.get_label() for l in plot_lines]
ax1.legend(plot_lines,
           labels,
           fontsize=10,
           loc='lower right')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=25)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
#ax2.set_xlim(0.22,0.33)
#ax2.set_ylim(-7e-4,3e-3)
ax2.set_xlim(0.23,0.33)
ax2.set_ylim(-9e-5,3e-4)
#ax2.set_xlim(0.25,0.37)
```

```
        #ax2.set_ylim(-8e-6,2e-5)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,
                'piecewise_regression_symmetric_LJ',
                '60percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



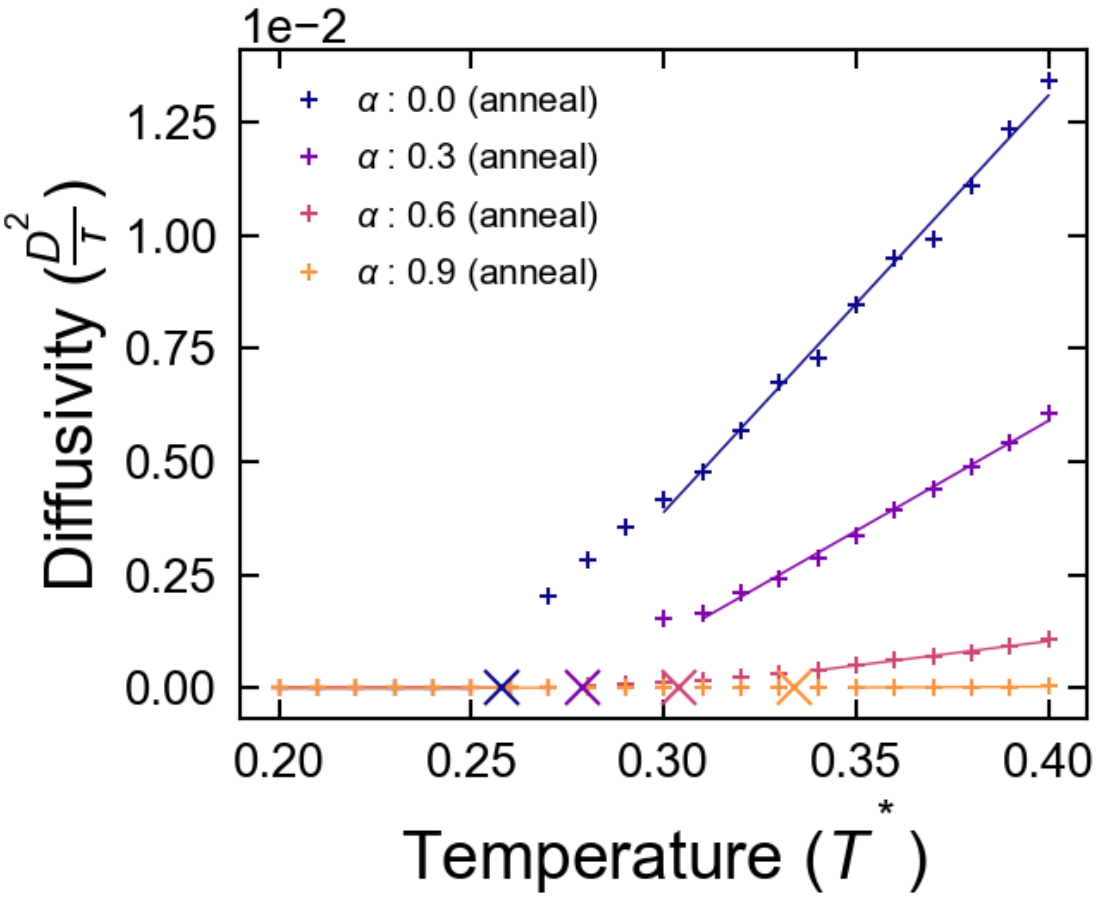
In [12]:
```
        import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
```

```
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,9
0]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='anneal'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
df_filtered=df[(df.quench_T<=0.4)&(df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=0
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f} ({})'.format(sap/100,cooling_method))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                    Ds_scaled,
                                    method='use_viscous_region',
                                    min_D=0,
```

```
                                        ver=4,
                                        viscous_line_index=0,
                                        l1_T_bounds=custom_ranges_l1[sap],
                                        l2_T_bounds=custom_ranges_l2[sap])
                    xs = Ts#np.linspace(0.1,4)
                    plt.plot(Tg,
                             Tg_prop/mul_fact,
                             marker=Tg_marker,
                             color=colors[i],#cooling_colors[j],
                             markersize=15)#,
                    l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                    l_colors=['r','g']
                    for li,line_val in enumerate(line_vals):
                        xs=line_val[0]
                        ys=line_val[1]/mul_fact
                        plt.plot(xs,
                                 ys,
                                 color=colors[i],#cooling_colors[j],
                                 zorder=0,
                                 linewidth=1)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=15)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 1.000479937
cure_percent 31.00138092
cure_percent 61.0007782
cure_percent 91.00018311
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−2
1.25
1.00
0.75
0.50
0.25
0.00
Diffusivity ($\frac{D^2}{T}$)
α : 0.0 (anneal)
α : 0.3 (anneal)
α : 0.6 (anneal)
α : 0.9 (anneal)
0.20
0.25
0.30
0.35
0.40
Temperature (T^*)

In [12]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [13]:
```
def get_custom_ranges(cooling_method):
    if cooling_method=='quench':
        custom_ranges_l1={00.0:[0.1,0.25],
                          30.0:[0.1,0.25],
                          60.0:[0.1,0.25],
                          90.0:[0.1,0.25]}
        custom_ranges_l2={00.0:[0.28,0.32],
                          30.0:[0.29,0.33],
                          60.0:[0.28,0.36],
                          90.0:[0.30,0.37]}
    elif cooling_method=='anneal':
        custom_ranges_l1={00.0:[0.1,0.25],
                    30.0:[0.1,0.25],
                    60.0:[0.1,0.25],
                    90.0:[0.1,0.30]}
        custom_ranges_l2={00.0:[0.26,0.31],
                        30.0:[0.29,0.35],
                        60.0:[0.27,0.36],
                        90.0:[0.30,0.37]}
    else:
        raise ValueError(cooling_method+'is unknown')
    return custom_ranges_l1, custom_ranges_l2
```

In [14]:
```
df_filtered=df[df.cooling_method=='anneal']
df_filtered.stop_after_percent
```

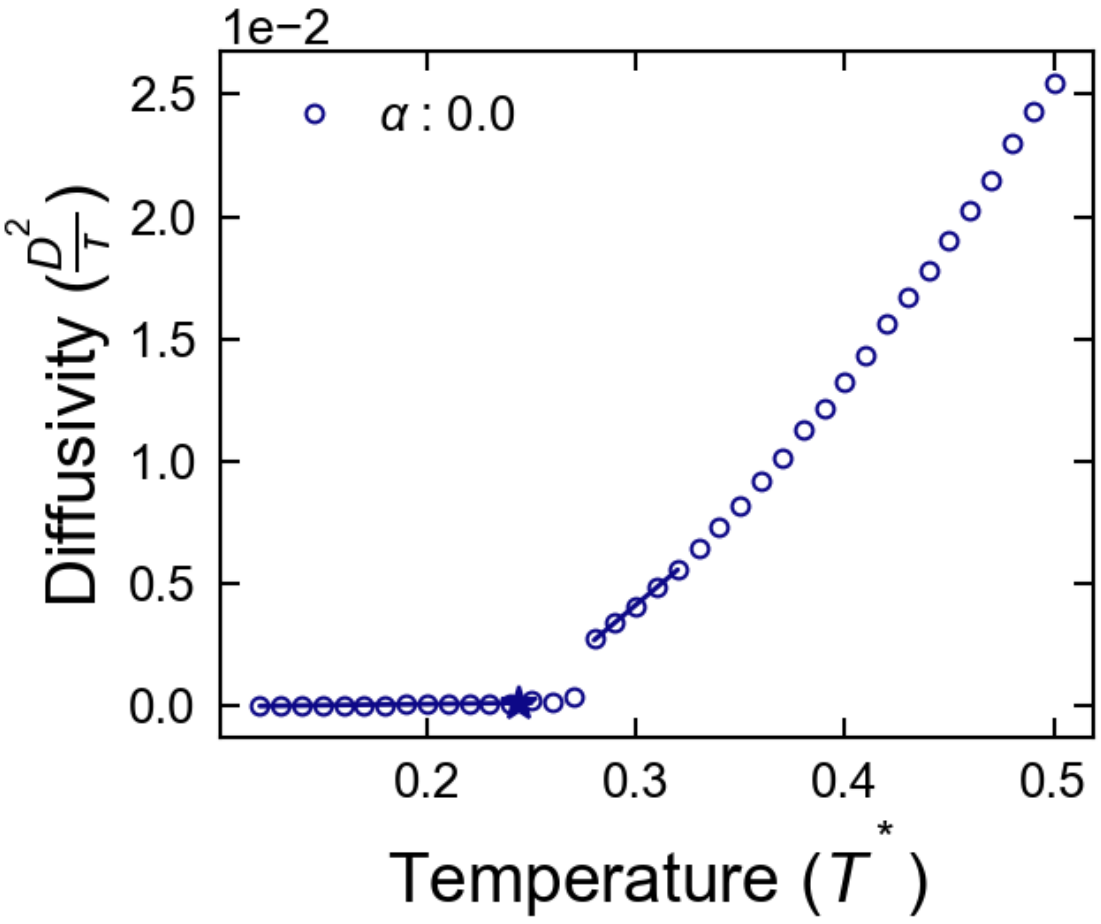
```
Out[14]: 90b99ca468d650a13ade4d55a27ce7e1    30
         ffd54826a80437a447973bbf73be3f68    90
         644d27b8f740cc6df4a628957245df0c    60
         646df3239f44bb1705e0ff4c9a58106e     0
         Name: stop_after_percent, dtype: object
```

```
In [15]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}

         Tgs_dict={}
         df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
         (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
         for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
             Tgs=[]
             cure_percents = []
             for i,sap in enumerate(filter_saps):
                 #print(cooling_method,sap)
                 df_curing = df_grp[(df_grp.bond==False)&
                                    (df_grp.cooling_method==cooling_method)&
                                    (df_grp.stop_after_percent==sap)]
                 cure_percent = df_curing.cure_percent.mean()
                 print('cure_percent',cure_percent)
                 cure_percents.append(cure_percent)
                 custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
                 if cooling_method == 'anneal':
                     quench_time = None
                     marker='o'
                     zorder=1
                     markersize=7
                     Tg_marker='x'
                 else:
                     quench_time = 5e6
                     marker='o'
                     zorder=1
                     markersize=7
                     Tg_marker='*'
                 Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
                 Cure_Ts.append(Ts)
                 #print('Ds',Ds)
                 mul_fact=1
                 Ds_scaled=Ds*mul_fact
                 #print('custom_ranges_l2',custom_ranges_l2[i])
```

```
                plt.plot(Ts,
                         Ds,
                         marker=marker,
                         markersize=markersize,
                         markerfacecolor='w',
                         zorder=zorder,
                         color=colors[i],#cooling_colors[j],
                         linewidth=0.0,
                         label='$\\alpha$ : {:.1f}'.format(sap/100))
                if True:
                    Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                                Ds_scaled,
                                                method='use_viscous_region',
                                                min_D=0,
                                                ver=4,
                                                viscous_line_index=0,
                                                l1_T_bounds=custom_ranges_l1[sap],
                                                l2_T_bounds=custom_ranges_l2[sap])
                    xs = Ts#np.linspace(0.1,4)
                    plt.plot(Tg,
                             Tg_prop/mul_fact,
                             marker=Tg_marker,
                             #markerfacecolor='w',
                             color=colors[i],#cooling_colors[j],
                             markersize=15)#,
                    l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                    l_colors=['r','g']
                    for li,line_val in enumerate(line_vals):
                        xs=line_val[0]
                        ys=line_val[1]/mul_fact
                        plt.plot(xs,
                                 ys,
                                 color=colors[i],#cooling_colors[j],
                                 zorder=1,
                                 linewidth=2)
                    Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench_custom_0_percent.pd
        f')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 1.000479937
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

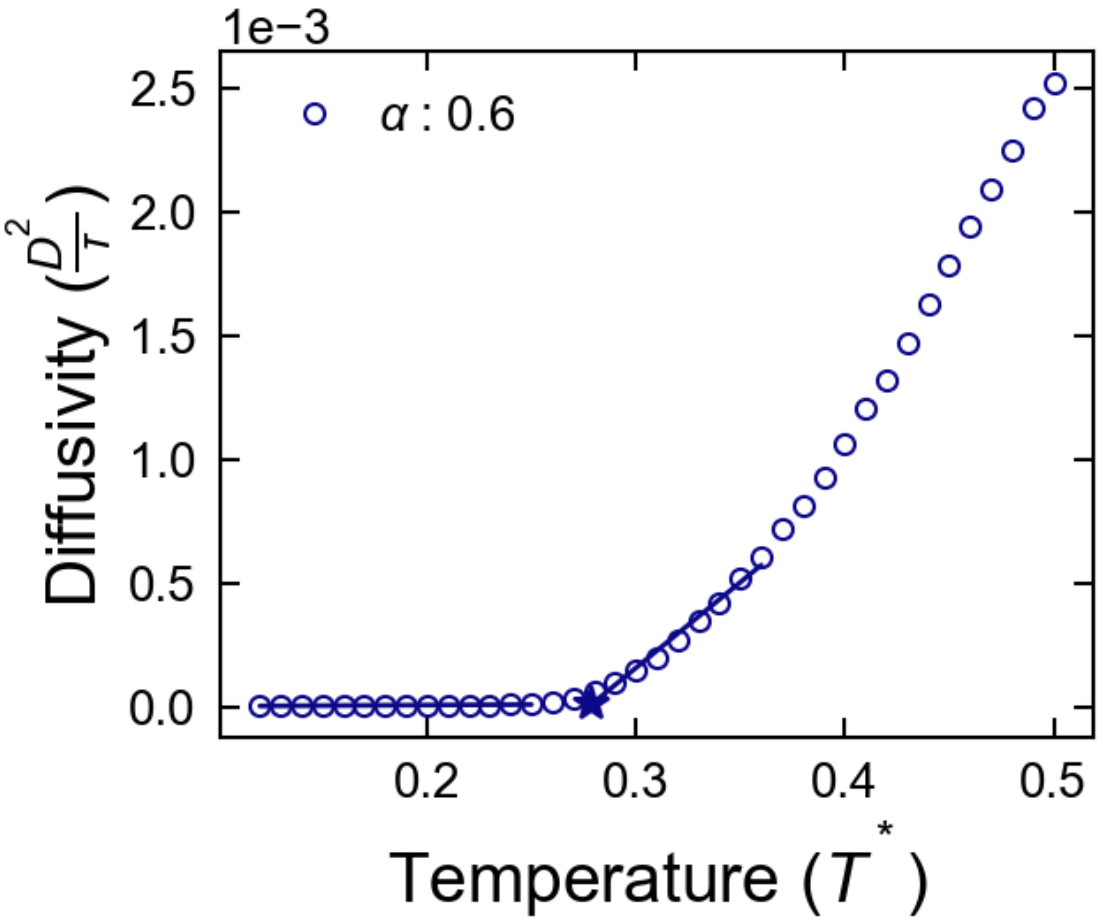


In [16]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[60.]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
```

```
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = 6e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                    Ds_scaled,
                                    method='use_viscous_region',
                                    min_D=0,
                                    ver=4,
                                    viscous_line_index=0,
                                    l1_T_bounds=custom_ranges_l1[sap],
                                    l2_T_bounds=custom_ranges_l2[sap])
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=colors[i],#cooling_colors[j],
```

```
                                zorder=1,
                                linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=20)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench_custom_60_percent.p
            df')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 61.0007782


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



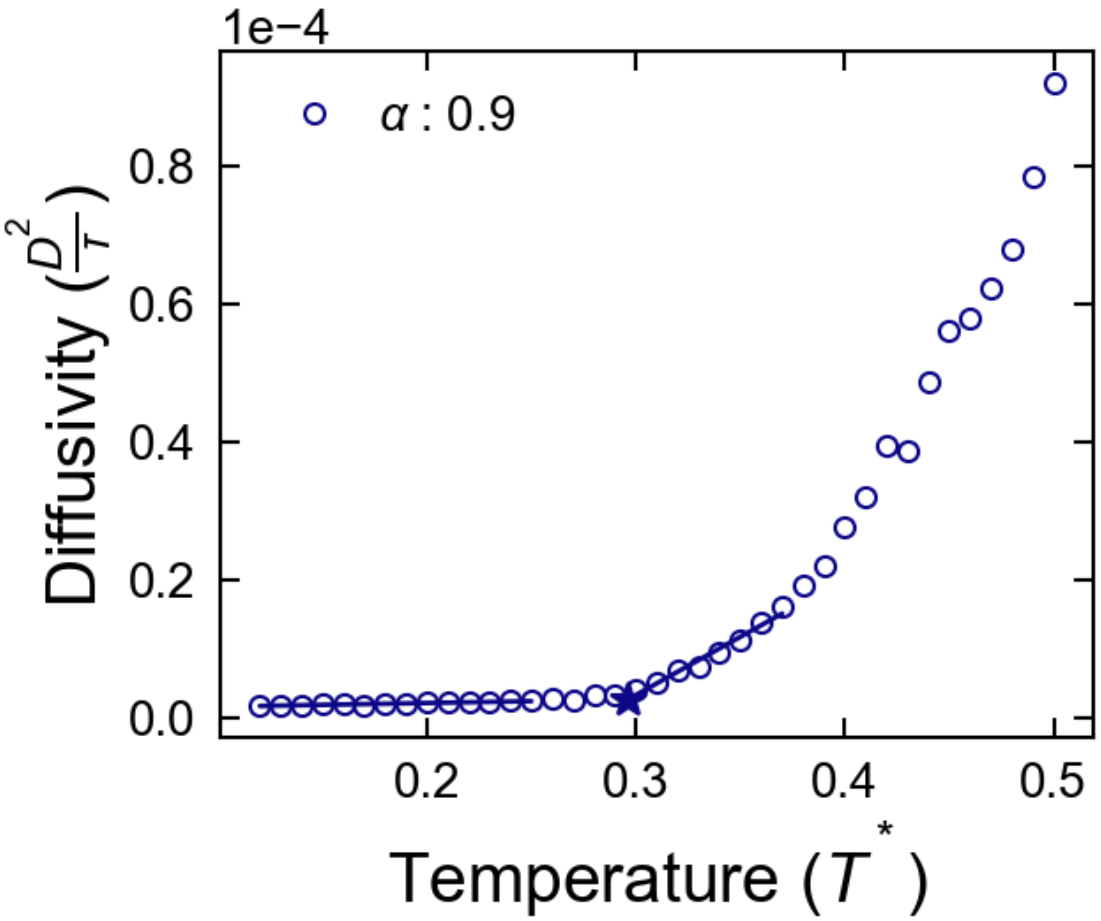

```
In [17]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[90.]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]

         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1={00.0:[0.1,0.25],
                           30.0:[0.1,0.25],
                           60.0:[0.1,0.26],
                           90.0:[0.1,0.27]}
         custom_ranges_l2={00.0:[0.29,0.4],
                           30.0:[0.30,0.4],
                           60.0:[0.33,0.4],
                           90.0:[0.35,0.4]}

         Tgs_dict={}
```

```
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = 6e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                    Ds_scaled,
                                    method='use_viscous_region',
                                    min_D=0,
                                    ver=4,
                                    viscous_line_index=0,
                                    l1_T_bounds=custom_ranges_l1[sap],
                                    l2_T_bounds=custom_ranges_l2[sap])
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=colors[i],#cooling_colors[j],
```

```
                                zorder=1,
                                linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench_custom_90_percent.p
        df')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 91.00018311
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



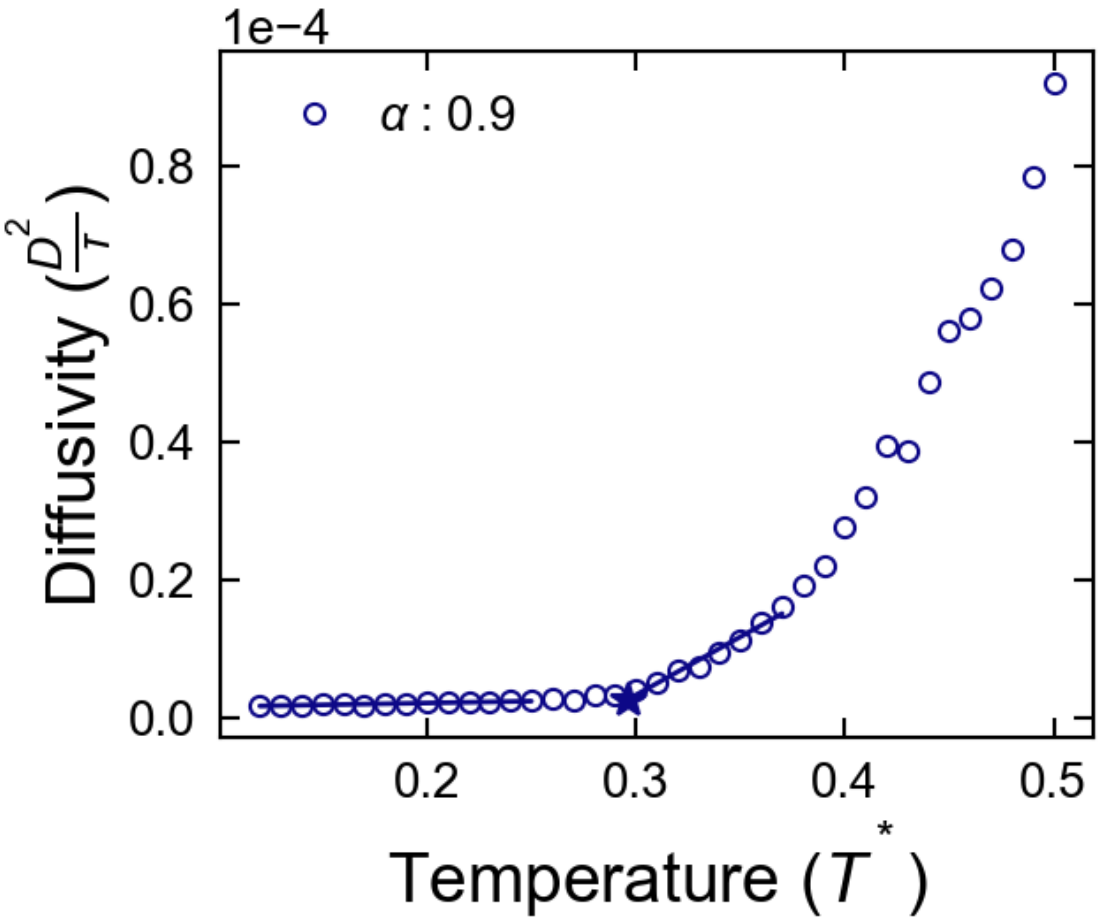
In [18]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[90.]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
```

```
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = 6e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                    Ds_scaled,
                                    method='use_viscous_region',
                                    min_D=0,
                                    ver=4,
                                    viscous_line_index=0,
                                    l1_T_bounds=custom_ranges_l1[sap],
                                    l2_T_bounds=custom_ranges_l2[sap])
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     #markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=colors[i],#cooling_colors[j],
```

```
                                zorder=1,
                                linewidth=2)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=20)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench_custom_all_Ts_90_pe
            rcent.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 91.00018311
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [ ]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[80.]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
```

```
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = 6e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=8
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                    Ds_scaled,
                                    method='use_viscous_region',
                                    min_D=0,
                                    ver=4,
                                    viscous_line_index=0,
                                    l1_T_bounds=custom_ranges_l1[sap],
                                    l2_T_bounds=custom_ranges_l2[sap])
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker=Tg_marker,
                     markerfacecolor='w',
                     color=colors[i],#cooling_colors[j],
                     markersize=15)#,
            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=colors[i],#cooling_colors[j],
```

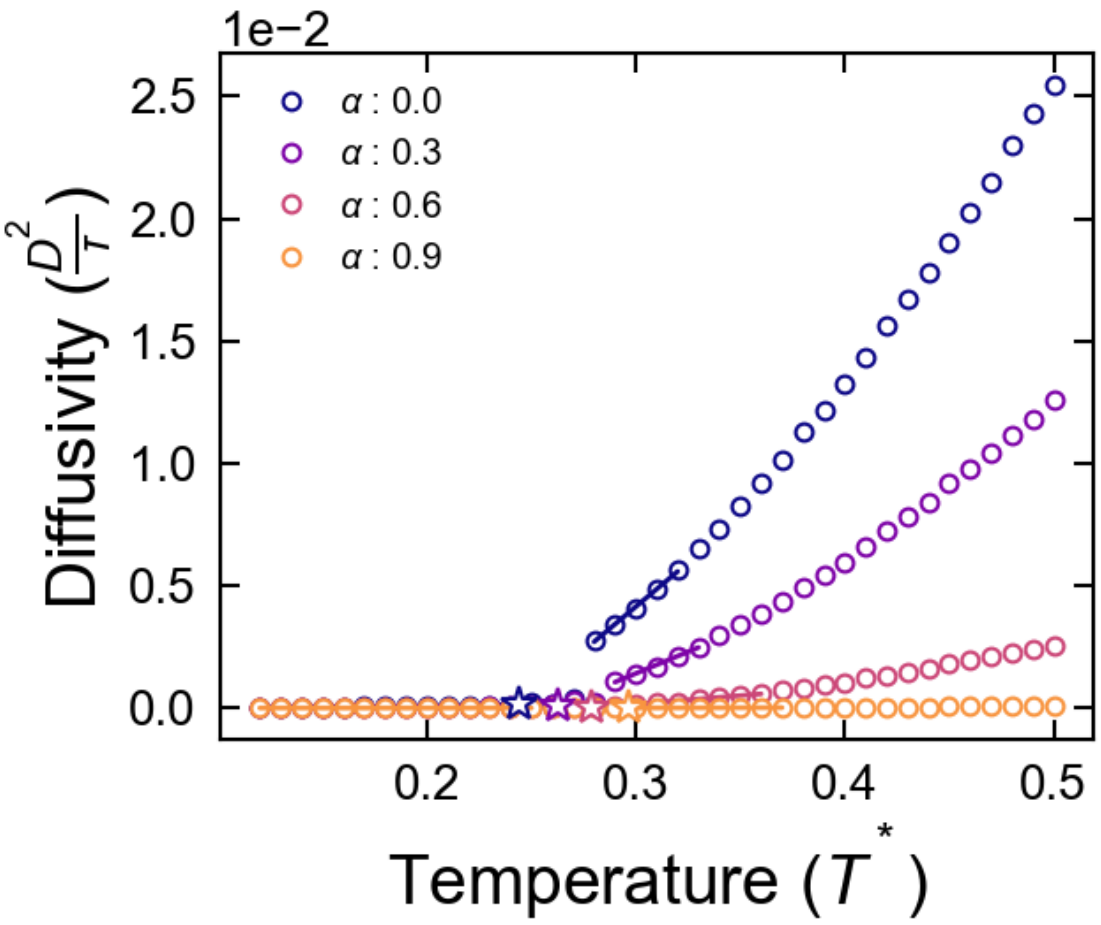
```
                            zorder=1,
                            linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=20)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench_custom_80_percent.p
        df')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()

In [8]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        import collections
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,9
        0]

        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'


        Tgs_dict=collections.OrderedDict()
        df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
        for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
            Tgs=[]
            cure_percents = []
            for i,sap in enumerate(filter_saps):
                #print(cooling_method,sap)
                df_curing = df_grp[(df_grp.bond==False)&
                               (df_grp.cooling_method==cooling_method)&
                               (df_grp.stop_after_percent==sap)]
                cure_percent = df_curing.cure_percent.mean()
                print('cure_percent',cure_percent)
                cure_percents.append(cure_percent)

                custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)

                if cooling_method == 'anneal':
                    quench_time = None
                    marker='o'
                    zorder=1
                    markersize=7
                    Tg_marker='x'
                else:
```

```
                quench_time = 5e6
                marker='o'
                zorder=1
                markersize=7
                Tg_marker='*'
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
            Cure_Ts.append(Ts)
            #print('Ds',Ds)
            mul_fact=1
            Ds_scaled=Ds*mul_fact
            #print('custom_ranges_l2',custom_ranges_l2[i])

            plt.plot(Ts,
                     Ds,
                     marker=marker,
                     markersize=markersize,
                     markerfacecolor='w',
                     zorder=zorder,
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0,
                     label='$\\alpha$ : {:.1f}'.format(sap/100))
            if True:
                Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='use_viscous_region',
                                        min_D=0,
                                        ver=4,
                                        viscous_line_index=0,
                                        l1_T_bounds=custom_ranges_l1[sap],
                                        l2_T_bounds=custom_ranges_l2[sap])
                xs = Ts#np.linspace(0.1,4)
                plt.plot(Tg,
                         Tg_prop/mul_fact,
                         marker=Tg_marker,
                         markerfacecolor='w',
                         color=colors[i],#cooling_colors[j],
                         markersize=15)#,
                l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                l_colors=['r','g']
                for li,line_val in enumerate(line_vals):
                    xs=line_val[0]
                    ys=line_val[1]/mul_fact
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        #break
    plt.legend(fontsize=15)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    #plt.xlim(0.5,1.5)
    #plt.ylim(-1e-4,5e-4)
    plt.xlabel('Temperature ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
    #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
    savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','quench.pdf')
    #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
    spose(data))

    plt.show()
```

```
cure_percent 1.000479937
cure_percent 31.00138092
```

```
cure_percent 61.0007782
cure_percent 91.00018311
```

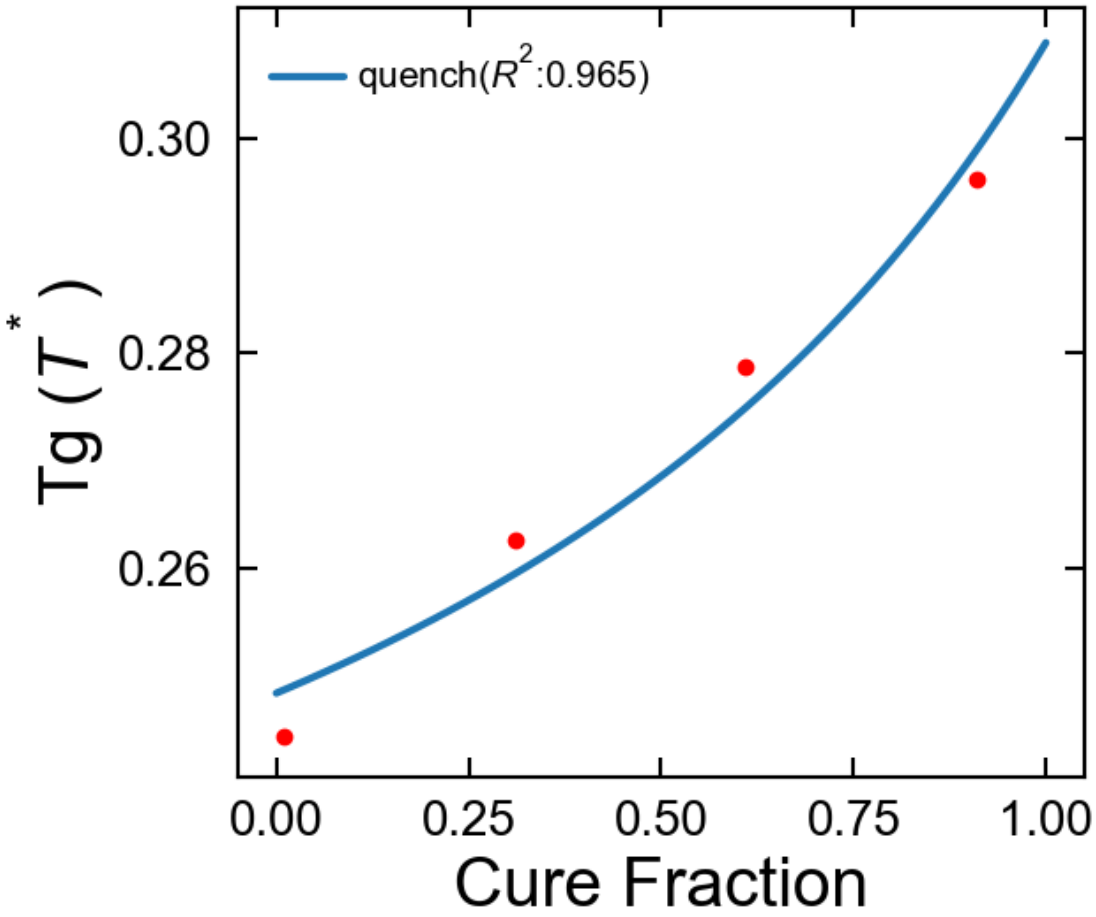
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [9]:
```
fig, ax1 = plt.subplots()


        cure_percents,Tgs = Tgs_dict['quench']
        cure_percents = np.asarray(cure_percents)
        Tgs = np.asarray(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        alphas = np.linspace(0,1)
```

```
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,
                 fit_ydata,
                 label='{}($R^2$:{})'.format(cooling_method,round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])


        ax1.set_xlabel('Cure Fraction')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','dibeneditto_quench_PRMc.pd
        f')
        #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

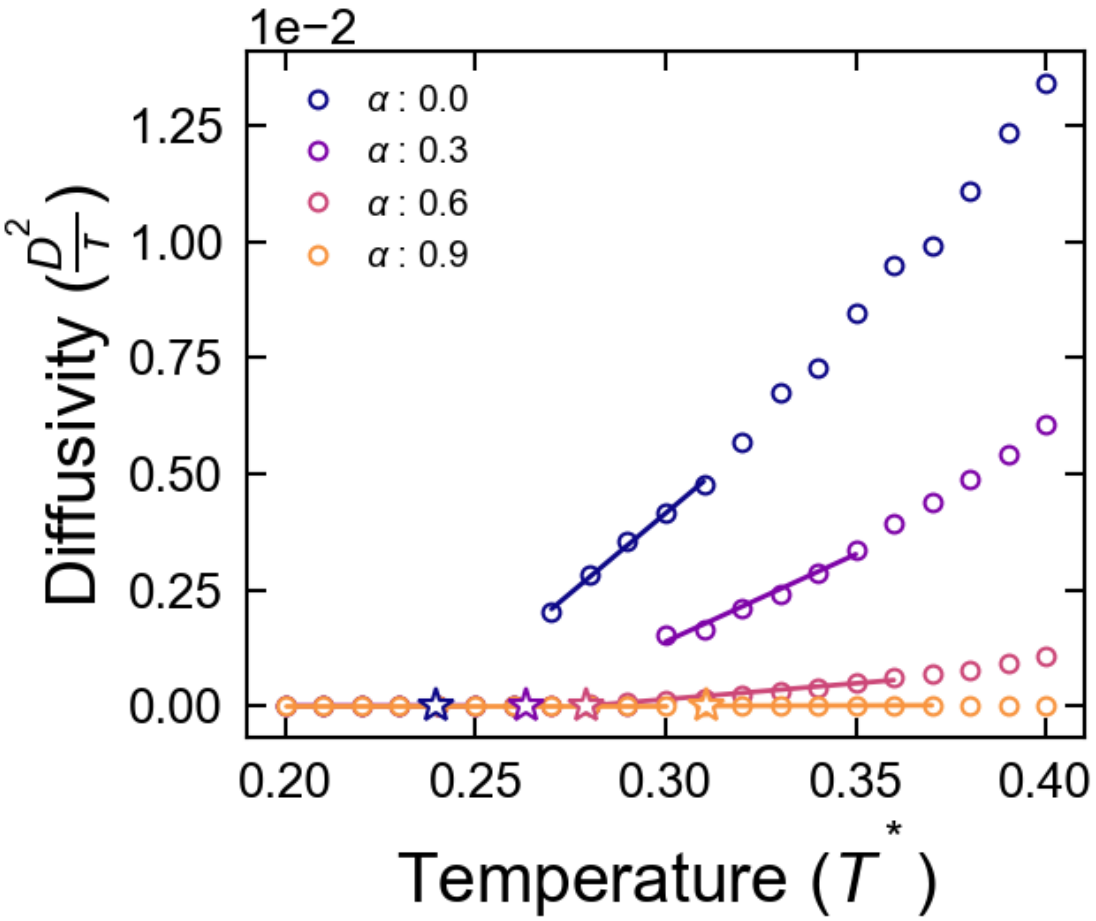


In [10]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,9
0]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='anneal'


#Tgs_dict={}
df_filtered=df[(df.quench_T<=0.5)&(df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)

        custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)

        if cooling_method == 'anneal':
            quench_time = None
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                               quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 markerfacecolor='w',
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
```

```
                    label='$\\alpha$ : {:.1f}'.format(sap/100))
            if True:
                Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        method='use_viscous_region',
                                        min_D=0,
                                        ver=4,
                                        viscous_line_index=0,
                                        l1_T_bounds=custom_ranges_l1[sap],
                                        l2_T_bounds=custom_ranges_l2[sap])
                xs = Ts#np.linspace(0.1,4)
                plt.plot(Tg,
                         Tg_prop/mul_fact,
                         marker=Tg_marker,
                         markerfacecolor='w',
                         color=colors[i],#cooling_colors[j],
                         markersize=15)#,
                l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                l_colors=['r','g']
                for li,line_val in enumerate(line_vals):
                    xs=line_val[0]
                    ys=line_val[1]/mul_fact
                    plt.plot(xs,
                             ys,
                             color=colors[i],#cooling_colors[j],
                             zorder=1,
                             linewidth=2)
                Tgs.append(Tg)
            Tgs_dict[cooling_method]=[cure_percents,Tgs]
            #break
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','anneal.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
cure_percent 1.000479937
cure_percent 31.00138092
cure_percent 61.0007782
cure_percent 91.00018311
```

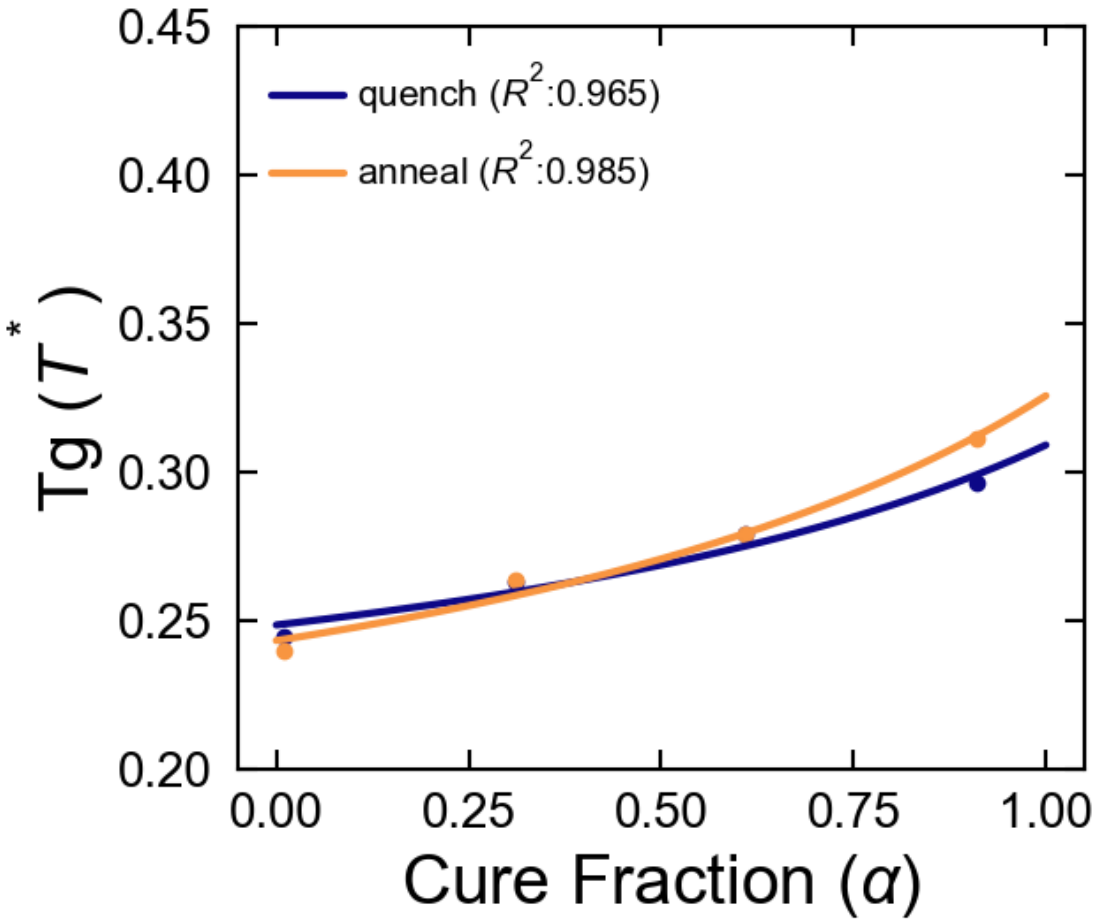
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [11]: fig, ax1 = plt.subplots()

         colors = plt.cm.plasma(np.linspace(0,0.75,len(Tgs_dict.items())))
         for i,(cooling_method,[cure_percents,Tgs]) in enumerate(Tgs_dict.items()):
             cure_percents = np.asarray(cure_percents)
             Tgs = np.asarray(Tgs)
             Tg_data = np.asarray([cure_percents/100.,Tgs])
             cure_percents_ss = cure_percents#[:-1]
             Tgs_ss = Tgs#[:-1]
             R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                                 Tgs_ss,
                                                                 T1=None,
                                                                 T0=None)
             alphas = np.linspace(0,1)
             fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
             ax1.plot(alphas,
                      fit_ydata,
                      color=colors[i],
                      label='{} ($R^2$:{})'.format(cooling_method,round(R2,3)))
             ax1.scatter(cure_percents/100.,
                         Tgs,
                         color=colors[i])


         ax1.set_xlabel('Cure Fraction ($\\alpha$)')
         ax1.set_ylabel('Tg ($T^*$)')
```

```
        ax1.set_ylim(0.2,0.45)
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range_symmetric_LJ','dibeneditto_anneal_quench.
        pdf')
        #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [12]:
```
        import matplotlib
        #from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.30, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])
```

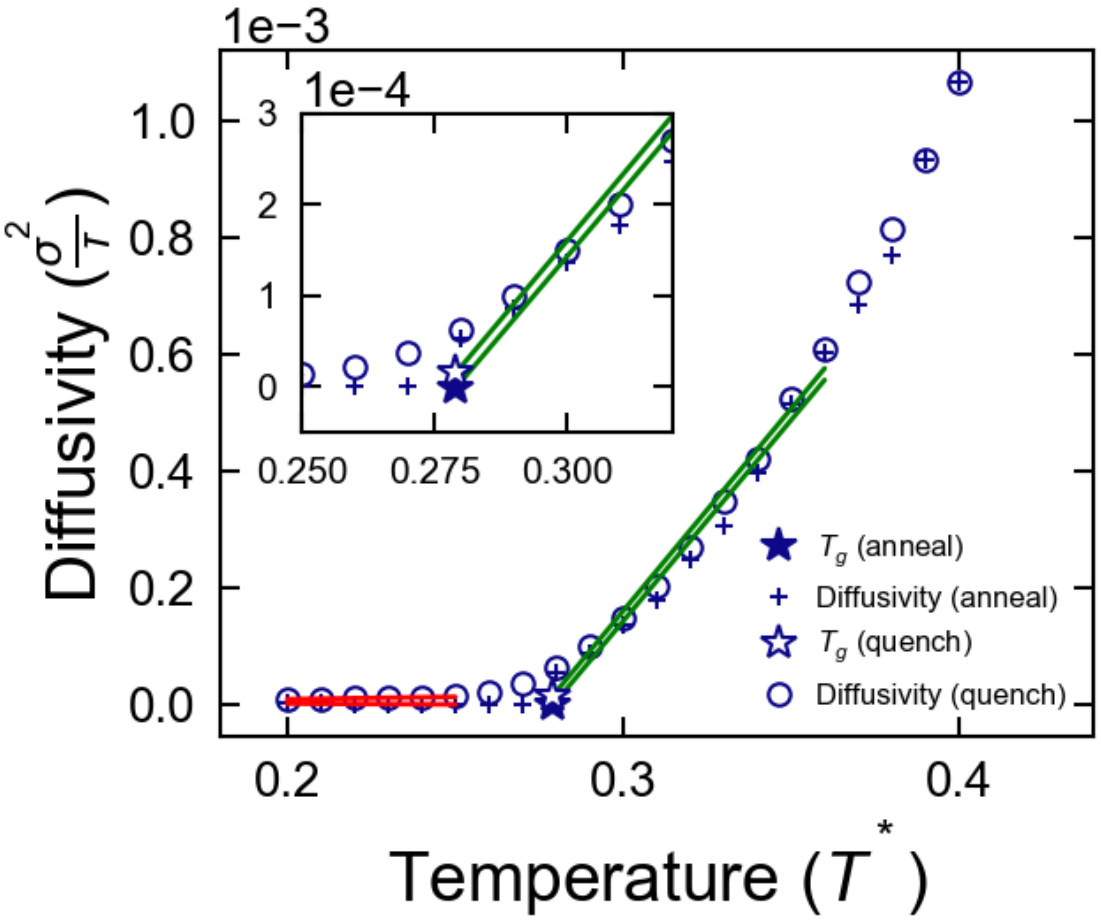
```
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[60.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]

cooling_method='quench'
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)

        custom_ranges_l1,custom_ranges_l2 = get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        if len(Ds)==0:
            raise ValueError('getDifusivities did not find Ds')
        Cure_Ts.append(Ts)
        print(Ts,cooling_method)
        avD = np.mean(Ds)
        mul_fact=1/avD
        Ds_scaled=Ds*mul_fact
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                Ds_scaled,
                                method='use_viscous_region',
                                min_D=0,
                                ver=4,
                                viscous_line_index=0,
                                l1_T_bounds=custom_ranges_l1[sap],
                                l2_T_bounds=custom_ranges_l2[sap])
        xs = Ts#np.linspace(0.1,4)
        plot_lines += ax1.plot(Tg,
                               Tg_prop/mul_fact,
                               marker='*',
                               markerfacecolor=Tg_markerfacecolor,
                               color=colors[i],
                               zorder=3,
```

```
                            markersize=15,
                            linewidth=0,
                            label='$T_g$ ({})'.format(cooling_method))#,
        ax2.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 markerfacecolor=Tg_markerfacecolor,
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,
        plot_lines += ax1.plot(Ts,
                            Ds,
                            marker=marker,
                            markerfacecolor='w',
                            markersize=markersize,
                            color=colors[i],#cooling_colors[j],
                            linewidth=0.0,
                            zorder=zorder,
                            label='Diffusivity ({})'.format(cooling_method))
        ax2.plot(Ts,
                 Ds,
                 marker=marker,
                 markerfacecolor='w',
                 markersize=markersize,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=zorder)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            ax1.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=2,
                     linewidth=2)
            ax2.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=2,
                     linewidth=2)
        Tgs.append(Tg)
    #break
labels = [l.get_label() for l in plot_lines]
ax1.legend(plot_lines,
           labels,
           fontsize=12,
           loc='lower right')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
ax2.set_xlim(0.25,0.32)
ax2.set_ylim(-5e-5,3e-4)
ax1.set_xlim(0.18,0.44)
ax1.set_xlabel('Temperature ($T^*$)')
ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
#savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
savefig(plt,
        'piecewise_regression_custom_range_symmetric_LJ',
        'quench_anneal_60_percent.pdf')
#np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
```

```
        spose(data))

        plt.show()
```

```
[ 0.4   0.39  0.38  0.37  0.36  0.35  0.34  0.33  0.32  0.31  0.3   0.29
  0.28  0.27  0.26  0.25  0.24  0.23  0.22  0.21  0.2 ] anneal
[ 0.2   0.21  0.22  0.23  0.24  0.25  0.26  0.27  0.28  0.29  0.3   0.31
  0.32  0.33  0.34  0.35  0.36  0.37  0.38  0.39  0.4 ] quench
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [37]:
```
import matplotlib
#from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
fig = plt.figure()
```

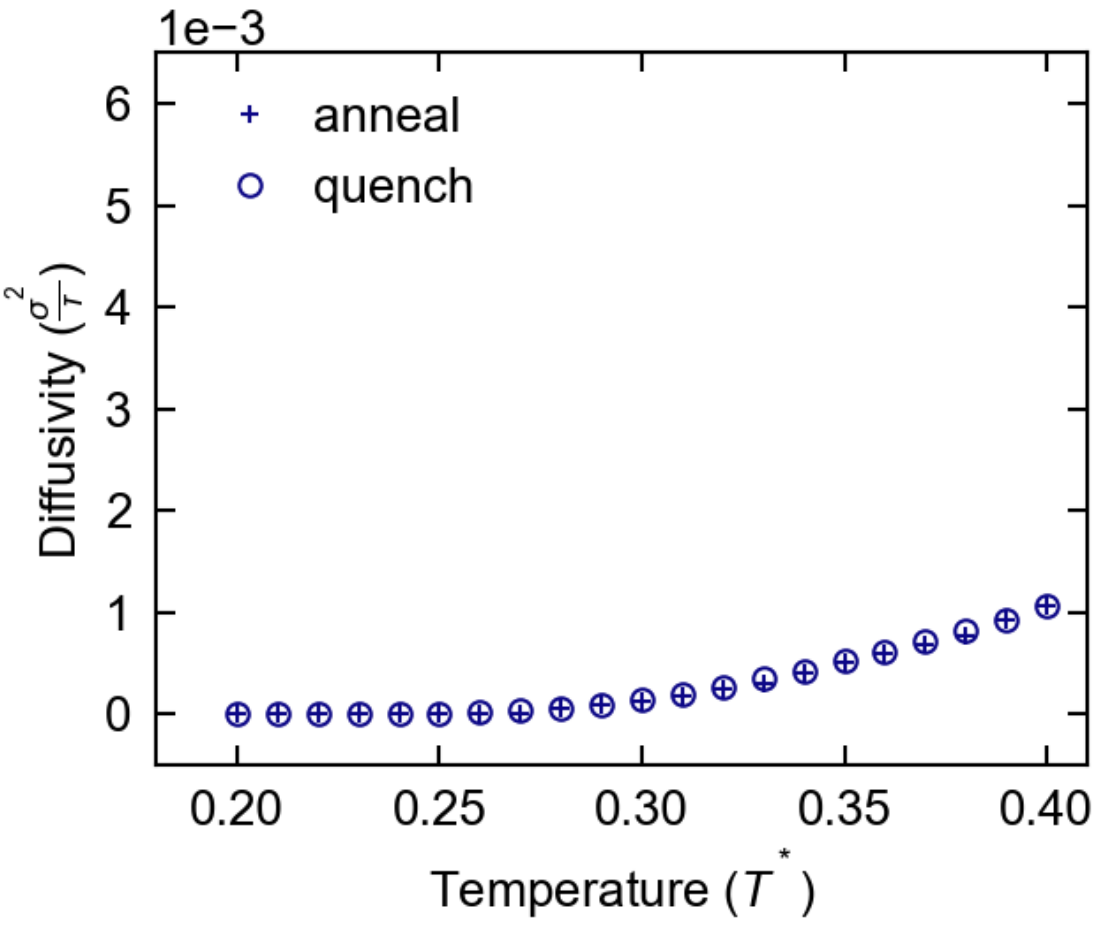
```
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.30, 0.53, 0.30, 0.30]
#ax2 = fig.add_axes([left, bottom, width, height])
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[60]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]

cooling_method='quench'
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)

        custom_ranges_l1,custom_ranges_l2 = get_custom_ranges(cooling_method)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        if len(Ds)==0:
            raise ValueError('getDifusivities did not find Ds')
        Cure_Ts.append(Ts)
        print(Ts,cooling_method)
        avD = np.mean(Ds)
        mul_fact=1/avD
        Ds_scaled=Ds*mul_fact
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                Ds_scaled,
                                method='use_viscous_region',
                                min_D=0,
                                ver=4,
                                viscous_line_index=0,
                                l1_T_bounds=custom_ranges_l1[sap],
                                l2_T_bounds=custom_ranges_l2[sap])
        xs = Ts#np.linspace(0.1,4)
        #plot_lines += ax1.plot(Tg,
        #                       Tg_prop/mul_fact,
        #                       marker='*',
```

```
        #                       markerfacecolor=Tg_markerfacecolor,
        #                       color=colors[i],
        #                       zorder=3,
        #                       markersize=15,
        #                       linewidth=0,
        #                       label='$T_g$ ({})'.format(cooling_method))#,
        #ax2.plot(Tg,
        #         Tg_prop/mul_fact,
        #         marker='*',
        #         markerfacecolor=Tg_markerfacecolor,
        #         color=colors[i],
        #         zorder=3,
        #         markersize=15)#,
        plot_lines += ax1.plot(Ts,
                               Ds,
                               marker=marker,
                               markerfacecolor='w',
                               markersize=markersize,
                               color=colors[i],#cooling_colors[j],
                               linewidth=0.0,
                               zorder=zorder,
                               label='{}'.format(cooling_method))
        #ax2.plot(Ts,
        #         Ds,
        #         marker=marker,
        #         markerfacecolor='w',
        #         markersize=markersize,
        #         color=colors[i],#cooling_colors[j],
        #         linewidth=0.0,
        #         zorder=zorder)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        #l_colors=['r','g']
        #for li,line_val in enumerate(line_vals):
        #    xs=line_val[0]
        #    ys=line_val[1]/mul_fact
        #    ax1.plot(xs,
        #             ys,
        #             color=l_colors[li],
        #             zorder=2,
        #             linewidth=2)
            #ax2.plot(xs,
            #         ys,
            #         color=l_colors[li],
            #         zorder=2,
            #         linewidth=2)
        Tgs.append(Tg)
    #break
labels = [l.get_label() for l in plot_lines]
ax1.legend(plot_lines,
           labels,
           fontsize=20,
           loc='upper left')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
#ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
#ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
#ax2.set_xlim(0.25,0.32)
#ax2.set_ylim(-5e-5,3e-4)
ax1.set_xlim(0.18,0.41)
ax1.set_ylim(-5e-4,6.5e-3)
ax1.set_xlabel('Temperature ($T^*$)',fontsize=20)
ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)',fontsize=20)
#savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
```

```
        savefig(plt,
                'piecewise_regression_custom_range_symmetric_LJ',
                'diffusivity_quench_anneal_60_percent.pdf')
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
[ 0.4   0.39  0.38  0.37  0.36  0.35  0.34  0.33  0.32  0.31  0.3   0.29
  0.28  0.27  0.26  0.25  0.24  0.23  0.22  0.21  0.2 ] anneal
[ 0.2   0.21  0.22  0.23  0.24  0.25  0.26  0.27  0.28  0.29  0.3   0.31
  0.32  0.33  0.34  0.35  0.36  0.37  0.38  0.39  0.4 ] quench
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



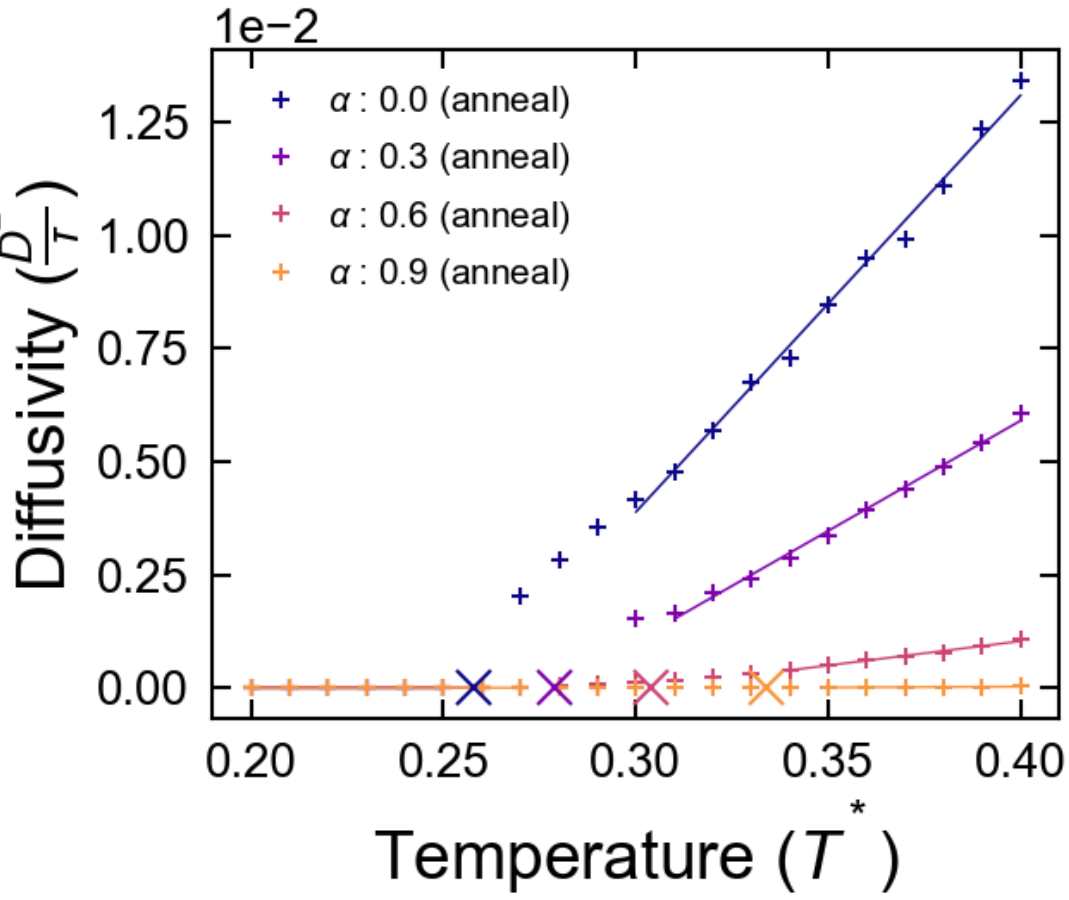

```
In [14]: import matplotlib
         from common import *
         %matplotlib inline
```

```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.,30,60,90]#[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,9
0]

Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='anneal'
custom_ranges_l1={00.0:[0.1,0.25],
                  30.0:[0.1,0.25],
                  60.0:[0.1,0.26],
                  90.0:[0.1,0.27]}
custom_ranges_l2={00.0:[0.29,0.4],
                  30.0:[0.30,0.4],
                  60.0:[0.33,0.4],
                  90.0:[0.35,0.4]}

Tgs_dict={}
df_filtered=df[(df.quench_T<=0.4)&(df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    Tgs=[]
    cure_percents = []
    for i,sap in enumerate(filter_saps):
        #print(cooling_method,sap)
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print('cure_percent',cure_percent)
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=1
            markersize=7
            Tg_marker='x'
        else:
            quench_time = 5e6
            marker='o'
            zorder=0
            markersize=7
            Tg_marker='*'
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print('Ds',Ds)
        mul_fact=1
        Ds_scaled=Ds*mul_fact
        #print('custom_ranges_l2',custom_ranges_l2[i])

        plt.plot(Ts,
                 Ds,
                 marker=marker,
                 markersize=markersize,
                 zorder=zorder,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f} ({})'.format(sap/100,cooling_method))
        if True:
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
```

```
                                        Ds_scaled,
                                        method='use_viscous_region',
                                        min_D=0,
                                        ver=4,
                                        viscous_line_index=0,
                                        l1_T_bounds=custom_ranges_l1[sap],
                                        l2_T_bounds=custom_ranges_l2[sap])
                    xs = Ts#np.linspace(0.1,4)
                    plt.plot(Tg,
                             Tg_prop/mul_fact,
                             marker=Tg_marker,
                             color=colors[i],#cooling_colors[j],
                             markersize=15)#,
                    l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
                    l_colors=['r','g']
                    for li,line_val in enumerate(line_vals):
                        xs=line_val[0]
                        ys=line_val[1]/mul_fact
                        plt.plot(xs,
                                 ys,
                                 color=colors[i],#cooling_colors[j],
                                 zorder=0,
                                 linewidth=1)
                    Tgs.append(Tg)
                Tgs_dict[cooling_method]=[cure_percents,Tgs]
                #break
            plt.legend(fontsize=15)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            #plt.xlim(0.5,1.5)
            #plt.ylim(-1e-4,5e-4)
            plt.xlabel('Temperature ($T^*$)')
            plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
            #savefig(plt,'piecewise_regression_custom_range','all_alphas.pdf')
            #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
cure_percent 1.000479937
cure_percent 31.00138092
cure_percent 61.0007782
cure_percent 91.00018311
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−2
1.25
1.00
0.75
0.50
0.25
0.00
Diffusivity ($\frac{D^2}{T}$)
α : 0.0 (anneal)
α : 0.3 (anneal)
α : 0.6 (anneal)
α : 0.9 (anneal)
0.20
0.25
0.30
0.35
0.40
Temperature (T^*)

In [2]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [3]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *

def H0(x,x0,c):
        return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

def hyper(x,x0,y0,a,b,c):
    #a=0.0
    return y0+a*(x-x0)+b*H0(x,x0,c)

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,x0,a,b,c):
    #a=0.0
    return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

def get_tangent(x,x0,y0,a,b,c):
    m=slope_of_tangent(x,x0,a,b,c)
```

```
            y=hyper(x,x0,y0,a,b,c)
            b=y-(m*x)
            return m,b
```

In [4]:
```
df_filtered=df[df.cooling_method=='anneal']
df_filtered.D_A
```

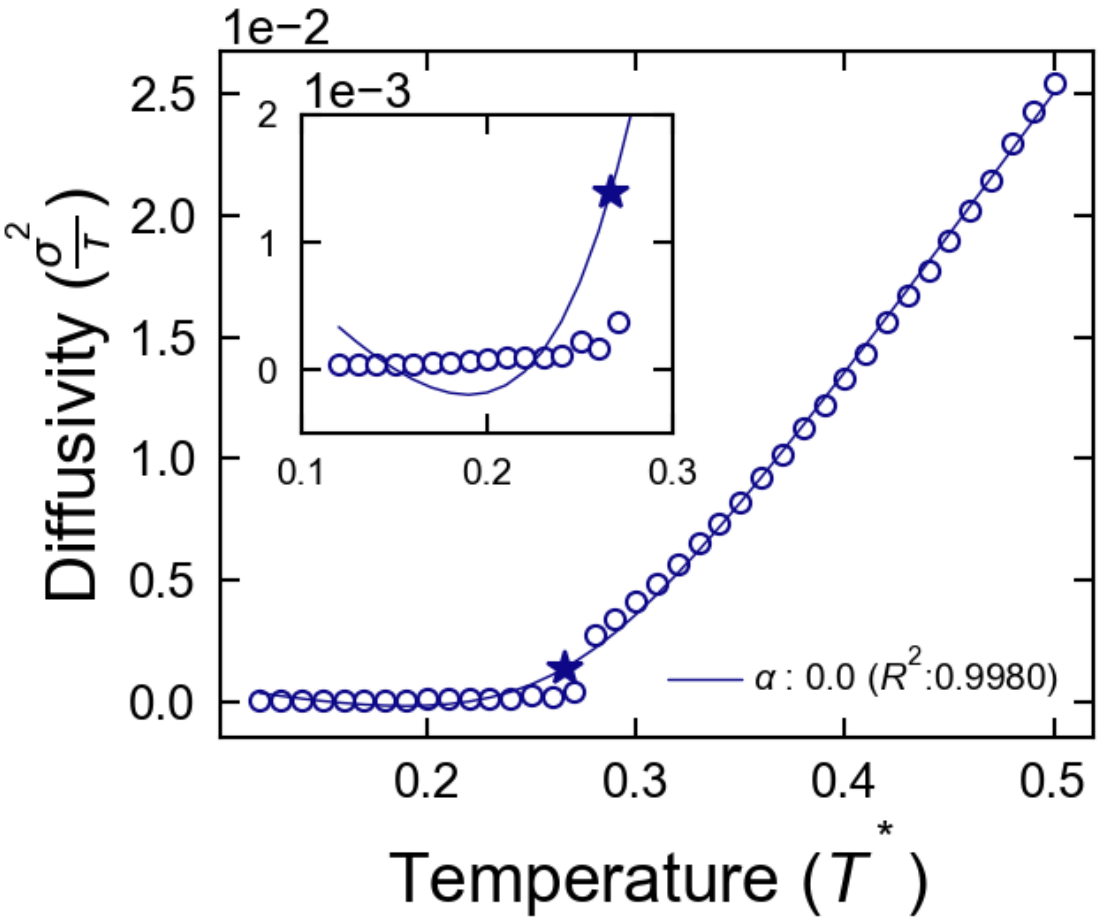
Out[4]:
```
90b99ca468d650a13ade4d55a27ce7e1    3.617847e-04
ffd54826a80437a447973bbf73be3f68    9.086983e-07
644d27b8f740cc6df4a628957245df0c    3.637193e-05
646df3239f44bb1705e0ff4c9a58106e    2.057987e-03
Name: D_A, dtype: float64
```

In [5]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

fig = plt.figure()
ax1 = fig.add_subplot(111)

left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
ax2 = fig.add_axes([left, bottom, width, height])

#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        #popt, pcov = curve_fit(hyper,
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
```

```
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
ax1.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

ax1.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

ax2.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

ax2.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
ax1.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
        color=colors[i],#cooling_colors[j],
         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
ax2.plot(xs,
         yHYP/mul_fact,
```

```
                        linewidth=1.0,
                        zorder=0,
                       color=colors[i],#cooling_colors[j],
                        label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
        ax1.legend(fontsize=15,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'hyp_0_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.2663917452
a -0.0205159027228 b 0.14115866065
Ps 0.0561794536329 0.975499130072


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

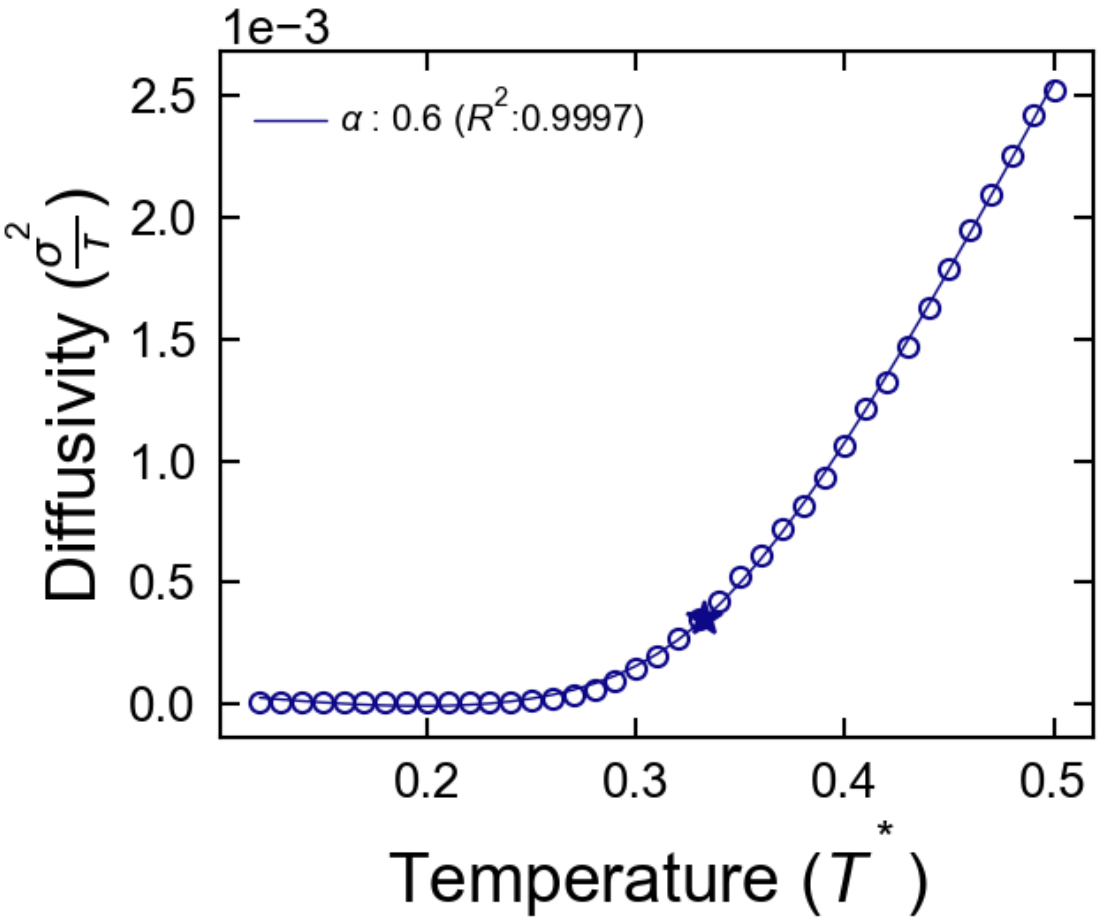



```
In [6]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.5)&
                       (df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                df_curing = df_grp[(df_grp.bond==False)&
```

```
                (df_grp.cooling_method==cooling_method)&
                (df_grp.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.mean()
cure_percents.append(cure_percent)
Ts,Ds=getDiffusivities(project,
                       df_curing,
                       name=PROP_NAME,
                      quench_time=5e6)
Cure_Ts.append(Ts)
#print(Ds)
mul_fact=10000
Ds_scaled=Ds*mul_fact
popt, pcov = curve_fit(hyper,
                       Ts,
                       Ds_scaled,
                      maxfev=200000)

T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
plt.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

plt.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
plt.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
```

```
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'hyp_60_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.332819321193
a -0.00139479953875 b 0.0180679951011
Ps 0.0359625206175 0.94527494216


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

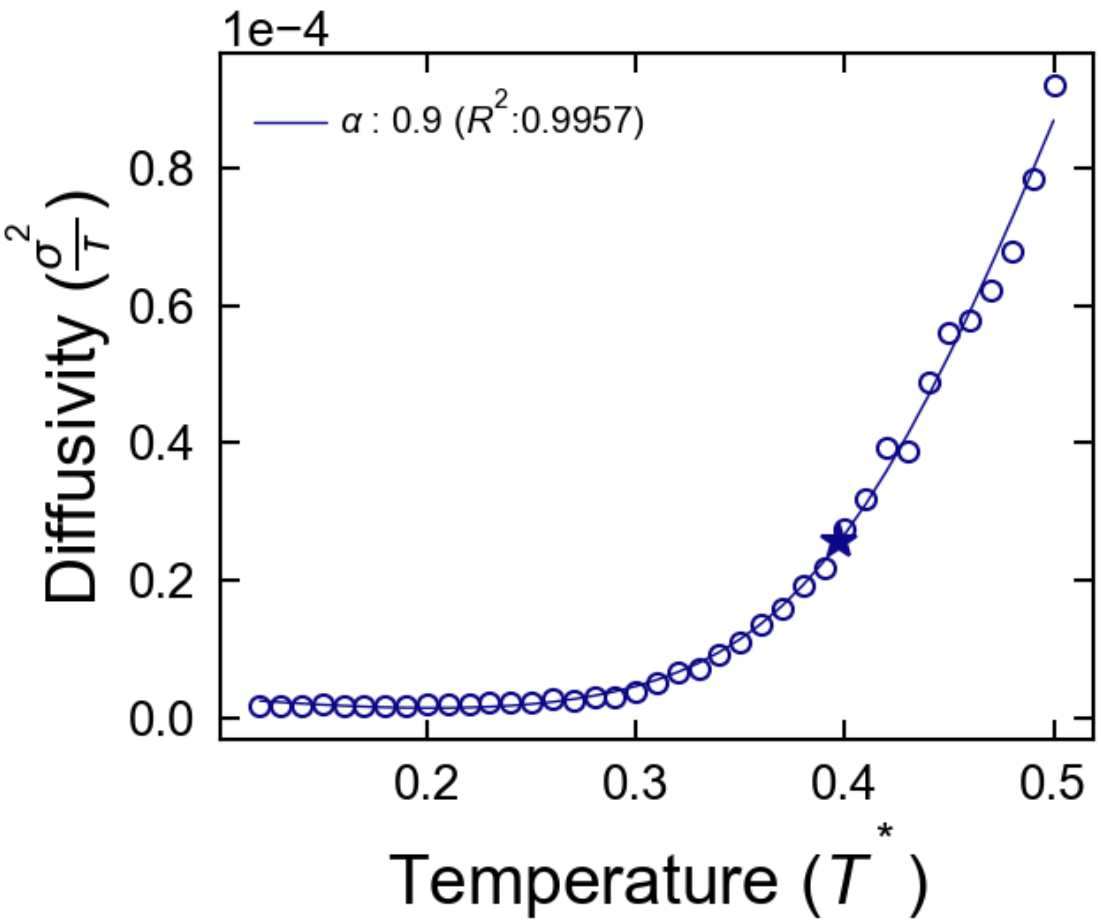



```
In [7]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.5)&
                       (df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                df_curing = df_grp[(df_grp.bond==False)&
```

```
                (df_grp.cooling_method==cooling_method)&
                (df_grp.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.mean()
cure_percents.append(cure_percent)
Ts,Ds=getDiffusivities(project,
                       df_curing,
                       name=PROP_NAME,
                      quench_time=5e6)
Cure_Ts.append(Ts)
#print(Ds)
mul_fact=10000
Ds_scaled=Ds*mul_fact
popt, pcov = curve_fit(hyper,
                       Ts,
                       Ds_scaled,
                      maxfev=200000)

T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
plt.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

plt.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
plt.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
```

```
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'hyp_90_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.397521051971
a -4.72095009638e-05 b 0.000901200140543
Ps 0.0279802426992 0.863195474072


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



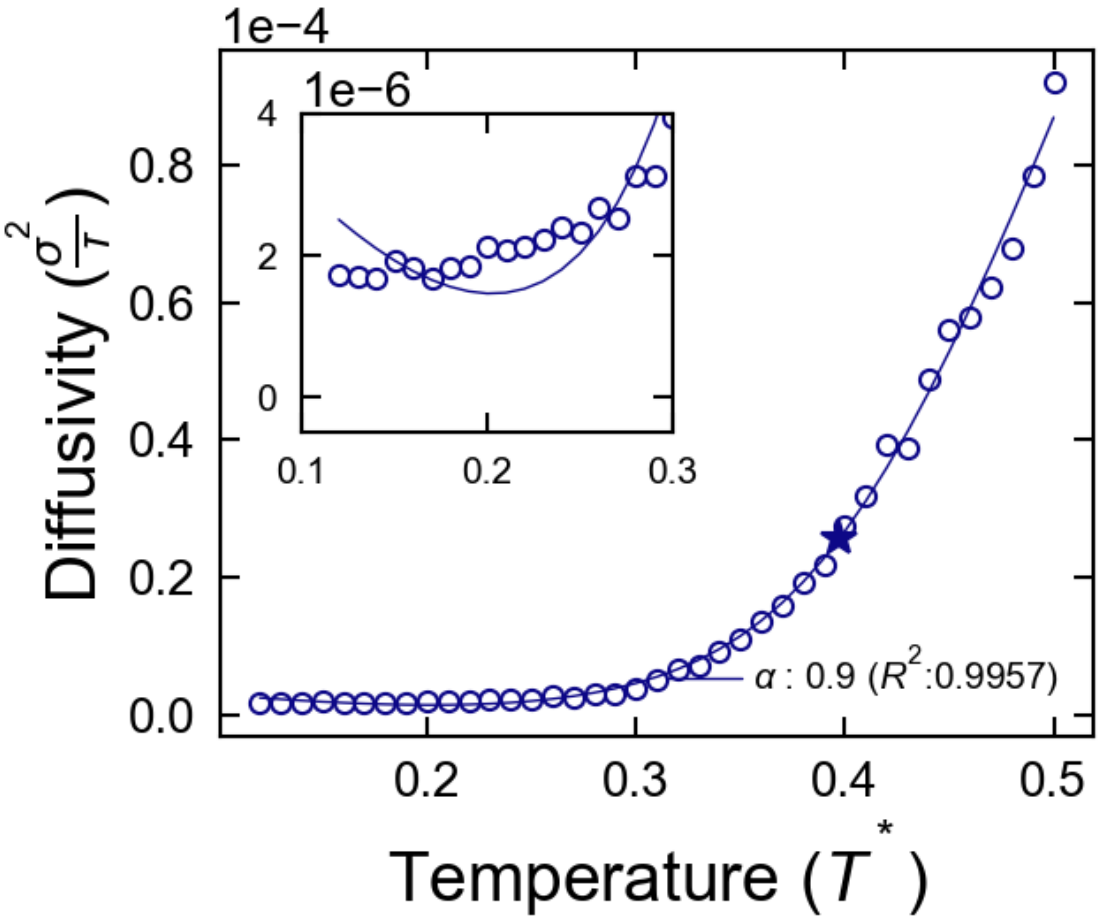

```
In [28]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mean_D = np.mean(Ds)
        mul_fact=1/mean_D
        Ds_scaled=Ds*mul_fact
        #popt, pcov = curve_fit(hyper,
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        ax1.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        ax2.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax2.plot(Ts,
                 Ds,
```

```
                    marker='o',
                   markerfacecolor='w',
                    markersize=8,zorder=0,
                    color=colors[i],#cooling_colors[j],
                    linewidth=0.0)

            m1,b1=get_tangent(Ts[0],*popt)
            m2,b2=get_tangent(Ts[-1],*popt)

            tgx,tgy=line_intersect(m1,b1,m2,b2)
            l1x=np.linspace(Ts[0],tgx+0.1)
            l1y=(m1*l1x)+b1
            #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            ax1.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            ax2.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        ax1.legend(fontsize=15,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-6,4e-6)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'hyp_90_percent.pdf')
        #np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tra
        nspose(data))

        plt.show()
```

```
To 0.39752090821
a -4.72093488062e-05 b 0.00090119884175
Ps 0.0279801838337 0.863196000272


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

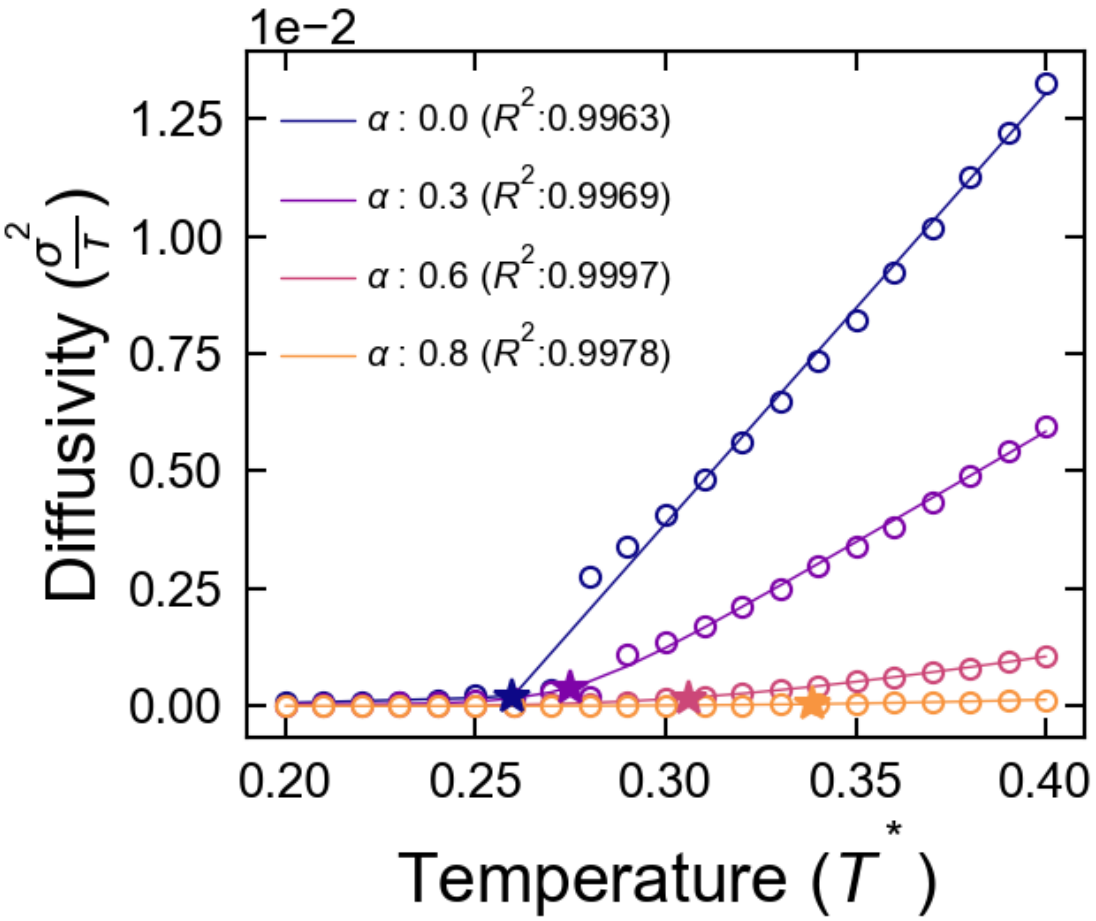



```
In [9]: import matplotlib
        from common import *
        import collections
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.,60.,80.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.4)&
```

```
            (df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
Tgs_dict = collections.OrderedDict()
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        avD = np.mean(Ds)

        #print('avD',avD,1/avD)
        mul_fact=1/avD#1000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plt.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
```

```
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        print(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'all_alphas_quench.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

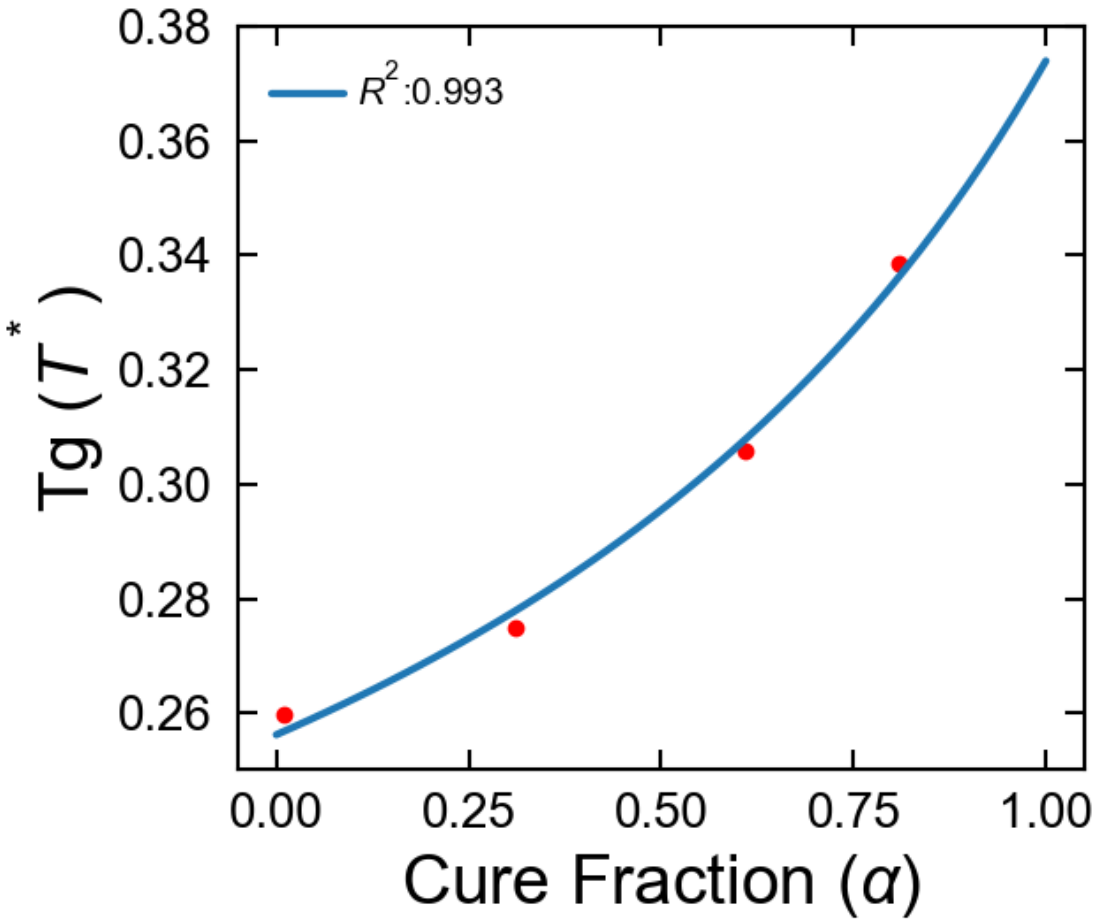
```
To 0.259676083556
a 0.00212052607476 b 0.0894124108243
Ps 1.66533453694e-16 1.0
To 0.275017814374
a -0.00194270966042 b 0.0497229642643
Ps 0.0226185833286 0.991485367288
To 0.305841942978
a -0.00205968079111 b 0.0152242043048
Ps 0.0863554317586 0.897444610783
To 0.338467649888
a -0.000261782953119 b 0.00262769623614
Ps 0.0609532164482 0.815996049877
[ 0.25967608  0.27501781  0.30584194  0.33846765]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [10]: cure_percents = np.asarray(cure_percents)
         #Tgs=[0.65,0.75,0.9,2.1]
         fig, ax1 = plt.subplots()
         Tgs = np.asarray(Tgs)
         Tgs_tangent = np.asarray(Tgs_tangent)
         print(Tgs)
         Tg_data = np.asarray([cure_percents/100.,Tgs])
         #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
         cure_percents_ss = cure_percents#[:-1]
         Tgs_ss = Tgs#[:-1]
         print(cure_percents_ss)
         print(Tgs_ss)
         R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                             Tgs_ss,
                                                             T1=None,
                                                             T0=None)
         print('T1',T1,'lambda',inter_parm)
         #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
         #print(cure_percents_ss)
         alphas = np.linspace(0,1)
         fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
         ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
         ax1.scatter(cure_percents/100.,
                     Tgs,
                     #label='$E_a$:{}'.format(activation_energy),
                     color='r')#colors[i])
```

```
        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)

        sim_low_lim = 0.5
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.5
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym','dibeneditto.pdf')
```

```
[ 0.25967608  0.27501781  0.30584194  0.33846765]
[  1.00047994  31.00138092  61.0007782   81.00138092]
[ 0.25967608  0.27501781  0.30584194  0.33846765]
T1 0.373975194402 lambda 0.5
300 K in T*: 0.233734496501


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

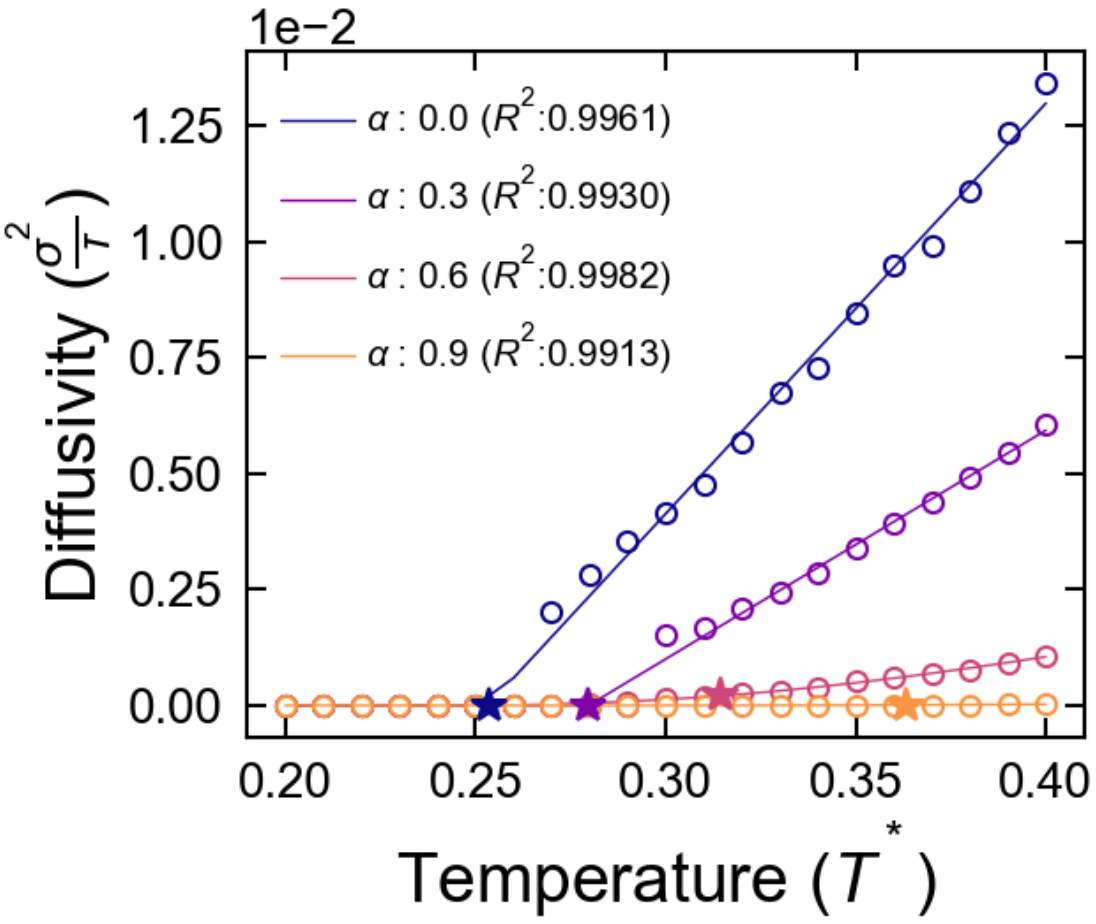



```
In [11]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='anneal'
         df_filtered=df[(df.quench_T<=0.4)&
                        (df.quench_T>=0.2)&
         (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
         for i,sap in enumerate(filter_saps):
             cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
             for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                 df_curing = df_grp[(df_grp.bond==False)&
```

```
                (df_grp.cooling_method==cooling_method)&
                (df_grp.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.mean()
cure_percents.append(cure_percent)
Ts,Ds=getDiffusivities(project,
                       df_curing,
                       name=PROP_NAME,
                      quench_time=6e6)
Cure_Ts.append(Ts)
#print(Ds)
avD = np.mean(Ds)

#print('avD',avD,1/avD)
mul_fact=1/avD#1000
Ds_scaled=Ds*mul_fact
popt, pcov = curve_fit(hyper,
                       Ts,
                       Ds_scaled,
                      maxfev=2000000)

T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
plt.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

plt.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
plt.plot(xs,
```

```
                        yHYP/mul_fact,
                        linewidth=1.0,
                        zorder=0,
                       color=colors[i],#cooling_colors[j],
                        label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                'all_alphas_anneal.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/scipy/optimize/minpack.py:785: OptimizeWarning: Covariance of the parameters
could not be estimated
  category=OptimizeWarning)


To 0.253321092758
a 2.91660884851e-06 b 0.0885605018245
Ps 1.0 0.0
To 0.279712831835
a -2.81780997253e-05 b 0.0493954222181
Ps 1.0 1.11022302463e-16
To 0.31462131525
a -0.00369816282275 b 0.0198230737251
Ps 0.824306491868 0.123539863683
To 0.363253104845
a -4.07876228168e-05 b 0.000714864616644
Ps 0.772906863927 0.0274187548925


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

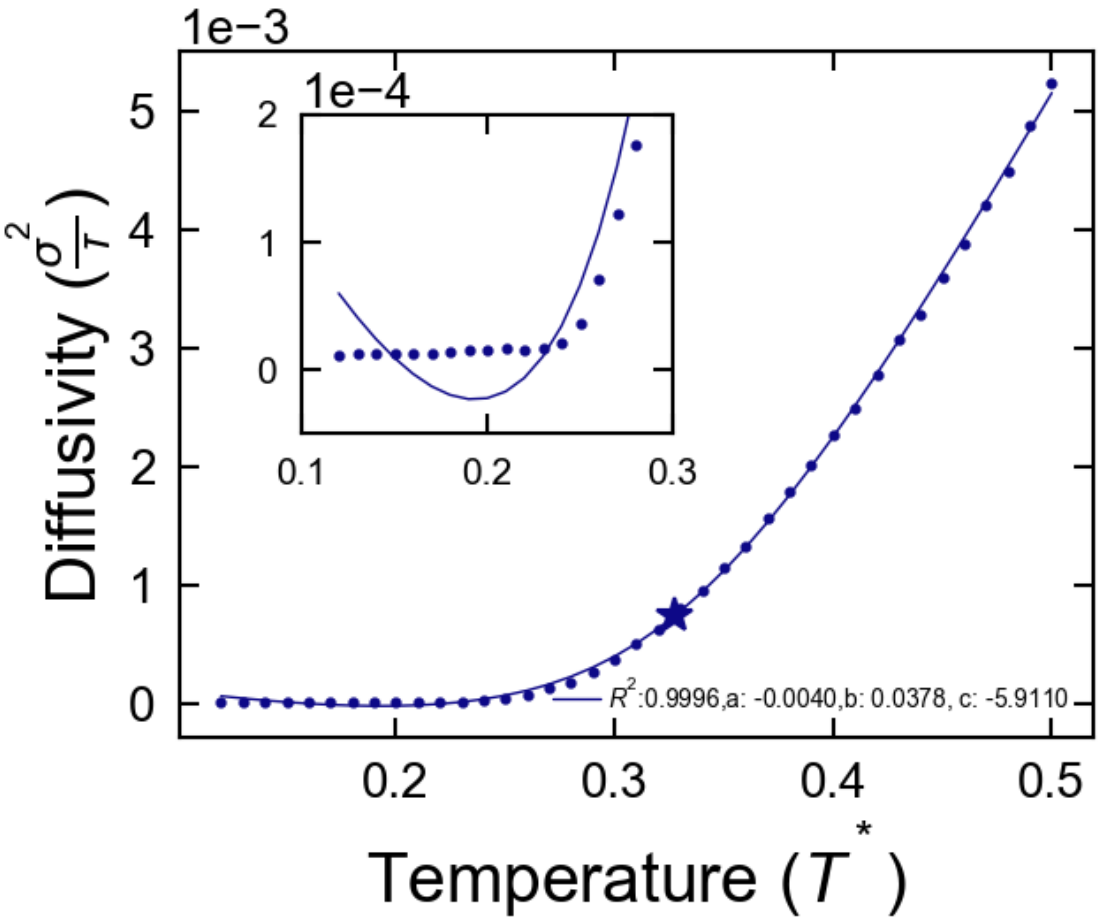



```
In [12]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=5.0)&
```

```
            (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               quench_time=5e6,
                               name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        ax1.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
        ax2.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
```

```
                tgx,tgy=line_intersect(m1,b1,m2,b2)
                l1x=np.linspace(Ts[0],tgx+0.1)
                l1y=(m1*l1x)+b1
                #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
                l2x=np.linspace(tgx-0.1,Ts[-1])
                l2y=(m2*l2x)+b2
                #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

                Tgs_tangent.append(tgx)
                #print(yHYP/mul_fact)
                ax1.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                        color=colors[i],#cooling_colors[j],
                         label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                                   a,
                                                                                   b,
                                                                                   c))
                ax2.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         color=colors[i],#cooling_colors[j],
                         label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                                   a,
                                                                                   b,
                                                                                   c))

                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
        ax1.legend(fontsize=10,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-4,2e-4)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                '50percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.327483697655
a -0.00397604311073 b 0.0378448477566
Ps 0.0530986785922 0.928094786613


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



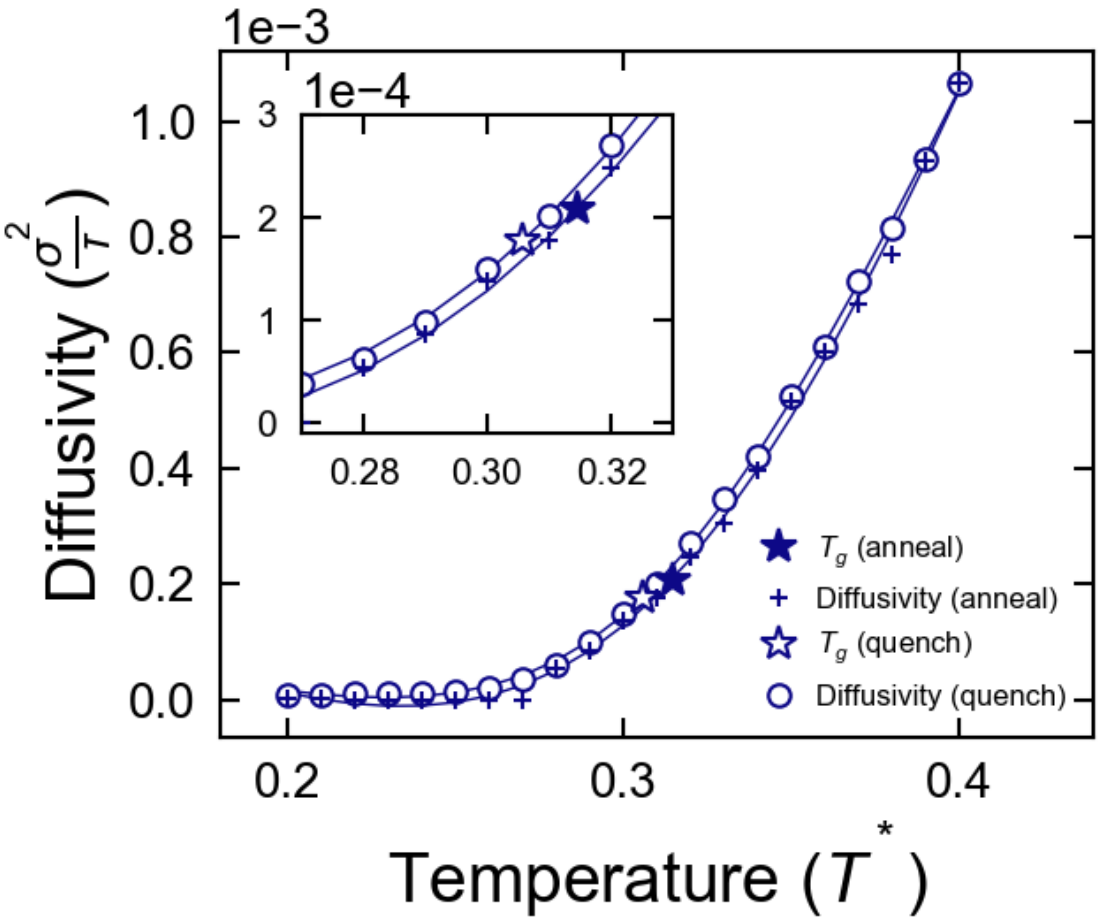

```
In [13]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0

        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=quench_time)
        Cure_Ts.append(Ts)
        avD = np.mean(Ds)

        print('avD',avD,1/avD)
        mul_fact=1/avD#1000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plot_lines += ax1.plot(T0,
                               d/mul_fact,
                               marker='*',
```

```
                        markerfacecolor=Tg_markerfacecolor,
                        color=colors[i],
                        zorder=3,
                        markersize=15,
                        linewidth=0.0,
                       label='$T_g$ ({})'.format(cooling_method))#,

plot_lines += ax1.plot(Ts,
                        Ds,
                        marker=marker,
                        markerfacecolor='w',
                        markersize=markersize,
                        color=colors[i],#cooling_colors[j],
                        linewidth=0.0,
                        zorder=zorder,
                        label='Diffusivity ({})'.format(cooling_method))

ax2.plot(T0,
           d/mul_fact,
           marker='*',
           markerfacecolor=Tg_markerfacecolor,
           color=colors[i],
           zorder=3,
         linewidth=0.0,
           markersize=15,
          label='$T_g$ ({})'.format(cooling_method))#,

ax2.plot(Ts,
           Ds,
           marker=marker,
           markerfacecolor='w',
           markersize=markersize,
           color=colors[i],#cooling_colors[j],
           linewidth=0.0,
           zorder=zorder,
           label='Diffusivity ({})'.format(cooling_method))

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
ax1.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
        color=colors[i],#cooling_colors[j],
         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
ax2.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
        color=colors[i],#cooling_colors[j],
         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

xvals = np.linspace(-3,4)
yfits = hyper(xvals, *popt)
if True:
    Tg=popt[0]
    Tgs.append(Tg)
```

```
                else:
                    Tgs.append(tgx)
            labels = [l.get_label() for l in plot_lines]
            ax1.legend(plot_lines,
                       labels,
                       fontsize=12,
                       loc='lower right')
            ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
            ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
            ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            ax2.set_xlim(0.27,0.33)
            ax2.set_ylim(-1e-5,3e-4)
            ax1.set_xlim(0.18,0.44)
            ax1.set_xlabel('Temperature ($T^*$)')
            ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
            #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
            #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
            savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                    'hyp_60_percent_quench_anneal.pdf')
            np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

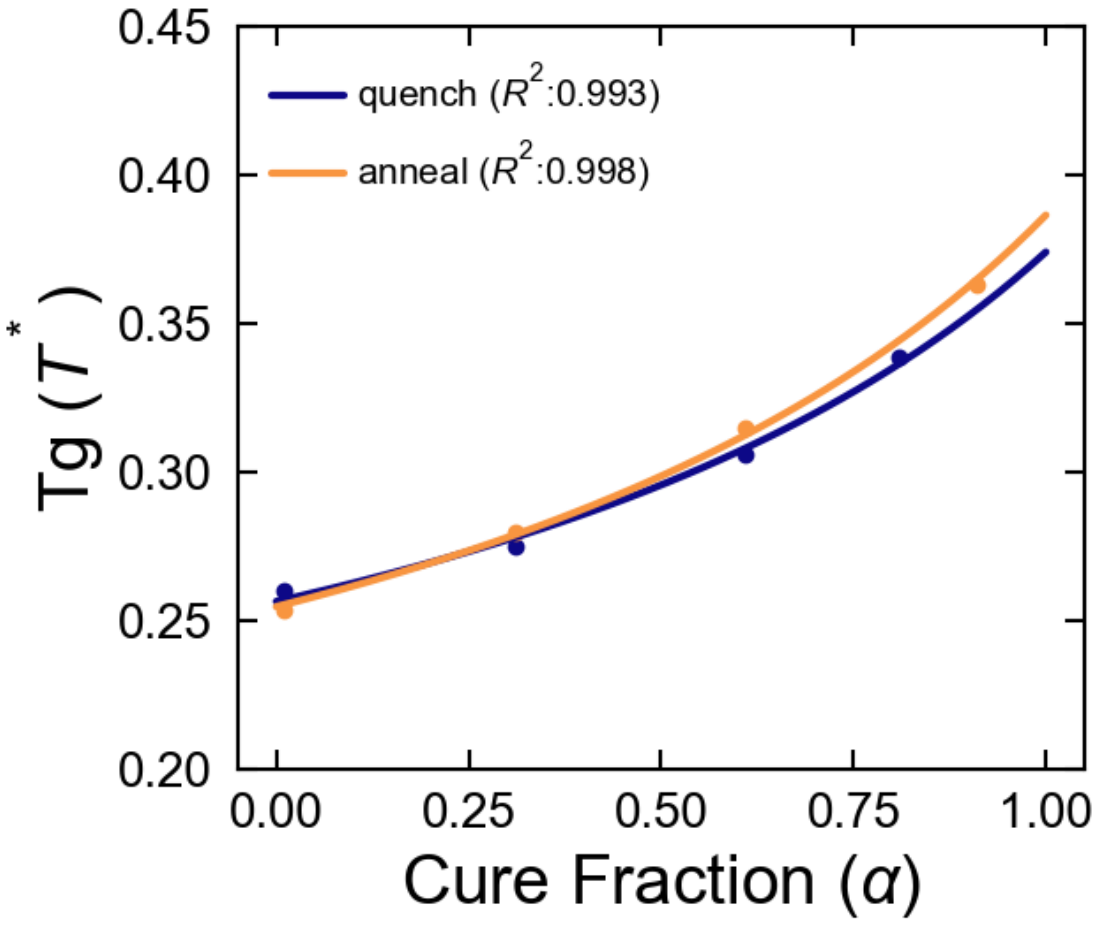
```
avD 0.000284628737507 3513.34868278
To 0.31462131525
a -0.00369816282275 b 0.0198230737251
Ps 0.824306491868 0.123539863683
avD 0.000302388392061 3307.00524972
To 0.305841942978
a -0.00205968079111 b 0.0152242043048
Ps 0.0863554317586 0.897444610783


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [14]: fig, ax1 = plt.subplots()

         colors = plt.cm.plasma(np.linspace(0,0.75,len(Tgs_dict.items())))
         for i,(cooling_method,[cure_percents,Tgs]) in enumerate(Tgs_dict.items()):
             cure_percents = np.asarray(cure_percents)
             Tgs = np.asarray(Tgs)
             Tg_data = np.asarray([cure_percents/100.,Tgs])
             cure_percents_ss = cure_percents#[:-1]
             Tgs_ss = Tgs#[:-1]
             R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                                 Tgs_ss,
                                                                 T1=None,
                                                                 T0=None)
             alphas = np.linspace(0,1)
             fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
             ax1.plot(alphas,
                      fit_ydata,
                      color=colors[i],
                      label='{} ($R^2$:{})'.format(cooling_method,round(R2,3)))
             ax1.scatter(cure_percents/100.,
                         Tgs,
                         color=colors[i])


         ax1.set_xlabel('Cure Fraction ($\\alpha$)')
         ax1.set_ylabel('Tg ($T^*$)')
```

```
        ax1.set_ylim(0.2,0.45)
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'hyperbola_fitting_unbounded_lj_sym','dibeneditto_anneal_quench.pdf')
        #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [15]: `df[df.cooling_method=='anneal'].stop_after_percent`

Out[15]:
```
90b99ca468d650a13ade4d55a27ce7e1    30
ffd54826a80437a447973bbf73be3f68    90
644d27b8f740cc6df4a628957245df0c    60
646df3239f44bb1705e0ff4c9a58106e     0
Name: stop_after_percent, dtype: object
```

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *

def H0(x,x0,c):
        return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

def hyper(x,x0,y0,a,b,c):
    #a=0.0
    return y0+a*(x-x0)+b*H0(x,x0,c)

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,x0,a,b,c):
    #a=0.0
    return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

def get_tangent(x,x0,y0,a,b,c):
    m=slope_of_tangent(x,x0,a,b,c)
```

```
            y=hyper(x,x0,y0,a,b,c)
            b=y-(m*x)
            return m,b
```

In [3]:
```
df_filtered=df[df.cooling_method=='anneal']
df_filtered.D_A
```

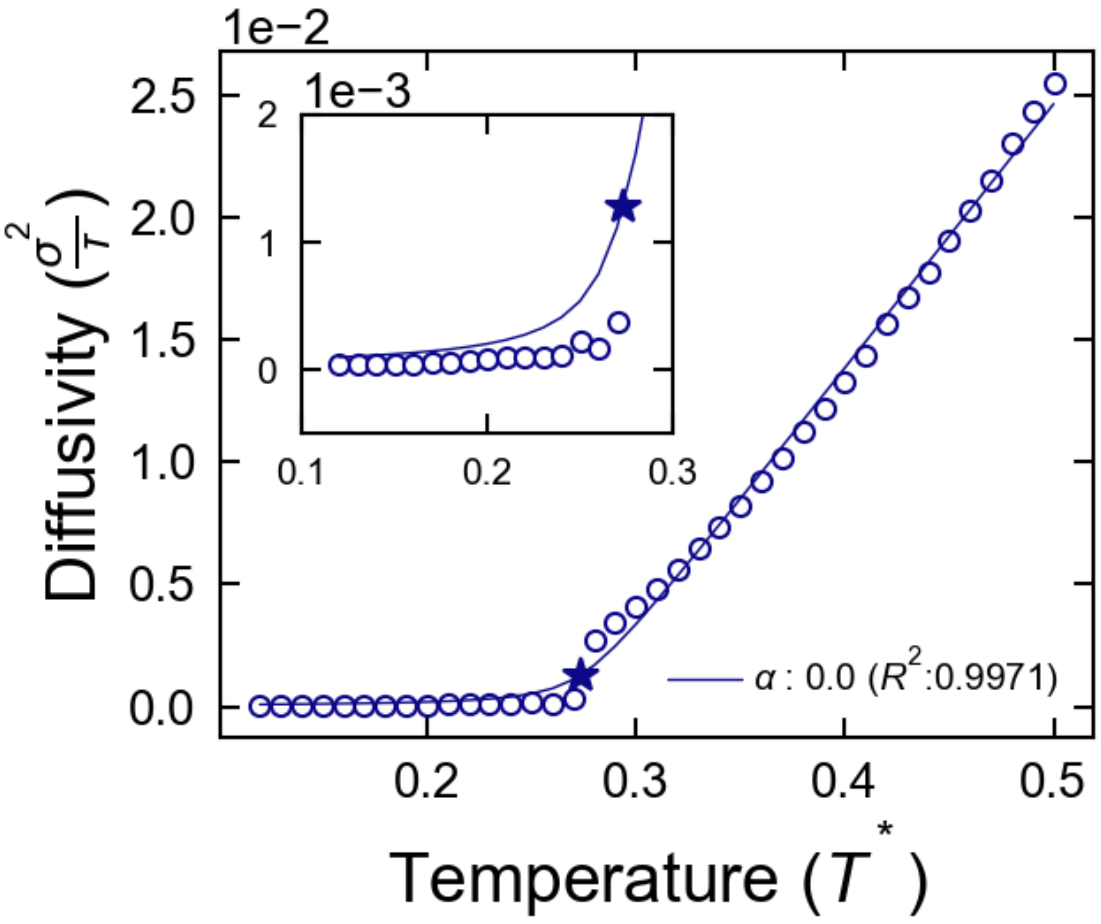
Out[3]:
```
90b99ca468d650a13ade4d55a27ce7e1    3.617847e-04
ffd54826a80437a447973bbf73be3f68    9.086983e-07
644d27b8f740cc6df4a628957245df0c    3.637193e-05
646df3239f44bb1705e0ff4c9a58106e    2.057987e-03
Name: D_A, dtype: float64
```

In [4]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

fig = plt.figure()
ax1 = fig.add_subplot(111)

left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
ax2 = fig.add_axes([left, bottom, width, height])

#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                               #    x0,y0,  a,  b,  c
                               p0=[1.0,1.0,1e-2,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0,0,0,0,-np.infty],
```

```
                                [np.infty,np.infty,1e-1,np.infty,np.infty]))

T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
ax1.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

ax1.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

ax2.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

ax2.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
#print(yHYP/mul_fact)
ax1.plot(xs,
         yHYP/mul_fact,
         linewidth=1.0,
         zorder=0,
        color=colors[i],#cooling_colors[j],
```

```
                    label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            ax2.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        ax1.legend(fontsize=15,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'hyp_0_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.273366792545
a 1e-05 b 0.108306301916
Ps 0.00586349248865 0.997289054166


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



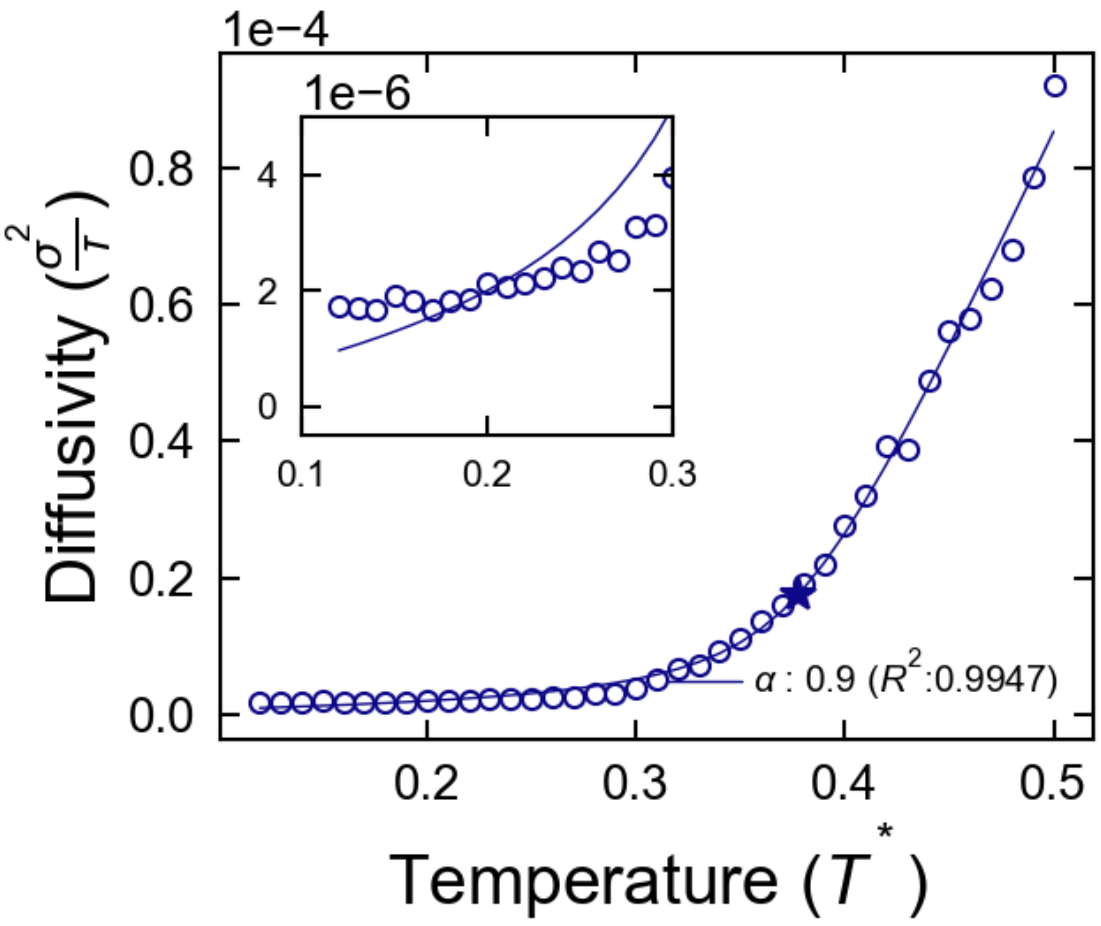

```
In [17]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                               #    x0,y0,  a,  b,  c
                               p0=[1.0,1.0,1e-2,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0,0,0,0,-np.infty],
                               [np.infty,np.infty,1e-1,np.infty,np.infty]))

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        ax1.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        ax2.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
```

```
                ax2.plot(Ts,
                         Ds,
                         marker='o',
                        markerfacecolor='w',
                         markersize=8,zorder=0,
                         color=colors[i],#cooling_colors[j],
                         linewidth=0.0)

                m1,b1=get_tangent(Ts[0],*popt)
                m2,b2=get_tangent(Ts[-1],*popt)

                tgx,tgy=line_intersect(m1,b1,m2,b2)
                l1x=np.linspace(Ts[0],tgx+0.1)
                l1y=(m1*l1x)+b1
                #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
                l2x=np.linspace(tgx-0.1,Ts[-1])
                l2y=(m2*l2x)+b2
                #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

                Tgs_tangent.append(tgx)
                #print(yHYP/mul_fact)
                ax1.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         zorder=0,
                        color=colors[i],#cooling_colors[j],
                         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
                ax2.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         zorder=0,
                        color=colors[i],#cooling_colors[j],
                         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
        ax1.legend(fontsize=15,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-6,5e-6)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'hyp_90_percent.pdf')
        #np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tra
        nspose(data))

        plt.show()
```

```
To 0.377974933611
a 3.21161867782e-06 b 0.000665415896651
Ps 0.0102204148902 0.958642482559


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

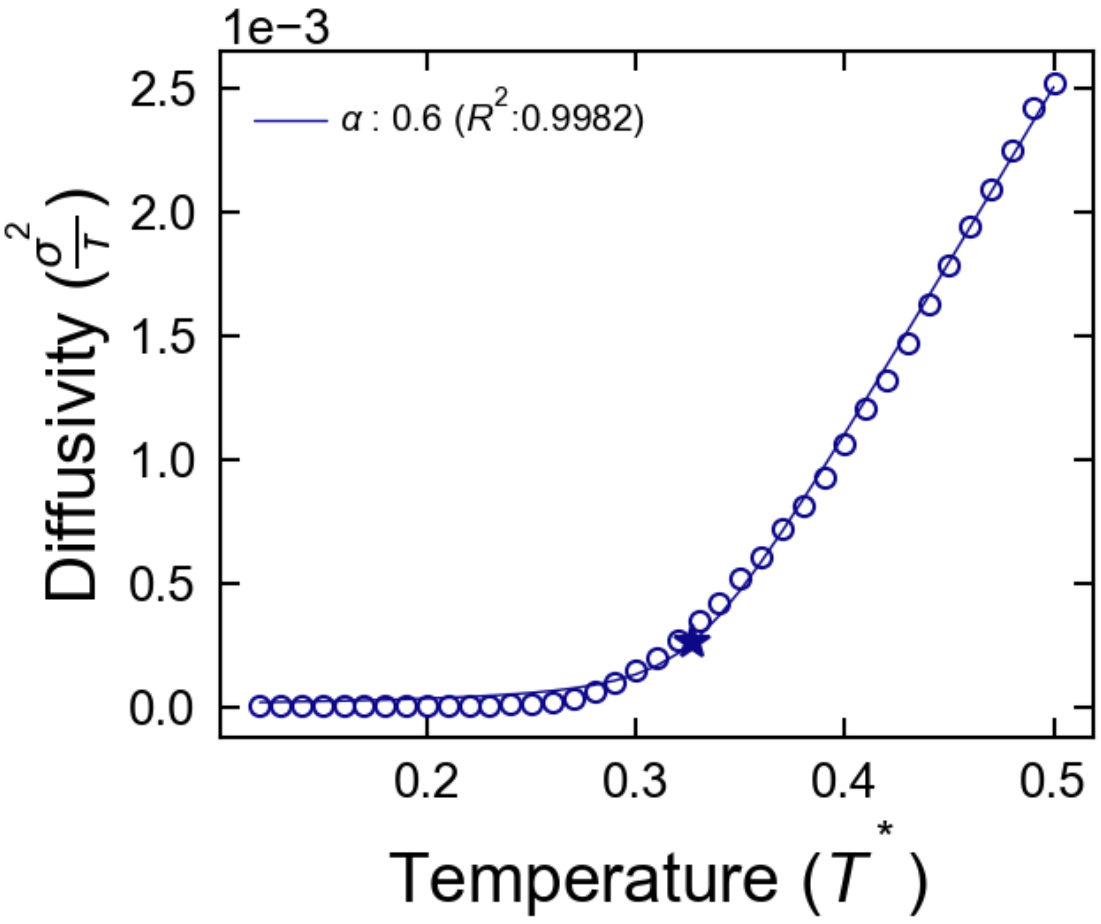



```
In [5]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.5)&
                       (df.quench_T>=0.1)&
```

```
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                               #    x0,y0,  a,  b,  c
                               p0=[1.0,1.0,1e-2,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0,0,0,0,-np.infty],
                               [np.infty,np.infty,1e-1,np.infty,np.infty]))

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plt.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
```

```
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
        {:.2f})'.format(sap/100,
            #
        r_squared,
            #
        a,
            #
        b,
            #
        c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'hyp_60_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.327371286807
a 1e-05 b 0.0143568830439
Ps 0.00781342383223 0.988840257628


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

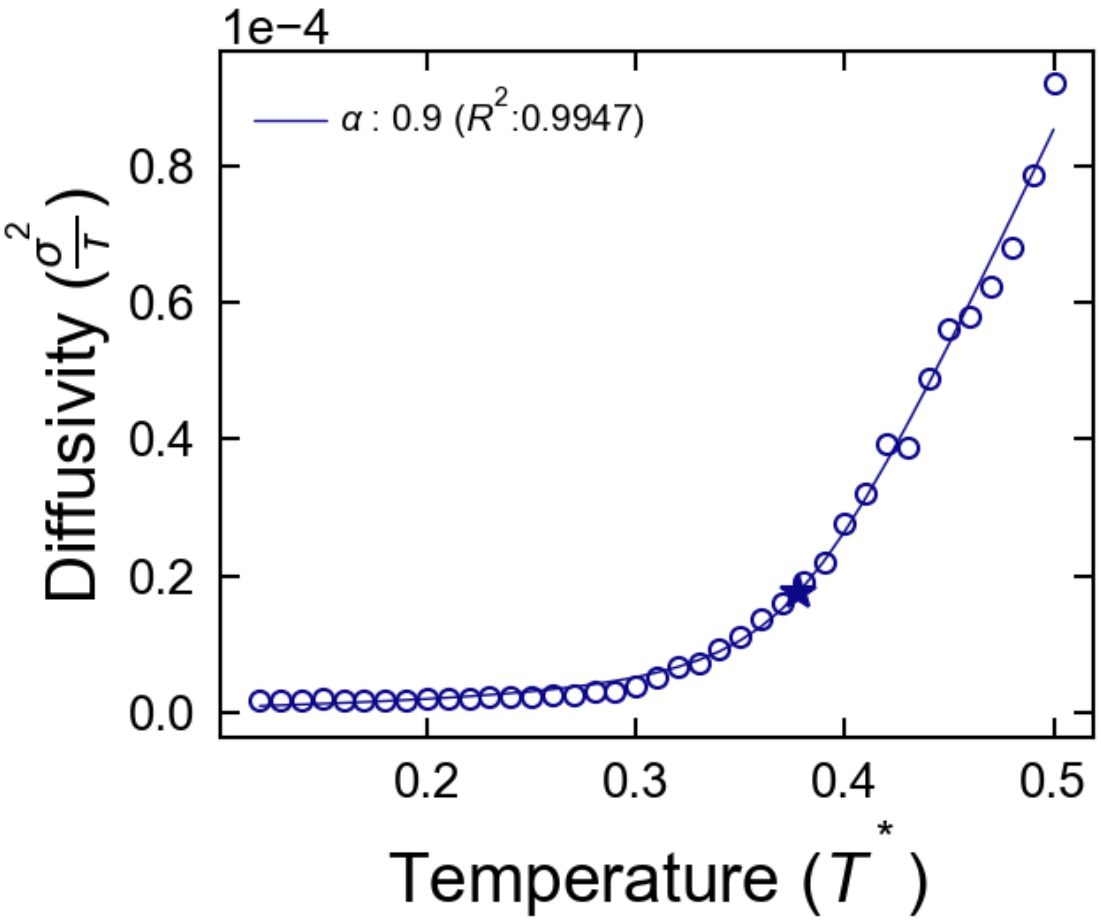



```
In [6]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.5)&
                       (df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                df_curing = df_grp[(df_grp.bond==False)&
```

```
                (df_grp.cooling_method==cooling_method)&
                (df_grp.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.mean()
cure_percents.append(cure_percent)
Ts,Ds=getDiffusivities(project,
                       df_curing,
                       name=PROP_NAME,
                      quench_time=5e6)
Cure_Ts.append(Ts)
#print(Ds)
mul_fact=10000
Ds_scaled=Ds*mul_fact
popt, pcov = curve_fit(hyper,
                       Ts,
                       Ds_scaled,
                       #    x0,y0,  a,  b,  c
                       p0=[1.0,1.0,1e-2,1.0,1.0],
                       maxfev=2000000,
                       bounds=([0,0,0,0,-np.infty],
                       [np.infty,np.infty,1e-1,np.infty,np.infty]))


T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
plt.plot(T0,
         d/mul_fact,
         marker='*',
         color=colors[i],
         markersize=15)#,

plt.plot(Ts,
         Ds,
         marker='o',
        markerfacecolor='w',
         markersize=8,zorder=0,
         color=colors[i],#cooling_colors[j],
         linewidth=0.0)

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)
```

```
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
        {:.2f})'.format(sap/100,
            #
        r_squared,
            #
        a,
            #
        b,
            #
        c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'hyp_90_percent.pdf')
        np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
To 0.377974933611
a 3.21161867782e-06 b 0.000665415896651
Ps 0.0102204148902 0.958642482559


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



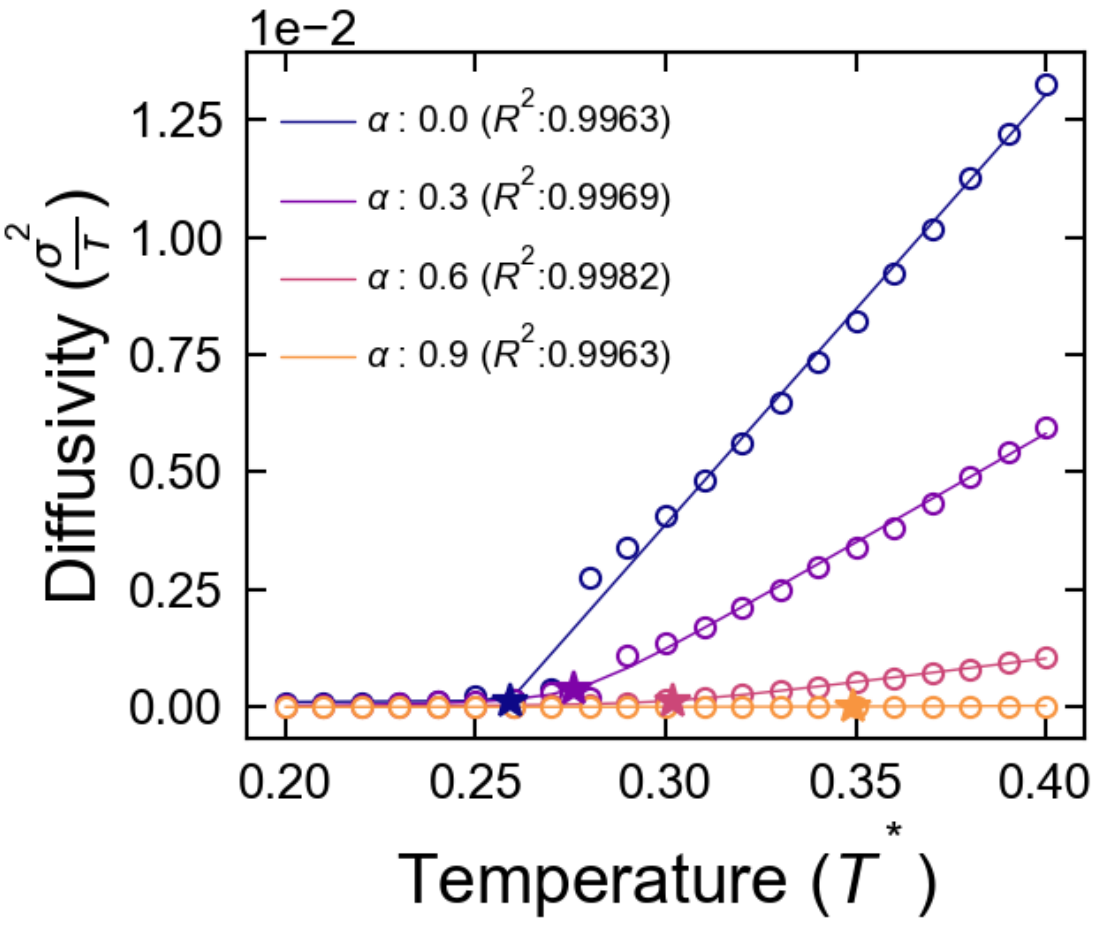

```
In [7]: import matplotlib
        import collections
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.4)&
                       (df.quench_T>=0.2)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        Tgs_dict=collections.OrderedDict()
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
```

```
for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
    df_curing = df_grp[(df_grp.bond==False)&
                   (df_grp.cooling_method==cooling_method)&
                   (df_grp.stop_after_percent==sap)]
    cure_percent = df_curing.cure_percent.mean()
    cure_percents.append(cure_percent)
    Ts,Ds=getDiffusivities(project,
                           df_curing,
                           name=PROP_NAME,
                          quench_time=5e6)
    Cure_Ts.append(Ts)
    #print(Ds)
    avD = np.mean(Ds)

    #print('avD',avD,1/avD)
    mul_fact=1/avD#1000
    Ds_scaled=Ds*mul_fact
    popt, pcov = curve_fit(hyper,
                           Ts,
                           Ds_scaled,
                           #    x0,y0,  a,  b,  c
                           p0=[1.0,1.0,1e-2,1.0,1.0],
                           maxfev=2000000,
                           bounds=([0,0,0,0,-np.infty],
                           [np.infty,np.infty,1e-1,np.infty,np.infty]))


    T0=popt[0]#x0
    D0=popt[1]/mul_fact#y0
    a=popt[2]/mul_fact
    b=popt[3]/mul_fact
    c=popt[4]
    Ps=Pt(Ts,T0,popt[4])
    print('To',T0)
    print('a',a,'b',b)
    print('Ps',Ps[0],Ps[-1])
    xs = Ts#np.linspace(0.1,4)
    yHYP = hyper(xs, *popt)
    residuals = Ds_scaled - yHYP
    ss_res = np.sum(residuals**2)
    ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
    if ss_tot == 0:
        r_squared = 0
    else:
        r_squared = 1 - (ss_res / ss_tot)

    d=hyper(T0,*popt)
    plt.plot(T0,
             d/mul_fact,
             marker='*',
             color=colors[i],
             markersize=15)#,

    plt.plot(Ts,
             Ds,
             marker='o',
            markerfacecolor='w',
             markersize=8,zorder=0,
             color=colors[i],#cooling_colors[j],
             linewidth=0.0)

    m1,b1=get_tangent(Ts[0],*popt)
    m2,b2=get_tangent(Ts[-1],*popt)

    tgx,tgy=line_intersect(m1,b1,m2,b2)
    l1x=np.linspace(Ts[0],tgx+0.1)
    l1y=(m1*l1x)+b1
    #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
```

```
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
        {:.2f})'.format(sap/100,
            #
        r_squared,
            #
        a,
            #
        b,
            #
        c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'all_alphas_quench.pdf')
        np.savetxt('hyperbola_fitting_bounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.transp
        ose(data))

        plt.show()
```

```
To 0.259050416383
a 0.000477033904718 b 0.0910559437524
Ps 2.6645352591e-15 1.0
To 0.276033763106
a 0.000128148738884 b 0.0466252497011
Ps 0.0115223186203 0.995570423525
To 0.301931740239
a 3.02388392061e-05 b 0.0103285836401
Ps 0.0160049421355 0.982774874831
To 0.349187263896
a 4.32907092324e-29 b 0.000440239216907
Ps 0.0245127072329 0.861626118117


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
```

```
  warnings.warn("This figure includes Axes that are not "
```



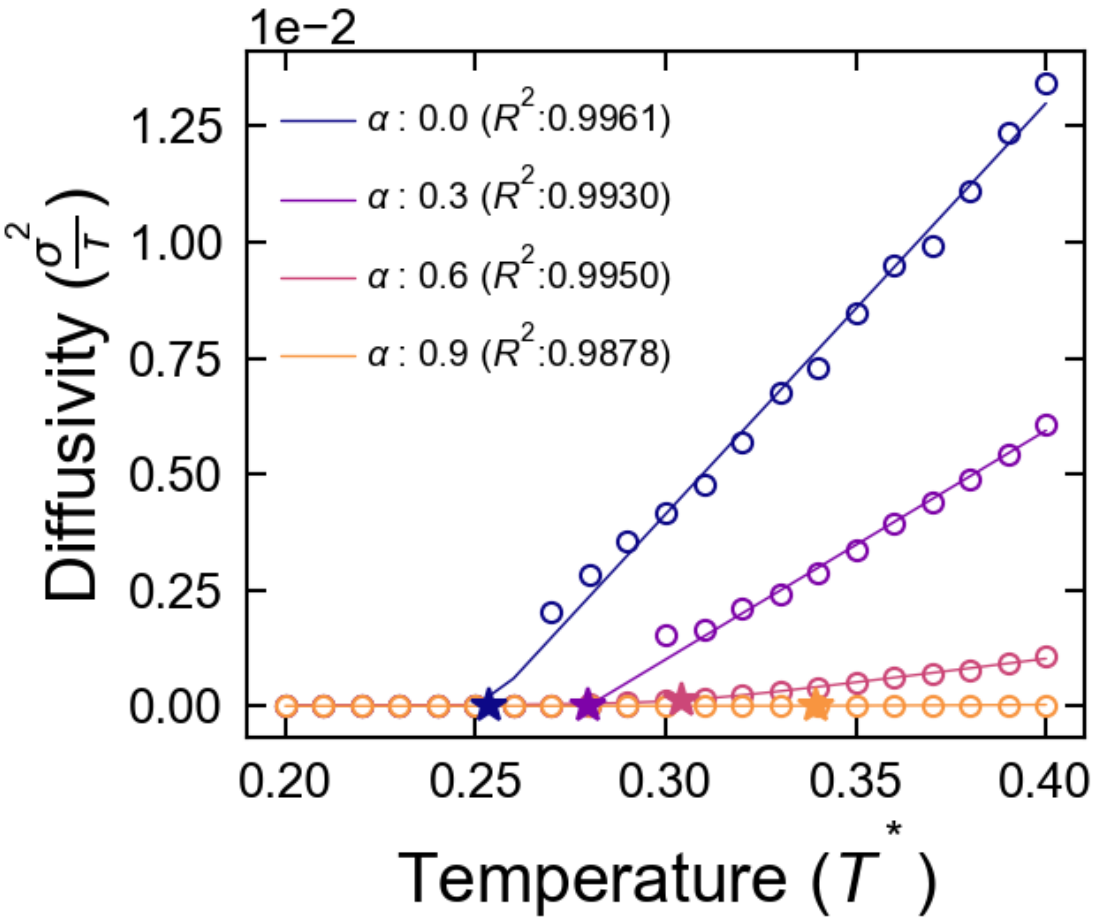

```
In [8]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.,60.,90.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='anneal'
        df_filtered=df[(df.quench_T<=0.4)&
                       (df.quench_T>=0.2)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
```

```
#Tgs_dict={}
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=6e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        avD = np.mean(Ds)

        #print('avD',avD,1/avD)
        mul_fact=1/avD#1000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                               #    x0,y0,  a,  b,  c
                               p0=[1.0,1.0,1e-2,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0,0,0,0,-np.infty],
                               [np.infty,np.infty,1e-1,np.infty,np.infty]))


        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plt.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
```

```
            l1x=np.linspace(Ts[0],tgx+0.1)
            l1y=(m1*l1x)+b1
            #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
        {:.2f})'.format(sap/100,
            #
        r_squared,
            #
        a,
            #
        b,
            #
        c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=15)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #plt.xlim(0.2,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'all_alphas_anneal.pdf')
        np.savetxt('hyperbola_fitting_bounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.transp
        ose(data))

        plt.show()
```

```
To 0.253322253648
a 3.40746141687e-11 b 0.0885622028031
Ps 1.0 2.16493489802e-15
To 0.279715126989
a 4.29573877116e-08 b 0.0493414277108
Ps 1.0 0.0
To 0.304225702318
a 2.84628737507e-05 b 0.0104707486121
Ps 0.985770880498 0.0120947001034
To 0.339360068034
a 5.33501438388e-07 b 0.00040599652744
Ps 0.969461886134 0.00623969952753


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
```

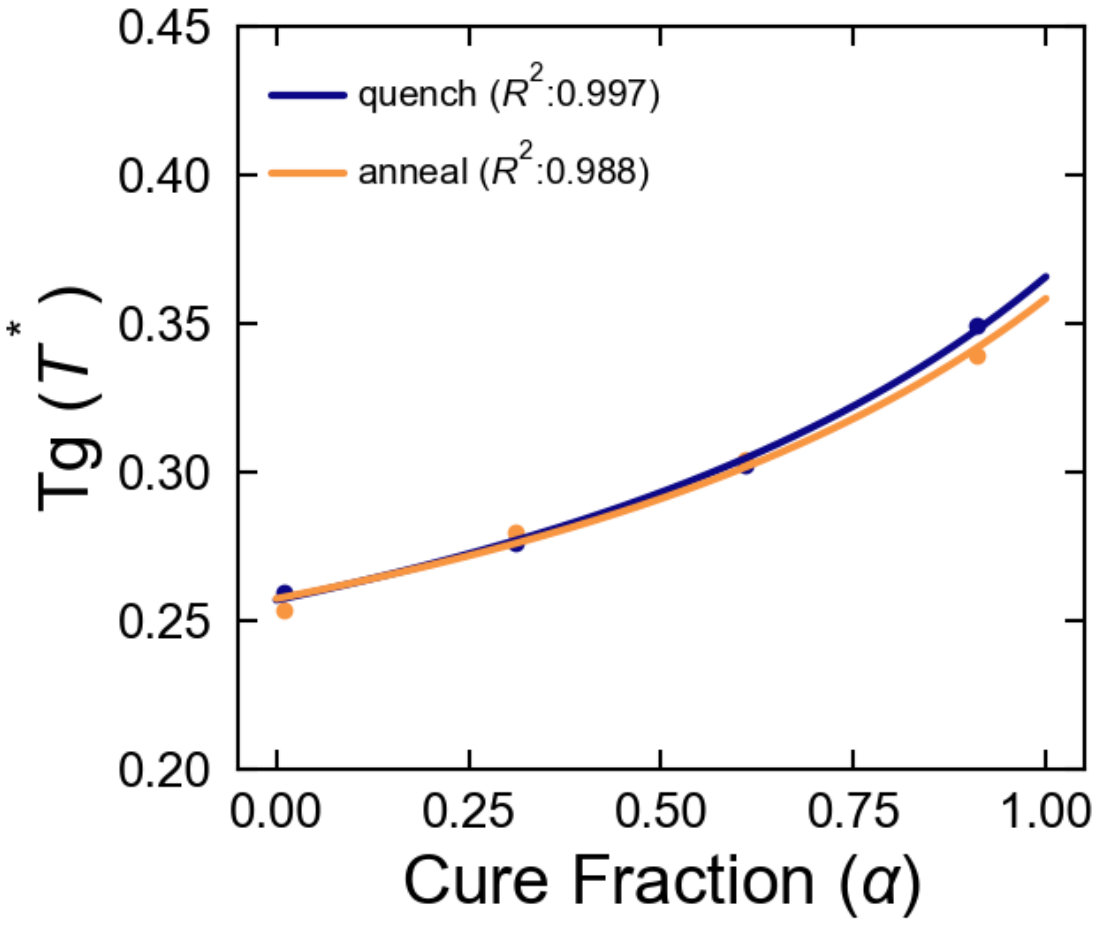
```
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [9]:
```
fig, ax1 = plt.subplots()

colors = plt.cm.plasma(np.linspace(0,0.75,len(Tgs_dict.items())))
for i,(cooling_method,[cure_percents,Tgs]) in enumerate(sorted(list(Tgs_dict.items()),
key=lambda x:x[0].lower(), reverse=True)):
#for i,(cooling_method,[cure_percents,Tgs]) in enumerate(Tgs_dict.items()):
    cure_percents = np.asarray(cure_percents)
    Tgs = np.asarray(Tgs)
    Tg_data = np.asarray([cure_percents/100.,Tgs])
    cure_percents_ss = cure_percents#[:-1]
    Tgs_ss = Tgs#[:-1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                        Tgs_ss,
                                                        T1=None,
                                                        T0=None)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    ax1.plot(alphas,
             fit_ydata,
             color=colors[i],
```

```
                label='{} ($R^2$:{})'.format(cooling_method,round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    color=colors[i])


    ax1.set_xlabel('Cure Fraction ($\\alpha$)')
    ax1.set_ylabel('Tg ($T^*$)')
    ax1.set_ylim(0.2,0.45)
    ax1.legend(fontsize=15,loc='upper left')
    #ax2.legend(fontsize=15,loc='lower right')
    plt.ticklabel_format(axis='y',style='plain')
    savefig(plt,'hyperbola_fitting_bounded_lj_sym','dibeneditto_anneal_quench.pdf')
    #savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

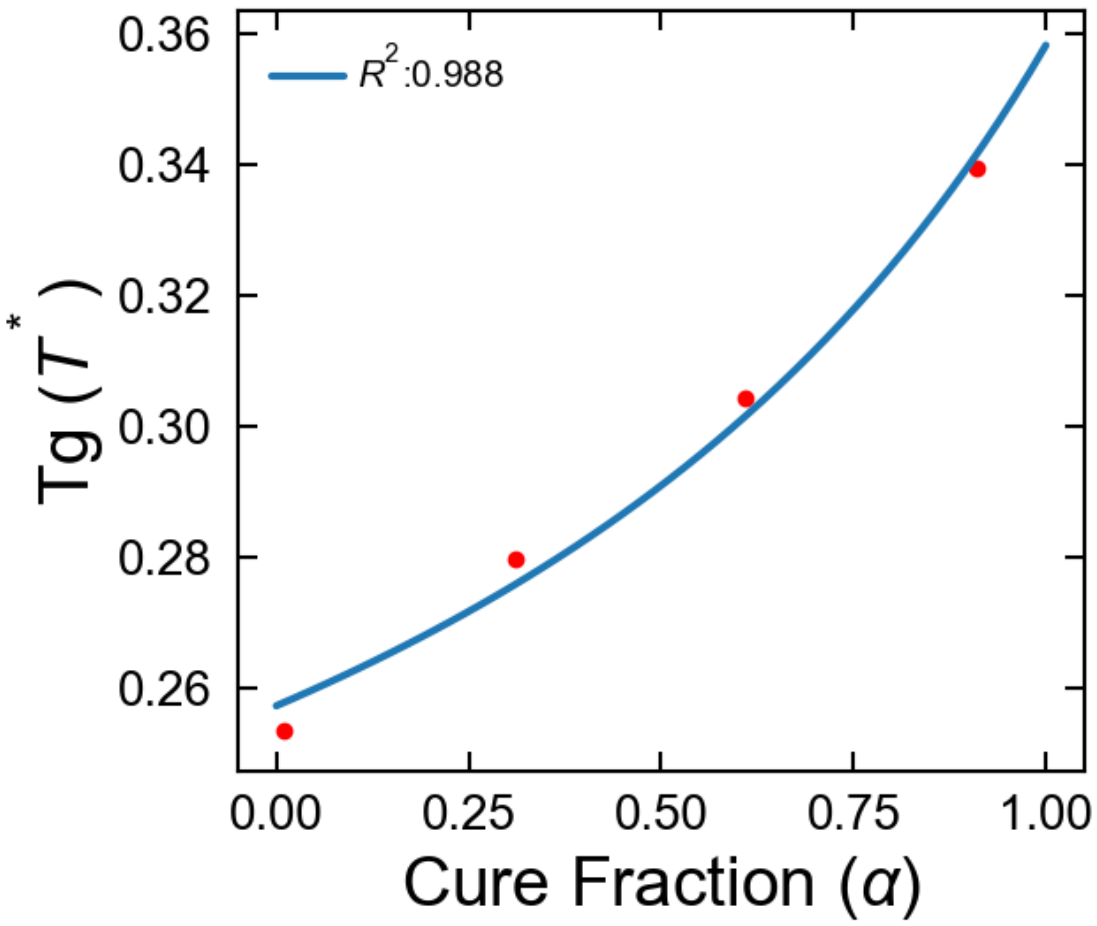
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [10]:
```
cure_percents = np.asarray(cure_percents)
#Tgs=[0.65,0.75,0.9,2.1]
fig, ax1 = plt.subplots()
```

```
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)

        sim_low_lim = 0.5
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.5
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym','dibeneditto.pdf')
```

```
[ 0.25332225  0.27971513  0.3042257   0.33936007]
[  1.00047994  31.00138092  61.0007782   91.00018311]
[ 0.25332225  0.27971513  0.3042257   0.33936007]
T1 0.358298404761 lambda 0.5
300 K in T*: 0.223936502976


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

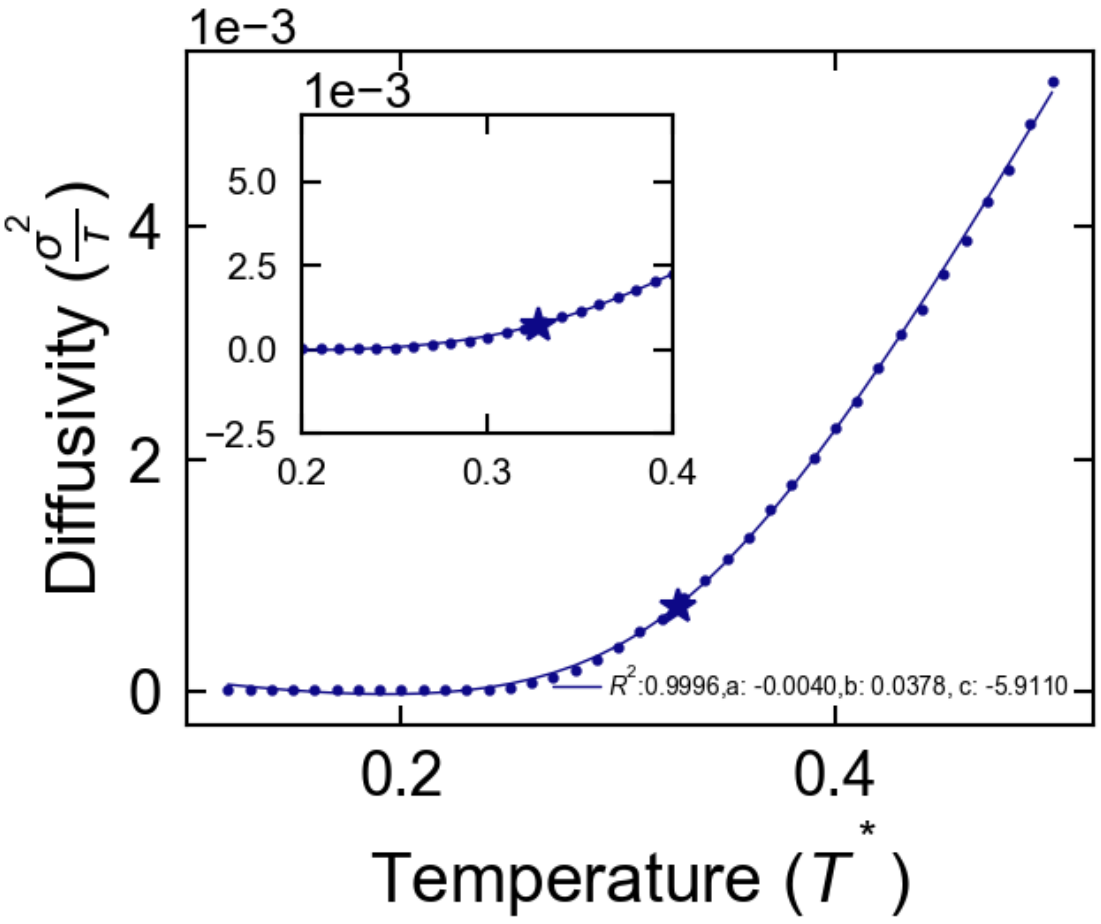



```
In [11]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=5.0)&
```

```
            (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               quench_time=5e6,
                               name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        ax1.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
        ax2.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
```

```
                tgx,tgy=line_intersect(m1,b1,m2,b2)
                l1x=np.linspace(Ts[0],tgx+0.1)
                l1y=(m1*l1x)+b1
                #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
                l2x=np.linspace(tgx-0.1,Ts[-1])
                l2y=(m2*l2x)+b2
                #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

                Tgs_tangent.append(tgx)
                #print(yHYP/mul_fact)
                ax1.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                        color=colors[i],#cooling_colors[j],
                         label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                                   a,
                                                                                   b,
                                                                                   c))
                ax2.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         color=colors[i],#cooling_colors[j],
                         label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                                   a,
                                                                                   b,
                                                                                   c))

                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
            ax1.legend(fontsize=10,loc='lower right')
            ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
            ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
            ax1.tick_params(axis = 'both', which = 'major',labelsize=25)
            Tgs = np.asarray(Tgs)
            cure_percents = np.asarray(cure_percents)
            data=[cure_percents,Tgs]
            ax2.set_xlim(0.2,0.4)
            ax2.set_ylim(-2.5e-3,7e-3)
            ax1.set_xlabel('Temperature ($T^*$)')
            ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
            #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
            savefig(plt,'hyperbola_fitting_unbounded_lj_sym',
                    '50percent.pdf')
            np.savetxt('hyperbola_fitting_unbounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.tran
            spose(data))

            plt.show()
```

```
To 0.327483697655
a -0.00397604311073 b 0.0378448477566
Ps 0.0530986785922 0.928094786613


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



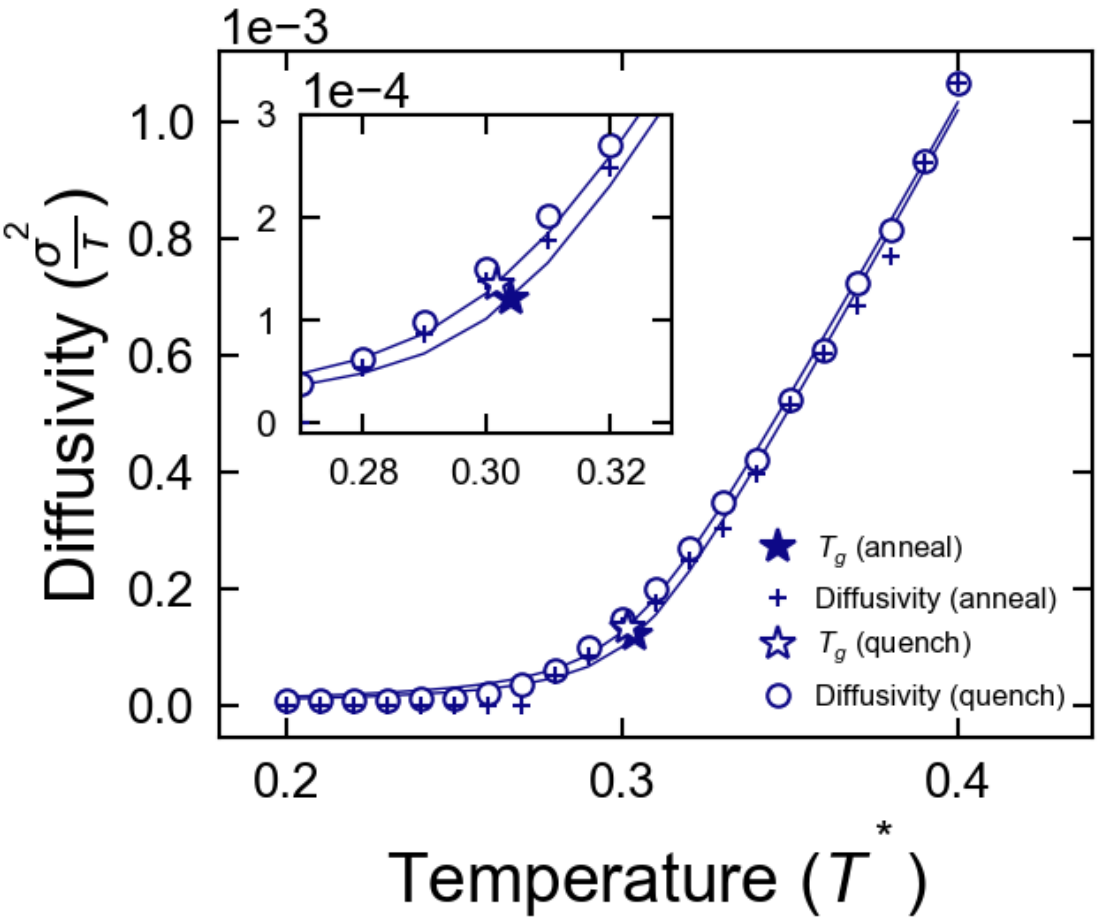

```
In [12]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0

        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                               #    x0,y0,  a,  b,  c
                               p0=[1.0,1.0,1e-2,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0,0,0,0,-np.infty],
                               [np.infty,np.infty,1e-1,np.infty,np.infty]))

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plot_lines += ax1.plot(T0,
```

```
                                d/mul_fact,
                                marker='*',
                                markerfacecolor=Tg_markerfacecolor,
                                color=colors[i],
                                zorder=3,
                                markersize=15,
                                linewidth=0.0,
                               label='$T_g$ ({})'.format(cooling_method))#,

        plot_lines += ax1.plot(Ts,
                                Ds,
                                marker=marker,
                                markerfacecolor='w',
                                markersize=markersize,
                                color=colors[i],#cooling_colors[j],
                                linewidth=0.0,
                                zorder=zorder,
                                label='Diffusivity ({})'.format(cooling_method))

        ax2.plot(T0,
                   d/mul_fact,
                   marker='*',
                   markerfacecolor=Tg_markerfacecolor,
                   color=colors[i],
                   zorder=3,
                 linewidth=0.0,
                   markersize=15,
                  label='$T_g$ ({})'.format(cooling_method))#,

        ax2.plot(Ts,
                   Ds,
                   marker=marker,
                   markerfacecolor='w',
                   markersize=markersize,
                   color=colors[i],#cooling_colors[j],
                   linewidth=0.0,
                   zorder=zorder,
                   label='Diffusivity ({})'.format(cooling_method))

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        Tgs_tangent.append(tgx)
        #print(yHYP/mul_fact)
        ax1.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        ax2.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
{:.2f})'.format(sap/100,
        #
r_squared,
```

```
            #
        a,
            #
        b,
            #
        c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        labels = [l.get_label() for l in plot_lines]
        ax1.legend(plot_lines,
                   labels,
                   fontsize=12,
                   loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.27,0.33)
        ax2.set_ylim(-1e-5,3e-4)
        ax1.set_xlim(0.18,0.44)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'hyperbola_fitting_bounded_lj_sym',
                'hyp_60_percent_quench_anneal.pdf')
        np.savetxt('hyperbola_fitting_bounded_lj_sym/Tg_{}.txt'.format(cooling_method),np.transp
        ose(data))

        plt.show()
```

```
To 0.303939545679
a 1e-05 b 0.0104570420585
Ps 0.986258681678 0.0118073647929
To 0.301606545611
a 1e-05 b 0.0103149433262
Ps 0.0156396841978 0.983373417618


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−3
1e−4
Diffusivity ($\frac{\sigma^2}{T}$)
Temperature (T^*)
T_g (anneal)
Diffusivity (anneal)
T_g (quench)
Diffusivity (quench)

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *


def custom(x,a,b,p):
    return (a*x**p) / (b + x**(p-1))

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,a,b,p):
    return a*x**p*(b*p*x+x**p)/(b*x+x**p)**2

def get_tangent(x,a,b,p):
    m=slope_of_tangent(x,a,b,p)
    y=custom(x,a,b,p)
    b=y-(m*x)
    return m,b
```

In [3]:
```
df_filtered=df[df.cooling_method=='anneal']
```

```
df_filtered.D_A
```

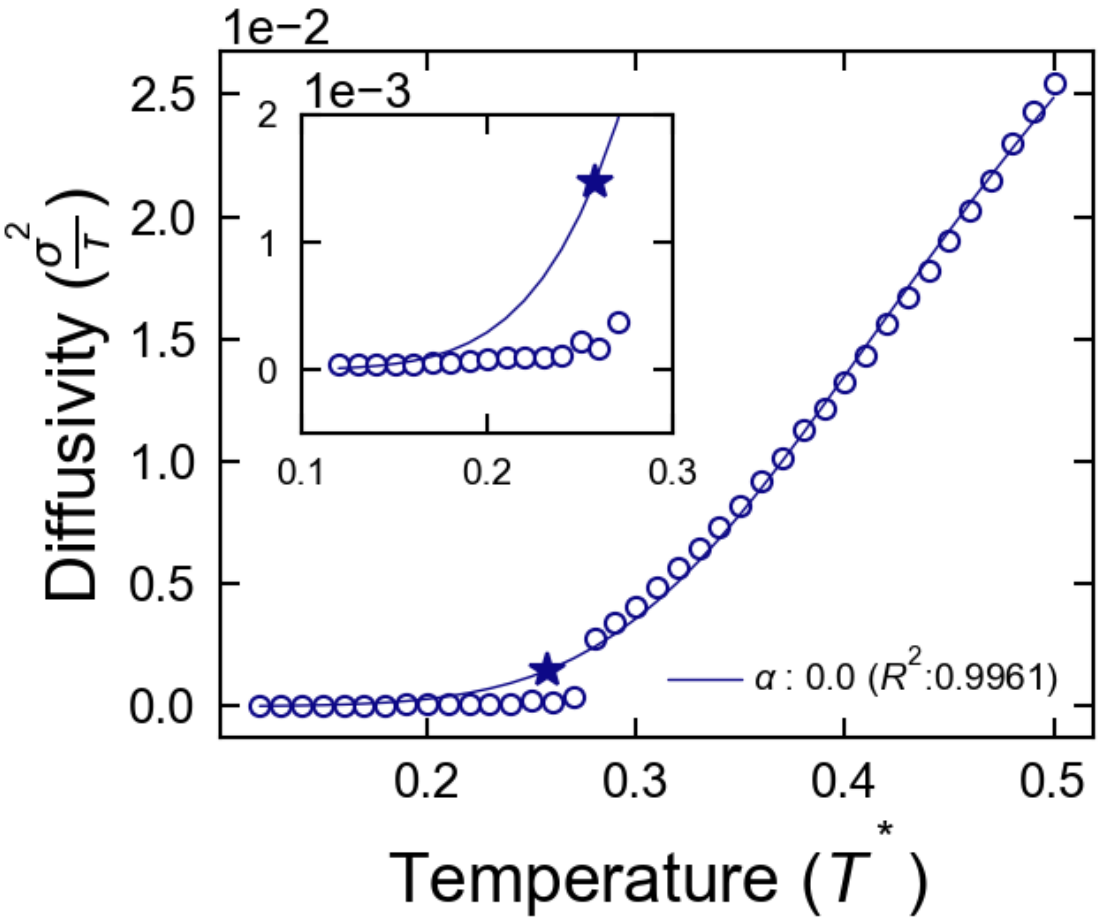
```
Out[3]: 90b99ca468d650a13ade4d55a27ce7e1    3.617847e-04
        ffd54826a80437a447973bbf73be3f68    9.086983e-07
        644d27b8f740cc6df4a628957245df0c    3.637193e-05
        646df3239f44bb1705e0ff4c9a58106e    2.057987e-03
        Name: D_A, dtype: float64
```

```
In [4]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])

        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=0.5)&
                       (df.quench_T>=0.1)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                df_curing = df_grp[(df_grp.bond==False)&
                                   (df_grp.cooling_method==cooling_method)&
                                   (df_grp.stop_after_percent==sap)]
                cure_percent = df_curing.cure_percent.mean()
                cure_percents.append(cure_percent)
                Ts,Ds=getDiffusivities(project,
                                       df_curing,
                                       name=PROP_NAME,
                                      quench_time=5e6)
                Cure_Ts.append(Ts)
                #print(Ds)
                mul_fact=100
                Ds_scaled=Ds*mul_fact
                popt, pcov = curve_fit(custom,
                                       Ts,
                                       Ds_scaled,
                                       #    a,     b,          p
                                       p0=[1.0,1.0,1.0],
                                       maxfev=2000000,
                                       bounds=([0.0,0.0,0.0],
                                       [np.infty,np.infty,np.infty]))

                a=popt[0]
                b=popt[1]
                p=popt[2]
                xs = Ts#np.linspace(0.1,4)
```

```
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        ax2.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax2.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        #print(yHYP/mul_fact)
        ax1.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        ax2.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        Tgs.append(tgx)

ax1.legend(fontsize=15,loc='lower right')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
```

```
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                'plf_0_percent.pdf')
        np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



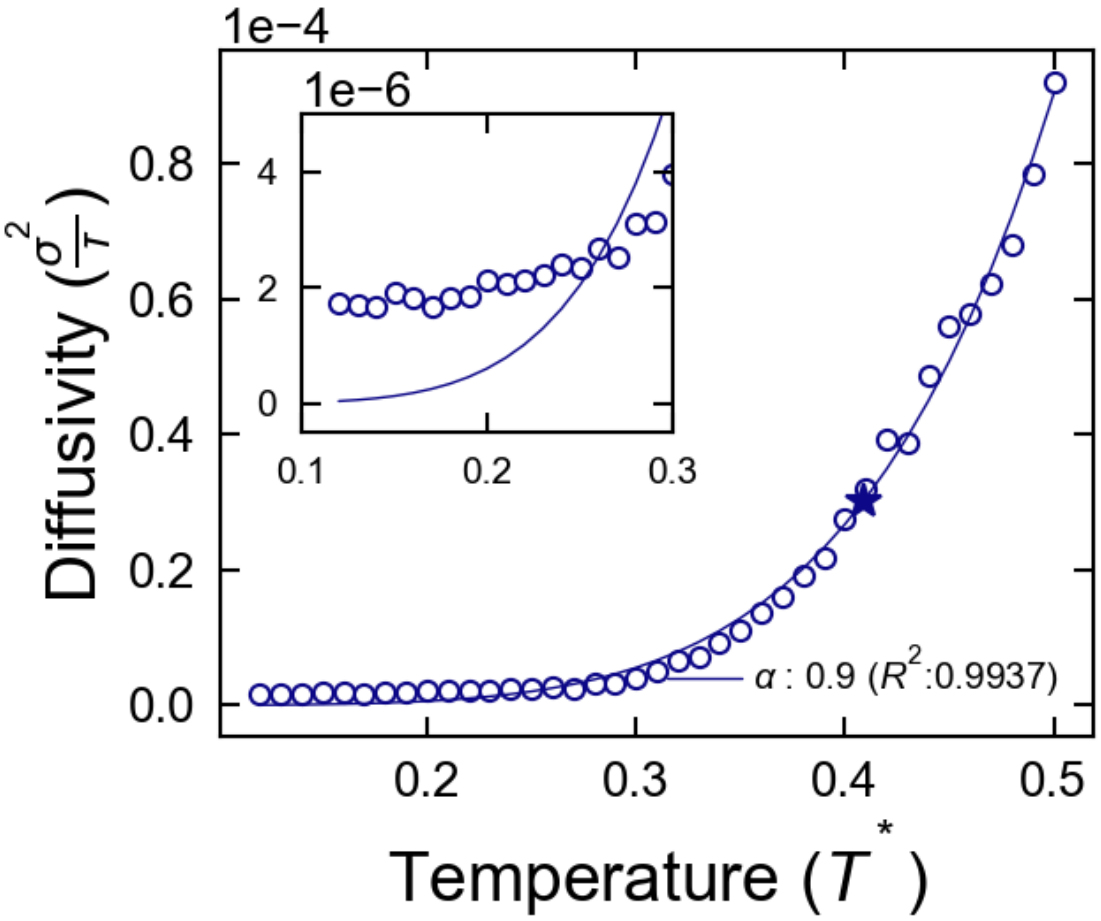

```
In [13]: import matplotlib
         from common import *
         %matplotlib inline
```

```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

fig = plt.figure()
ax1 = fig.add_subplot(111)

left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
ax2 = fig.add_axes([left, bottom, width, height])

#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
```

```
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        ax2.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax2.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        #print(yHYP/mul_fact)
        ax1.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        ax2.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        Tgs.append(tgx)

ax1.legend(fontsize=15,loc='lower right')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
ax2.set_xlim(0.1,0.3)
ax2.set_ylim(-0.5e-6,5e-6)
ax1.set_xlabel('Temperature ($T^*$)')
ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
#savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
#savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
savefig(plt,'custom_fn_lj_sym',
        'plf_90_percent.pdf')
#np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))
```

```
        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

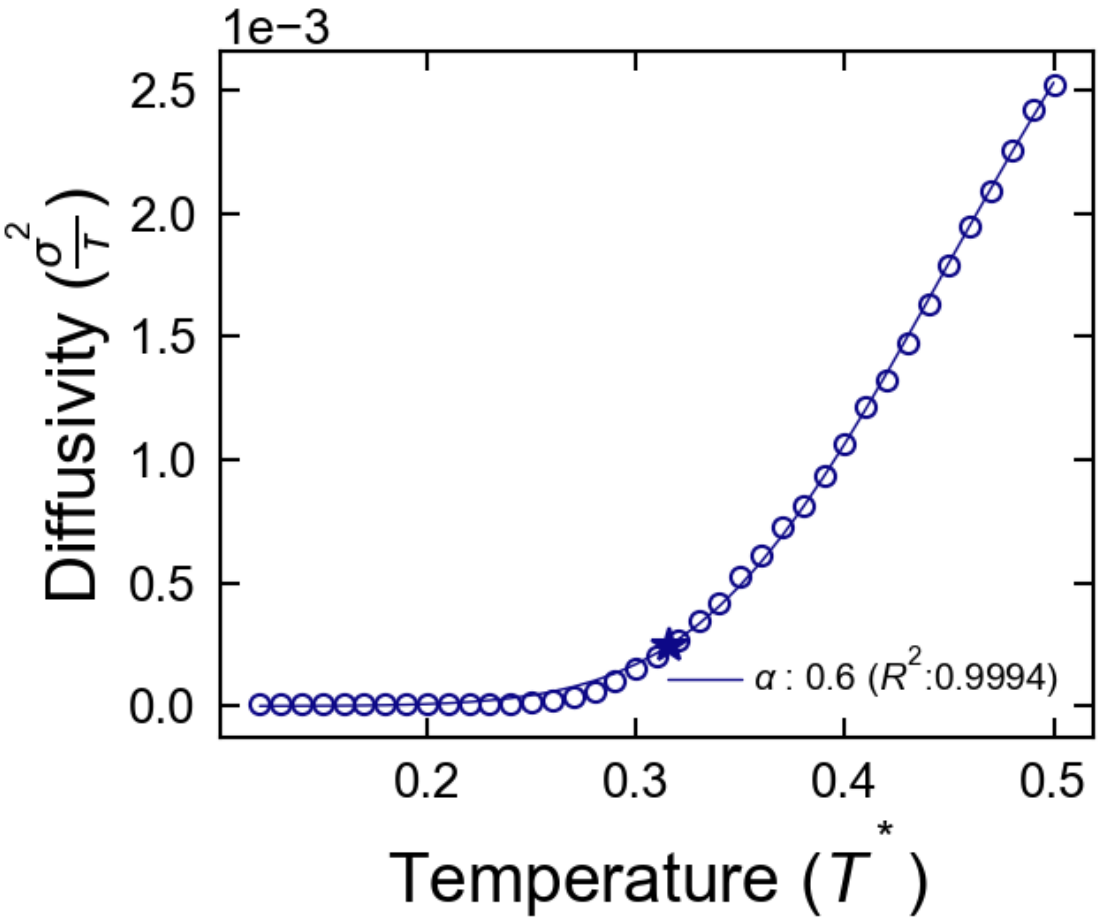



```
In [5]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        #left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        #ax2 = fig.add_axes([left, bottom, width, height])

        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
```

```
#plt.figure()
filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
```

```
            ax1.plot(Ts,
                     Ds,
                     marker='o',
                    markerfacecolor='w',
                     markersize=8,zorder=0,
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0)

            #ax2.plot(tgx,
            #         d/mul_fact,
            #         marker='*',
            #         color=colors[i],
            #         markersize=15)#,
            #
            #ax2.plot(Ts,
            #         Ds,
            #         marker='o',
            #        markerfacecolor='w',
            #         markersize=8,zorder=0,
            #         color=colors[i],#cooling_colors[j],
            #         linewidth=0.0)

            #print(yHYP/mul_fact)
            ax1.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #ax2.plot(xs,
            #         yHYP/mul_fact,
            #         linewidth=1.0,
            #         zorder=0,
            #        color=colors[i],#cooling_colors[j],
            #         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            Tgs.append(tgx)

        ax1.legend(fontsize=15,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        #ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        #ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #ax2.set_xlim(0.1,0.3)
        #ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                'plf_60_percent.pdf')
        np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

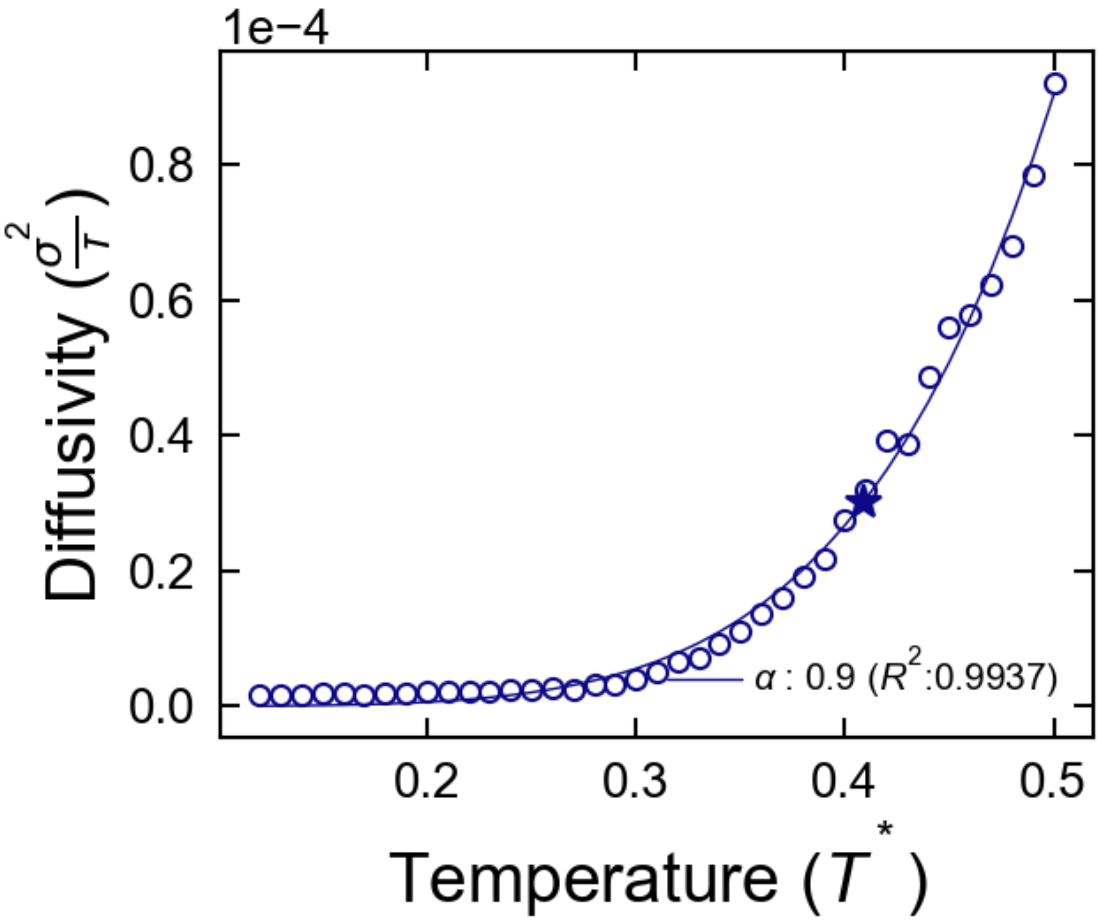



```
In [6]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        #left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        #ax2 = fig.add_axes([left, bottom, width, height])

        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.5)&
               (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        #ax2.plot(tgx,
        #         d/mul_fact,
```

```
            #          marker='*',
            #          color=colors[i],
            #          markersize=15)#,
            #
            #ax2.plot(Ts,
            #          Ds,
            #          marker='o',
            #         markerfacecolor='w',
            #          markersize=8,zorder=0,
            #          color=colors[i],#cooling_colors[j],
            #          linewidth=0.0)

            #print(yHYP/mul_fact)
            ax1.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #ax2.plot(xs,
            #          yHYP/mul_fact,
            #          linewidth=1.0,
            #          zorder=0,
            #         color=colors[i],#cooling_colors[j],
            #          label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            Tgs.append(tgx)

    ax1.legend(fontsize=15,loc='lower right')
    ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    #ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
    #ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
    ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    #ax2.set_xlim(0.1,0.3)
    #ax2.set_ylim(-0.5e-3,2e-3)
    ax1.set_xlabel('Temperature ($T^*$)')
    ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
    #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
    savefig(plt,'custom_fn_lj_sym',
            'plf_90_percent.pdf')
    np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

    plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

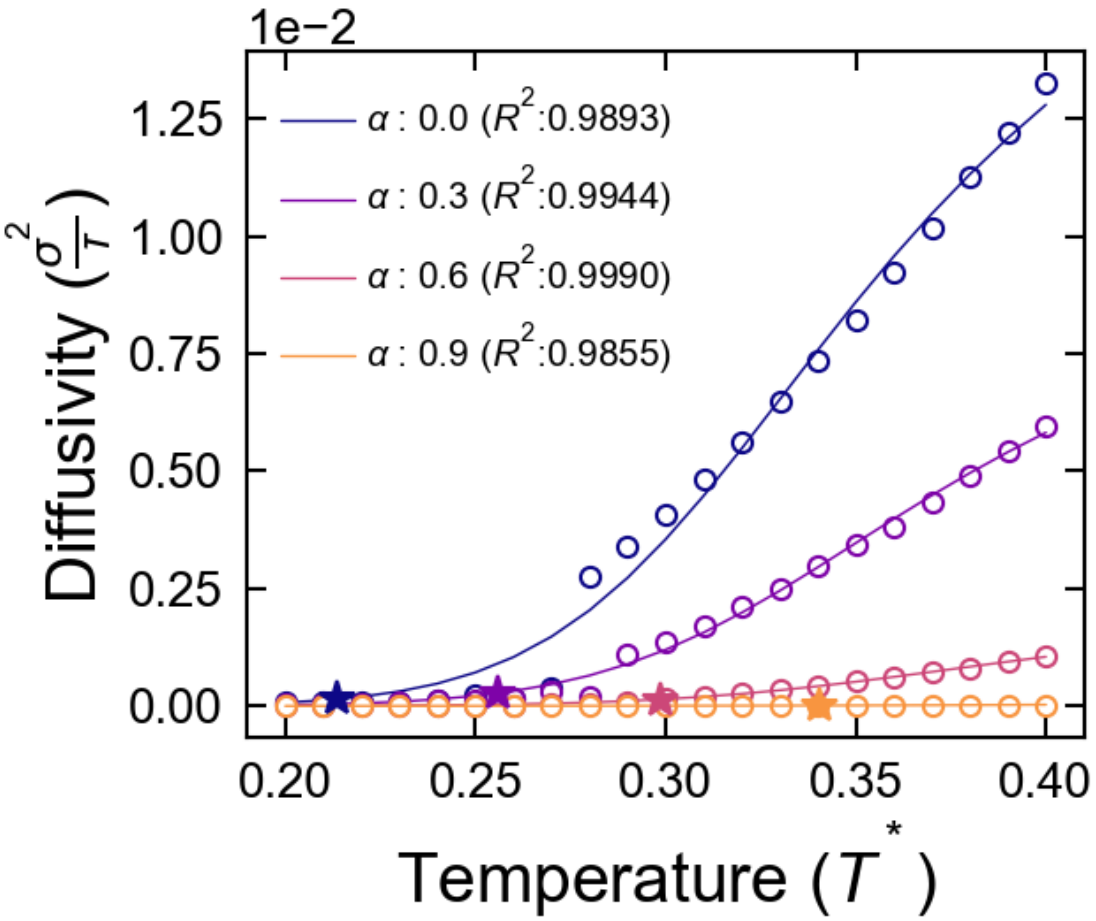



```
In [7]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        import collections
        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        #left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        #ax2 = fig.add_axes([left, bottom, width, height])

        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.0,60.0,90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
```

```
Tgs_dict = collections.OrderedDict()

cooling_method='quench'
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=5e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
```

```
                #ax2.plot(tgx,
                #         d/mul_fact,
                #         marker='*',
                #         color=colors[i],
                #         markersize=15)#,
                #
                #ax2.plot(Ts,
                #         Ds,
                #         marker='o',
                #        markerfacecolor='w',
                #         markersize=8,zorder=0,
                #         color=colors[i],#cooling_colors[j],
                #         linewidth=0.0)

                #print(yHYP/mul_fact)
                ax1.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         zorder=0,
                        color=colors[i],#cooling_colors[j],
                         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
                #ax2.plot(xs,
                #         yHYP/mul_fact,
                #         linewidth=1.0,
                #         zorder=0,
                #        color=colors[i],#cooling_colors[j],
                #         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
                Tgs.append(tgx)

        ax1.legend(fontsize=15,loc='best')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        #ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        #ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #ax2.set_xlim(0.1,0.3)
        #ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                'all_alpha_quenches.pdf')
        np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

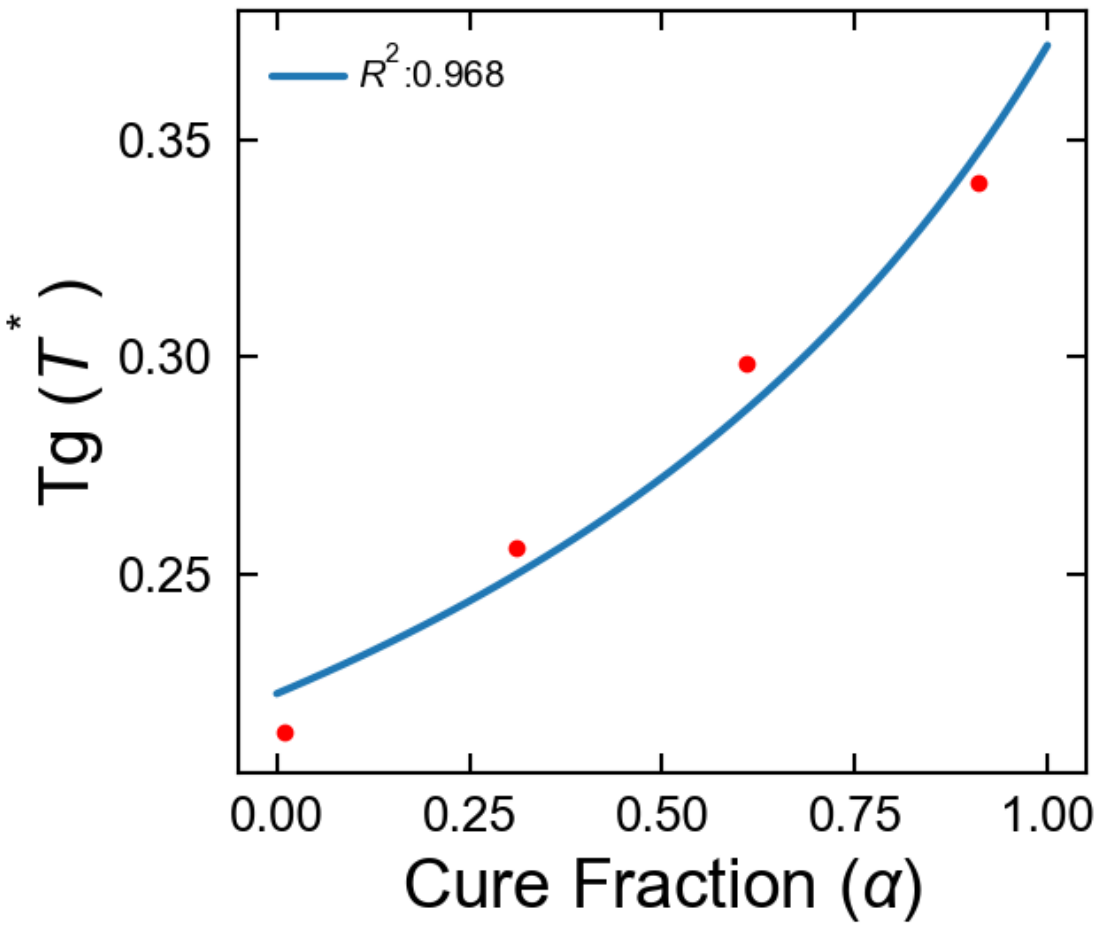
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [8]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
```

```
        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)

        sim_low_lim = 0.5
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.5
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'custom_fn_lj_sym','dibeneditto.pdf')
```

```
[ 0.21357302  0.25597607  0.29850292  0.34027198]
[  1.00047994  31.00138092  61.0007782   91.00018311]
[ 0.21357302  0.25597607  0.29850292  0.34027198]
T1 0.371678771585 lambda 0.5
300 K in T*: 0.232299232241


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

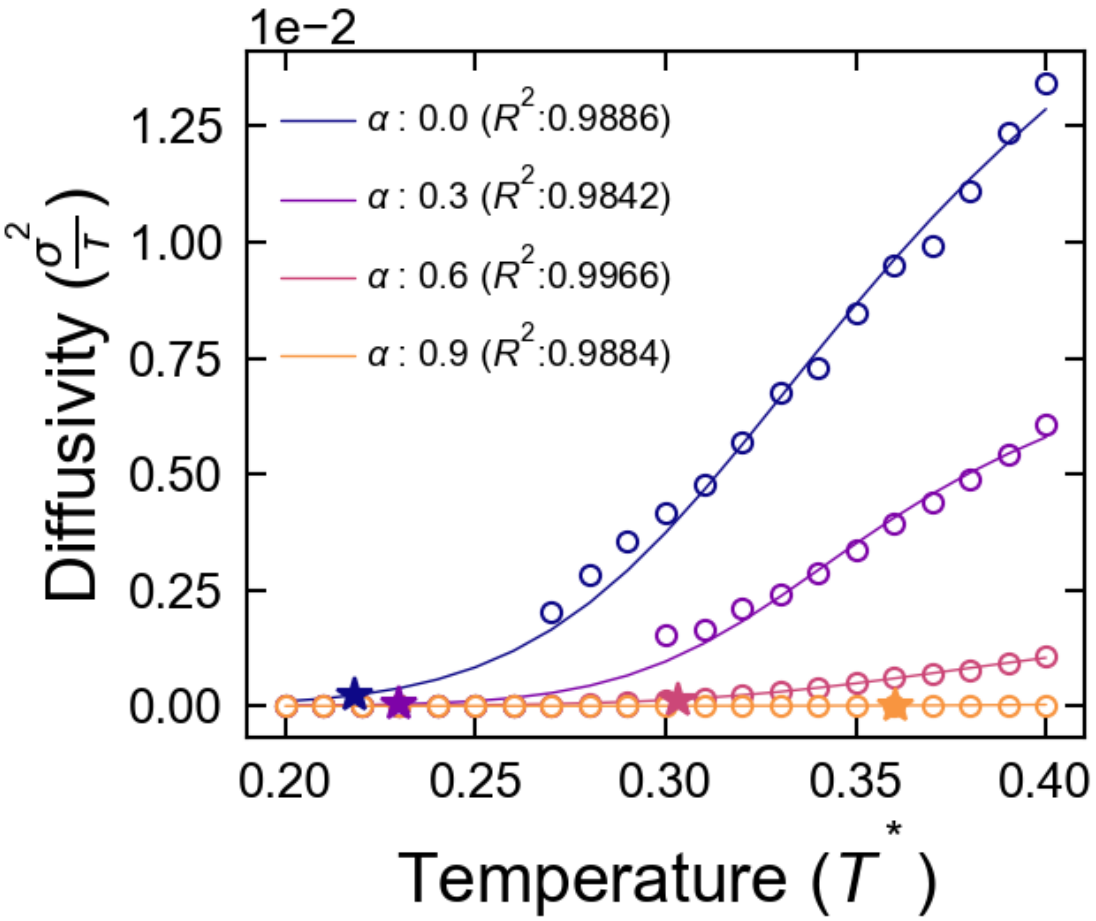



```
In [9]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        #left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        #ax2 = fig.add_axes([left, bottom, width, height])

        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.0,60.0,90.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='anneal'
```

```
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=6e6)
        Cure_Ts.append(Ts)
        #print(Ds)
        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        ax1.plot(tgx,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='o',
                markerfacecolor='w',
                 markersize=8,zorder=0,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        #ax2.plot(tgx,
        #         d/mul_fact,
```

```
            #         marker='*',
            #         color=colors[i],
            #         markersize=15)#,
            #
            #ax2.plot(Ts,
            #         Ds,
            #         marker='o',
            #        markerfacecolor='w',
            #         markersize=8,zorder=0,
            #         color=colors[i],#cooling_colors[j],
            #         linewidth=0.0)

            #print(yHYP/mul_fact)
            ax1.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            #ax2.plot(xs,
            #         yHYP/mul_fact,
            #         linewidth=1.0,
            #         zorder=0,
            #        color=colors[i],#cooling_colors[j],
            #         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
            Tgs.append(tgx)

        ax1.legend(fontsize=15,loc='best')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        #ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        #ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        Tgs_dict[cooling_method]=[cure_percents,Tgs]
        data=[cure_percents,Tgs]
        #ax2.set_xlim(0.1,0.3)
        #ax2.set_ylim(-0.5e-3,2e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                'all_alpha_anneal.pdf')
        np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

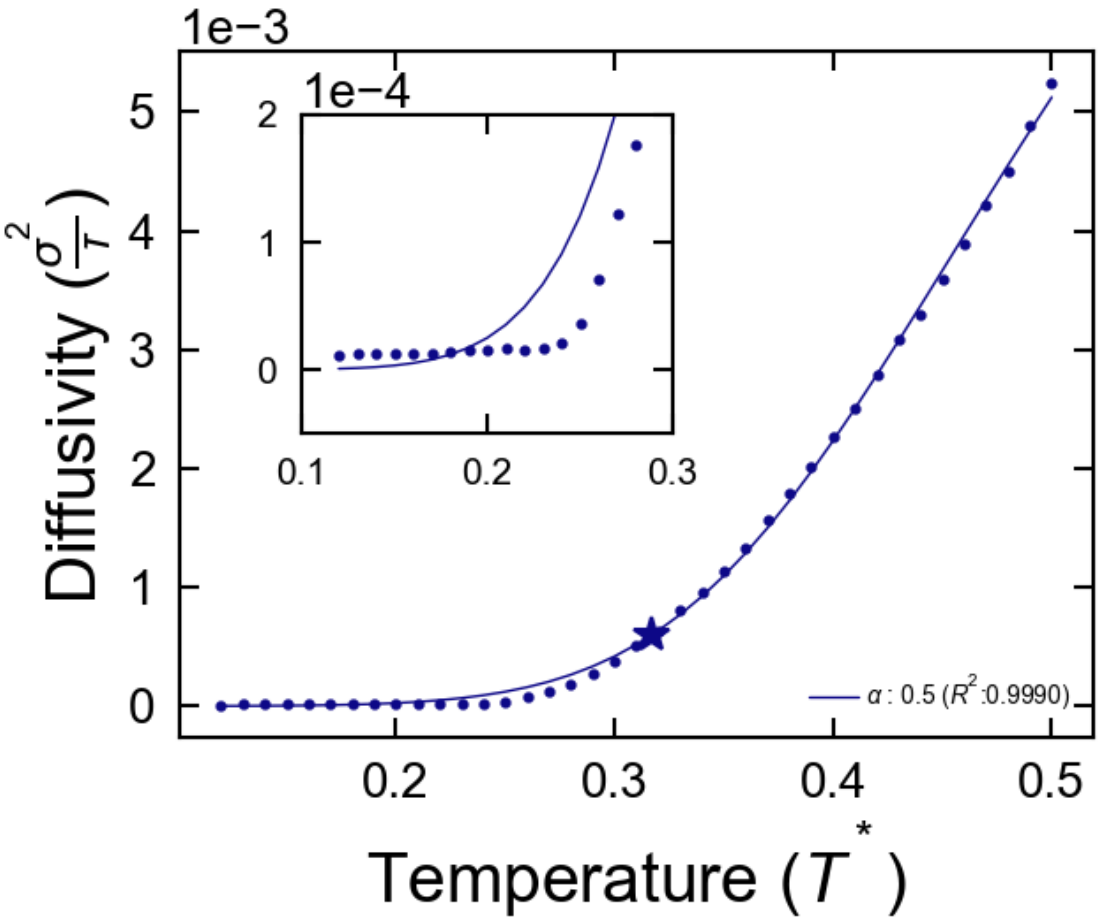



```
In [10]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         df_filtered=df[(df.quench_T<=5.0)&
```

```
            (df.quench_T>=0.1)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               quench_time=5e6,
                               name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        T0=tgx

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
```

```
                l1x=np.linspace(Ts[0],tgx+0.1)
                l1y=(m1*l1x)+b1
                #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
                l2x=np.linspace(tgx-0.1,Ts[-1])
                l2y=(m2*l2x)+b2
                #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

                Tgs_tangent.append(tgx)

                ax1.plot(T0,
                         d/mul_fact,
                         marker='*',
                         color=colors[i],
                         markersize=15)#,
                ax2.plot(T0,
                         d/mul_fact,
                         marker='*',
                         color=colors[i],
                         markersize=15)#,
                #print(yHYP/mul_fact)
                ax1.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         zorder=0,
                        color=colors[i],#cooling_colors[j],
                         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
                ax2.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                         zorder=0,
                        color=colors[i],#cooling_colors[j],
                         label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))

                Tgs.append(tgx)
        ax1.legend(fontsize=10,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,0.3)
        ax2.set_ylim(-0.5e-4,2e-4)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                '50percent.pdf')
        np.savetxt('custom_fn_lj_sym/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



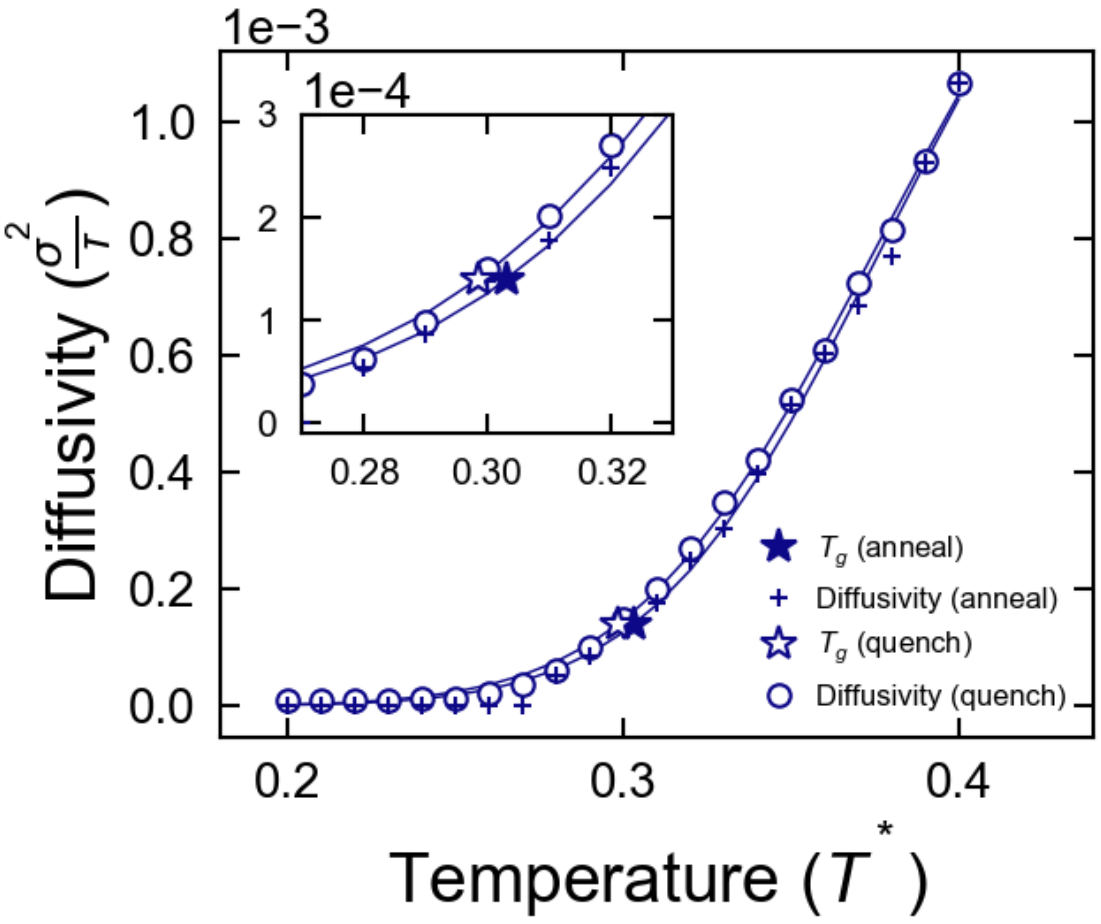

```
In [11]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)

         left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
         ax2 = fig.add_axes([left, bottom, width, height])

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[60.0]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered=df[(df.quench_T<=0.4)&
               (df.quench_T>=0.2)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
plot_lines = []
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        if cooling_method == 'anneal':
            quench_time = None
            marker='+'
            zorder=2
            markersize=7
            Tg_marker='x'
            Tg_markerfacecolor=None
            markerfacecolor='w',
            linewidth=0
        else:
            quench_time = 5e6
            marker='o'
            zorder=1
            markersize=9
            Tg_markerfacecolor='w'
            Tg_marker='*'
            markerfacecolor='w',
            linewidth=0

        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                              quench_time=quench_time)
        Cure_Ts.append(Ts)
        #print(Ds)
        avDs=np.mean(Ds)
        mul_fact=1/avDs#100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(custom,
                               Ts,
                               Ds_scaled,
                               #    a,     b,          p
                               p0=[1.0,1.0,1.0],
                               maxfev=2000000,
                               bounds=([0.0,0.0,0.0],
                               [np.infty,np.infty,np.infty]))

        a=popt[0]
        b=popt[1]
        p=popt[2]
        xs = Ts#np.linspace(0.1,4)
        yHYP = custom(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)
        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
        l2x=np.linspace(tgx-0.1,Ts[-1])
```

```
        l2y=(m2*l2x)+b2
        #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

        d=custom(tgx,*popt)
        T0=tgx
        plot_lines += ax1.plot(T0,
                               d/mul_fact,
                               marker='*',
                               markerfacecolor=Tg_markerfacecolor,
                               color=colors[i],
                               zorder=3,
                               markersize=15,
                               linewidth=0.0,
                              label='$T_g$ ({})'.format(cooling_method))#,

        plot_lines += ax1.plot(Ts,
                               Ds,
                               marker=marker,
                               markerfacecolor='w',
                               markersize=markersize,
                               color=colors[i],#cooling_colors[j],
                               linewidth=0.0,
                               zorder=zorder,
                               label='Diffusivity ({})'.format(cooling_method))

        ax2.plot(T0,
                   d/mul_fact,
                   marker='*',
                   markerfacecolor=Tg_markerfacecolor,
                   color=colors[i],
                   zorder=3,
                 linewidth=0.0,
                   markersize=15,
                    label='$T_g$ ({})'.format(cooling_method))#,

        ax2.plot(Ts,
                   Ds,
                   marker=marker,
                   markerfacecolor='w',
                   markersize=markersize,
                   color=colors[i],#cooling_colors[j],
                   linewidth=0.0,
                   zorder=zorder,
                   label='Diffusivity ({})'.format(cooling_method))



        Tgs_tangent.append(tgx)
        #print(yHYP/mul_fact)
        ax1.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        ax2.plot(xs,
                 yHYP/mul_fact,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='$\\alpha$ : {:.1f} ($R^2$:{:.4f})'.format(sap/100,r_squared))
        #label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.3f},b: {:.2f}, c:
{:.2f})'.format(sap/100,
        #
r_squared,
        #
a,
        #
```

```
        b,
                #
        c))

                #xvals = np.linspace(-3,4)
                #yfits = hyper(xvals, *popt)
                Tgs.append(tgx)
        labels = [l.get_label() for l in plot_lines]
        ax1.legend(plot_lines,
                   labels,
                   fontsize=12,
                   loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.27,0.33)
        ax2.set_ylim(-1e-5,3e-4)
        ax1.set_xlim(0.18,0.44)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')

        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'custom_fn_lj_sym',
                'hyp_60_percent_quench_anneal.pdf')
        np.savetxt('v/Tg_{}.txt'.format(cooling_method),np.transpose(data))

        plt.show()
```

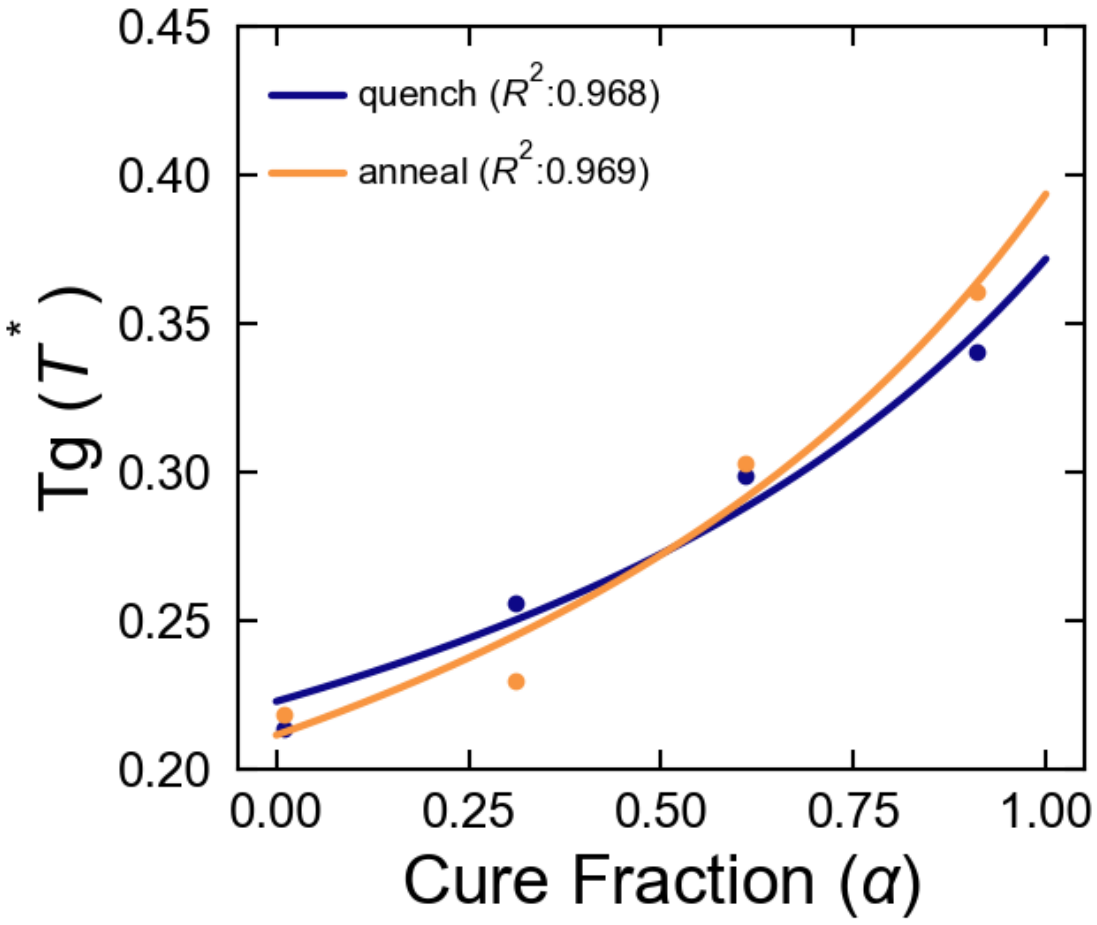
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [12]: fig, ax1 = plt.subplots()

         colors = plt.cm.plasma(np.linspace(0,0.75,len(Tgs_dict.items())))
         for i,(cooling_method,[cure_percents,Tgs]) in enumerate(Tgs_dict.items()):
             cure_percents = np.asarray(cure_percents)
             Tgs = np.asarray(Tgs)
             Tg_data = np.asarray([cure_percents/100.,Tgs])
             cure_percents_ss = cure_percents#[:-1]
             Tgs_ss = Tgs#[:-1]
             R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                                 Tgs_ss,
                                                                 T1=None,
                                                                 T0=None)
             alphas = np.linspace(0,1)
             fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
             ax1.plot(alphas,
                      fit_ydata,
                      color=colors[i],
                      label='{} ($R^2$:{})'.format(cooling_method,round(R2,3)))
             ax1.scatter(cure_percents/100.,
                         Tgs,
                         color=colors[i])


         ax1.set_xlabel('Cure Fraction ($\\alpha$)')
         ax1.set_ylabel('Tg ($T^*$)')
```

```
ax1.set_ylim(0.2,0.45)
ax1.legend(fontsize=15,loc='upper left')
#ax2.legend(fontsize=15,loc='lower right')
plt.ticklabel_format(axis='y',style='plain')
savefig(plt,'custom_fn_lj_sym','dibeneditto_anneal_quench.pdf')
#savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [1]:
```
from common import *
import numpy as np
```

In [2]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA/'
```

In [3]:
```
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

1 Parameters for blend

In [4]:
```
tau=1
tauP=10
num_c10=0 #0, 2
use_curing_job_P=False
quench_time=6e6
cooling_method = 'anneal'
P = 4.84#[4.84 , 5.44, 6.29]
stop_after_percents = [0,30,60,90]
```

In [5]:
```
df_curing = df[(df.bond==True)]#&
                #(df.tau==tau)&
                #(df.tauP==tauP)&
                #(df.cooling_method==cooling_method)&
                #(df.P==P)&
                #(df.num_c10==num_c10)&
                #(df.stop_after_percent==100)]
#df_curing.quench_temp_prof[-1][1][0]/df_curing.dcd_write
df_curing.FO_model_R2.describe()
```

Out[5]:
```
count    21.000000
mean      0.910970
std       0.129217
min       0.421453
25%       0.903635
50%       0.946459
75%       0.982772
max       1.000000
Name: FO_model_R2, dtype: float64
```

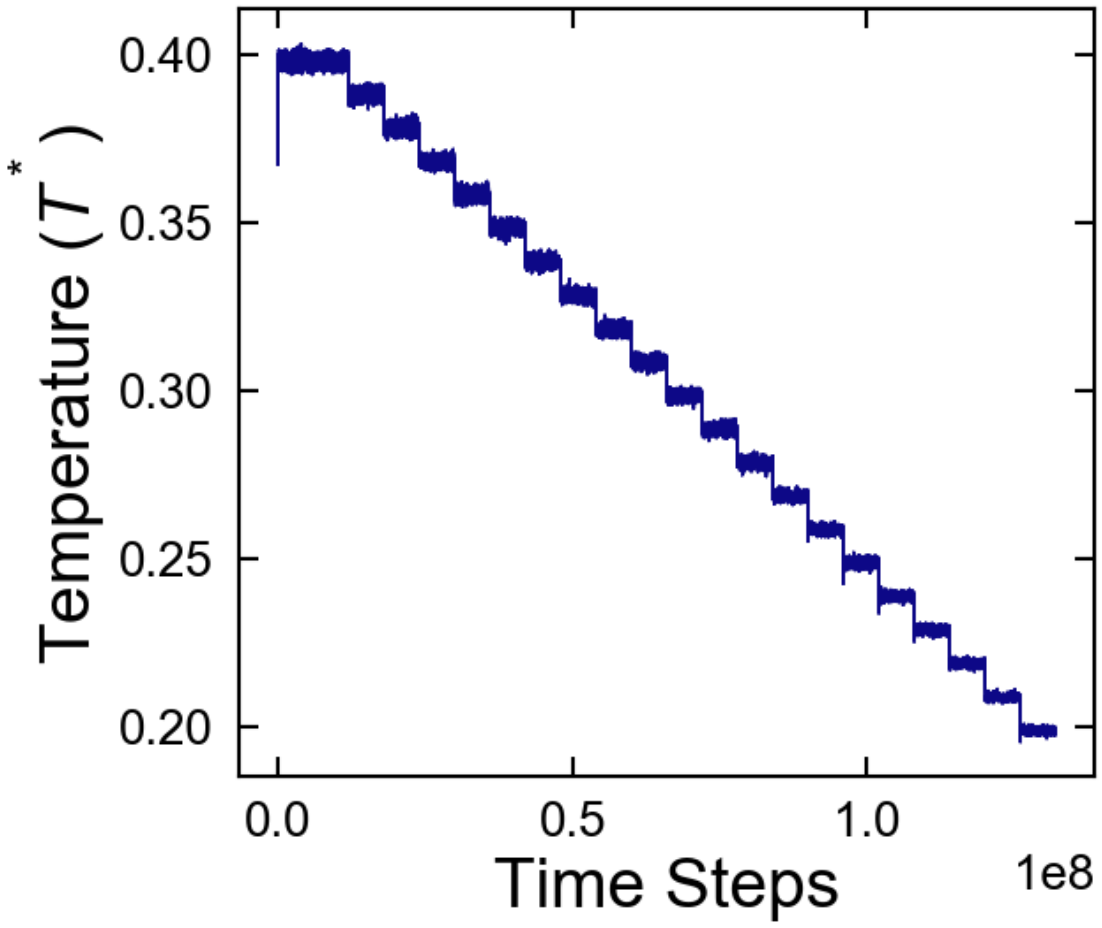
```
In [6]: df_curing = df[(df.bond==False)&
                       #(df.tau==tau)&
                       #(df.tauP==tauP)&
                       (df.cooling_method==cooling_method)]
                       #(df.P==P)&
                       #(df.num_c10==num_c10)&
                       #(df.stop_after_percent==100)]
        #df_curing.quench_temp_prof[-1][1][0]/df_curing.dcd_write
        df_curing.stop_after_percent
```

```
Out[6]: 90b99ca468d650a13ade4d55a27ce7e1     30
        ffd54826a80437a447973bbf73be3f68     90
        644d27b8f740cc6df4a628957245df0c     60
        646df3239f44bb1705e0ff4c9a58106e      0
        Name: stop_after_percent, dtype: object
```

```
In [12]: #stop_after_percents = [100.]#np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='temperature'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()
         for sap in stop_after_percents:
             print(sap)
             df_curing = df[(df.bond==False)&
                            #(df.tau==tau)&
                            #(df.use_curing_job_P==use_curing_job_P)&
                            #(df.tauP==tauP)&
                            #(df.P==P)&
                            (df.quench_time==quench_time)&
                            (df.num_c10==num_c10)&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==0.0)]
             plot_equilibriation(df_curing,
                                 project,PROP_NAME,
                                 draw_equilibrium_window=False,
                                 draw_decorrelated_samples=False)

             #break
         plt.xlabel('Time Steps')
         plt.ylabel('Temperature ($T^*$)')
         savefig(plt,
                 'anneal_DGEBA_DDS_LJ',
                'anneal_temperature.png')
         #plt.xlim(87610000.0, 90000000.0)
         #plt.ylim(44000,45000)
         #plt.legend(fontsize=5)
         #plt.xlim(0,100)
         #plt.ylim(34000,41000)
         plt.show()
```

```
0
30
60
90
```

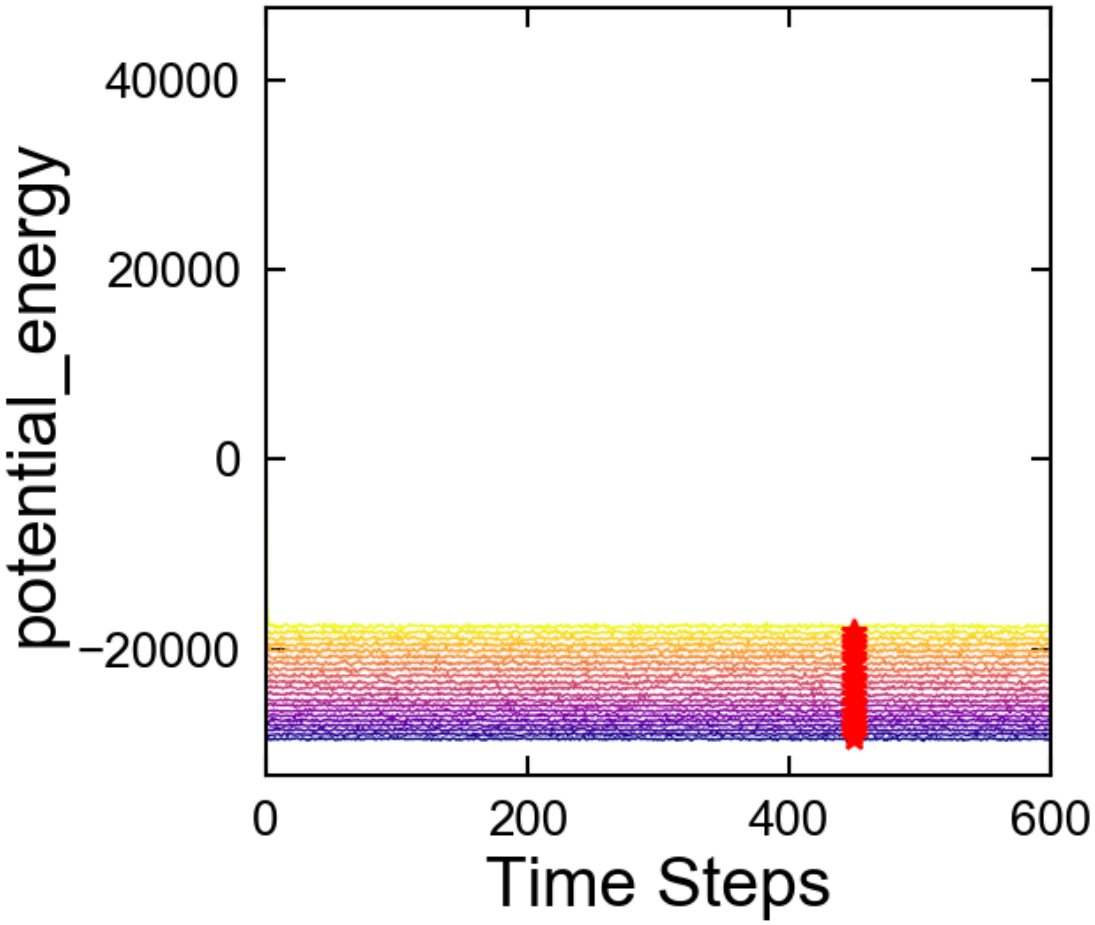
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [8]: #stop_after_percents = [30.]#np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='potential_energy'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        for sap in stop_after_percents:
            df_curing = df[(df.bond==False)&
                           #(df.tau==tau)&
                           #(df.use_curing_job_P==use_curing_job_P)&
                           #(df.tauP==tauP)&
                           #(df.P==P)&
                       (df.quench_time==quench_time)&
                           (df.num_c10==num_c10)&
                           (df.cooling_method==cooling_method)&
                           (df.stop_after_percent==90)]
            split_log(df_curing,project,PROP_NAME,filter_temp=1.0,rtol=0.01,show_all=True)
            break
        plt.xlabel('Time Steps')
        plt.ylabel(PROP_NAME)
        #plt.legend(fontsize=5)
        plt.xlim(0,600)
        ##plt.ylim(1,2)
        plt.show()
```

ffd54826a80437a447973bbf73be3f68

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

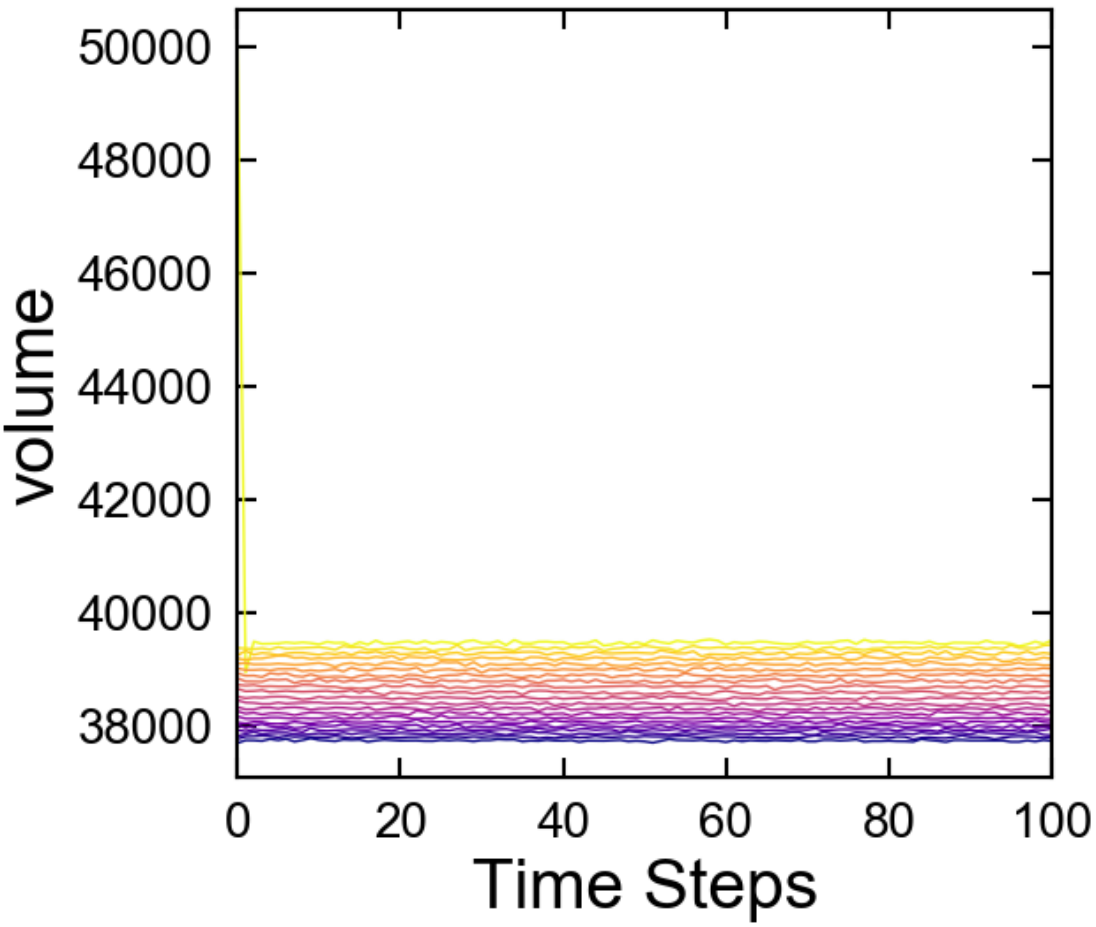


In [9]:
```
#stop_after_percents = [30.]#np.arange(10,105,15,dtype=float)
PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
plt.figure()
for sap in stop_after_percents:
    df_curing = df[(df.bond==False)&
                   #(df.tau==tau)&
                   #(df.use_curing_job_P==use_curing_job_P)&
                   #(df.tauP==tauP)&
                   #(df.P==P)&
                (df.quench_time==quench_time)&
                   (df.num_c10==num_c10)&
                   (df.cooling_method==cooling_method)&
                   (df.stop_after_percent==90)]
    split_log(df_curing,project,PROP_NAME,filter_temp=1.0,rtol=0.01,show_all=True)
    break
plt.xlabel('Time Steps')
plt.ylabel(PROP_NAME)
#plt.legend(fontsize=5)
```

```
        plt.xlim(0,100)
        ##plt.ylim(1,2)
        plt.show()
```

```
ffd54826a80437a447973bbf73be3f68
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



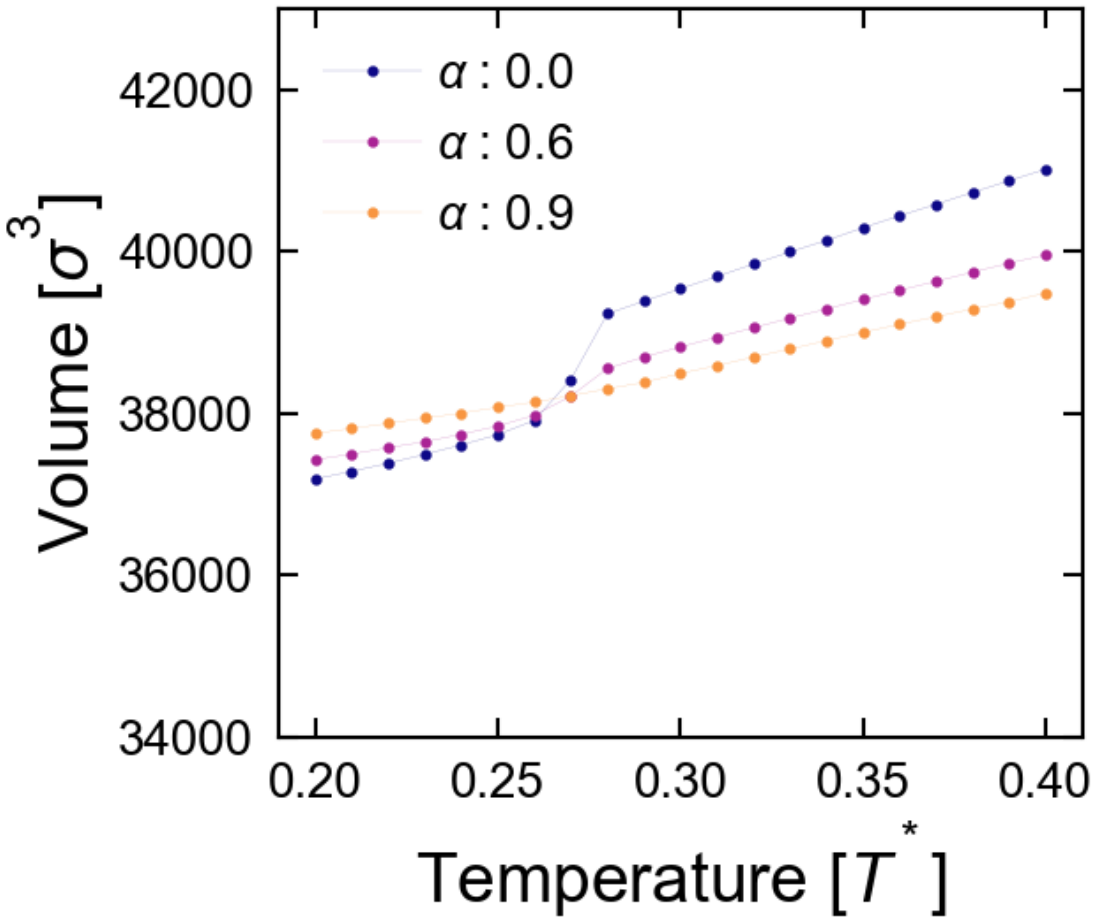

```
In [10]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()
         filter_saps = [0,60,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             df_curing = df[(df.bond==False)&
```

```
                        #(df.tau==tau)&
                        #(df.use_curing_job_P==use_curing_job_P)&
                        #(df.tauP==tauP)&
                        #(df.P==P)&
                    (df.quench_time==quench_time)&
                        (df.num_c10==num_c10)&
                        (df.cooling_method==cooling_method)&
                        (df.stop_after_percent==sap)]
    for job_id in df_curing.index:
        job = project.open_job(id=job_id)
        print(job.sp.P,job.sp.stop_after_percent)
        means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
        mean_vols = np.asarray(means)
        density = mean_vols/job.sp.n_particles
        #print(temps,means)

        if False:
            model = piecewise(temps, mean_vols)
            #print(model)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
            else:
                print('WARNING: found more or less than 2 line segments in regression!')

        Tg,Tg_prop_val = find_Tg(temps,mean_vols)
        show_transition = False
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop_val,
                        marker='*',
                        s=100,
                        color='r',
                       zorder=0)#colors[i])
        plt.plot(temps,
                 mean_vols,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha$ : {}'.format(job.sp.stop_after_percent/100),
                 linewidth=0.1)
plt.xlabel('Temperature [$T^*$]')
plt.ylabel('Volume [$\\sigma^3$]')
plt.legend(fontsize=20)
#plt.xlim(0.11,0.51)
plt.ylim(34000,43000)
```

```
        savefig(plt,
               'anneal_DGEBA_DDS_LJ',
               'A-B_annealV_qT.pdf')
        plt.show()
```

```
6.0 0.0
646df3239f44bb1705e0ff4c9a58106e
using line iftting
6.0 60.0
644d27b8f740cc6df4a628957245df0c
using line iftting
6.0 90.0
ffd54826a80437a447973bbf73be3f68
using line iftting
```

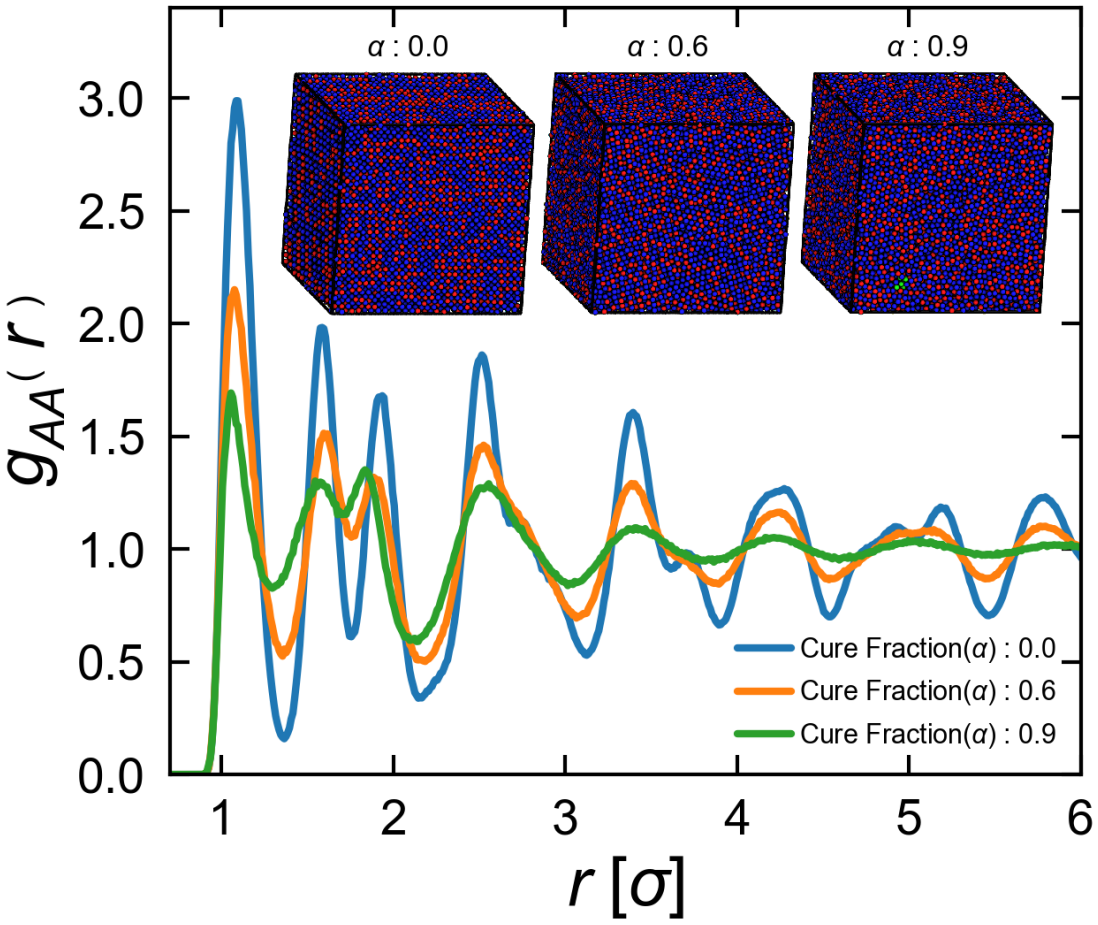
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [11]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.27, 0.64, 0.23, 0.23]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.48, 0.64, 0.23, 0.23]#max t2
ax3 = fig.add_axes([left, bottom, width, height])
ax3.axis('off')
left, bottom, width, height = [0.69, 0.64, 0.23, 0.23]#max t2
ax4 = fig.add_axes([left, bottom, width, height])
ax4.axis('off')
figure_axes=[ax2,ax3,ax4]

filter_saps = [0.,60.,90.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
qTs=[0.2]
colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
for i,sap in enumerate(filter_saps):
    #plt.figure()
    if sap not in filter_saps:
        continue
    df_filtered = df[(df.bond==False)&
                     #(df.tau==tau)&
                     #(df.use_curing_job_P==use_curing_job_P)&
                     #(df.tauP==tauP)&
                     #(df.P==P)&
                 (df.quench_time==quench_time)&
                     (df.num_c10==num_c10)&
                     (df.cooling_method==cooling_method)&
                     (df.stop_after_percent==sap)]
    df_sorted = df_filtered.sort_values('quench_T')
    for j,qT in enumerate(qTs):
        df_xstal = df_sorted[(df_sorted.quench_T==qT)]
        for jobid in df_xstal.index:
            job=project.open_job(id=jobid)
            print(job.isfile('final.hoomdxml'),job)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax1.plot(r,
                     gr,
                     label='Cure Fraction($\\alpha$) : {}'.format(sap/100))
            im = plt.imread(job.fn('final_snapshot.png'))
            figure_axes[i].set_title('$\\alpha$ : {}'.format(sap/100),fontsize=12)
            figure_axes[i].imshow(im)

#plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax1.set_xlabel(r"$r\ [\sigma]$")
ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
ax1.set_ylim(0,3.4)
ax1.set_xlim(0.7,6)
ax1.legend(fontsize=11,loc='lower right')
savefig(plt,
        'anneal_DGEBA_DDS_LJ',
        'xstal_anneal_A-B.pdf')
#plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
#plt.show()
```

```
False 646df3239f44bb1705e0ff4c9a58106e
False 644d27b8f740cc6df4a628957245df0c
False ffd54826a80437a447973bbf73be3f68
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



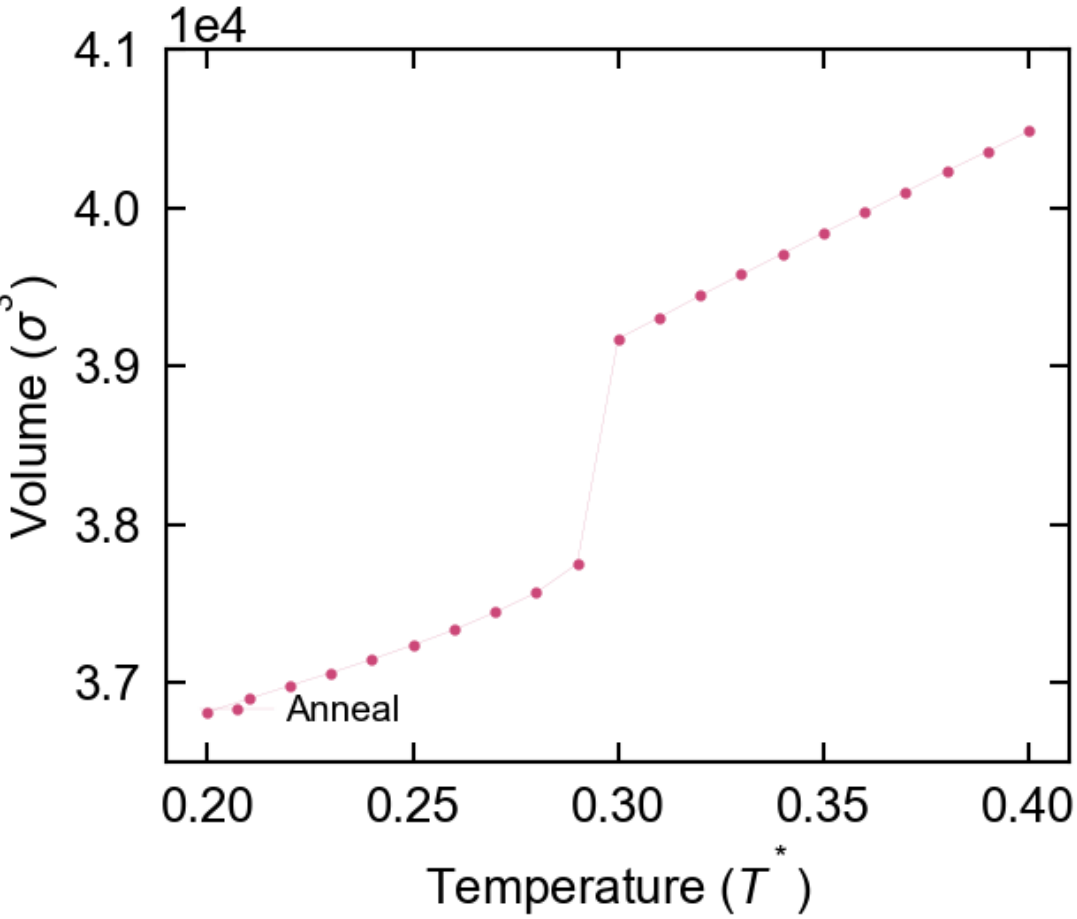

```
In [28]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()
         filter_saps = [30]
         colors = plt.cm.plasma(np.linspace(0.5,0.75,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             df_curing = df[(df.bond==False)&
                            #(df.tau==tau)&
                            #(df.use_curing_job_P==use_curing_job_P)&
                            #(df.tauP==tauP)&
                            #(df.P==P)&
                        (df.quench_time==quench_time)&
                            (df.num_c10==num_c10)&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==sap)]
```

```
    for job_id in df_curing.index:
        job = project.open_job(id=job_id)
        print(job.sp.P,job.sp.stop_after_percent)
        means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
        mean_vols = np.asarray(means)
        density = mean_vols/job.sp.n_particles
        #print(temps,means)

        if False:
            model = piecewise(temps, mean_vols)
            #print(model)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
            else:
                print('WARNING: found more or less than 2 line segments in regression!')

        Tg,Tg_prop_val = find_Tg(temps,mean_vols)
        show_transition = False
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop_val,
                        marker='*',
                        s=100,
                        color='r',
                       zorder=0)#colors[i])
        plt.plot(temps,
                 mean_vols,
                 marker='.',
                 color=colors[i],
                 label='Anneal',
                 linewidth=0.1)
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

plt.xlabel('Temperature ($T^*$)',fontsize=20)
plt.ylabel('Volume ($\\sigma^3$)',fontsize=20)
plt.legend(fontsize=15,loc='lower left')
#plt.xlim(0.11,0.51)
plt.ylim(36500,41000)
savefig(plt,
       'anneal_DGEBA_DDS_LJ',
       'A-B_annealV_qT_alpha_30.pdf')
plt.show()
```

```
6.0 30.0
```

```
90b99ca468d650a13ade4d55a27ce7e1
using line iftting
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



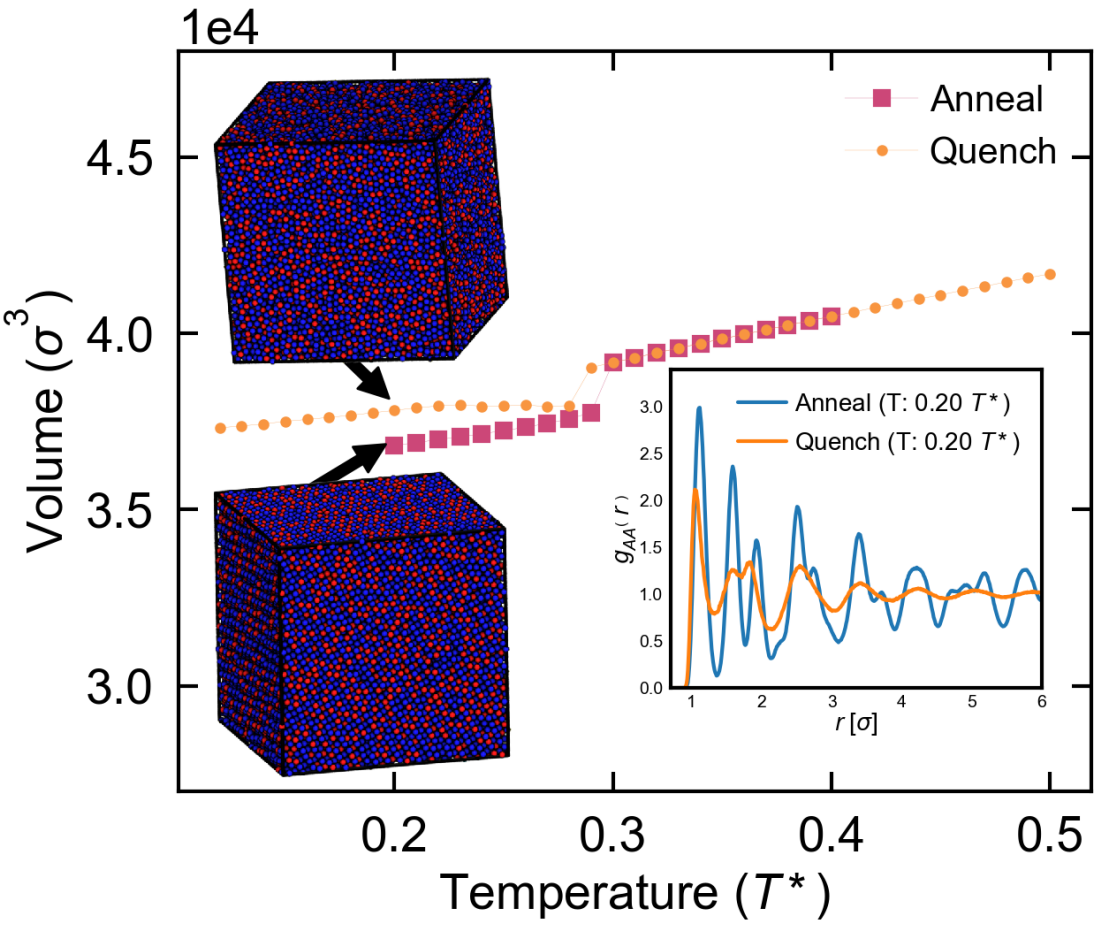
In [122]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
from scipy import ndimage
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'

fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.20, 0.20, 0.3, 0.3]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.20, 0.58, 0.3, 0.3]#max t2
ax3 = fig.add_axes([left, bottom, width, height])
```

```
ax3.axis('off')
left, bottom, width, height = [0.6, 0.29, 0.3, 0.3]#max t2
ax4 = fig.add_axes([left, bottom, width, height])
#ax4.axis('off')
figure_axes=[ax2,ax3,ax4]
#cooling_labels={"anneal":"Anneal","quench","Quench"}
colors = plt.cm.plasma(np.linspace(0.5,0.75,2))
df_curing = df[(df.bond==False)&
                        #(df.tau==tau)&
                        #(df.use_curing_job_P==use_curing_job_P)&
                        #(df.tauP==tauP)&
                        #(df.P==P)&
                    ((df.quench_time==quench_time)|(df.quench_time==5e6))&
                        (df.num_c10==num_c10)&
                        #(df.cooling_method==cooling_method)&
                        (df.stop_after_percent==30)]
for i,(cool_method,df_cooling) in enumerate(df_curing.groupby('cooling_method')):
    print(cool_method,df_cooling.quench_time.mean())
    if cool_method == 'quench':
        temps,mean_vals,val_stds = get_values_for_quenchTs(df_cooling,
                                                           project,
                                                           PROP_NAME,
mean_from_second_half=True)
        mean_vols = np.asarray(mean_vals)
        ax1.plot(temps,
                 mean_vols,
                 marker='.',
                 color=colors[i],
                 label=cool_method.title(),
                 linewidth=0.1)

        df_quench_T = df_cooling[df_cooling.quench_T==0.2]
        for job_id in df_quench_T.index:
            job = project.open_job(id=job_id)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, -85,prefilter=False)
            #figure_axes[i].set_title('T: {:.2f}
$T^*$,{}'.format(job.sp.quench_T,job.sp.cooling_method),fontsize=9)
            figure_axes[i].imshow(rotated_img)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax4.plot(r,
                     gr,
                     label='{} (T: {:.2f}
$T*$)'.format(cool_method.title(),job.sp.quench_T),
                    linewidth=1.5)
    else:
        for job_id in df_cooling.index:
            job = project.open_job(id=job_id)
            print(job.sp.P,job.sp.stop_after_percent,cool_method)
            means,stds,times,temps =
get_split_quench_job_property_mean_std(job,PROP_NAME)
            mean_vols = np.asarray(means)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, 5,prefilter=False)
            #figure_axes[i].set_title('T: {:.2f}
$T^*$,{}'.format(job.sp.quench_T,job.sp.cooling_method),fontsize=9)
            figure_axes[i].imshow(rotated_img)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
```

```
                ax4.plot(r,
                         gr,
                         label='{} (T: {:.2f}
        $T*$)'.format(cool_method.title(),job.sp.quench_T),
                        linewidth=1.5)
            ax1.plot(temps,
                     mean_vols,
                     marker='s',
                     color=colors[i],
                     label=cool_method.title(),
                     linewidth=0.1)

        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel('Temperature ($T*$)',fontsize=20)
        ax1.set_ylabel('Volume ($\\sigma^3$)',fontsize=20)
        ax1.legend(fontsize=15)#,loc='lower left')
        #plt.xlim(0.11,0.51)
        ax1.set_ylim(27000,48000)
        ax1.annotate('', xy=(0.2, 3.7e4), xytext=(0.12,
        3.4e4),arrowprops=dict(facecolor='black', shrink=0.05))
        ax1.annotate('', xy=(0.2, 3.8e4), xytext=(0.15,
        4.1e4),arrowprops=dict(facecolor='black', shrink=0.05))

        ax4.set_xlabel(r"$r\ [\sigma]$",fontsize=10,labelpad=0)
        ax4.set_ylabel(r"$g_{AA}\left(\ r\ \right)$",fontsize=10,labelpad=0)
        ax4.set_ylim(0,3.4)
        ax4.set_xlim(0.7,6)

        ax4.tick_params(axis='both', labelsize=7,length=0,pad=3)
        ax4.legend(fontsize=10,loc='upper right')
        savefig(plt,
               'anneal_DGEBA_DDS_LJ',
               'A-B_anneal_and_quench_V_qT_alpha_30.pdf')
        plt.show()
```

```
anneal 6000000.0
6.0 30.0 anneal
90b99ca468d650a13ade4d55a27ce7e1
quench 5000000.0
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

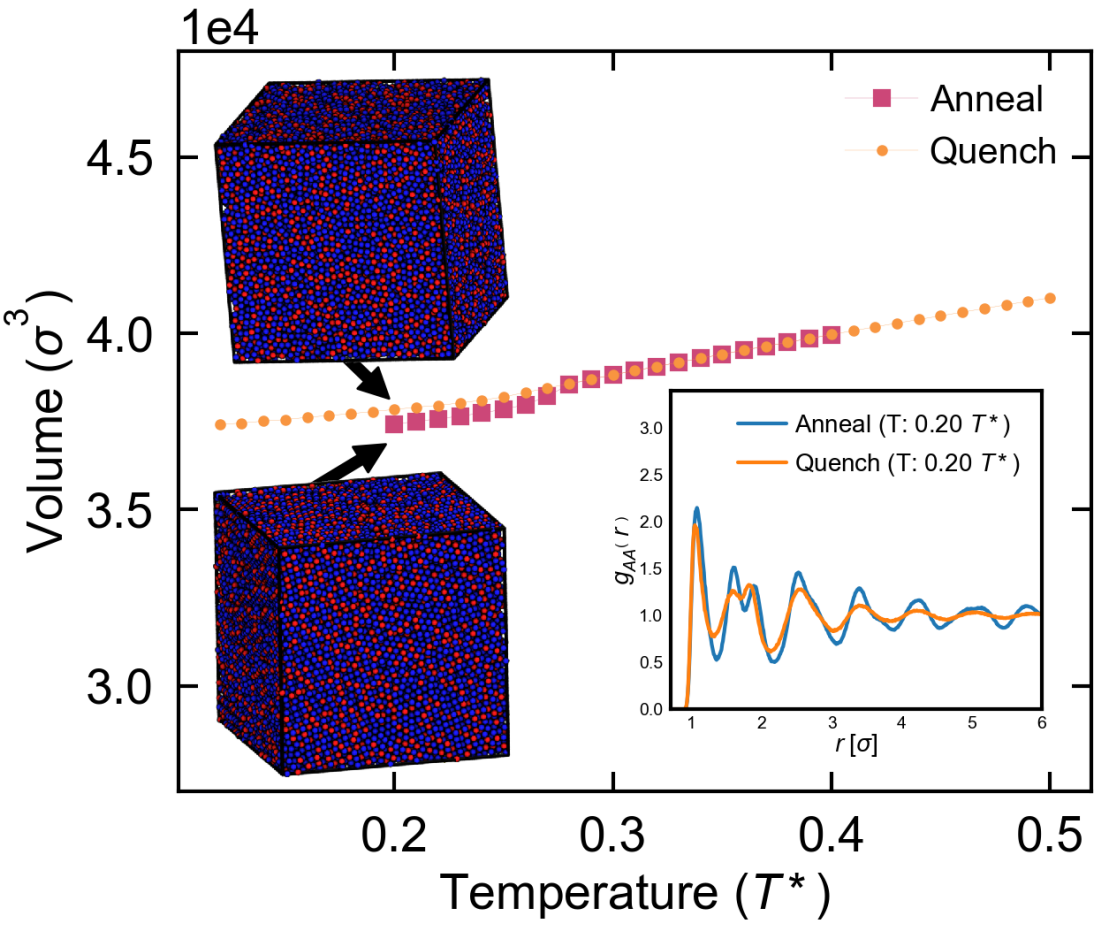



```
In [125]: from piecewise.regressor import piecewise
          #https://www.datadoghq.com/blog/engineering/piecewise-regression/
          from piecewise.plotter import plot_data_with_regression
          from scipy import ndimage
          #stop_after_percents = np.arange(10,105,15,dtype=float)
          PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'

          fig = plt.figure(dpi=200)
          ax1 = fig.add_subplot(111)
          left, bottom, width, height = [0.20, 0.20, 0.3, 0.3]#max t2
          ax2 = fig.add_axes([left, bottom, width, height])
          ax2.axis('off')
          left, bottom, width, height = [0.20, 0.58, 0.3, 0.3]#max t2
          ax3 = fig.add_axes([left, bottom, width, height])
          ax3.axis('off')
          left, bottom, width, height = [0.6, 0.27, 0.3, 0.3]#max t2
          ax4 = fig.add_axes([left, bottom, width, height])
          #ax4.axis('off')
          figure_axes=[ax2,ax3,ax4]
          #cooling_labels={"anneal":"Anneal","quench","Quench"}
          colors = plt.cm.plasma(np.linspace(0.5,0.75,2))
          df_curing = df[(df.bond==False)&
                         #(df.tau==tau)&
                         #(df.use_curing_job_P==use_curing_job_P)&
                         #(df.tauP==tauP)&
                         #(df.P==P)&
```

```
                    ((df.quench_time==quench_time)|(df.quench_time==5e6))&
                        (df.num_c10==num_c10)&
                        #(df.cooling_method==cooling_method)&
                        (df.stop_after_percent==60)]
for i,(cool_method,df_cooling) in enumerate(df_curing.groupby('cooling_method')):
    print(cool_method,df_cooling.quench_time.mean())
    if cool_method == 'quench':
        temps,mean_vals,val_stds = get_values_for_quenchTs(df_cooling,
                                                           project,
                                                           PROP_NAME,
mean_from_second_half=True)
        mean_vols = np.asarray(mean_vals)
        ax1.plot(temps,
                 mean_vols,
                 marker='.',
                 color=colors[i],
                 label=cool_method.title(),
                 linewidth=0.1)

        df_quench_T = df_cooling[df_cooling.quench_T==0.2]
        for job_id in df_quench_T.index:
            job = project.open_job(id=job_id)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, -85,prefilter=False)
            #figure_axes[i].set_title('T: {:.2f}
$T^*$,{}'.format(job.sp.quench_T,job.sp.cooling_method),fontsize=9)
            figure_axes[i].imshow(rotated_img)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax4.plot(r,
                     gr,
                     label='{} (T: {:.2f}
$T*$)'.format(cool_method.title(),job.sp.quench_T),
                    linewidth=1.5)
    else:
        for job_id in df_cooling.index:
            job = project.open_job(id=job_id)
            print(job.sp.P,job.sp.stop_after_percent,cool_method)
            means,stds,times,temps =
get_split_quench_job_property_mean_std(job,PROP_NAME)
            mean_vols = np.asarray(means)
            im = plt.imread(job.fn('final_snapshot.png'))
            rotated_img = ndimage.interpolation.rotate(im, 5,prefilter=False)
            #figure_axes[i].set_title('T: {:.2f}
$T^*$,{}'.format(job.sp.quench_T,job.sp.cooling_method),fontsize=9)
            figure_axes[i].imshow(rotated_img)
            if job.isfile('rdf_DDS_DDS.txt'):
                data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                r=data[:,0]
                gr=data[:,1]
            else:
                raise FileNotFoundError('Dont have the rdf file for',job)
            ax4.plot(r,
                     gr,
                     label='{} (T: {:.2f}
$T*$)'.format(cool_method.title(),job.sp.quench_T),
                    linewidth=1.5)
        ax1.plot(temps,
                 mean_vols,
                 marker='s',
                 color=colors[i],
                 label=cool_method.title(),
                 linewidth=0.1)
```

```
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel('Temperature ($T*$)',fontsize=20)
        ax1.set_ylabel('Volume ($\\sigma^3$)',fontsize=20)
        ax1.legend(fontsize=15)#,loc='lower left')
        #plt.xlim(0.11,0.51)
        ax1.set_ylim(27000,48000)
        ax1.annotate('', xy=(0.2, 3.7e4), xytext=(0.12,
        3.4e4),arrowprops=dict(facecolor='black', shrink=0.05))
        ax1.annotate('', xy=(0.2, 3.8e4), xytext=(0.15,
        4.1e4),arrowprops=dict(facecolor='black', shrink=0.05))

        ax4.set_xlabel(r"$r\ [\sigma]$",fontsize=10,labelpad=0)
        ax4.set_ylabel(r"$g_{AA}\left(\ r\ \right)$",fontsize=10,labelpad=0)
        ax4.set_ylim(0,3.4)
        ax4.set_xlim(0.7,6)

        ax4.tick_params(axis='both', labelsize=7,length=0,pad=3)
        ax4.legend(fontsize=10,loc='upper right')
        savefig(plt,
               'anneal_DGEBA_DDS_LJ',
               'A-B_anneal_and_quench_V_qT_alpha_60.pdf')
        plt.show()
```

```
anneal 6000000.0
6.0 60.0 anneal
644d27b8f740cc6df4a628957245df0c
quench 5000000.0


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

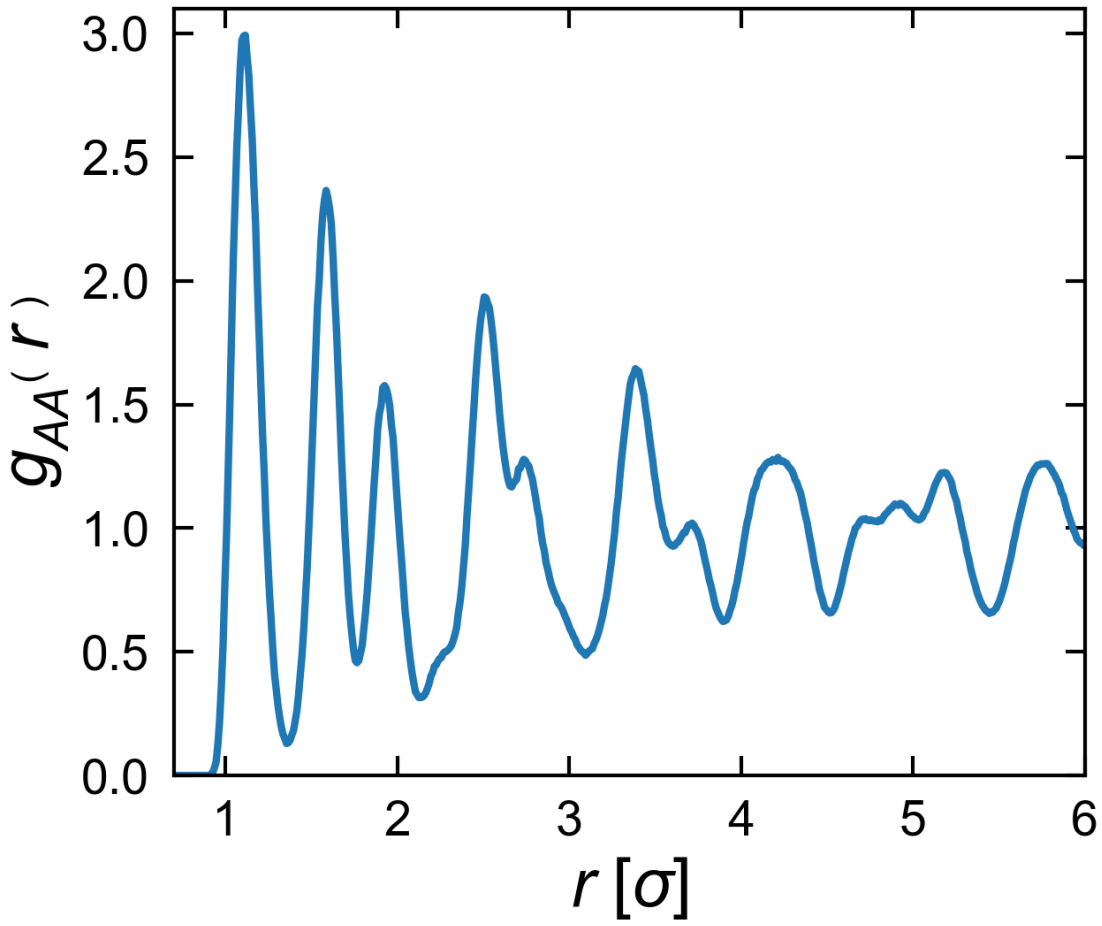



```
In [33]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.27, 0.64, 0.23, 0.23]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.48, 0.64, 0.23, 0.23]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.69, 0.64, 0.23, 0.23]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')
         figure_axes=[ax2,ax3,ax4]

         filter_saps = [30]#[0.,60.,90.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         qTs=[0.2]
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.bond==False)&
                              #(df.tau==tau)&
```

```
                            #(df.use_curing_job_P==use_curing_job_P)&
                            #(df.tauP==tauP)&
                            #(df.P==P)&
                        (df.quench_time==quench_time)&
                            (df.num_c10==num_c10)&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==sap)]
            df_sorted = df_filtered.sort_values('quench_T')
            for j,qT in enumerate(qTs):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    print(job.isfile('final.hoomdxml'),job)
                    if job.isfile('rdf_DDS_DDS.txt'):
                        data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr)#,label='Cure Fraction($\\alpha$) : {}'.format(sap/100))
                    #im = plt.imread(job.fn('final_snapshot.png'))
                    #figure_axes[i].set_title('$\\alpha$ : {}'.format(sap/100),fontsize=12)
                    #figure_axes[i].imshow(im)

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
        ax1.set_ylim(0,3.1)
        ax1.set_xlim(0.7,6)
        ax1.legend(fontsize=11,loc='lower right')
        savefig(plt,
                'anneal_DGEBA_DDS_LJ',
                'xstal_anneal_A-B_alpha_30.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False 90b99ca468d650a13ade4d55a27ce7e1
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



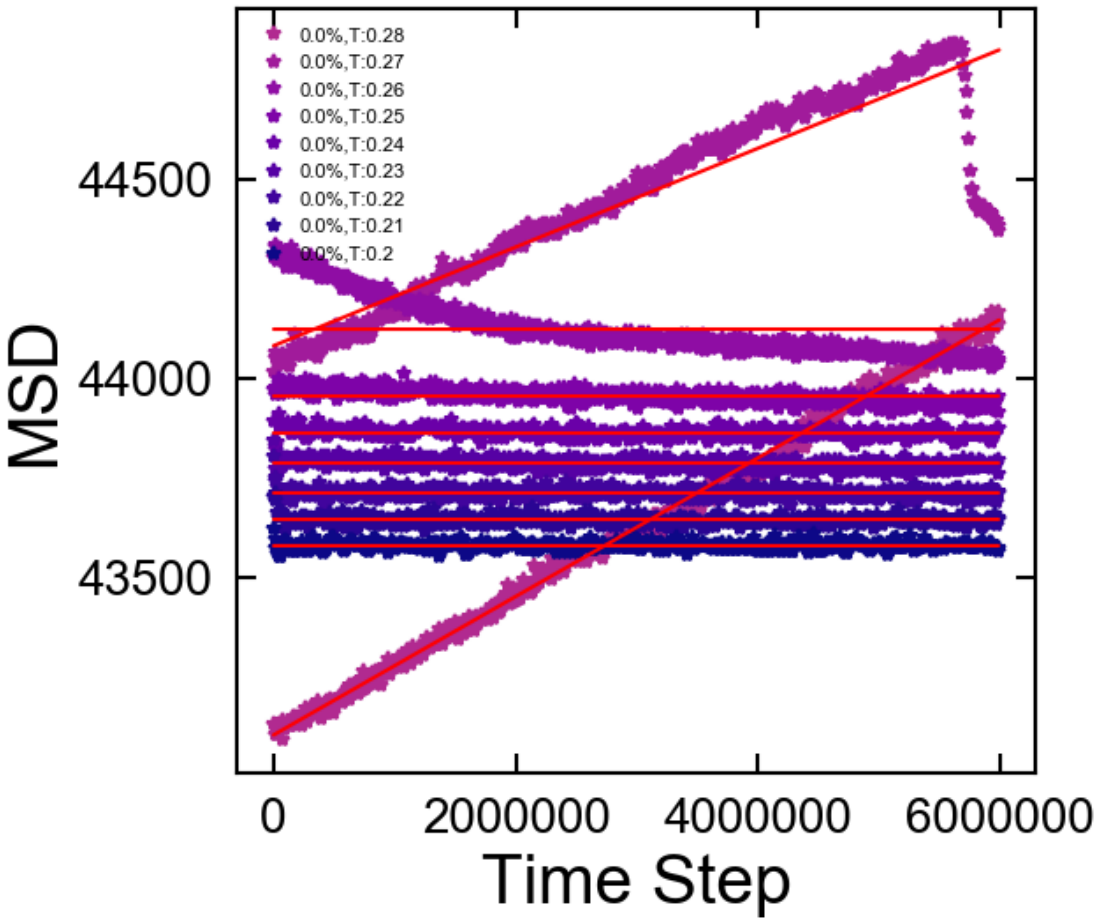

```
In [12]: def get_split_quench_job_msd(job,prop_name):
             times = []
             prop_vals = []
             qTs=[]
             if job.isfile('msd.log'):
                 log_path = job.fn('msd.log')
                 data = np.genfromtxt(log_path, names=True)
                 PROP_NAME =prop_name
                 prop_values = data[PROP_NAME]#'pair_lj_energy']
                 time_steps = data['timestep']
                 len_prof = len(job.sp.quench_temp_prof)
                 for i in range(0,len_prof,2):
                     current_point = job.sp.quench_temp_prof[i]
                     next_point = job.sp.quench_temp_prof[i+1]
                     start_time = current_point[0]
                     end_time = next_point[0]
                     if current_point[1]!=next_point[1]:
                         print('WARNING! Detected a non isothermal step')
                     target_T = current_point[1]
                     #print(time_steps)
                     #print(start_time,end_time)
                     indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
                     start_index = indices[0][0]
                     end_index = indices[0][-1]
                     sliced_ts = time_steps[start_index:end_index+1]
                     sliced_prop_vals = prop_values[start_index:end_index+1]
```

```
                #sliced_pe = pe[start_index:end_index+1]
                #mean,std = get_mean_and_std(job,sliced_ts,sliced_prop_vals,sliced_pe)
                #means.append(mean)
                #stds.append(std)
                times.append(sliced_ts)
                prop_vals.append(sliced_prop_vals)
                qTs.append(target_T)
        return times,prop_vals,qTs
```

```
In [13]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()

         for i,sap in enumerate(stop_after_percents):
             filter_saps=[0.0]
             if sap not in filter_saps:
                 continue
             df_curing = df[(df.bond==False)&
                            #(df.tau==tau)&
                            #(df.use_curing_job_P==use_curing_job_P)&
                            #(df.tauP==tauP)&
                            #(df.quench_T<0.25)&
                            (df.quench_time==quench_time)&
                            (df.num_c10==num_c10)&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==sap)]
             for job_id in df_curing.index:
                 job = project.open_job(id=job_id)
                 #print(job.sp.P,job.sp.stop_after_percent)
                 times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
                 colors = plt.cm.plasma(np.linspace(1,0,len(msds)))
                 for j,msd in enumerate(msds):
                     #print(len(msd))
                     start_index = int(len(times[j])*0.0)
                     time=times[j][start_index:]
                     print('using the last {} timesteps of MSD'.format(len(time)))
                     normalized_time = times[0][:len(time)]
                     eq_msd = msd[start_index:]
                     quench_T = qTs[j]
                     if quench_T<0.28:
                         if True:

                             popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                                    normalized_time,
                                                    eq_msd,
                                                    p0=[1.,0.0],
                                                    bounds=([0.0,0.0],[np.infty,np.infty]))

                             # determine the slop using linear regression
                             par = np.polyfit(normalized_time, eq_msd, 1, full=True)
                             drdt_A = popt[0]#par[0][0]#0-slope, 1-intercept
                             x=normalized_time
                             if False:
                                 y = popt[0]*x**(popt[1])
                                 #y = x**(popt[0])
                             else:
                                 y=popt[0]*x+popt[1]
                                 #y=par[0][0]*x+par[0][1]
                             #calculate the diffusion coefficient


                             plt.figure(0)

                             if True:
```

```
                        #print(x,y)
                        plt.plot(x,
                                 y,
                                 color='r',#colors[i],
                                 linewidth=1.5,
                                 zorder=1)#,
#label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))

                #Tg,Tg_prop_val = find_Tg(temps,mean_vols,sap)
                show_transition = False
                if show_transition:
                    #plt.axvline(x=Tg,
                    #            linewidth=1.0,
                    #            color=colors[i])
                    plt.scatter(Tg,
                                Tg_prop_val,
                                marker='*',
                                s=100,
                                color='r',
                               zorder=0)#colors[i])
                plt.plot(normalized_time,
                         eq_msd,
                         marker='*',
                         color=colors[j],
label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                         zorder=0,
                         linewidth=0.0)
plt.xlabel('Time Step')
plt.ylabel('MSD')
#plt.xscale('log')
#plt.yscale('log')
plt.legend(fontsize=8)
#plt.xlim(0.11,0.51)
#plt.ylim(34000,43000)
plt.show()
```

```
using the last 1201 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD
using the last 600 timesteps of MSD


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
<matplotlib.figure.Figure at 0x116034a20>
```

```
In [14]: quench_time
```

```
Out[14]: 6000000.0
```

```
In [15]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()
         colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
         Tgs=[]
         cure_percents = []
         for i,sap in enumerate(stop_after_percents):
             filter_saps=[0.0,30,60,90]
             if sap not in filter_saps:
                 continue
             Ts=[]
             Ds=[]
             df_curing = df[(df.bond==False)&
```

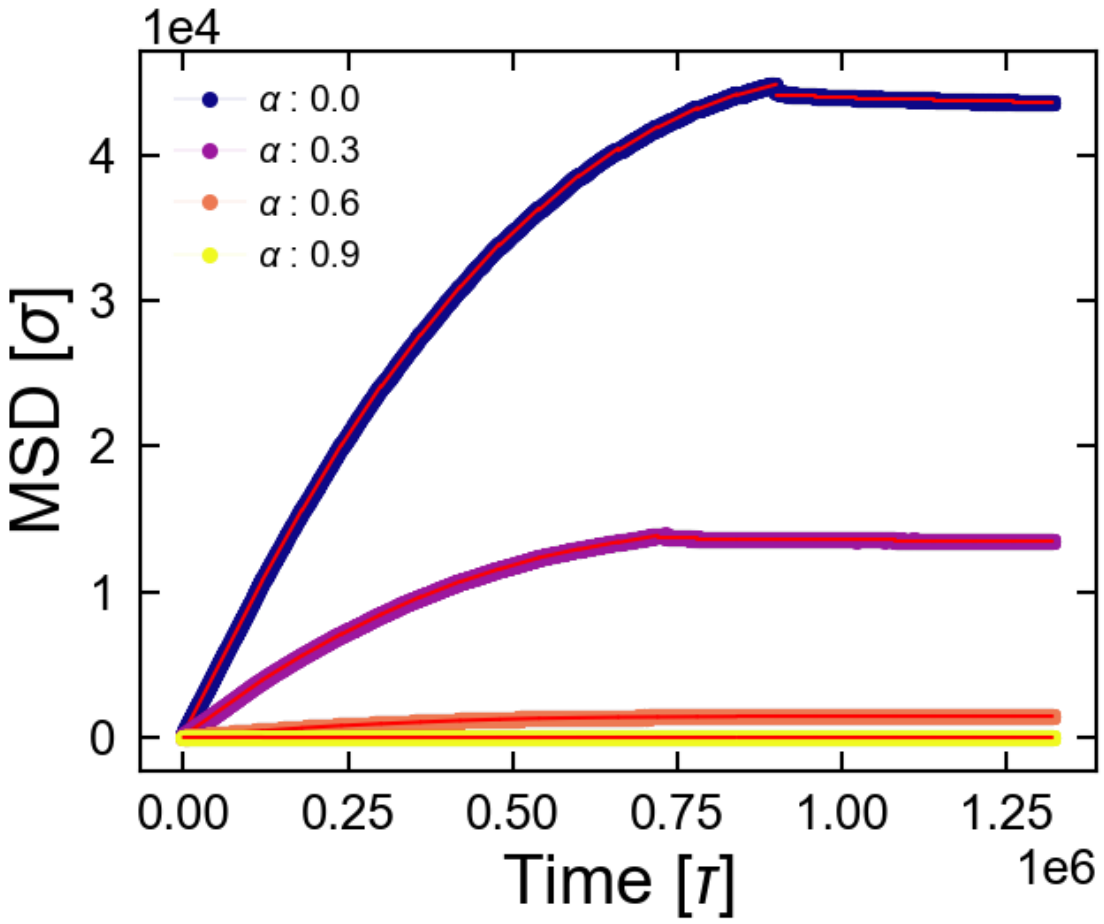
```
                (df.quench_time==quench_time)&
                (df.num_c10==num_c10)&
                (df.cooling_method==cooling_method)&
                (df.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.values[0]
cure_percents.append(cure_percent)
for job_id in df_curing.index:
    job = project.open_job(id=job_id)
    #print(job.sp.P,job.sp.stop_after_percent)
    log_path = job.fn('msd.log')
    data = np.genfromtxt(log_path, names=True)
    prop_values = data[PROP_NAME]#'pair_lj_energy']
    all_time_steps = data['timestep']
    if True:
        plt.figure(0)
        plt.plot(all_time_steps*job.sp.md_dt,#times[0][0:len(time)],
                 prop_values,
                 marker='o',
                 markersize=5,
                 color=colors[i],
                 label='$\\alpha$ : {}'.format(job.sp.stop_after_percent/100),
                 zorder=0,
                 linewidth=0.1)
        plt.ticklabel_format(axis='both', style='sci', scilimits=(-2,2))
    times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
    for j,msd in enumerate(msds):
        start_index = int(len(times[j])*0.0)
        time=times[j]*job.sp.md_dt
        quench_T = qTs[j]
        eq_msd = msd[start_index:]
        eq_time = time[start_index:]
        #print(quench_T)
        # determine the slop using linear regression
        fit_method='curve_fit'#,'curve_fit'#'power_law','poly_fit'
        if fit_method=='curve_fit':
            norm_eq_time = (eq_time-eq_time[0])
            #print(norm_eq_time,eq_msd)
            popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                   eq_time,
                                   eq_msd,
                                   p0=[1.,0.0],
                                   bounds=([0.0,0.0],[np.infty,np.infty]))
            drdt_A = popt[0]
            m=popt[0]
            b=popt[1]
        elif fit_method=='poly_fit':
            par = np.polyfit(time, msd, 1, full=True)
            drdt_A = par[0][0]#0-slope, 1-intercept
            m=par[0][0]
            b=par[0][1]
        elif fit_method=='power_law':
            popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                   time,
                                   msd,
                                   p0=[0.2,1.0],
                                   #p0=[1.0],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0],[1.0,4.0]))
            raise NotImplementedError('Diffusivity not determined')

        x=time
        y=m*x+b

        #calculate the diffusion coefficient
        dimensions=3
        D_A = drdt_A/(2*dimensions)
```

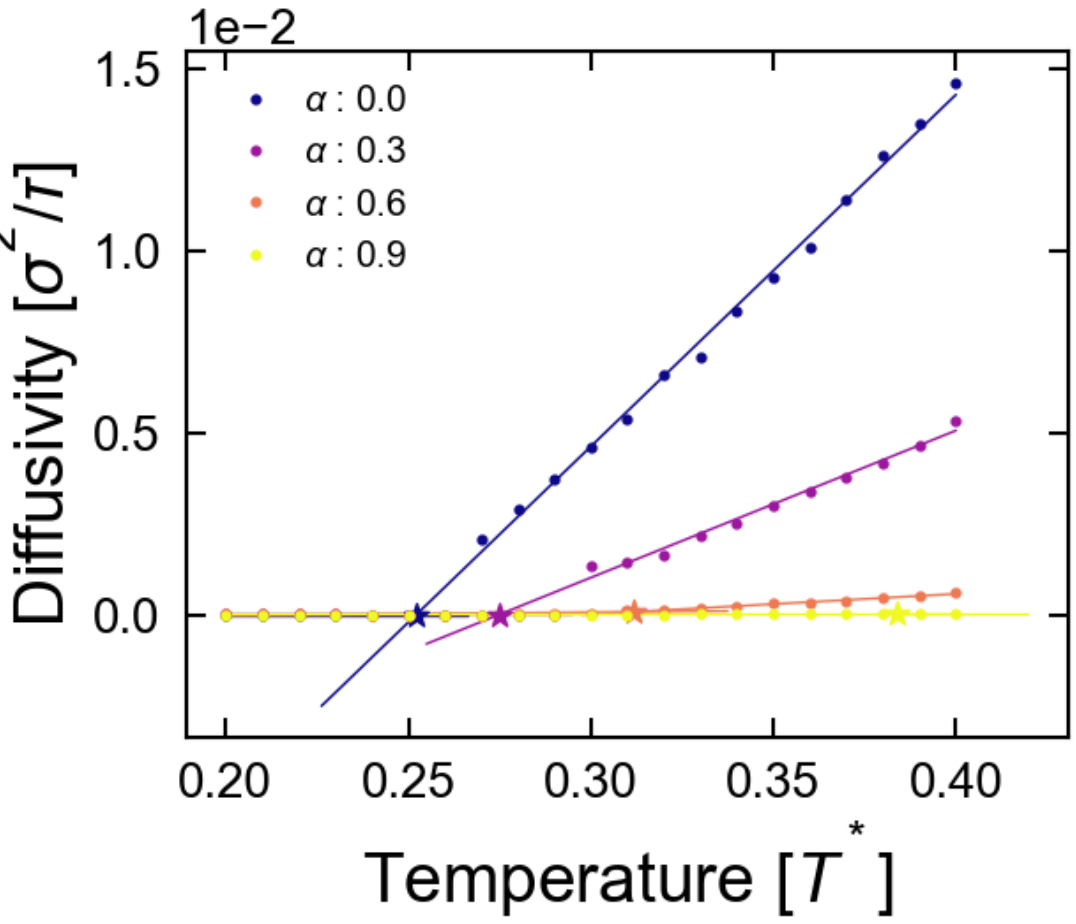
```
            Ts.append(quench_T)
            Ds.append(D_A)
            plt.figure(0)

            if False:
                plt.plot(time,#times[0][0:len(time)],
                         msd,
                         marker='o',
                         markersize=5,
                         color=colors[i],
#label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                         linewidth=0.1)
            if True:
                #print(x,y)
                plt.plot(x,
                         y,
                         color='r',#colors[i],
                         zorder=0,
                         linewidth=1.5)#,
                         #label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))
            #plt.xlim(0.835e8,0.845e8)

            #break
    plt.xlabel('Time [$\\tau$]')
    plt.ylabel('MSD [$\\sigma$]')
    #plt.xlim(87610000.0, 90000000.0)
    #plt.ylim(44000,45000)
    #plt.xscale('log')
    #plt.yscale('log')
    plt.legend(fontsize=15)
    plt.figure(1)
    #print(Ts,Ds)
    Ts=np.asarray(Ts)
    Ds=np.asarray(Ds)
    #Ds[Ds<0]=0
    plt.plot(Ts,
             Ds,
             marker='.',
             color=colors[i],
             linewidth=0,
            label='$\\alpha$ : {}'.format(sap/100))
    D_fit_method='line_fit'#'VLF'#'line_fit','power_law'
    if D_fit_method == 'VLF':
        popt, pcov = curve_fit(lambda t,x1: (t)**x1 ,
                               time,
                               msd,
                               p0=[1.0],
                               #p0=[1.0],
                               #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                               #bounds=([0],[4.0]))
                               maxfev=2000000,
                               bounds=([-1.0],[4.0]))
        print('popt',popt)
        x1=popt[0]
        xs=Ts
        ys=(xs)**x1
        To=ys**(1/x1)
        print('To',To)
        plt.plot(xs,
                 ys,
                 color=colors[i],
                 zorder=0,
                 linewidth=10)
        #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
        Tgs.append(To)
    elif D_fit_method == 'power_law':
        popt, pcov = curve_fit(lambda t,Do,To,B: Do*np.exp(B/(To-t)) ,
                               Ts,
```

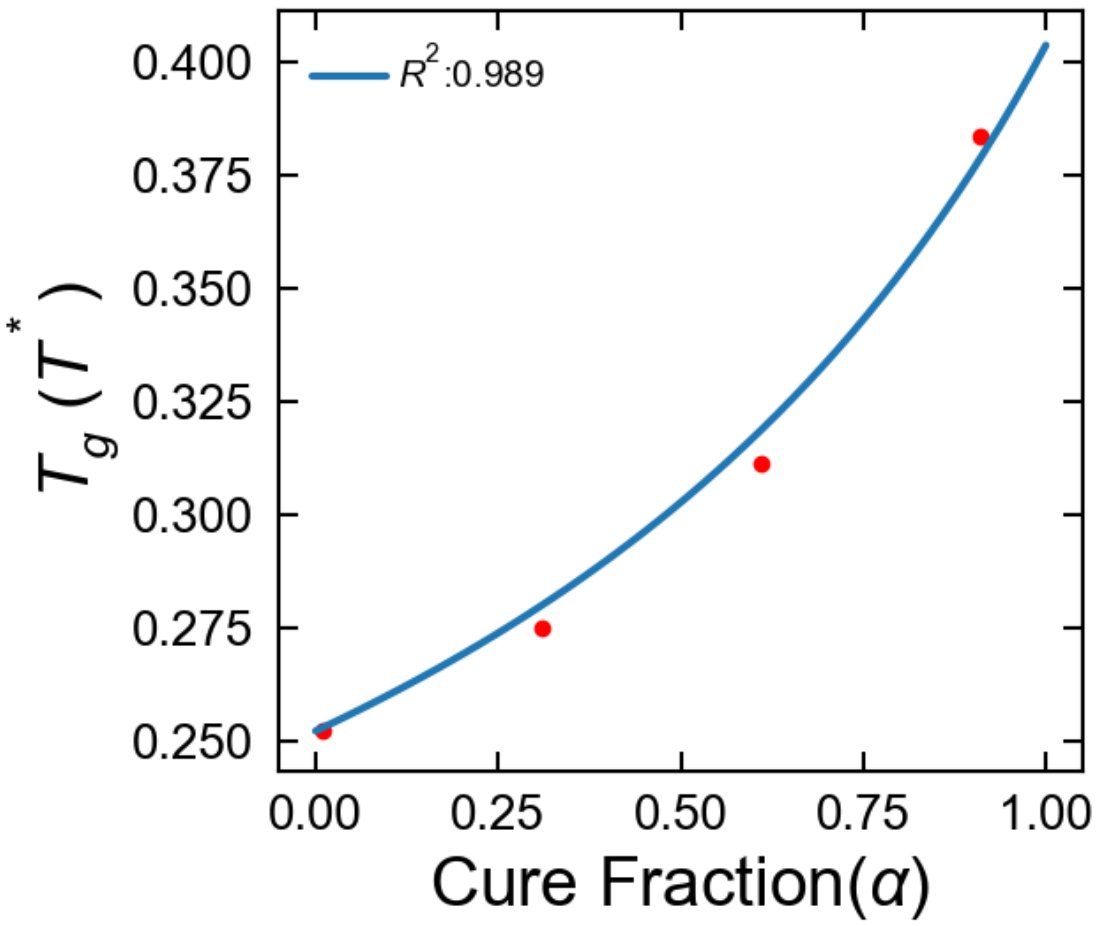
```
                                Ds,
                                p0=[0.1,1.0,1.0],
                                #p0=[1.0],
                                #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                #bounds=([0],[4.0]))
                                maxfev=2000000,
                                bounds=([0.0,0.0,0.0],[np.infty,4.0,np.infty]))
            print('popt',popt)
            Do=popt[0]
            To=popt[1]
            B=popt[2]
            xs=Ts
            ys=Do*np.exp(B/(To-xs))
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=10)
            #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            Tgs.append(To)
        elif D_fit_method == 'line_fit':
            model = piecewise(Ts, Ds)
            #print('len(model.segments)',len(model.segments))
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            Tgs.append(Tg)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=100,
                    color=colors[i],
                   zorder=1)#colors[i])
        plt.xlabel('Temperature [$T^*$]')
        plt.ylabel('Diffusivity [$\\sigma^2/\\tau$]')
        #plt.xscale('log')
        #plt.yscale('log')
        #plt.ylim(-1e-8,1e-8)
        plt.ticklabel_format(axis='both', style='sci', scilimits=(-2,2))
        plt.legend(fontsize=15)
    plt.figure(0)
    savefig(plt,
            'anneal_DGEBA_DDS_LJ',
            'msd.pdf')
    plt.figure(1)
    savefig(plt,
```

```
            'anneal_DGEBA_DDS_LJ',
            'D_vs_qT.pdf')

        plt.figure(2)
        cure_percents = np.asarray(cure_percents)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction($\\alpha$)')
        plt.ylabel('$T_g\ (T^*)$')
        plt.legend(fontsize=15)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        np.savetxt('DGEBA_DDS_Tg_anneal.txt',np.transpose(Tg_data))
        #plt.ylim(34000,43000)
        savefig(plt,
                'anneal_DGEBA_DDS_LJ',
                'dibeneditto.pdf')
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
T1 0.403859571448 lambda 0.5
[  1.00047994  31.00138092  61.0007782   91.00018311]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e4
α : 0.0
α : 0.3
α : 0.6
α : 0.9
MSD [σ]
4
3
2
1
0
0.00
0.25
0.50
0.75
1.00
1.25
1e6
Time [τ]


1e−2
1.5
1.0
0.5
0.0
Diffusivity [σ^2/τ]
α : 0.0
α : 0.3
α : 0.6
α : 0.9
0.20
0.25
0.30
0.35
0.40
Temperature [T^*]


R^2:0.989
0.400
0.375
0.350
0.325
0.300
0.275
0.250
T_g^*
0.00
0.25
0.50
0.75
1.00
Cure Fraction(α)

In [1]:
```
from common import *
import numpy as np
```

1 Parameters for DGEBA/DDS/PES (DPD)

In [2]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA_DDS_PES/'

tau=1
tauP=10
num_c10=0
kT = 1.7
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='DPD'
activation_energy=2.0
density=3.0
quench_time=5e6#1e6 #5e6
P = 23#[4.5, 6, 8]
stop_after_percents = np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.,40.,50.]#[60.,70.,
80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
```

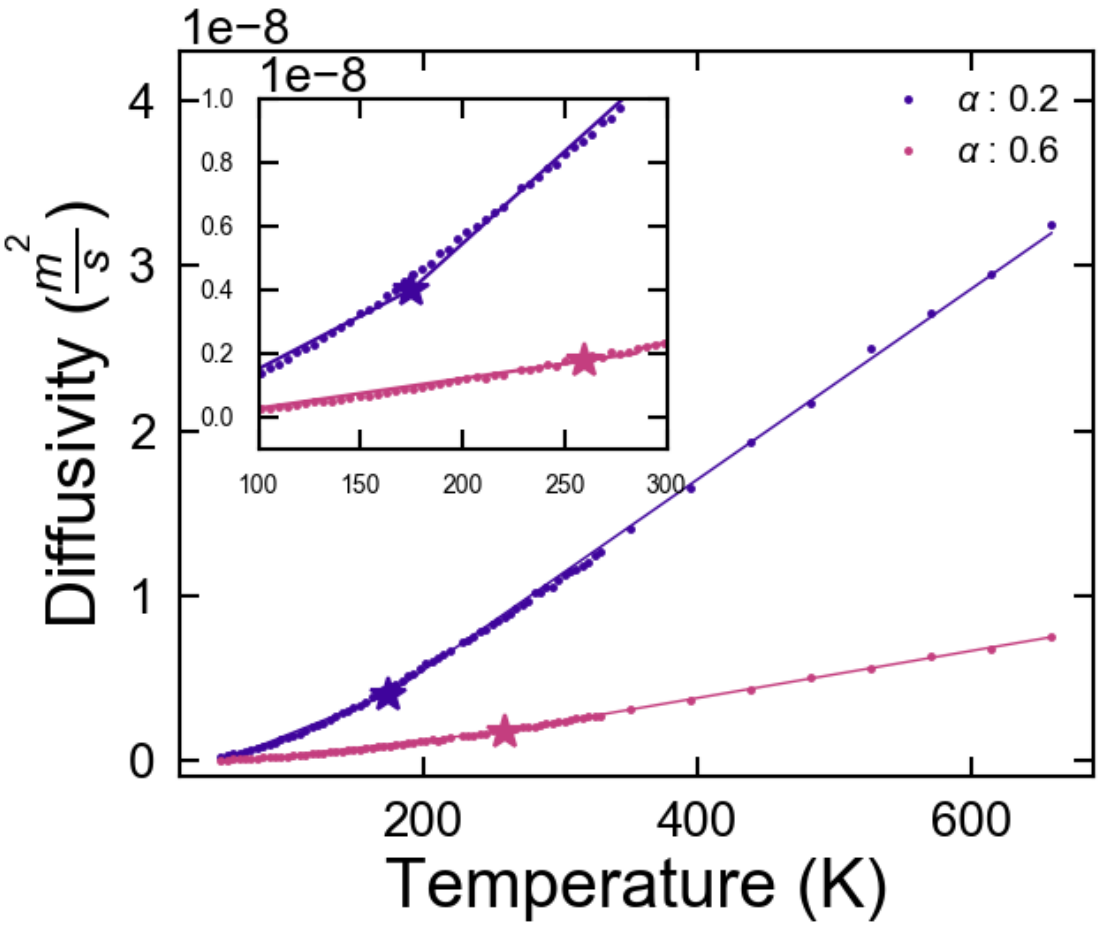
In [3]:
```
#data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_kestrel/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_fry/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_glass_transition/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_step_quench/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Tg/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [20]:
```
import matplotlib.pyplot as plt
 import matplotlib
```

```
%matplotlib inline

fig = plt.figure()
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.265, 0.515, 0.33, 0.33]
ax2 = fig.add_axes([left, bottom, width, height])
ax2.tick_params(labelsize=10)
cure_fractions_to_plot=[0.2,0.6]
PROP_NAME ='aparticles'
df_filtered = df[(df.integrator==integrator) &
                 (df.kT==kT) &
                 (df.quench_T<=1.5)&
                 #(df.quench_T>=0.1)&
                 #(df.quench_time ==quench_time)&
                 #(df.quench_method=='fine_quench')&
                 (df.tauP == tauP)&
                 (df.n_particles==N)&
                 (df.density==density)&
                 (df.activation_energy==activation_energy)&
                 (df.P==P)&
                 (df.cooling_method==cooling_method)&
                 (df.stop_after_percent<100)&
                 (df.stop_after_percent>0)&
                 #((df.stop_after_percent==20)|
                  #(df.stop_after_percent==50)|
                 #(df.stop_after_percent==60))&
                 (df.pot==pot)]

df_sorted=df_filtered.sort_values('quench_T')

Tgs=[]
cure_percents=[]
num_saps = len(df_sorted.groupby('stop_after_percent'))
colors = plt.cm.plasma(np.linspace(0,0.75,num_saps))
for i,(key,df_grp) in enumerate(df_sorted.groupby('stop_after_percent')):
    Ds=[]
    Ts=[]

    for jobid in df_grp.index:
        job=project.open_job(id=jobid)
        if job.isfile('msd.log'):
            log_path = job.fn('msd.log')
            data = np.genfromtxt(log_path, names=True)
            distance_unit=1.06e-9#m
            time_unit=9.29e-12#s
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            all_time_steps = data['timestep']*job.sp.md_dt
            D,m,b,r2=Calc_Diffusivity(all_time_steps,prop_values)
            Ds.append(D*(distance_unit**2)/time_unit)
            Ts.append(job.sp.quench_T*job.sp.calibrationT)
    Ts=np.asarray(Ts)
    Ds=np.asarray(Ds)
    Tg,Tg_prop,xs1,ys1,xs2,ys2=Fit_Diffusivity1(Ts,
                                                Ds,
                                                ver=1,
                                                 min_D=0,
                                                method='intersection')
    Tgs.append(Tg)
    cure_percents.append(df_grp.cure_percent.values[0])
    if key/100 in cure_fractions_to_plot:
        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 linewidth=0.0,
                 markersize=4,
                 zorder=0,
                 color=colors[i],
                label='$\\alpha$ : {}'.format(round(key/100,2)))
```

```
            ax2.plot(Ts,
                     Ds,
                     marker='.',
                     markersize=4,
                     linewidth=0.0,
                     zorder=0,
                     color=colors[i],
                    label='$\\alpha$ : {}'.format(round(key/100,2)))
            if True:
                ax1.plot(xs1,ys1,linewidth=1.0,color=colors[i],zorder=0)
                ax1.plot(xs2,ys2,linewidth=1.0,color=colors[i],zorder=0)
                ax1.plot(Tg,
                         Tg_prop,
                         color=colors[i],
                         #markerfacecolor='w',
                         markersize=15.0,
                         marker='*',
                         zorder=1,
                         linewidth=0.0)
                ax2.plot(xs1,ys1,linewidth=1.5,color=colors[i],zorder=0)
                ax2.plot(xs2,ys2,linewidth=1.5,color=colors[i],zorder=0)
                ax2.plot(Tg,
                         Tg_prop,
                         color=colors[i],
                         #markerfacecolor='w',
                         markersize=15.0,
                         marker='*',
                         zorder=1,
                         linewidth=0.0)
        ax1.legend(fontsize=15,loc='upper right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax1.set_xlabel('Temperature (K)')
        ax1.set_ylabel('Diffusivity ($\\frac{m^2}{s}$)')
        ax2.set_xlim(100,300)
        ax2.set_ylim(-1e-9,1e-8)
        ax1.set_ylim(-1e-9,4.3e-8)
        #ax1.set_ylim()
        #py.plot_mpl(fig)
        savefig(plt,
                'Tg_DPD',
               'piecewise_regression.pdf')
        plt.show()
```

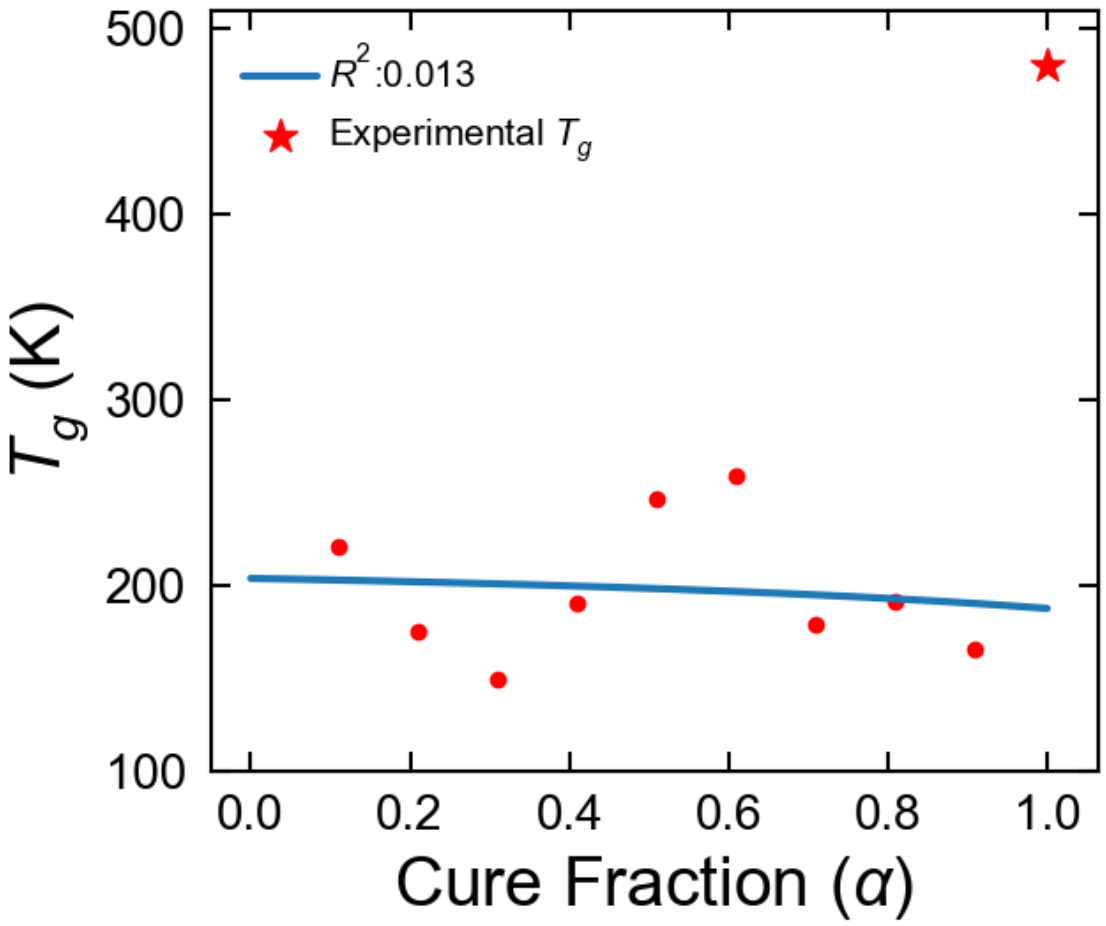
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [14]: cure_percents = np.asarray(cure_percents)
         #Tgs=[0.65,0.75,0.9,2.1]
         fig, ax1 = plt.subplots()
         #ax2=ax1.twinx()
         Tgs = np.asarray(Tgs)
         #Tgs_tangent = np.asarray(Tgs_tangent)
         print(Tgs)
         Tg_data = np.asarray([cure_percents/100.,Tgs])
         #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
         cure_percents_ss = cure_percents#[:-1]
         Tgs_ss = Tgs#[:-1]
         print(cure_percents_ss)
         print(Tgs_ss)
         try:
             R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                                 Tgs_ss,
                                                                 T1=None,
                                                                 T0=None)
             print('T1',T1,'lambda',inter_parm)
             #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
             #print(cure_percents_ss)
             alphas = np.linspace(0,1)
             fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
             ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
         except:
             print('Exception caught')
```

```
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax1.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental $T_g$')
        #ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.1
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 0.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax2.set_ylim(ex_low_lim,ex_up_lim)
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        ax1.set_ylim(100,510)
        show_roomT=False
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g$ (K)')
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'Tg_DPD',
                'piecewise_regression_dibeneditto.pdf')
```

```
[ 220.80058954  175.16073694  149.06432993  190.374292    246.3846229
  259.42598998  178.4563685   191.43419009  165.49618494]
[ 11.  21.  31.  41.  51.  61.  71.  81.  91.]
[ 220.80058954  175.16073694  149.06432993  190.374292    246.3846229
  259.42598998  178.4563685   191.43419009  165.49618494]
T1 187.396543005 lambda 0.5
300 K in T*: 117.122839378


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [8]: import numpy as np
        import math
        from scipy.optimize import curve_fit
        import matplotlib.pyplot as plt
        import matplotlib
        %matplotlib inline
        from common import *

        def H0(x,x0,c):
                return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

        def hyper(x,x0,y0,a,b,c):
            #a=0.0
            return y0+a*(x-x0)+b*H0(x,x0,c)

        def Pt(x,x0,c):
            return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

        def slope_of_tangent(x,x0,a,b,c):
            #a=0.0
            return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

        def get_tangent(x,x0,y0,a,b,c):
            m=slope_of_tangent(x,x0,a,b,c)
            y=hyper(x,x0,y0,a,b,c)
            b=y-(m*x)
```

```
        return m,b


    def custom(x,a,b,p):
        return (a*x**p) / (b + x**(p-1))

    def slope_of_tangent_custom(x,a,b,p):
        return a*x**p*(b*p*x+x**p)/(b*x+x**p)**2

    def get_tangent_custom(x,a,b,p):
        m=slope_of_tangent_custom(x,a,b,p)
        y=custom(x,a,b,p)
        b=y-(m*x)
        return m,b
```

In [21]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

fig = plt.figure()
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.265, 0.515, 0.33, 0.33]
ax2 = fig.add_axes([left, bottom, width, height])
ax2.tick_params(labelsize=10)

#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0,10.,20.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
cure_fractions_to_plot=[0.2,0.6]
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered = df[(df.integrator==integrator) &
                 (df.kT==kT) &
                 #(df.quench_T<=0.75)&
                 #(df.quench_T>=0.1)&
                 #(df.quench_time ==quench_time)&
                 #(df.quench_method=='fine_quench')&
                 (df.tauP == tauP)&
                 (df.n_particles==N)&
                 (df.density==density)&
                 (df.activation_energy==activation_energy)&
                 (df.P==P)&
                 (df.cooling_method==cooling_method)&
                 (df.stop_after_percent>10)&
                 (df.stop_after_percent<90)&
                 #((df.stop_after_percent==20)|
                 # (df.stop_after_percent==60))&
                 (df.pot==pot)]
df_sorted=df_filtered.sort_values('quench_T')
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('stop_after_percent'))))
calibrationT=df_sorted.calibrationT.mean()
for i,(sap,df_curing) in enumerate(df_sorted.groupby('stop_after_percent')):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    cure_percent = df_curing.cure_percent.mean()
    cure_percents.append(cure_percent)
    Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=None)
    #print(Ds)
    Ts=Ts#*calibrationT
```

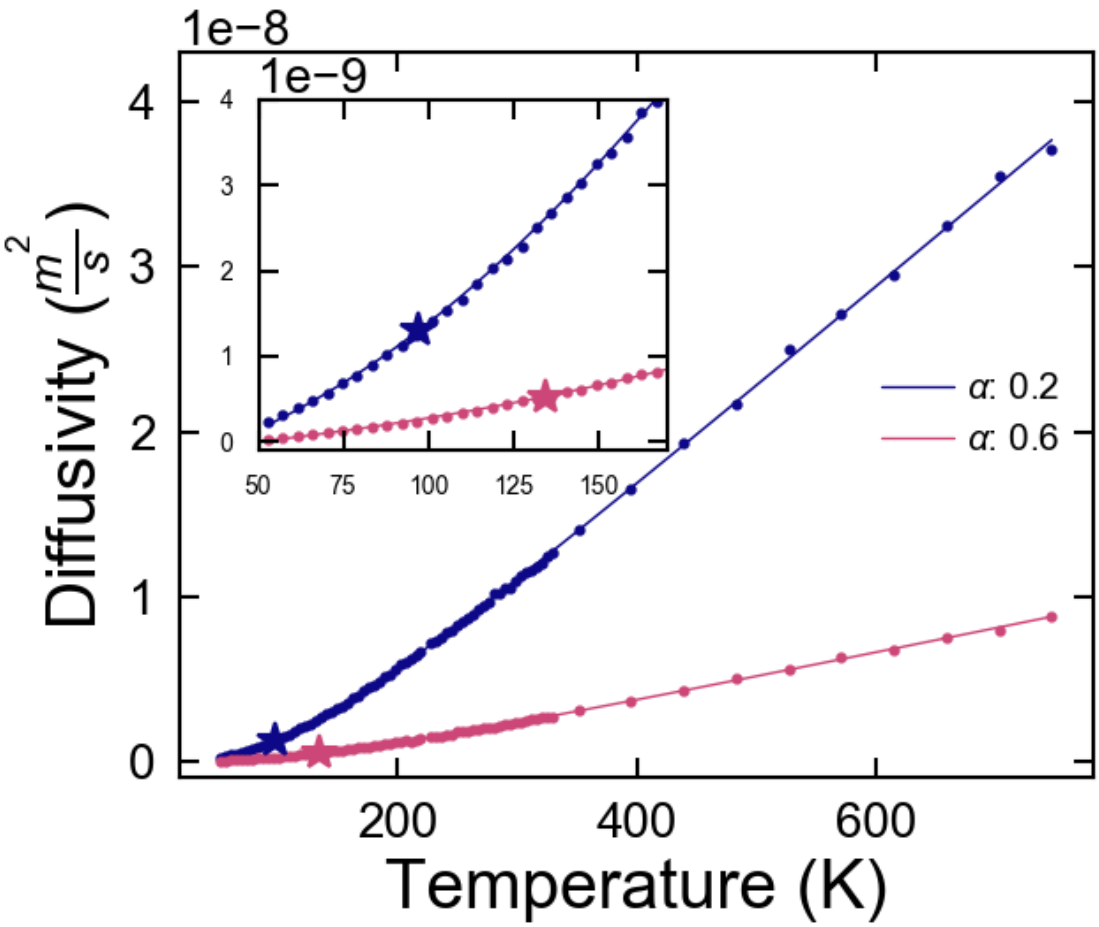
```
Cure_Ts.append(Ts)

mul_fact=10000
Ds_scaled=Ds*mul_fact
popt, pcov = curve_fit(hyper,
                       Ts,
                       Ds_scaled,
                       #    x0,y0,  a,  b,  c
                       p0=[1.0,1.0,1.0,1.0,1.0],
                       maxfev=2000000,
                       bounds=([0,-np.infty,0,0,-np.infty],
                       [np.infty,np.infty,np.infty,np.infty,np.infty]))
#popt, pcov = curve_fit(hyper,
#                        Ts,
#                        Ds_scaled,
#                       maxfev=200000)

T0=popt[0]#x0
D0=popt[1]/mul_fact#y0
a=popt[2]/mul_fact
b=popt[3]/mul_fact
c=popt[4]
Ps=Pt(Ts,T0,popt[4])
print('To',T0)
print('a',a,'b',b)
print('Ps',Ps[0],Ps[-1])
xs = Ts#np.linspace(0.1,4)
yHYP = hyper(xs, *popt)
residuals = Ds_scaled - yHYP
ss_res = np.sum(residuals**2)
ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
if ss_tot == 0:
    r_squared = 0
else:
    r_squared = 1 - (ss_res / ss_tot)

d=hyper(T0,*popt)
fitD0=d/mul_fact
distance_unit=1.06e-9#m
time_unit=9.29e-12#s
fit_D0_SI=fitD0*(distance_unit**2)/time_unit
Ds_SI=Ds*(distance_unit**2)/time_unit
T0_SI=T0*calibrationT
Ts_SI=Ts*calibrationT
fit_Ds = yHYP/mul_fact
fit_Ds_SI=fit_Ds*(distance_unit**2)/time_unit

m1,b1=get_tangent(Ts[0],*popt)
m2,b2=get_tangent(Ts[-1],*popt)

tgx,tgy=line_intersect(m1,b1,m2,b2)
l1x=np.linspace(Ts[0],tgx+0.1)
l1y=(m1*l1x)+b1
#plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
l2x=np.linspace(tgx-0.1,Ts[-1])
l2y=(m2*l2x)+b2
#plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

Tgs_tangent.append(tgx)

if sap/100 in cure_fractions_to_plot:
    ax1.plot(T0_SI,
             fit_D0_SI,
             marker='*',
             #markerfacecolor='w',
             color=colors[i],
             zorder=2,
             markersize=15)#,
```

```
            ax2.plot(T0_SI,
                     fit_D0_SI,
                     marker='*',
                     #markerfacecolor='w',
                     color=colors[i],
                     zorder=2,
                     markersize=15)#,

            ax1.plot(Ts_SI,
                     Ds_SI,
                     marker='.',
                     zorder=1,
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0)
            ax2.plot(Ts_SI,
                     Ds_SI,
                     marker='.',
                     zorder=1,
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0)
            ax1.plot(Ts_SI,
                     fit_Ds_SI,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                    label='$\\alpha$: {:.1f}'.format(sap/100))

            ax2.plot(Ts_SI,
                     fit_Ds_SI,
                     linewidth=1.0,
                     zorder=0,
                    color=colors[i],#cooling_colors[j],
                     label='{:.2f}% cured($R^2$:{:.4f},a: {:.4f},b: {:.4f}, c:
        {:.4f})'.format(sap,
              r_squared,
              a,
              b,
              c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)

            Tg=popt[0]
            Tg_SI=Tg*calibrationT
            Tgs.append(Tg_SI)
        ax1.legend(fontsize=15,loc='center right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(50,170)
        ax2.set_ylim(-1e-10,0.4e-8)
        ax1.set_ylim(-1e-9,4.3e-8)
        ax1.set_xlabel('Temperature (K)')
        ax1.set_ylabel('Diffusivity ($\\frac{m^2}{s}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'Tg_DPD',
                'bounded_hyperbola_fitting.pdf')

        plt.show()
```

```
To 0.221224933348
a 9.66087851197e-17 b 0.221776341287
Ps 0.32361160851 0.991957101732
To 0.204528970144
a 4.61805194739e-17 b 0.178050742368
```

```
Ps 0.389947738243 0.985013635095
To 0.217206896576
a 9.5875842477e-14 b 0.129564843865
Ps 0.360809065507 0.987681035387
To 0.24114272268
a 2.26511315072e-14 b 0.088893672034
Ps 0.33431761614 0.986583963359
To 0.30576161256
a 0.00447141077931 b 0.0498249453403
Ps 0.259754308888 0.985840557349
To 0.303195459885
a 9.60917106039e-24 b 0.0339310233341
Ps 0.28794578888 0.981475074583
To 0.381305762322
a 0.00339722941778 b 0.0126789507726
Ps 0.124101669723 0.992615262459
```

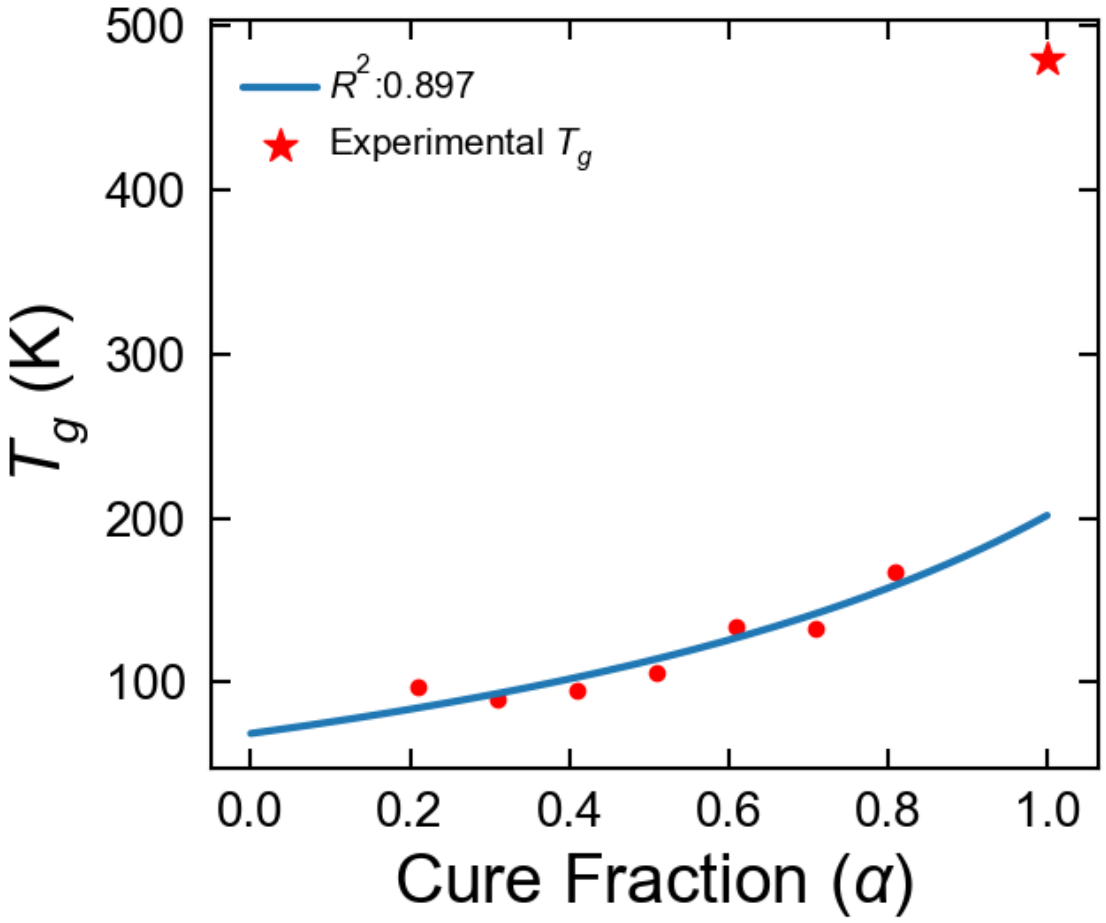
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [12]:
```
cure_percents = np.asarray(cure_percents)
#Tgs=[0.65,0.75,0.9,2.1]
fig, ax1 = plt.subplots()
#ax2=ax1.twinx()
Tgs = np.asarray(Tgs)
Tgs_tangent = np.asarray(Tgs_tangent)
print(Tgs)
Tg_data = np.asarray([cure_percents/100.,Tgs])
#np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
print(cure_percents_ss)
print(Tgs_ss)
R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                    Tgs_ss,
                                                    T1=None,
                                                    T0=None)
print('T1',T1,'lambda',inter_parm)
#plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
#print(cure_percents_ss)
alphas = np.linspace(0,1)
fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
ax1.scatter(cure_percents/100.,
            Tgs,
            #label='$E_a$:{}'.format(activation_energy),
            color='r')#colors[i])

Tg_sim = T1#0.851796418313
Tg_exp = 480
roomT_exp = 300
Tex_toTsim = Tg_exp/Tg_sim
roomT_sim =  Tg_sim*roomT_exp/Tg_exp
Tg0_exp = Tg_exp*T0/Tg_sim
print('300 K in T*:',roomT_sim)
ax1.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental $T_g$')
#ax2.set_ylabel('Tg (K)')

sim_low_lim = 0.1
ex_low_lim = sim_low_lim*Tex_toTsim
sim_up_lim = 0.8
ex_up_lim = sim_up_lim*Tex_toTsim
#ax2.set_ylim(ex_low_lim,ex_up_lim)
#ax1.set_ylim(sim_low_lim,sim_up_lim)
#ax1.set_ylim(0,510)
show_roomT=False
if show_roomT:
    ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
ax1.set_xlabel('Cure Fraction ($\\alpha$)')
ax1.set_ylabel('$T_g$ (K)')
ax1.legend(fontsize=15,loc='upper left')
#ax2.legend(fontsize=15,loc='lower right')
plt.ticklabel_format(axis='y',style='plain')
savefig(plt,'Tg_DPD',
        'bounded_hyperbola_dibeneditto.pdf')
```

```
[  97.19818313   89.86258463   95.43280402  105.94933475  134.34052284
  133.21304876  167.53187243]
[ 21.  31.  41.  51.  61.  71.  81.]
[  97.19818313   89.86258463   95.43280402  105.94933475  134.34052284
  133.21304876  167.53187243]
T1 201.640091017 lambda 0.5
300 K in T*: 126.025056885


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

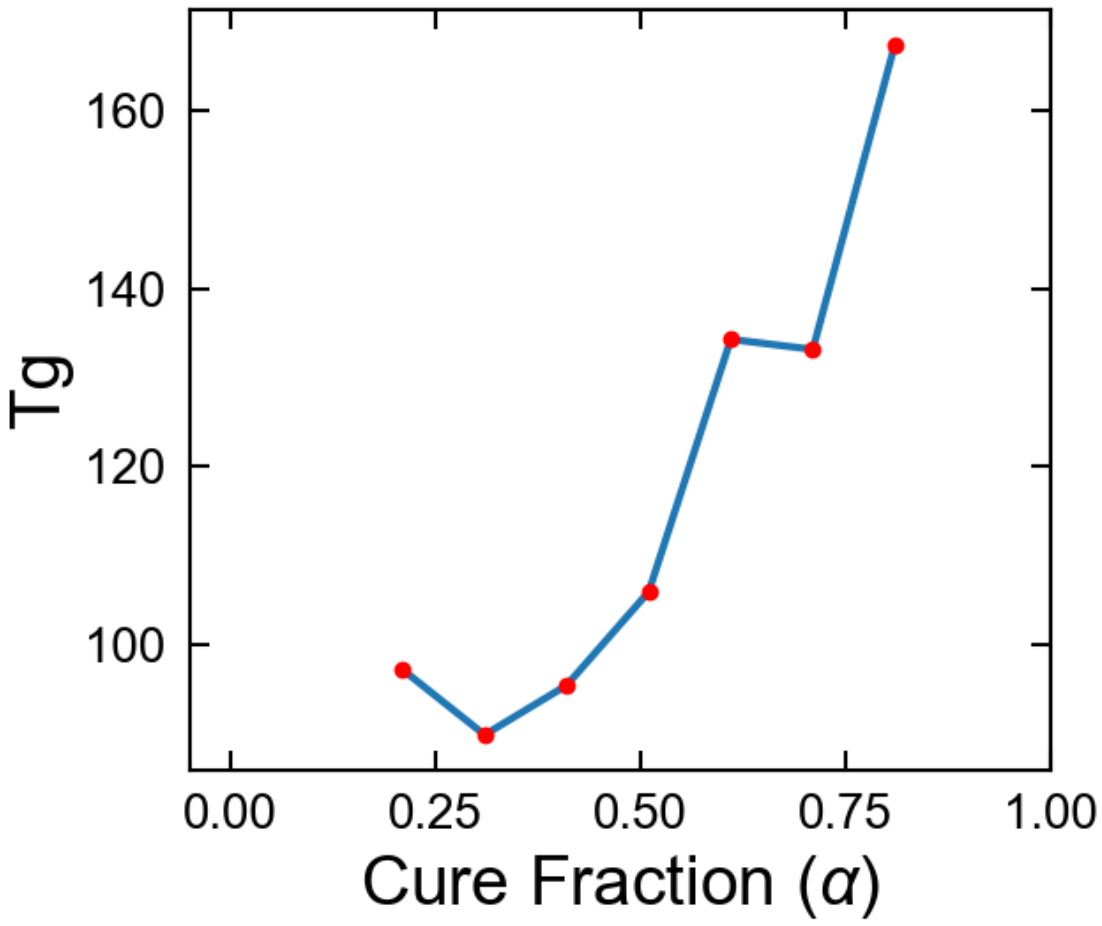
```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [10]: cure_percents = np.asarray(cure_percents)
         plt.scatter(cure_percents/100.,
                     Tgs,
                    marker='o',
                    color='r',
                       zorder=1)
         plt.plot(cure_percents/100.,
                     Tgs,
                    zorder=0)
         plt.legend(fontsize=20)
         plt.xlim(-0.05,1)
         plt.xlabel('Cure Fraction ($\\alpha$)')
         plt.ylabel('Tg')
         savefig(plt,
                 'Tg_DPD',
                'DPD_Tg.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
```

```
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



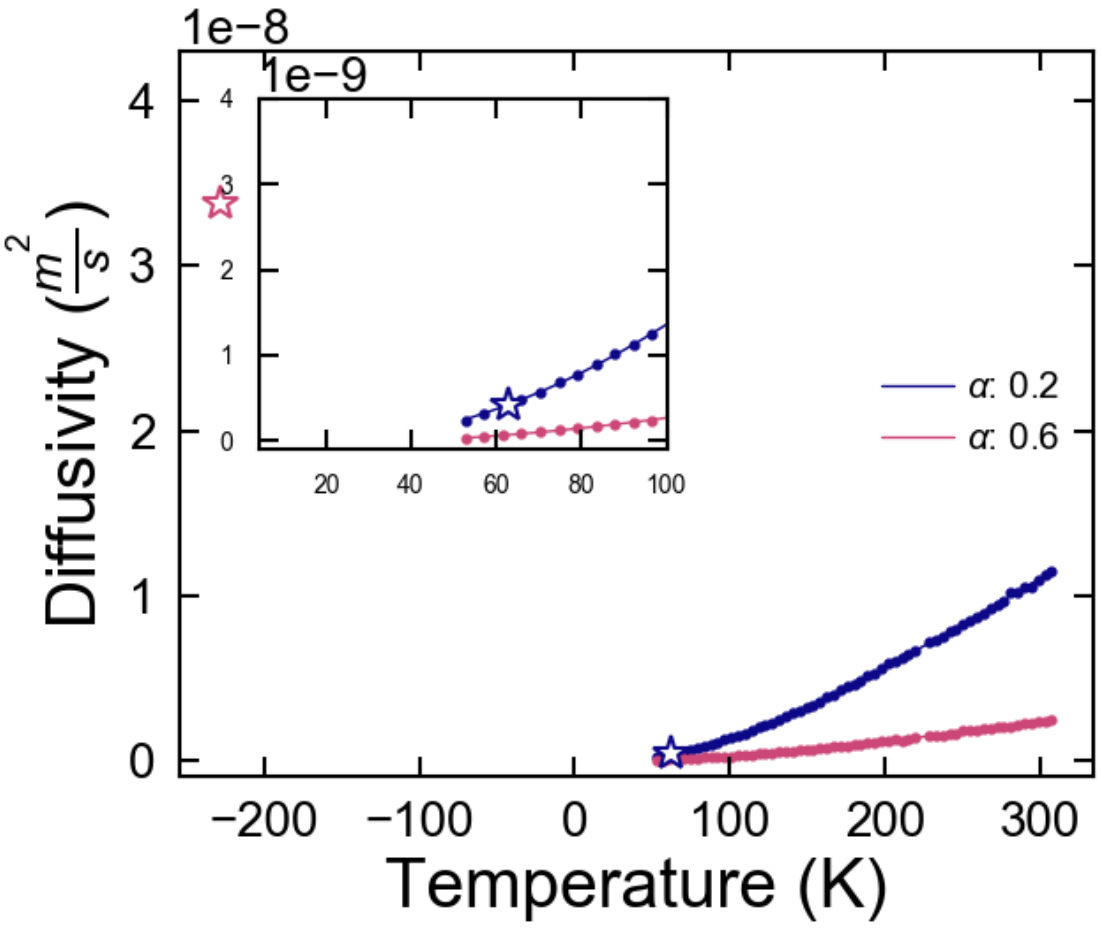

```
In [15]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.265, 0.515, 0.33, 0.33]
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.tick_params(labelsize=10)

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
```

```
filter_saps=[0.0,10.,20.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
cure_fractions_to_plot=[0.2,0.6]#[0.2,0.3,0.4,0.5,0.6,0.7,0.8]
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered = df[(df.integrator==integrator) &
                 (df.kT==kT) &
                 (df.quench_T<=0.7)&
                 #(df.quench_T>=0.1)&
                 #(df.quench_time ==quench_time)&
                 #(df.quench_method=='fine_quench')&
                 (df.tauP == tauP)&
                 (df.n_particles==N)&
                 (df.density==density)&
                 (df.activation_energy==activation_energy)&
                 (df.P==P)&
                 (df.cooling_method==cooling_method)&
                 (df.stop_after_percent>10)&
                 (df.stop_after_percent<90)&
                 #((df.stop_after_percent==20)|
                 # (df.stop_after_percent==60))&
                 (df.pot==pot)]
df_sorted=df_filtered.sort_values('quench_T')
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('stop_after_percent'))))
calibrationT=df_sorted.calibrationT.mean()
for i,(sap,df_curing) in enumerate(df_sorted.groupby('stop_after_percent')):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    cure_percent = df_curing.cure_percent.mean()
    cure_percents.append(cure_percent)
    Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=None)
    #print(Ds)
    Ts=Ts#*calibrationT
    Cure_Ts.append(Ts)

    mul_fact=10000
    Ds_scaled=Ds*mul_fact
    popt, pcov = curve_fit(hyper,
                           Ts,
                           Ds_scaled,
                          maxfev=200000)

    T0=popt[0]#x0
    D0=popt[1]/mul_fact#y0
    a=popt[2]/mul_fact
    b=popt[3]/mul_fact
    c=popt[4]
    Ps=Pt(Ts,T0,popt[4])
    print('To',T0)
    print('a',a,'b',b)
    print('Ps',Ps[0],Ps[-1])
    xs = Ts#np.linspace(0.1,4)
    yHYP = hyper(xs, *popt)
    residuals = Ds_scaled - yHYP
    ss_res = np.sum(residuals**2)
    ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
    if ss_tot == 0:
        r_squared = 0
    else:
        r_squared = 1 - (ss_res / ss_tot)

    d=hyper(T0,*popt)
    fitD0=d/mul_fact
    distance_unit=1.06e-9#m
    time_unit=9.29e-12#s
```

```
    fit_D0_SI=fitD0*(distance_unit**2)/time_unit
    Ds_SI=Ds*(distance_unit**2)/time_unit
    T0_SI=T0*calibrationT
    Ts_SI=Ts*calibrationT
    fit_Ds = yHYP/mul_fact
    fit_Ds_SI=fit_Ds*(distance_unit**2)/time_unit

    m1,b1=get_tangent(Ts[0],*popt)
    m2,b2=get_tangent(Ts[-1],*popt)

    tgx,tgy=line_intersect(m1,b1,m2,b2)
    l1x=np.linspace(Ts[0],tgx+0.1)
    l1y=(m1*l1x)+b1
    #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
    l2x=np.linspace(tgx-0.1,Ts[-1])
    l2y=(m2*l2x)+b2
    #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

    Tgs_tangent.append(tgx)

    if sap/100 in cure_fractions_to_plot:
        ax1.plot(T0_SI,
                 fit_D0_SI,
                 marker='*',
                 markerfacecolor='w',
                 color=colors[i],
                 zorder=2,
                 markersize=15)#,
        ax2.plot(T0_SI,
                 fit_D0_SI,
                 marker='*',
                 markerfacecolor='w',
                 color=colors[i],
                 zorder=2,
                 markersize=15)#,

        ax1.plot(Ts_SI,
                 Ds_SI,
                 marker='.',
                 zorder=1,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts_SI,
                 Ds_SI,
                 marker='.',
                 zorder=1,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax1.plot(Ts_SI,
                 fit_Ds_SI,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                label='$\\alpha$: {:.1f}'.format(sap/100))

        ax2.plot(Ts_SI,
                 fit_Ds_SI,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='{:.2f}% cured($R^2$:{:.4f},a: {:.4f},b: {:.4f}, c:
{:.4f})'.format(sap,
      r_squared,
      a,
      b,
      c))

    xvals = np.linspace(-3,4)
```

```
            yfits = hyper(xvals, *popt)

            Tg=popt[0]
            Tg_SI=Tg*calibrationT
            Tgs.append(Tg_SI)
        ax1.legend(fontsize=15,loc='center right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(4,100)
        ax2.set_ylim(-1e-10,0.4e-8)
        ax1.set_ylim(-1e-9,4.3e-8)
        ax1.set_xlabel('Temperature (K)')
        ax1.set_ylabel('Diffusivity ($\\frac{m^2}{s}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'Tg_DPD',
                'unbounded_hyperbola_fitting.pdf')

        plt.show()
```

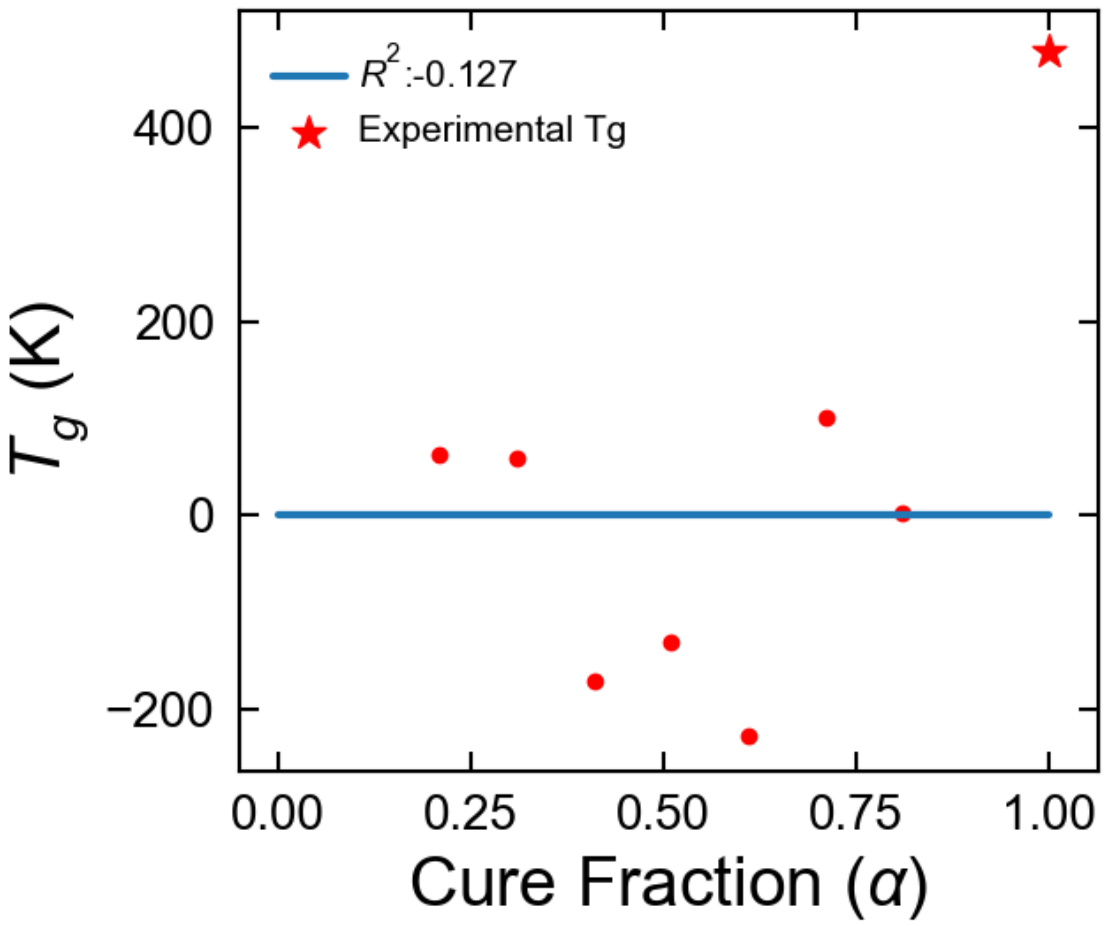
```
To 0.142614850241
a -0.0836701343693 b 0.303045624149
Ps 0.457268169519 0.951987766605
To 0.136740397598
a -0.0595586870079 b 0.228961750902
Ps 0.469673079982 0.94915458631
To -0.390913185113
a -9.85270178928 b 9.99733210916
Ps 0.988704757946 0.997455526974
To -0.298984348275
a -0.784763853835 b 0.882044057519
Ps 0.911242128217 0.98020909132
To -0.518031856812
a -5.78785336778 b 5.84900001253
Ps 0.991401099124 0.997595713538
To 0.231577392136
a -0.00334762212457 b 0.0332695965729
Ps 0.315049426578 0.929072449957
To 0.00632939702258
a -0.0226174592465 b 0.0434277667394
Ps 0.587238624357 0.867088233668


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [16]: cure_percents = np.asarray(cure_percents)
         #Tgs=[0.65,0.75,0.9,2.1]
         fig, ax1 = plt.subplots()
         #ax2=ax1.twinx()
         Tgs = np.asarray(Tgs)
         Tgs_tangent = np.asarray(Tgs_tangent)
         print(Tgs)
         Tg_data = np.asarray([cure_percents/100.,Tgs])
         #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
         cure_percents_ss = cure_percents#[:-1]
         Tgs_ss = Tgs#[:-1]
         print(cure_percents_ss)
         print(Tgs_ss)
         R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                             Tgs_ss,
                                                             T1=None,
                                                             T0=None)
         print('T1',T1,'lambda',inter_parm)
         #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
         #print(cure_percents_ss)
         alphas = np.linspace(0,1)
         fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
         ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
         ax1.scatter(cure_percents/100.,
                     Tgs,
                     #label='$E_a$:{}'.format(activation_energy),
```

```
                color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax1.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg')
        #ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.1
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 0.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax2.set_ylim(ex_low_lim,ex_up_lim)
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,510)
        show_roomT=False
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g$ (K)')
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'Tg_DPD',
                'unbounded_hyperbola_dibeneditto.pdf')
```

```
[  62.65977402   60.07875335 -171.7530243  -131.3628396  -227.60434152
  101.74667669    2.78090666]
[ 21.  31.  41.  51.  61.  71.  81.]
[  62.65977402   60.07875335 -171.7530243  -131.3628396  -227.60434152
  101.74667669    2.78090666]
T1 6.64585855447e-20 lambda 0.5
300 K in T*: 4.15366159654e-20


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



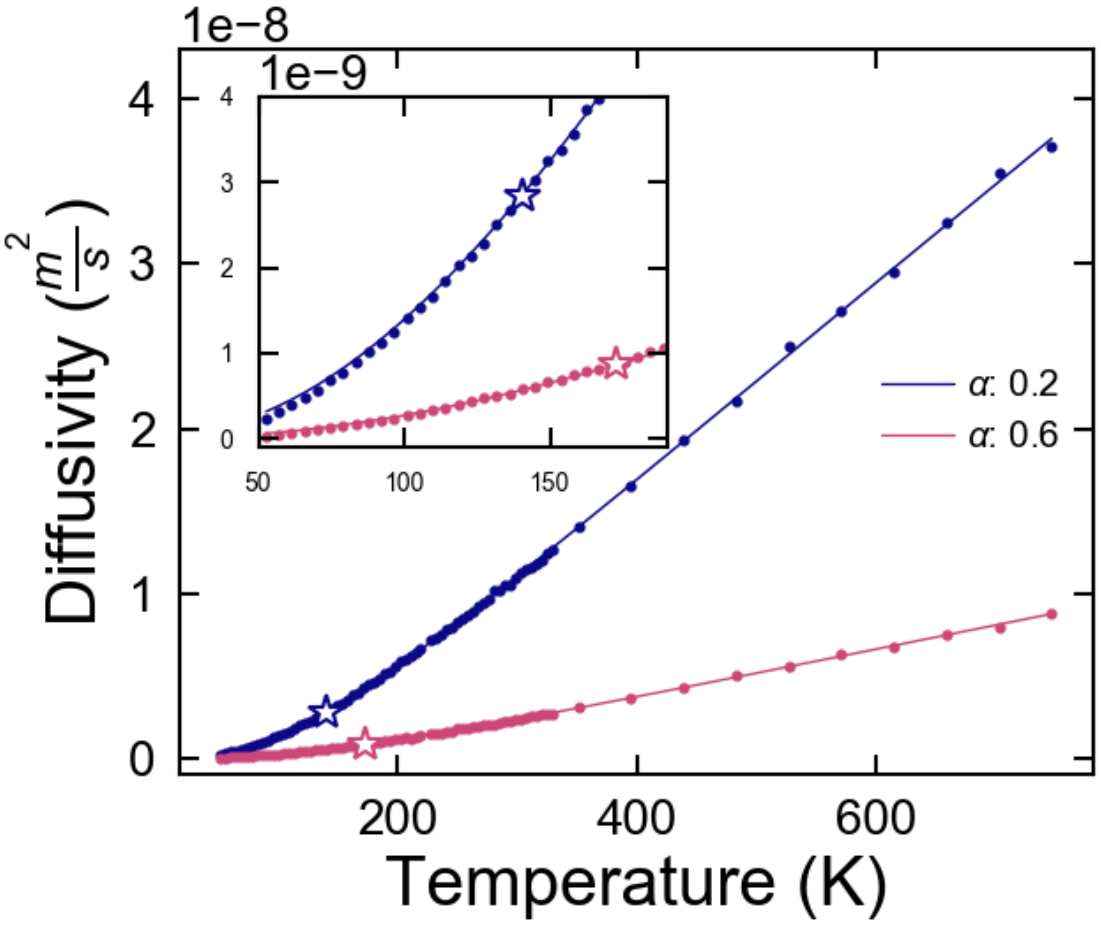

```
In [17]: import matplotlib
         from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.265, 0.515, 0.33, 0.33]
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.tick_params(labelsize=10)

         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.0,10.,20.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         cure_fractions_to_plot=[0.2,0.6]#[0.2,0.3,0.4,0.5,0.6,0.7,0.8]
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
```

```
df_filtered = df[(df.integrator==integrator) &
                 (df.kT==kT) &
                 #(df.quench_T<=0.7)&
                 #(df.quench_T>=0.1)&
                 #(df.quench_time ==quench_time)&
                 #(df.quench_method=='fine_quench')&
                 (df.tauP == tauP)&
                 (df.n_particles==N)&
                 (df.density==density)&
                 (df.activation_energy==activation_energy)&
                 (df.P==P)&
                 (df.cooling_method==cooling_method)&
                 (df.stop_after_percent>10)&
                 (df.stop_after_percent<90)&
                 #((df.stop_after_percent==20)|
                 # (df.stop_after_percent==60))&
                 (df.pot==pot)]
df_sorted=df_filtered.sort_values('quench_T')
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('stop_after_percent'))))
calibrationT=df_sorted.calibrationT.mean()
for i,(sap,df_curing) in enumerate(df_sorted.groupby('stop_after_percent')):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    cure_percent = df_curing.cure_percent.mean()
    cure_percents.append(cure_percent)
    Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME,quench_time=None)
    #print(Ds)
    Ts=Ts#*calibrationT
    Cure_Ts.append(Ts)

    mul_fact=1
    Ds_scaled=Ds*mul_fact
    popt, pcov = curve_fit(custom,
                           Ts,
                           Ds_scaled,
                           #    a,     b,          p
                           p0=[1.0,1.0,1.0],
                           maxfev=2000000,
                           bounds=([0.0,0.0,0.0],
                           [np.infty,np.infty,np.infty]))

    a=popt[0]
    b=popt[1]
    p=popt[2]
    xs = Ts#np.linspace(0.1,4)
    yHYP = custom(xs, *popt)
    residuals = Ds_scaled - yHYP
    ss_res = np.sum(residuals**2)
    ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
    if ss_tot == 0:
        r_squared = 0
    else:
        r_squared = 1 - (ss_res / ss_tot)

    m1,b1=get_tangent_custom(Ts[0],*popt)
    m2,b2=get_tangent_custom(Ts[-1],*popt)
    tgx,tgy=line_intersect(m1,b1,m2,b2)
    l1x=np.linspace(Ts[0],tgx+0.1)
    l1y=(m1*l1x)+b1
    #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
    l2x=np.linspace(tgx-0.1,Ts[-1])
    l2y=(m2*l2x)+b2
    #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)
    T0=tgx
    d=custom(tgx,*popt)
    fitD0=d/mul_fact
    distance_unit=1.06e-9#m
    time_unit=9.29e-12#s
    fit_D0_SI=fitD0*(distance_unit**2)/time_unit
```

```
    Ds_SI=Ds*(distance_unit**2)/time_unit
    T0_SI=T0*calibrationT
    Ts_SI=Ts*calibrationT
    fit_Ds = yHYP/mul_fact
    fit_Ds_SI=fit_Ds*(distance_unit**2)/time_unit


    Tgs_tangent.append(tgx)

    if sap/100 in cure_fractions_to_plot:
        ax1.plot(T0_SI,
                 fit_D0_SI,
                 marker='*',
                 markerfacecolor='w',
                 color=colors[i],
                 zorder=2,
                 markersize=15)#,
        ax2.plot(T0_SI,
                 fit_D0_SI,
                 marker='*',
                 markerfacecolor='w',
                 color=colors[i],
                 zorder=2,
                 markersize=15)#,

        ax1.plot(Ts_SI,
                 Ds_SI,
                 marker='.',
                 zorder=1,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts_SI,
                 Ds_SI,
                 marker='.',
                 zorder=1,
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax1.plot(Ts_SI,
                 fit_Ds_SI,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                label='$\\alpha$: {:.1f}'.format(sap/100))

        ax2.plot(Ts_SI,
                 fit_Ds_SI,
                 linewidth=1.0,
                 zorder=0,
                color=colors[i],#cooling_colors[j],
                 label='{:.2f}% cured($R^2$:{:.4f},a: {:.4f},b: {:.4f}, p:
{:.4f})'.format(sap,
      r_squared,
      a,
      b,
      p))
    Tg=tgx
    Tg_SI=Tg*calibrationT
    Tgs.append(Tg_SI)
ax1.legend(fontsize=15,loc='center right')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
ax2.set_xlim(50,190)
ax2.set_ylim(-1e-10,0.4e-8)
ax1.set_ylim(-1e-9,4.3e-8)
ax1.set_xlabel('Temperature (K)')
```

```
        ax1.set_ylabel('Diffusivity ($\\frac{m^2}{s}$)')
        #savefig(plt,'hyperbola_fitting_unbounded','all_alphas_zoomed.pdf')
        savefig(plt,'Tg_DPD',
                'power_law_fitting.pdf')

        plt.show()
```

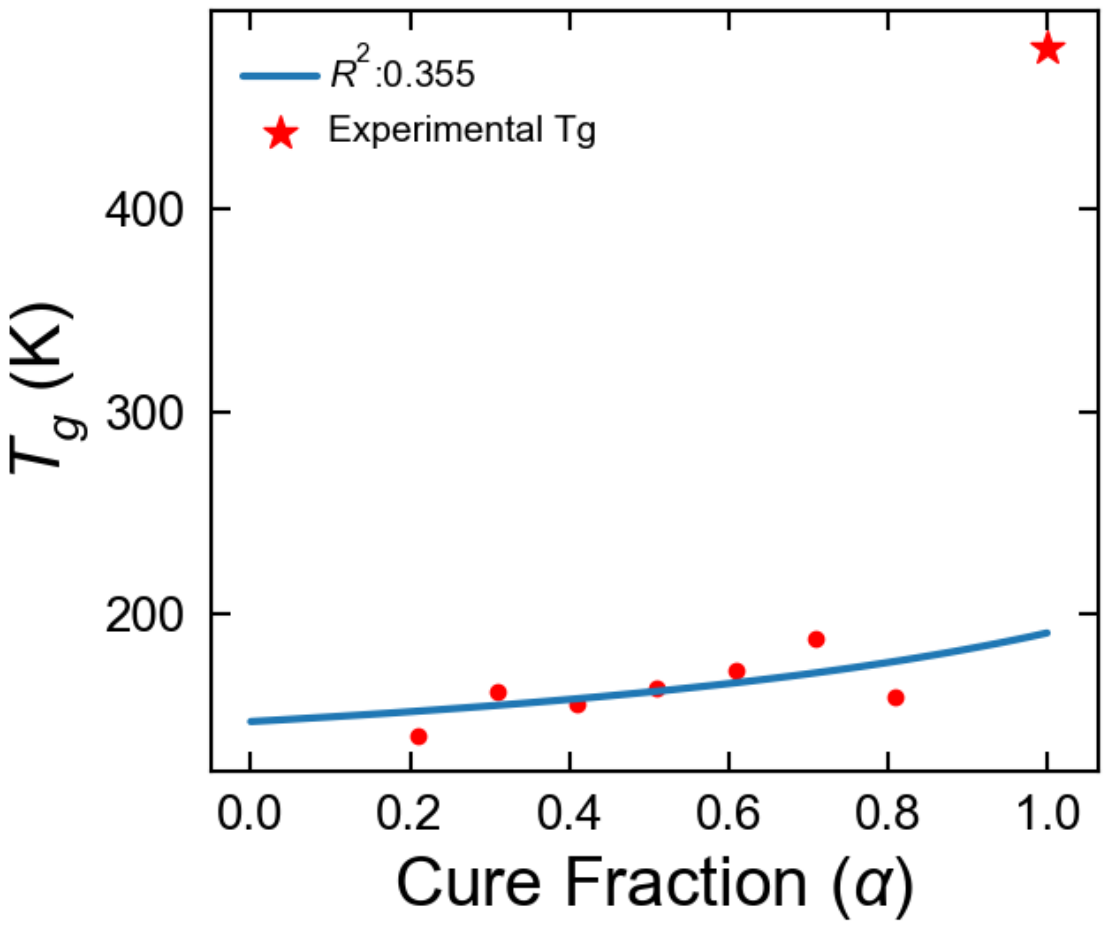
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [18]:
```
cure_percents = np.asarray(cure_percents)
#Tgs=[0.65,0.75,0.9,2.1]
fig, ax1 = plt.subplots()
#ax2=ax1.twinx()
Tgs = np.asarray(Tgs)
Tgs_tangent = np.asarray(Tgs_tangent)
print(Tgs)
Tg_data = np.asarray([cure_percents/100.,Tgs])
#np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
```

```
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax1.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg')
        #ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.1
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 0.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        #ax2.set_ylim(ex_low_lim,ex_up_lim)
        #ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,510)
        show_roomT=False
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g$ (K)')
        ax1.legend(fontsize=15,loc='upper left')
        #ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'Tg_DPD',
                'power_law_dibeneditto.pdf')
```

```
[ 140.36338564  162.1313821   155.53714761  163.98481988  172.72999887
  188.37481101  159.13393074]
[ 21.  31.  41.  51.  61.  71.  81.]
[ 140.36338564  162.1313821   155.53714761  163.98481988  172.72999887
  188.37481101  159.13393074]
T1 190.835514016 lambda 0.5
300 K in T*: 119.27219626


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


R^2:0.355
Experimental Tg
400
300
200
T_g (K)
0.0
0.2
0.4
0.6
0.8
1.0
Cure Fraction (α)

In [1]:
```
from common import *
import numpy as np
```

1 Parameters for DGEBA/DDS/PES (LJ) (quench_strategy="coarse_fine")

In [2]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA_DDS_PES/'

tau=1
tauP=10
num_c10=0
kT = 1.7
N=50000
use_curing_job_P=False
cooling_method = 'anneal'
integrator='NPT'
pot='LangH'
activation_energy=2.0
density=1.0
quench_time=3e6
coarse_quench_time=3e6
fine_quench_time=6e6
quench_strategy='coarse_fine'
P = 6.0#[4.5, 6, 8]
stop_after_percents = np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.,40.,50.]#[60.,70.,
80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
```

In [3]:
```
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

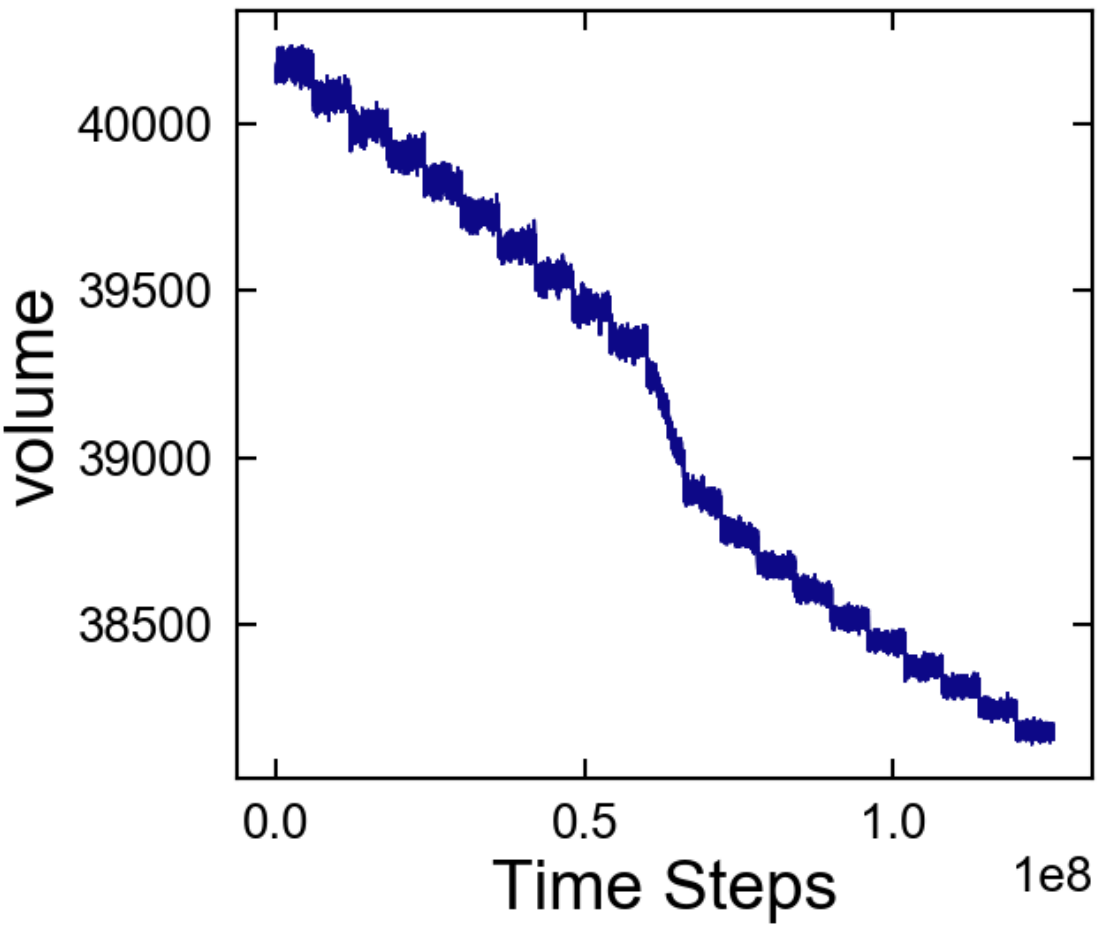
In [16]:
```
df_curing = df[(df.bond==False)&
                            (df.quench_time==fine_quench_time)&
                             (df.quench_method=='fine_quench')&
                            (df.cooling_method==cooling_method)]#&
                            #(df.stop_after_percent==80.)]
 df_curing.quench_temp_prof[-1][-1][0]/df_curing.dcd_write
 df_curing.pot
 #60000000.0/df_curing.dcd_write
```

Out[16]:
```
aaf20b058ecdad475a5349e35eed9f82    LangH
af39575dfd538b5a658b084611dcd49d    LangH
```

```
a7f6d499313e912f17a5fc83196f4edb    LangH
b8284cfadf869c1176f5c06ebe8c2405    LangH
f79c0a0626cb5e7e87ac105a80f9f2bd    LangH
c88721a9d284845da2754000bb76d46f    LangH
3a03184c70bc3330eb93850126804e46    LangH
3d49795d5d58f412c89848c04665a1f7    LangH
1ccba542af7a2a217511b03c826e604f    LangH
14a582d8ff7b32e005886ecc12e9c439    LangH
6308d796b634c6a17d12842db9b8408e    LangH
7932520b6d3b660a9f8bc0ddb8a9d0fb    LangH
587d75da08f1881c40caee3b1c6c4ece    LangH
5c78ffc9384d3e76b797c33f8a92f6f1    LangH
5663e6302c7c338ec0b0ccfba90fcd52    LangH
Name: pot, dtype: object
```

In [5]:
```
#stop_after_percents = [50.,80.,100.]#np.arange(10,105,15,dtype=float)
PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
df_curing = df[(df.bond==False)&
                        (df.quench_time==fine_quench_time)&
                         (df.quench_method=='fine_quench')&
                        (df.cooling_method==cooling_method)&
                        (df.stop_after_percent==80.)]
plt.figure()
for sap in stop_after_percents:
    plot_equilibriation(df_curing,
                        project,
                        PROP_NAME,
                        draw_decorrelated_samples=False,
                        draw_equilibrium_window=False,
                        mean_from_second_half=False)

    #break
plt.xlabel('Time Steps')
plt.ylabel(PROP_NAME)
#plt.legend(fontsize=5)
#plt.xlim(0,100)
#plt.ylim(34000,41000)
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

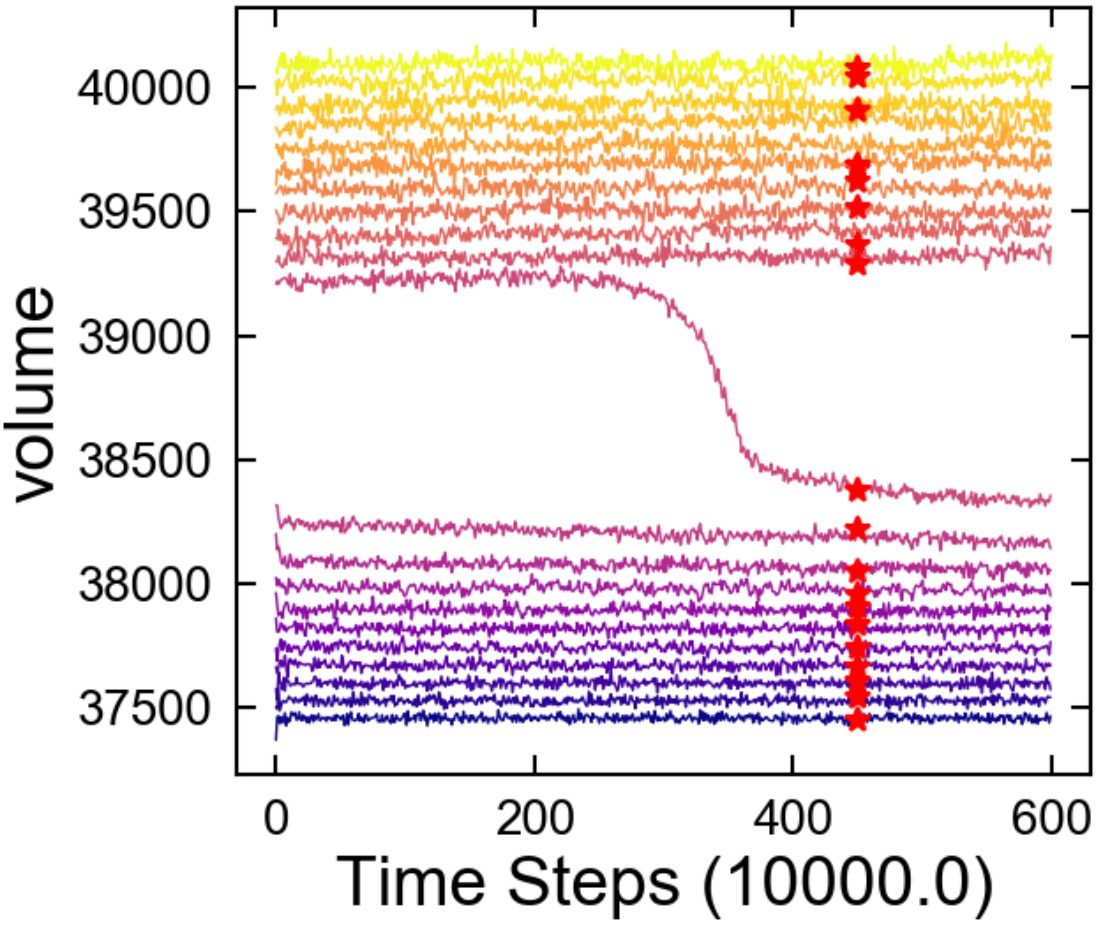



```
In [6]: #stop_after_percents = [100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        for sap in stop_after_percents:
            df_curing = df[(df.bond==False)&
                           (df.quench_time==fine_quench_time)&
                            (df.quench_method=='fine_quench')&
                           (df.cooling_method==cooling_method)&
                       (df.CC_bond_angle!=109.5)&
                           (df.stop_after_percent==50.)]
            split_log(df_curing,project,PROP_NAME,filter_temp=1.0,rtol=0.01,show_all=True)
            log_write = df_curing.log_write.values[0]
            break
        plt.xlabel('Time Steps ({})'.format(log_write))
        plt.ylabel(PROP_NAME)
        #plt.legend(fontsize=5)
        #plt.xlim(0,100)
        ##plt.ylim(1,2)
        plt.show()
        quench_time
```

```
af39575dfd538b5a658b084611dcd49d
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



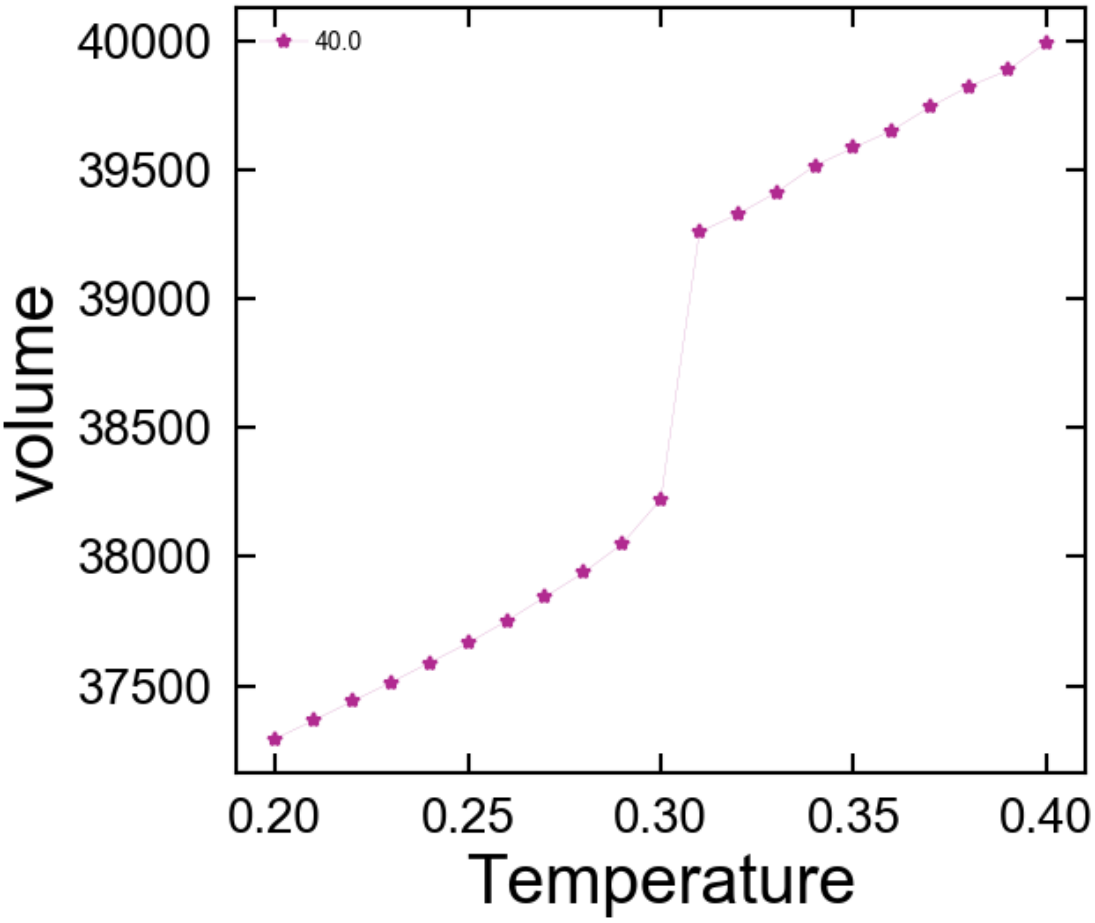
Out[6]: 3000000.0

```
In [7]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = [0.,40.,80.,100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
        for i,sap in enumerate(stop_after_percents):
            filter_saps=[40.]#[0.,10.,20.,30.]
            if sap not in filter_saps:
                continue
            df_curing = df[(df.bond==False)&
                           #(df.tau==tau)&
                           #(df.use_curing_job_P==use_curing_job_P)&
                           #(df.tauP==tauP)&
                           #(df.P==P)&
                           #(df.num_c10==num_c10)&
                            (df.quench_time==fine_quench_time)&
```

```
                         (df.quench_method=='fine_quench')&
                        (df.cooling_method==cooling_method)&
                        (df.stop_after_percent==sap)]
        for job_id in df_curing.index:
            job = project.open_job(id=job_id)
            #print(job.sp.P,job.sp.stop_after_percent)
            means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
            mean_vols = np.asarray(means)
            density = mean_vols/job.sp.n_particles
            #print(temps,means)

            if False:
                model = piecewise(temps, mean_vols)
                #print(model)
                if len(model.segments) == 2:
                    lines = []
                    l1 = model.segments[0]
                    m1 = l1.coeffs[1]
                    b1 = l1.coeffs[0]
                    l2 = model.segments[1]
                    m2 = l2.coeffs[1]
                    b2 = l2.coeffs[0]
                    x,y = line_intersect(m1,b1,m2,b2)
                    xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                    ys = l1.coeffs[1]*xs+l1.coeffs[0]
                    plt.plot(xs,
                             ys,
                             color=colors[i],
                             zorder=0,
                             linewidth=1)
                    xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                    ys = l2.coeffs[1]*xs+l2.coeffs[0]
                    plt.plot(xs,
                             ys,
                             color=colors[i],
                             zorder=0,
                             linewidth=1)
                else:
                    print('WARNING: found more or less than 2 line segments in regression!')

            Tg,Tg_prop_val = find_Tg(temps,mean_vols)
            show_transition = False
            if show_transition:
                #plt.axvline(x=Tg,
                #            linewidth=1.0,
                #            color=colors[i])
                plt.scatter(Tg,
                            Tg_prop_val,
                            marker='*',
                            s=100,
                            color='r',
                           zorder=0)#colors[i])
            plt.plot(temps,
                     mean_vols,
                     marker='*',
                     color=colors[i],
                     label=job.sp.stop_after_percent,
                     linewidth=0.1)
    plt.xlabel('Temperature')
    plt.ylabel(PROP_NAME)
    plt.legend(fontsize=10)
    #plt.xlim(0.11,0.51)
    #plt.ylim(34000,43000)
    plt.show()
```

```
587d75da08f1881c40caee3b1c6c4ece
using line iftting
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

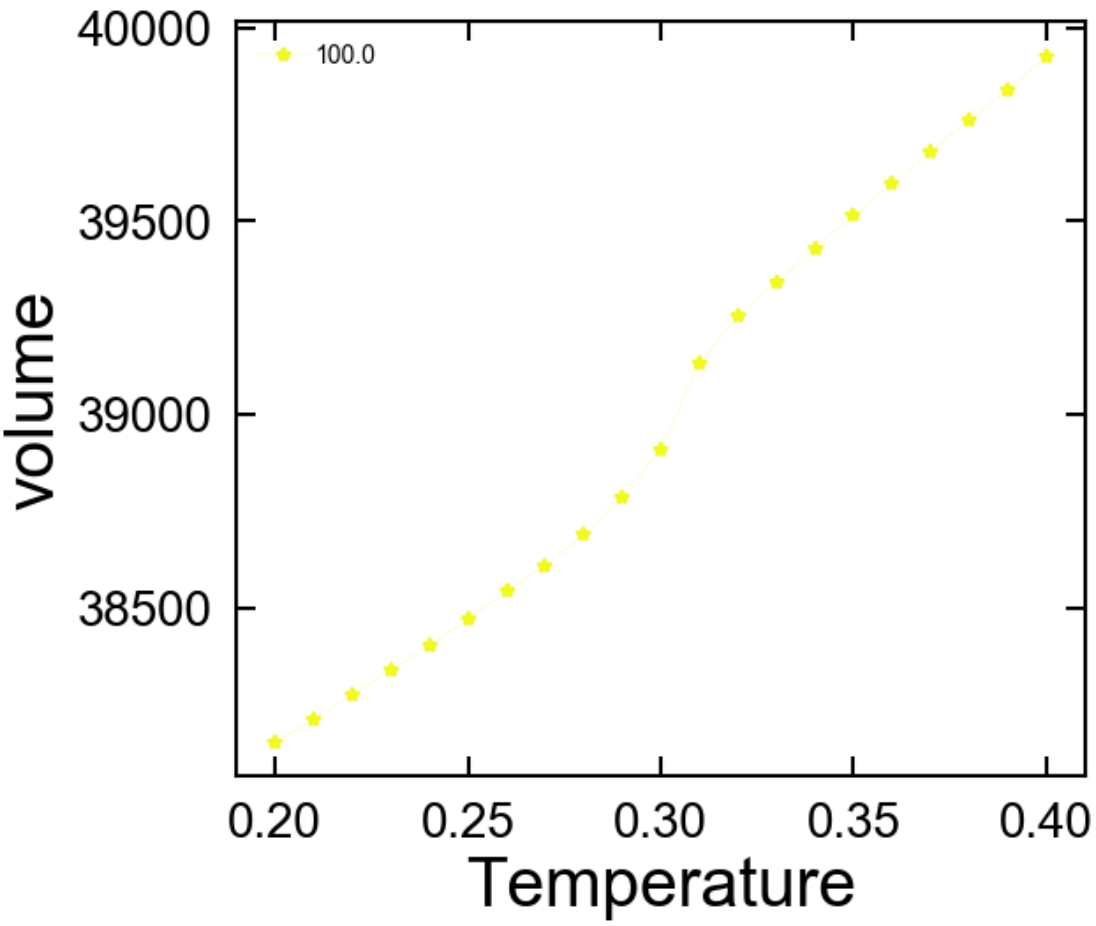



```
In [8]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = [0.,40.,80.,100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
        for i,sap in enumerate(stop_after_percents):
            filter_saps=[100.]
            if sap not in filter_saps:
                continue
            df_curing = df[(df.bond==False)&
                           #(df.tau==tau)&
                           #(df.use_curing_job_P==use_curing_job_P)&
                           #(df.tauP==tauP)&
                           #(df.P==P)&
                           #(df.num_c10==num_c10)&
                            (df.quench_time==fine_quench_time)&
                            (df.quench_method=='fine_quench')&
```

```
                    (df.cooling_method==cooling_method)&
                    (df.stop_after_percent==sap)]
        for job_id in df_curing.index:
            job = project.open_job(id=job_id)
            #print(job.sp.P,job.sp.stop_after_percent)
            means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
            mean_vols = np.asarray(means)
            density = mean_vols/job.sp.n_particles
            #print(temps,means)

            if False:
                model = piecewise(temps, mean_vols)
                #print(model)
                if len(model.segments) == 2:
                    lines = []
                    l1 = model.segments[0]
                    m1 = l1.coeffs[1]
                    b1 = l1.coeffs[0]
                    l2 = model.segments[1]
                    m2 = l2.coeffs[1]
                    b2 = l2.coeffs[0]
                    x,y = line_intersect(m1,b1,m2,b2)
                    xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                    ys = l1.coeffs[1]*xs+l1.coeffs[0]
                    plt.plot(xs,
                             ys,
                             color=colors[i],
                             zorder=0,
                             linewidth=1)
                    xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                    ys = l2.coeffs[1]*xs+l2.coeffs[0]
                    plt.plot(xs,
                             ys,
                             color=colors[i],
                             zorder=0,
                             linewidth=1)
                else:
                    print('WARNING: found more or less than 2 line segments in regression!')

            Tg,Tg_prop_val = find_Tg(temps,mean_vols)
            show_transition = False
            if show_transition:
                #plt.axvline(x=Tg,
                #            linewidth=1.0,
                #            color=colors[i])
                plt.scatter(Tg,
                            Tg_prop_val,
                            marker='*',
                            s=100,
                            color='r',
                           zorder=0)#colors[i])
            plt.plot(temps,
                     mean_vols,
                     marker='*',
                     color=colors[i],
                     label=job.sp.stop_after_percent,
                     linewidth=0.1)
    plt.xlabel('Temperature')
    plt.ylabel(PROP_NAME)
    plt.legend(fontsize=10)
    #plt.xlim(0.11,0.51)
    #plt.ylim(34000,43000)
    plt.show()
```

7932520b6d3b660a9f8bc0ddb8a9d0fb
using line iftting

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [9]: def get_split_quench_job_msd(job,prop_name):
            times = []
            prop_vals = []
            qTs=[]
            if job.isfile('msd.log'):
                log_path = job.fn('msd.log')
                data = np.genfromtxt(log_path, names=True)
                PROP_NAME =prop_name
                prop_values = data[PROP_NAME]#'pair_lj_energy']
                time_steps = data['timestep']
                len_prof = len(job.sp.quench_temp_prof)
                for i in range(0,len_prof,2):
                    current_point = job.sp.quench_temp_prof[i]
                    next_point = job.sp.quench_temp_prof[i+1]
                    start_time = current_point[0]
                    end_time = next_point[0]
                    if current_point[1]!=next_point[1]:
                        print('WARNING! Detected a non isothermal step')
                    target_T = current_point[1]
```

```
                #print(time_steps)
                #print(start_time,end_time)
                indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
                start_index = indices[0][0]
                end_index = indices[0][-1]
                sliced_ts = time_steps[start_index:end_index+1]
                sliced_prop_vals = prop_values[start_index:end_index+1]
                #sliced_pe = pe[start_index:end_index+1]
                #mean,std = get_mean_and_std(job,sliced_ts,sliced_prop_vals,sliced_pe)
                #means.append(mean)
                #stds.append(std)
                times.append(sliced_ts)
                prop_vals.append(sliced_prop_vals)
                qTs.append(target_T)
    return times,prop_vals,qTs
```

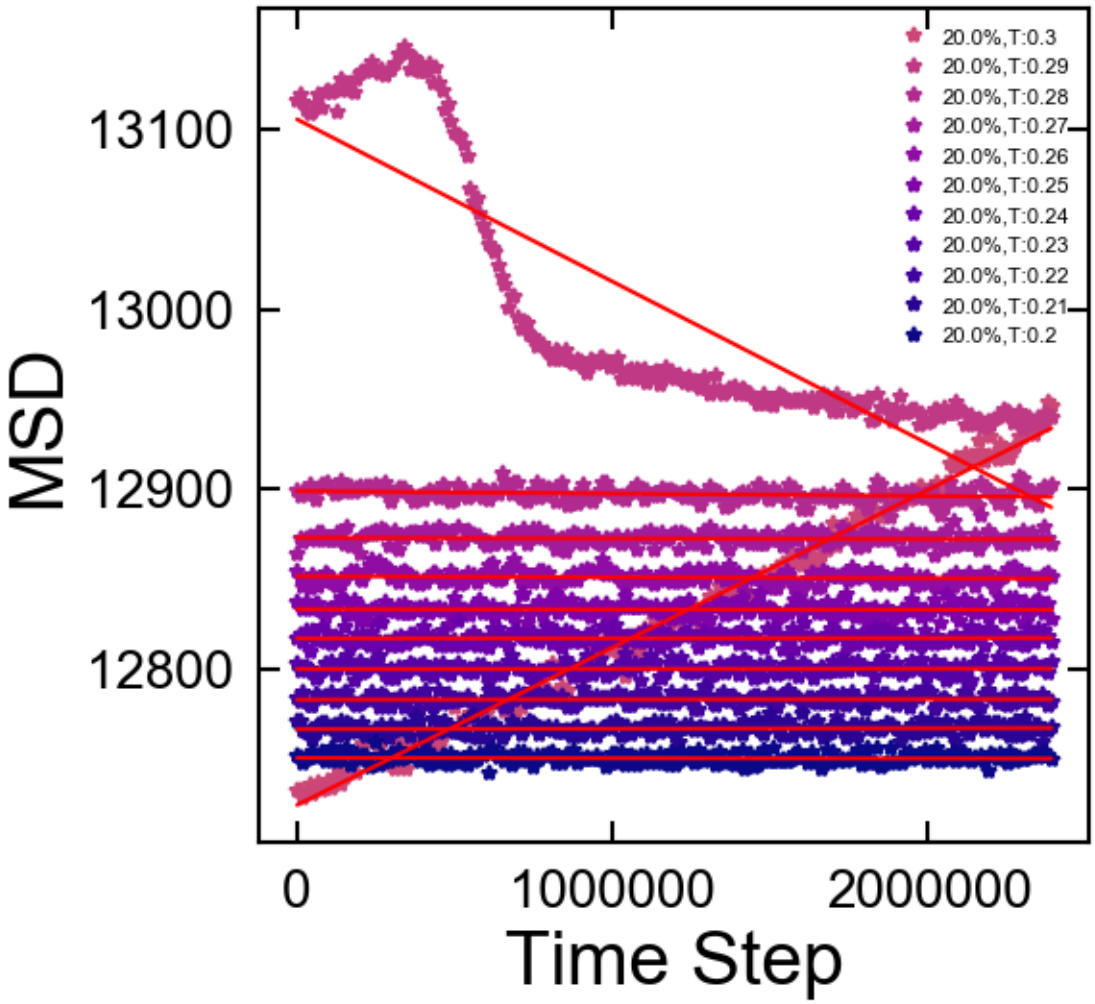
```
In [10]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()

         for i,sap in enumerate(stop_after_percents):
             filter_saps=[20.0]
             if sap not in filter_saps:
                 continue
             df_curing = df[(df.bond==False)&
                            (df.quench_time==fine_quench_time)&
                            (df.quench_method=='fine_quench')&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==sap)]
             for job_id in df_curing.index:
                 job = project.open_job(id=job_id)
                 #print(job.sp.P,job.sp.stop_after_percent)
                 times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
                 colors = plt.cm.plasma(np.linspace(1,0,len(msds)))
                 for j,msd in enumerate(msds):
                     #print(len(msd))
                     start_index = int(len(times[j])*0.6)
                     time=times[j][start_index:]
                     print('using the last {} timesteps of MSD'.format(len(time)))
                     normalized_time = times[0][:len(time)]
                     eq_msd = msd[start_index:]
                     quench_T = qTs[j]
                     if quench_T<0.3:
                         if True:
                             #print(time,msd,quench_T)
                             popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                                    time,
                                                    eq_msd,
                                                    p0=[0.2,1.0],
                                                    maxfev=2000000,
                                                    bounds=([0.0,0.0],[1.0,4.0]))
                             #print(time,eq_msd)
                             #print(time,eq_msd)
                             #print(time[0],time[-1],time[-1]-time[0])
                             popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                                    normalized_time,
                                                    eq_msd,
                                                    p0=[1.,0.0],
                                                    bounds=([-1,0.0],[np.infty,np.infty]))

                             # determine the slop using linear regression
                             par = np.polyfit(normalized_time, eq_msd, 1, full=True)
                             drdt_A = popt[0]#par[0][0]#0-slope, 1-intercept
                             x=normalized_time
```

```
                        if False:
                            y = popt[0]*x**(popt[1])
                            #y = x**(popt[0])
                        else:
                            y=popt[0]*x+popt[1]
                            #y=par[0][0]*x+par[0][1]
                        #calculate the diffusion coefficient


                        plt.figure(0)

                        if True:
                            #print(x,y)
                            plt.plot(x,
                                     y,
                                     color='r',#colors[i],
                                     linewidth=1.5,
                                     zorder=1)#,
    #label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))

                    #Tg,Tg_prop_val = find_Tg(temps,mean_vols,sap)
                    show_transition = False
                    if show_transition:
                        #plt.axvline(x=Tg,
                        #            linewidth=1.0,
                        #            color=colors[i])
                        plt.scatter(Tg,
                                    Tg_prop_val,
                                    marker='*',
                                    s=100,
                                    color='r',
                                   zorder=0)#colors[i])
                    plt.plot(normalized_time,
                             eq_msd,
                             marker='*',
                             color=colors[j],
    label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                             zorder=0,
                             linewidth=0.0)
    plt.xlabel('Time Step')
    plt.ylabel('MSD')
    #plt.xscale('log')
    #plt.yscale('log')
    plt.legend(fontsize=8)
    #plt.xlim(0.11,0.51)
    #plt.ylim(34000,43000)
    plt.show()
```

```
using the last 241 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
```

```
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
<matplotlib.figure.Figure at 0x11c926d30>
```

```
In [17]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
```

```
PROP_NAME
='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
plt.figure()
colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
Tgs=[]
cure_percents = []
for i,sap in enumerate(stop_after_percents):
    filter_saps=[0.0,20,30,40,50,60,70,80]#,90]
    if sap not in filter_saps:
        continue
    Ts=[]
    Ds=[]
    df_curing = df[(df.bond==False)&
                   (df.quench_time==fine_quench_time)&
                   (df.quench_method=='fine_quench')&
                   (df.CC_bond_angle!=109.5)&
                   (df.cooling_method==cooling_method)&
                   (df.stop_after_percent==sap)]
    cure_percent = df_curing.cure_percent.values[0]
    cure_percents.append(cure_percent)
    print(cure_percent,df_curing.quench_T)
    for job_id in df_curing.index:
        job = project.open_job(id=job_id)
        #print(job.sp.P,job.sp.stop_after_percent)
        log_path = job.fn('msd.log')
        data = np.genfromtxt(log_path, names=True)
        prop_values = data[PROP_NAME]#'pair_lj_energy']
        all_time_steps = data['timestep']
        if True:
            plt.figure(0)
            plt.plot(all_time_steps*job.sp.md_dt,#times[0][0:len(time)],
                        prop_values,
                        marker='o',
                        markersize=5,
                        color=colors[i],
                        label='$\\alpha$ {}'.format(job.sp.stop_after_percent/100),
                        linewidth=0.1)
            plt.ticklabel_format(axis='both', style='sci', scilimits=(-2,2))
            plt.legend(fontsize=15,ncol=2)
        times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
        for j,msd in enumerate(msds):
            start_index = int(len(times[j])*0.6)
            time=times[j]*job.sp.md_dt
            quench_T = qTs[j]
            eq_msd = msd[start_index:]
            eq_time = time[start_index:]
            #print(quench_T)
            # determine the slop using linear regression
            fit_method='curve_fit'#'power_law','poly_fit'
            if fit_method=='curve_fit':
                norm_eq_time = (eq_time-eq_time[0])
                #print(norm_eq_time,eq_msd)
                popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                       eq_time,
                                       eq_msd,
                                       p0=[1.,0.0],
                                       bounds=([0,0.0],[np.infty,np.infty]))
                drdt_A = popt[0]
                m=popt[0]
                b=popt[1]
            elif fit_method=='poly_fit':
                par = np.polyfit(time, msd, 1, full=True)
                drdt_A = par[0][0]#0-slope, 1-intercept
                m=par[0][0]
                b=par[0][1]
            elif fit_method=='power_law':
                popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                       time,
```

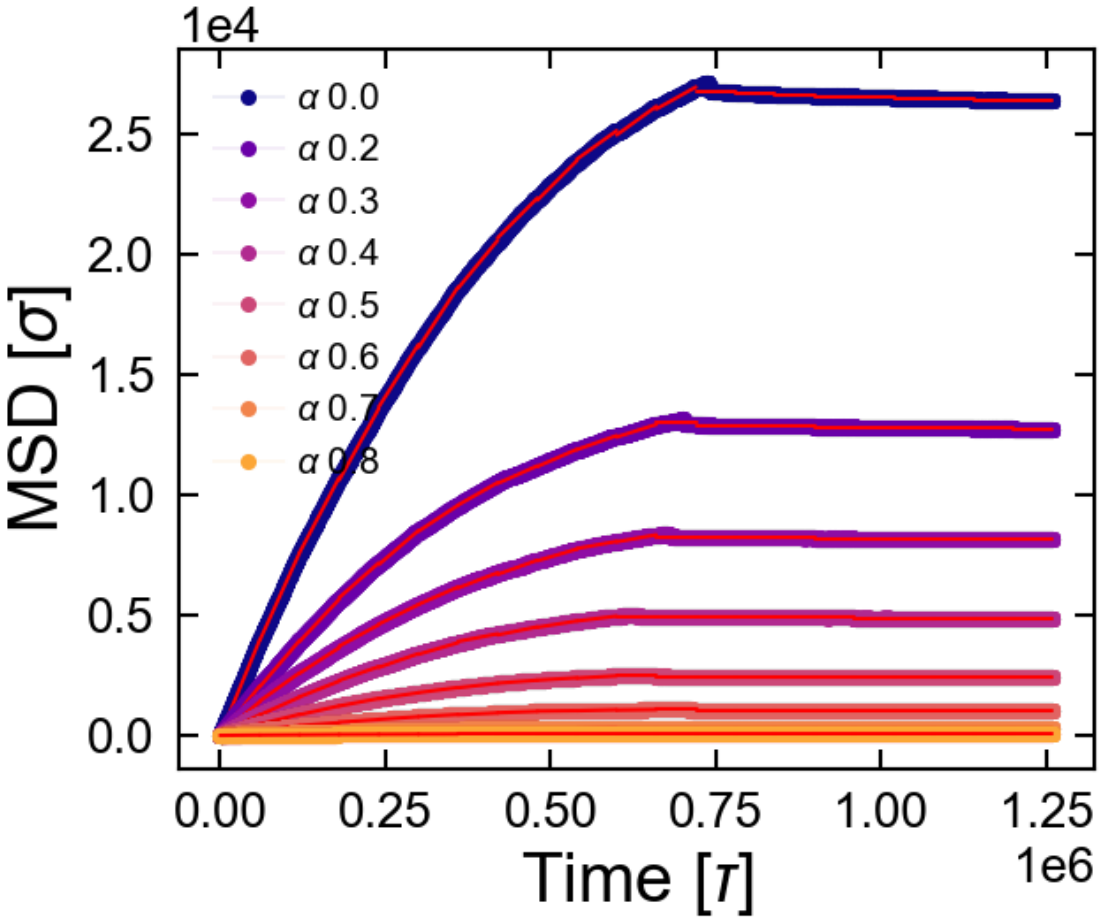
```
                                    msd,
                                    p0=[0.2,1.0],
                                    #p0=[1.0],
                                    #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                    #bounds=([0],[4.0]))
                                    maxfev=2000000,
                                    bounds=([0.0,0.0],[1.0,4.0]))
                raise NotImplementedError('Diffusivity not determined')

            x=time
            y=m*x+b

            #calculate the diffusion coefficient
            dimensions=3
            D_A = drdt_A/(2*dimensions)
            Ts.append(quench_T)
            Ds.append(D_A)
            plt.figure(0)

            if False:
                plt.plot(time,#times[0][0:len(time)],
                         msd,
                         marker='o',
                         markersize=5,
                         color=colors[i],
#label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                         linewidth=0.1)
            if True:
                #print(x,y)
                plt.plot(x,
                         y,
                         color='r',#colors[i],
                         linewidth=1.5)#,
                         #label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))
            #plt.xlim(0.835e8,0.845e8)

            #break
    plt.xlabel('Time [$\\tau$]')
    plt.ylabel('MSD [$\\sigma$]')
    #plt.xlim(87610000.0, 90000000.0)
    #plt.ylim(44000,45000)
    #plt.xscale('log')
    #plt.yscale('log')
    plt.legend(fontsize=15)
    plt.figure(1)
    #print(Ts,Ds)
    Ts=np.asarray(Ts)
    Ds=np.asarray(Ds)
    #Ds[Ds<0]=0
    plt.plot(Ts,
             Ds,
             marker='.',
             color=colors[i],
             linewidth=0,
            label='$\\alpha$ : {}'.format(sap/100))
    D_fit_method='line_fit'#'VLF'#'line_fit','power_law'
    if D_fit_method == 'power_law':
        popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                    Ts,
                                    Ds,
                                    p0=[1.0,1.0],
                                    #p0=[1.0],
                                    #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                    #bounds=([0],[4.0]))
                                    maxfev=2000000,
                                    bounds=([0.0,0.0],[np.infty,4.0]))
        print('popt',popt)
        w=popt[0]
```

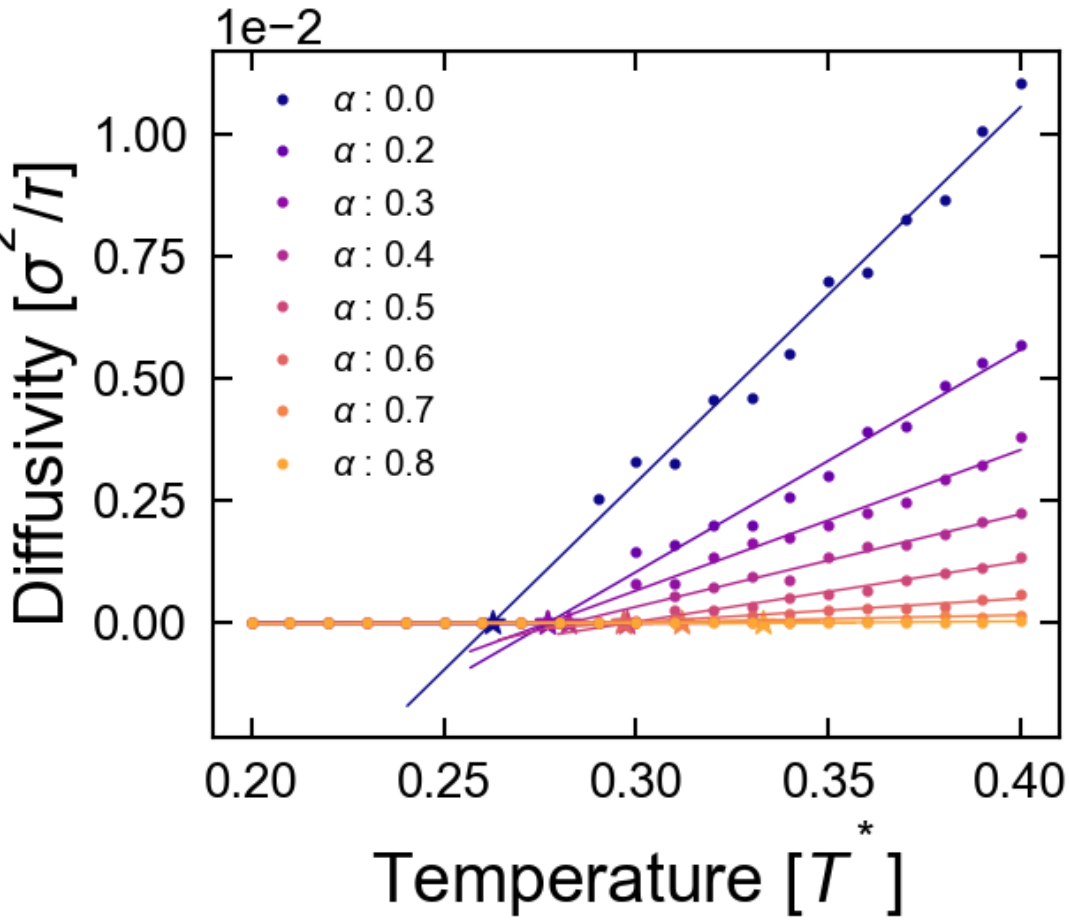
```
    x1=popt[1]
    xs=Ts
    ys=(w*xs)**x1
    To=ys**(1/x1)
    print('To',To,ys)
    plt.plot(xs,
             ys,
             color=colors[i],
             zorder=0,
             linewidth=1)
    #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
    Tgs.append(To)
elif D_fit_method == 'VLF':
    popt, pcov = curve_fit(lambda t,Do,To,B: Do*np.exp(B/(To-t)) ,
                           Ts,
                           Ds,
                           p0=[0.1,1.0,1.0],
                           #p0=[1.0],
                           #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                           #bounds=([0],[4.0]))
                           maxfev=2000000,
                           bounds=([0.0,0.0,0.0],[np.infty,4.0,np.infty]))
    print('popt',popt)
    Do=popt[0]
    To=popt[1]
    B=popt[2]
    xs=Ts
    ys=Do*np.exp(B/(To-xs))
    plt.plot(xs,
             ys,
             color=colors[i],
             zorder=0,
             linewidth=1)
    #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
    #Tgs.append(To)
elif D_fit_method == 'line_fit':
    model = piecewise(Ts, Ds)
    #print('len(model.segments)',len(model.segments))
    if len(model.segments) == 2:
        lines = []
        l1 = model.segments[0]
        m1 = l1.coeffs[1]
        b1 = l1.coeffs[0]
        l2 = model.segments[1]
        m2 = l2.coeffs[1]
        b2 = l2.coeffs[0]
        x,y = line_intersect(m1,b1,m2,b2)
        xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
        ys = l1.coeffs[1]*xs+l1.coeffs[0]
    plt.plot(xs,
             ys,
             color=colors[i],
             zorder=0,
             linewidth=1)
    xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
    ys = l2.coeffs[1]*xs+l2.coeffs[0]
    plt.plot(xs,
             ys,
             color=colors[i],
             zorder=0,
             linewidth=1)
    Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
    Tgs.append(Tg)
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
plt.scatter(Tg,
            Tg_prop,
            marker='*',
            s=100,
```

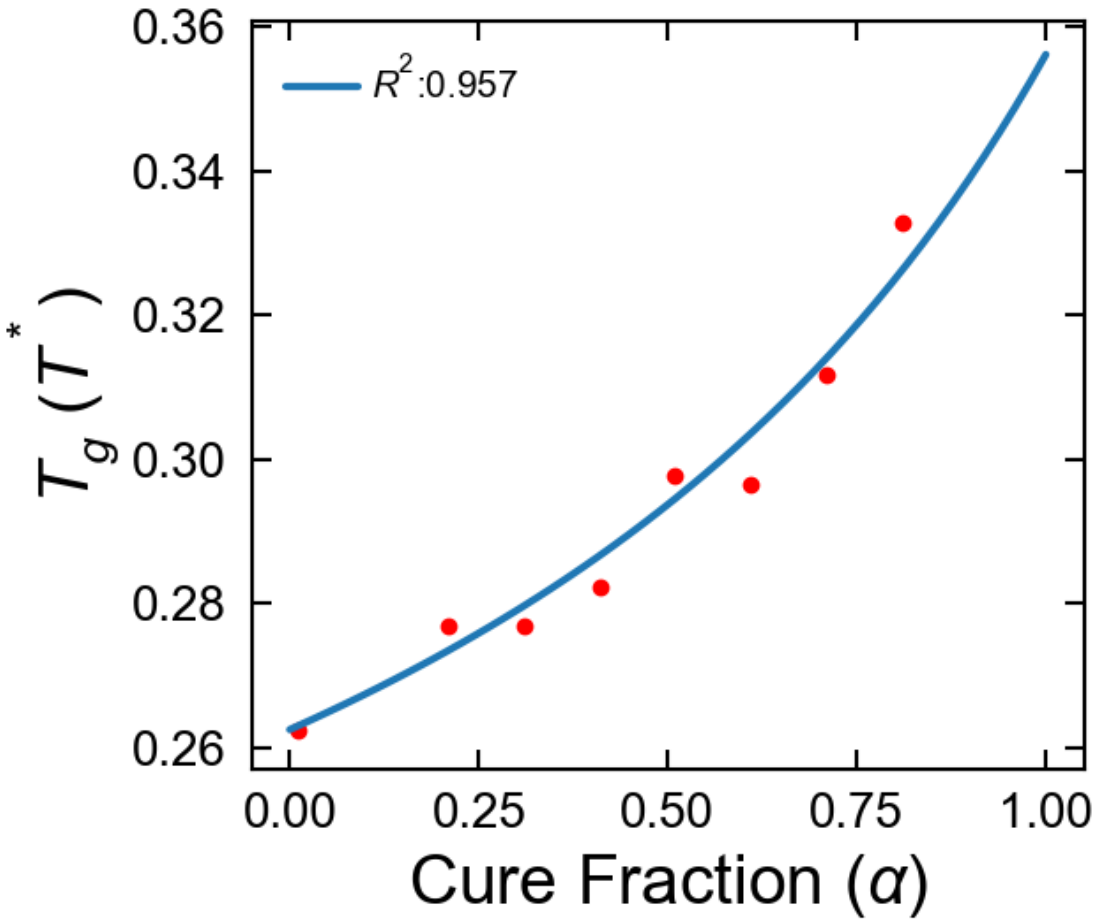
```
                    color=colors[i],
                   zorder=0)#colors[i])
        plt.xlabel('Temperature [$T^*$]')
        plt.ylabel(r'Diffusivity [$\sigma^2/\tau$]')
        #plt.xscale('log')
        #plt.yscale('log')
        #plt.ylim(-1e-8,1e-8)
        plt.legend(fontsize=15)
    plt.savefig('LJ_Diffusivity.png',transparent=True)
    savefig(plt,
           'Coarse_Fine_anneal_DGEBA_DDS_PES_6e6',
           'D_vs_qT.pdf')
    plt.figure(2)
    cure_percents = np.asarray(cure_percents)
    cure_percents_ss = cure_percents#[:-1]
    Tgs_ss = Tgs#[:-1]
    R2,fit_Tgs,T1,inter_parm =
    fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
    plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
    plt.scatter(cure_percents/100.,
                Tgs,
                #label='$E_a$:{}'.format(activation_energy),
                color='r')#colors[i])
    plt.legend(fontsize=20)
    plt.xlabel('Cure Fraction ($\\alpha$)')
    plt.ylabel('$T_g\ (T^*)$')
    plt.legend(fontsize=15)
    plt.savefig('LJ_Tg.png',transparent=True)
    Tg_data = np.asarray([cure_percents/100.,Tgs])
    np.savetxt('DGEBA_DDS_PES_Tg.txt',Tg_data)
    savefig(plt,
           'Coarse_Fine_anneal_DGEBA_DDS_PES_6e6',
           'dibeneditto.pdf')
    #plt.ylim(34000,43000)
    plt.show()
```

```
1.0 f79c0a0626cb5e7e87ac105a80f9f2bd    0.2
Name: quench_T, dtype: object
using line iftting
21.0 a7f6d499313e912f17a5fc83196f4edb    0.2
Name: quench_T, dtype: object
using line iftting
31.0 5663e6302c7c338ec0b0ccfba90fcd52    0.2
Name: quench_T, dtype: object
using line iftting
41.0 587d75da08f1881c40caee3b1c6c4ece    0.2
Name: quench_T, dtype: object
using line iftting
51.0 af39575dfd538b5a658b084611dcd49d    0.2
Name: quench_T, dtype: object
using line iftting
61.0 14a582d8ff7b32e005886ecc12e9c439    0.2
Name: quench_T, dtype: object
using line iftting
71.0 3d49795d5d58f412c89848c04665a1f7    0.2
Name: quench_T, dtype: object
using line iftting
81.0 aaf20b058ecdad475a5349e35eed9f82    0.2
Name: quench_T, dtype: object
```

```
using line iftting
T1 0.356100677422 lambda 0.5
[  1.  21.  31.  41.  51.  61.  71.  81.]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



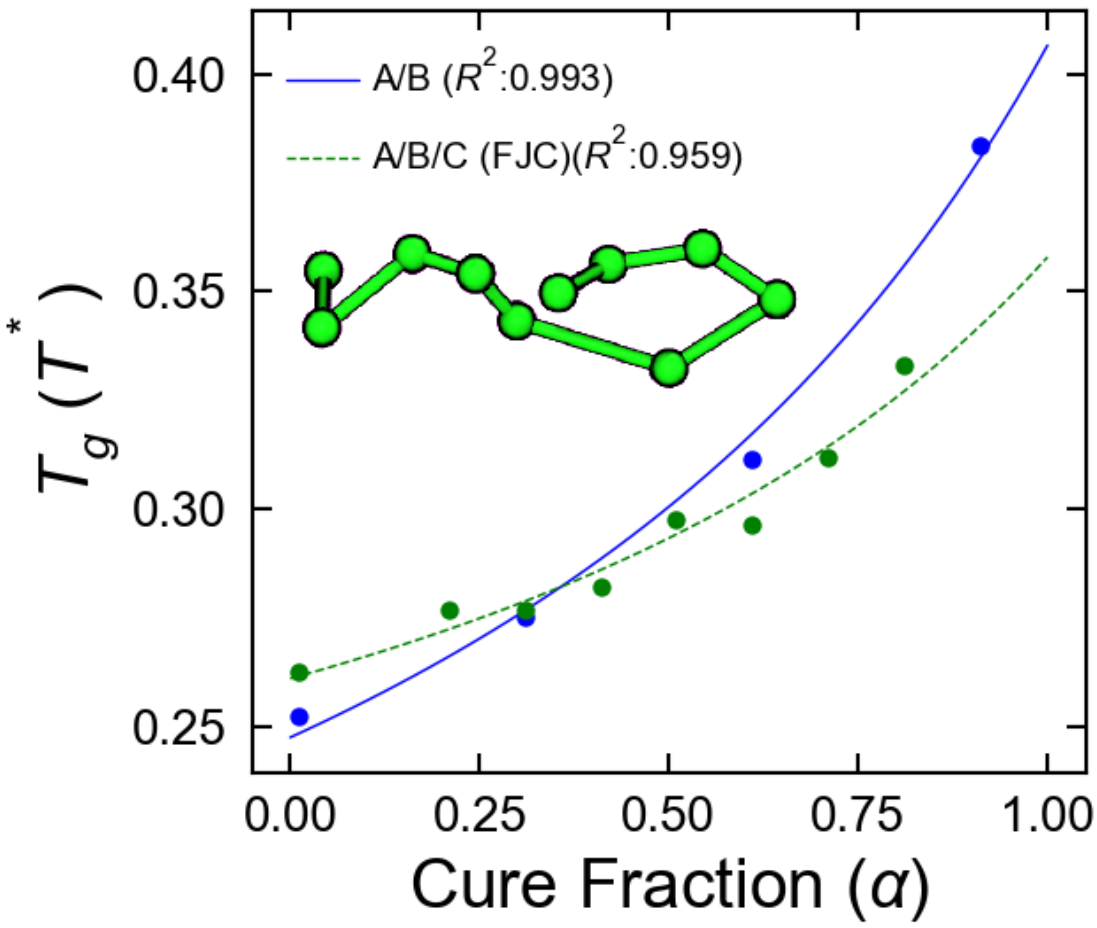

1e−2
1.00
0.75
0.50
0.25
0.00
Diffusivity [σ^2/π]
α : 0.0
α : 0.2
α : 0.3
α : 0.4
α : 0.5
α : 0.6
α : 0.7
α : 0.8
0.20
0.25
0.30
0.35
0.40
Temperature [T^*]




```
In [15]: data_DGEBA_DDS = np.genfromtxt('DGEBA_DDS_Tg_anneal.txt')  #anneal_DGEBA_DDS_LJ.ipynb
         print(data_DGEBA_DDS)
         data_DGEBA_DDS_PES = np.genfromtxt('DGEBA_DDS_PES_Tg.txt')
         print(data_DGEBA_DDS_PES)
         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.3, 0.45, 0.4, 0.4]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')

         alphas = np.linspace(0,1)
         Tgs_ss = data_DGEBA_DDS[:,1]#[:-1]
         print(Tgs_ss)
         cure_fractions = data_DGEBA_DDS[:,0]
         print(cure_fractions)
         R2,fit_Tgs,T1,inter_parm,T0 =
         fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
         print('T1',T1,'lambda',inter_parm)
         fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
         ax1.plot(alphas,
                  fit_ydata,
                 linestyle='-',
                 color='b',
                  linewidth=1,
                  label='A/B ($R^2$:{})'.format(round(R2,3)))
```

```
        Tgs_ss = data_DGEBA_DDS_PES[1]#[:-1]
        cure_fractions = data_DGEBA_DDS_PES[0]
        R2,fit_Tgs,T1,inter_parm,T0 =
        fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
        print('T1',T1,'lambda',inter_parm)
        fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
        ax1.plot(alphas,
                 fit_ydata,
                linestyle='--',
                 linewidth=1,
                color='g',
                 label='A/B/C (FJC)($R^2$:{})'.format(round(R2,3)))


        ax1.plot(data_DGEBA_DDS[:,0],
                 data_DGEBA_DDS[:,1],
                 linewidth=0.0,
                 color='b',
                 marker='o')
        ax1.plot(data_DGEBA_DDS_PES[0],
                 data_DGEBA_DDS_PES[1],
                 linewidth=0.0,
                 color='g',
                 marker='o')
        ax1.legend(fontsize=15)
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g\ (T^*)$')
        im = plt.imread('Coarse_Fine_anneal_DGEBA_DDS_PES_6e6/FJC.png')
        #ax2.set_title('FJC')
        ax2.imshow(im)

        savefig(plt,
               'Coarse_Fine_anneal_DGEBA_DDS_PES_6e6',
               'Tg_with_and_wo_PES.pdf')
        plt.savefig('Tg_with_and_wo_PES.png',transparent=True)
```

```
[[ 0.0100048   0.25224391]
 [ 0.31001381  0.27497143]
 [ 0.61000778  0.31146791]
 [ 0.91000183  0.38378896]]
[[ 0.01        0.21        0.31        0.41        0.51        0.61        0.71
   0.81      ]
 [ 0.26252358  0.27694041  0.27684044  0.2822291   0.29771358  0.29641493
   0.31181878  0.33289583]]
[ 0.25224391  0.27497143  0.31146791  0.38378896]
[ 0.0100048   0.31001381  0.61000778  0.91000183]
T1 0.406699386095 lambda 0.5
T1 0.357950752712 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



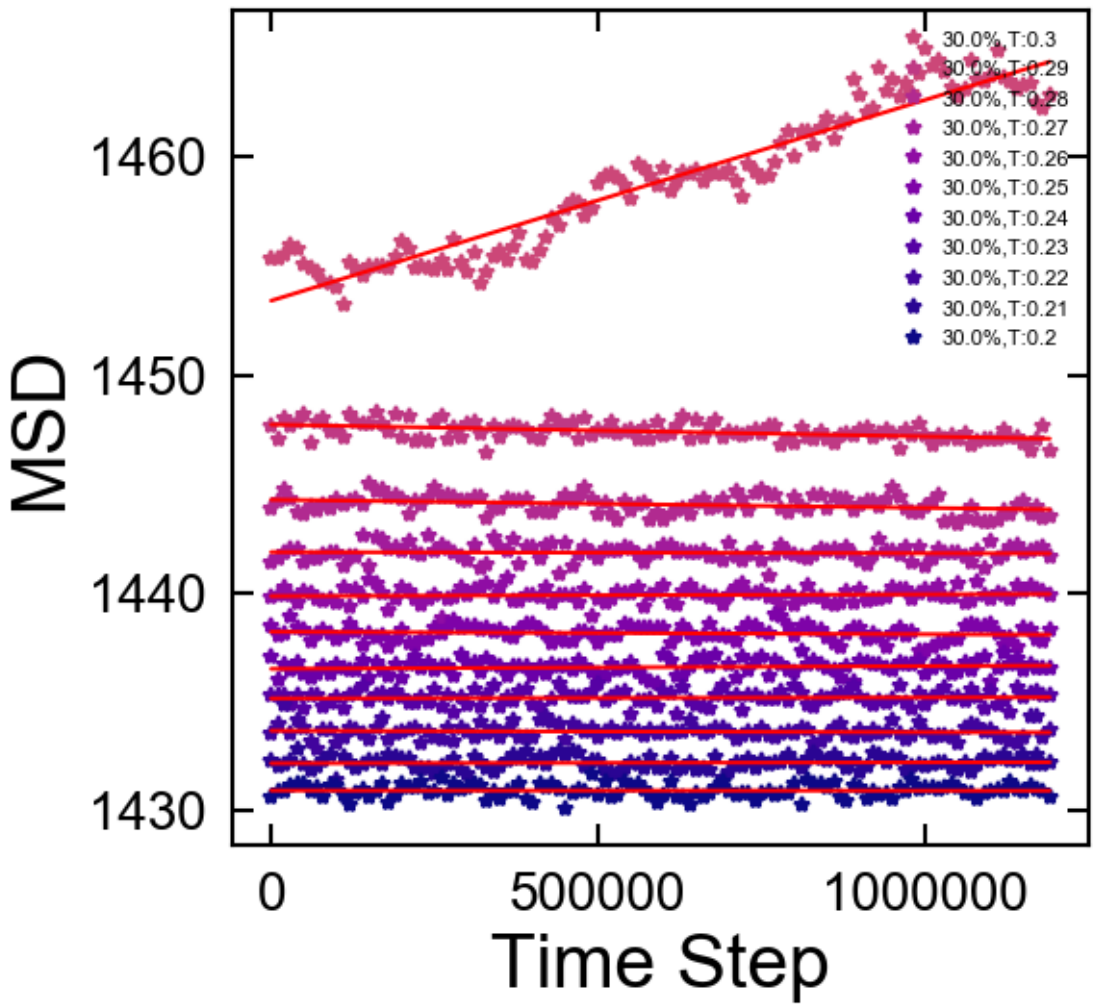

```
In [13]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='cparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()

         for i,sap in enumerate(stop_after_percents):
             filter_saps=[30.0]
             if sap not in filter_saps:
                 continue
             df_curing = df[(df.bond==False)&
                            (df.quench_time==fine_quench_time)&
                            (df.quench_method=='fine_quench')&
                            (df.CC_bond_angle!=109.5)&
                            (df.cooling_method==cooling_method)&
                            (df.stop_after_percent==sap)]
             for job_id in df_curing.index:
                 job = project.open_job(id=job_id)
                 #print(job.sp.P,job.sp.stop_after_percent)
                 times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
                 colors = plt.cm.plasma(np.linspace(1,0,len(msds)))
                 for j,msd in enumerate(msds):
                     #print(len(msd))
                     start_index = int(len(times[j])*0.8)
```

```
            time=times[j][start_index:]
            print('using the last {} timesteps of MSD'.format(len(time)))
            normalized_time = times[0][:len(time)]
            eq_msd = msd[start_index:]
            quench_T = qTs[j]
            if quench_T<0.3:
                if True:
                    #print(time,msd,quench_T)
                    popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                           time,
                                           eq_msd,
                                           p0=[0.2,1.0],
                                           maxfev=2000000,
                                           bounds=([0.0,0.0],[1.0,4.0]))
                    #print(time,eq_msd)
                    #print(time,eq_msd)
                    #print(time[0],time[-1],time[-1]-time[0])
                    popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                           normalized_time,
                                           eq_msd,
                                           p0=[1.,0.0],
                                           bounds=([-1,0.0],[np.infty,np.infty]))

                    # determine the slop using linear regression
                    par = np.polyfit(normalized_time, eq_msd, 1, full=True)
                    drdt_A = popt[0]#par[0][0]#0-slope, 1-intercept
                    x=normalized_time
                    if False:
                        y = popt[0]*x**(popt[1])
                        #y = x**(popt[0])
                    else:
                        y=popt[0]*x+popt[1]
                        #y=par[0][0]*x+par[0][1]
                    #calculate the diffusion coefficient


                    plt.figure(0)

                    if True:
                        #print(x,y)
                        plt.plot(x,
                                 y,
                                 color='r',#colors[i],
                                 linewidth=1.5,
                                 zorder=1)#,
#label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))

                    #Tg,Tg_prop_val = find_Tg(temps,mean_vols,sap)
                    show_transition = False
                    if show_transition:
                        #plt.axvline(x=Tg,
                        #            linewidth=1.0,
                        #            color=colors[i])
                        plt.scatter(Tg,
                                    Tg_prop_val,
                                    marker='*',
                                    s=100,
                                    color='r',
                                   zorder=0)#colors[i])
                    plt.plot(normalized_time,
                             eq_msd,
                             marker='*',
                             color=colors[j],
label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                             zorder=0,
                             linewidth=0.0)
plt.xlabel('Time Step')
plt.ylabel('MSD')
```

```
        #plt.xscale('log')
        #plt.yscale('log')
        plt.legend(fontsize=8)
        #plt.xlim(0.11,0.51)
        #plt.ylim(34000,43000)
        plt.show()
```

```
using the last 121 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
using the last 120 timesteps of MSD
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



<matplotlib.figure.Figure at 0x11cb93320>

```
In [14]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAMES =['aparticles','bparticles','cparticles']#'volume'#'pair_lj_energy','bond_har
         monic_energy'#'potential_energy'
         plt.figure()
         Tgcolors = plt.cm.plasma(np.linspace(0.,0.5,len(PROP_NAMES)))
         colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
         for k,PROP_NAME in enumerate(PROP_NAMES):
             Tgs=[]
             cure_percents = []
             for i,sap in enumerate(stop_after_percents):
                 filter_saps=[0.0,20,30,40,50,60,70,80]#,90]
                 if sap not in filter_saps:
                     continue
                 Ts=[]
                 Ds=[]
```

```
        df_curing = df[(df.bond==False)&
                       (df.quench_time==fine_quench_time)&
                       (df.quench_method=='fine_quench')&
                       (df.CC_bond_angle!=109.5)&
                       (df.cooling_method==cooling_method)&
                       (df.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.values[0]
        cure_percents.append(cure_percent)
        for job_id in df_curing.index:
            job = project.open_job(id=job_id)
            #print(job.sp.P,job.sp.stop_after_percent)
            log_path = job.fn('msd.log')
            data = np.genfromtxt(log_path, names=True)
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            all_time_steps = data['timestep']
            if True:
                plt.figure(0)
                plt.plot(all_time_steps,#times[0][0:len(time)],
                             prop_values,
                             marker='o',
                             markersize=5,
                             color=colors[i],
                             label='{}%'.format(job.sp.stop_after_percent),
                             linewidth=0.1)
            times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
            for j,msd in enumerate(msds):
                start_index = int(len(times[j])*0.6)
                time=times[j]
                quench_T = qTs[j]
                eq_msd = msd[start_index:]
                eq_time = time[start_index:]
                #print(quench_T)
                # determine the slop using linear regression
                fit_method='curve_fit'#'power_law','poly_fit'
                if fit_method=='curve_fit':
                    norm_eq_time = (eq_time-eq_time[0])
                    #print(norm_eq_time,eq_msd)
                    popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                                    eq_time,
                                                    eq_msd,
                                                    p0=[1.,0.0],
bounds=([-1,0.0],[np.infty,np.infty]))
                    drdt_A = popt[0]
                    m=popt[0]
                    b=popt[1]
                elif fit_method=='poly_fit':
                    par = np.polyfit(time, msd, 1, full=True)
                    drdt_A = par[0][0]#0-slope, 1-intercept
                    m=par[0][0]
                    b=par[0][1]
                elif fit_method=='power_law':
                    popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                           time,
                                           msd,
                                           p0=[0.2,1.0],
                                           #p0=[1.0],
#bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                           #bounds=([0],[4.0]))
                                           maxfev=2000000,
                                           bounds=([0.0,0.0],[1.0,4.0]))
                    raise NotImplementedError('Diffusivity not determined')

                x=time
                y=m*x+b

                #calculate the diffusion coefficient
                dimensions=3
                D_A = drdt_A/(2*dimensions)
```

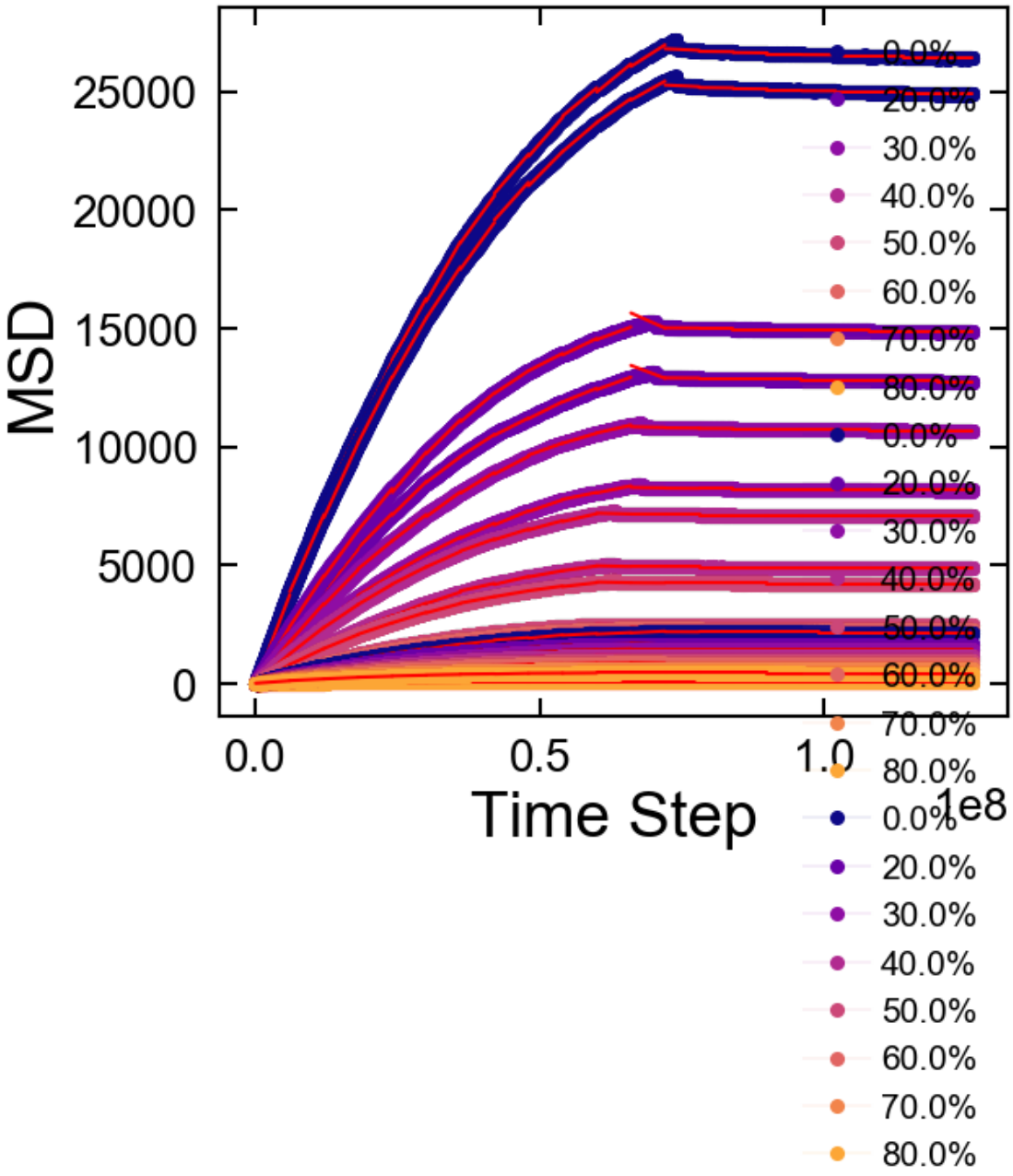
```
                Ts.append(quench_T)
                Ds.append(D_A)
                plt.figure(0)

                if False:
                    plt.plot(time,#times[0][0:len(time)],
                             msd,
                             marker='o',
                             markersize=5,
                             color=colors[i],
#label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                             linewidth=0.1)
                if True:
                    #print(x,y)
                    plt.plot(x,
                             y,
                             color='r',#colors[i],
                             linewidth=1.5)#,
#label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))
                #plt.xlim(0.835e8,0.845e8)

                #break
        plt.xlabel('Time Step')
        plt.ylabel('MSD')
        #plt.xlim(87610000.0, 90000000.0)
        #plt.ylim(44000,45000)
        #plt.xscale('log')
        #plt.yscale('log')
        plt.legend(fontsize=15)
        plt.figure(1)
        #print(Ts,Ds)
        Ts=np.asarray(Ts)
        Ds=np.asarray(Ds)
        #Ds[Ds<0]=0
        plt.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],
                 linewidth=0,
                label=sap)
        D_fit_method='line_fit'#'VLF'#'line_fit','power_law'
        if D_fit_method == 'power_law':
            popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                   Ts,
                                   Ds,
                                   p0=[1.0,1.0],
                                   #p0=[1.0],
#bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0],[np.infty,4.0]))
            print('popt',popt)
            w=popt[0]
            x1=popt[1]
            xs=Ts
            ys=(w*xs)**x1
            To=ys**(1/x1)
            print('To',To,ys)
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            Tgs.append(To)
        elif D_fit_method == 'VLF':
            popt, pcov = curve_fit(lambda t,Do,To,B: Do*np.exp(B/(To-t)) ,
                                   Ts,
```

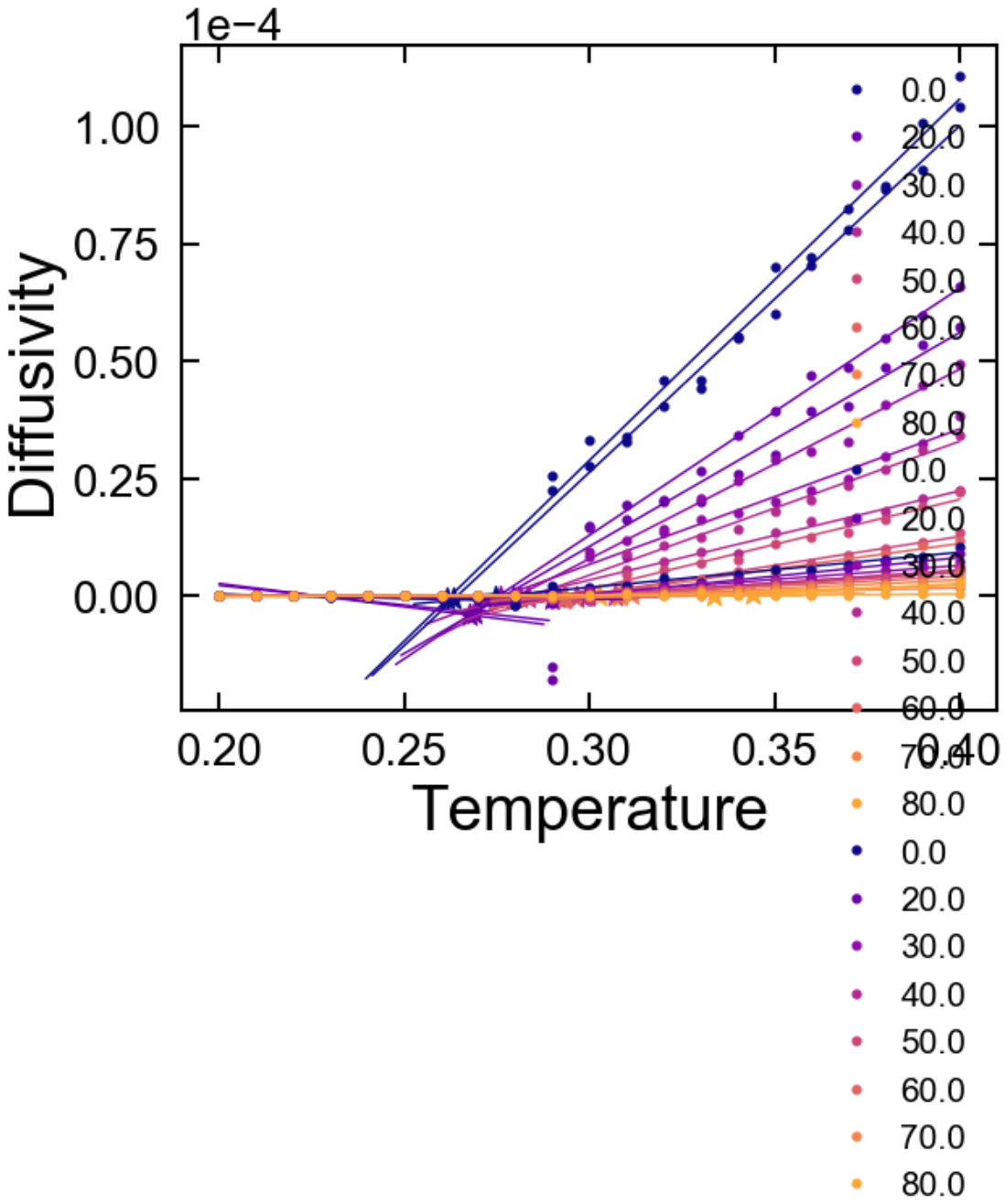
```
                                Ds,
                                p0=[0.1,1.0,1.0],
                                #p0=[1.0],
                                #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                #bounds=([0],[4.0]))
                                maxfev=2000000,
                                bounds=([0.0,0.0,0.0],[np.infty,4.0,np.infty]))
            print('popt',popt)
            Do=popt[0]
            To=popt[1]
            B=popt[2]
            xs=Ts
            ys=Do*np.exp(B/(To-xs))
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            #Tgs.append(To)
        elif D_fit_method == 'line_fit':
            model = piecewise(Ts, Ds)
            #print('len(model.segments)',len(model.segments))
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            Tgs.append(Tg)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=100,
                    color=colors[i],
                   zorder=0)#colors[i])
        plt.xlabel('Temperature')
        plt.ylabel('Diffusivity')
        #plt.xscale('log')
        #plt.yscale('log')
        #plt.ylim(-1e-8,1e-8)
        plt.legend(fontsize=15)


plt.figure(2)
cure_percents = np.asarray(cure_percents)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
R2,fit_Tgs,T1,inter_parm =
```

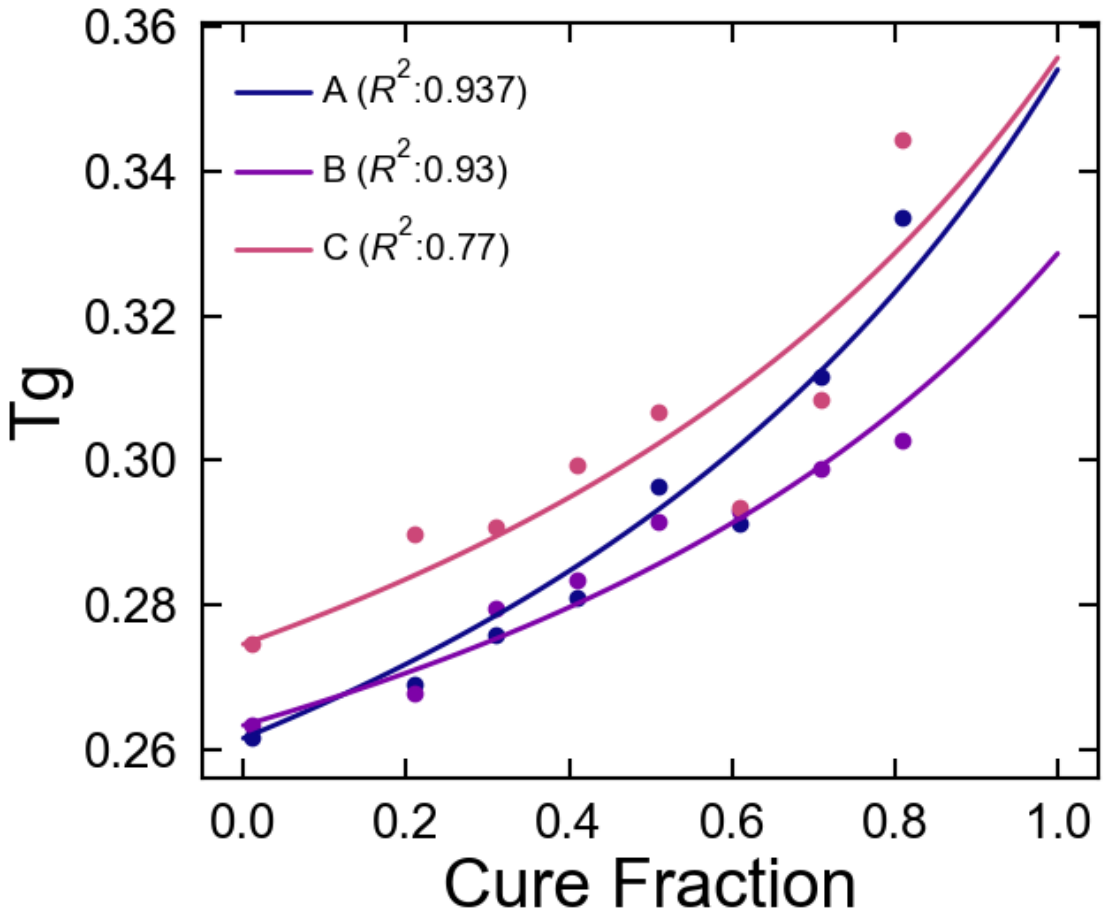
```
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
            print('T1',T1,'lambda',inter_parm)
            #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
            print(cure_percents_ss)
            alphas = np.linspace(0,1)
            fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
            if k==0:
                particle='A'
            elif k==1:
                particle='B'
            elif k==2:
                particle='C'
            plt.plot(alphas,
                     fit_ydata,
                     color=Tgcolors[k],
                     linewidth=2.0,
                     label='{} ($R^2$:{})'.format(particle,round(R2,3)))
            plt.scatter(cure_percents/100.,
                        Tgs,
                        #label='$E_a$:{}'.format(activation_energy),
                        color=Tgcolors[k])#colors[i])
            plt.legend(fontsize=20)
            plt.xlabel('Cure Fraction')
            plt.ylabel('Tg')
            plt.legend(fontsize=15)

        #plt.ylim(34000,43000)
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
T1 0.353997381364 lambda 0.5
[  1.  21.  31.  41.  51.  61.  71.  81.]
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
T1 0.328585190741 lambda 0.5
[  1.  21.  31.  41.  51.  61.  71.  81.]
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
using line iftting
T1 0.355636102352 lambda 0.5
[  1.  21.  31.  41.  51.  61.  71.  81.]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




1e−4
1.00
0.75
0.50
0.25
0.00
Diffusivity
0.20
0.25
0.30
0.35
0.40
Temperature
0.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
0.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
0.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0


A (R^2:0.937)
B (R^2:0.93)
C (R^2:0.77)
Tg
0.36
0.34
0.32
0.30
0.28
0.26
0.0
0.2
0.4
0.6
0.8
1.0
Cure Fraction

In [1]:
```
from common import *
import numpy as np
```

1 Parameters for DGEBA/DDS/PES (LJ) (quench_strategy="coarse_fine")

In [2]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/glass_transition_DGEBA_DDS_PES/'

tau=1
tauP=10
num_c10=0
kT = 1.7
N=50000
use_curing_job_P=False
cooling_method = 'anneal'
integrator='NPT'
pot='LangH'
activation_energy=2.0
density=1.0
quench_time=3e6
coarse_quench_time=3e6
fine_quench_time=6e6
quench_strategy='coarse_fine'
P = 6.0#[4.5, 6, 8]
stop_after_percents = np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.,40.,50.]#[60.,70.,
80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
```

In [3]:
```
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

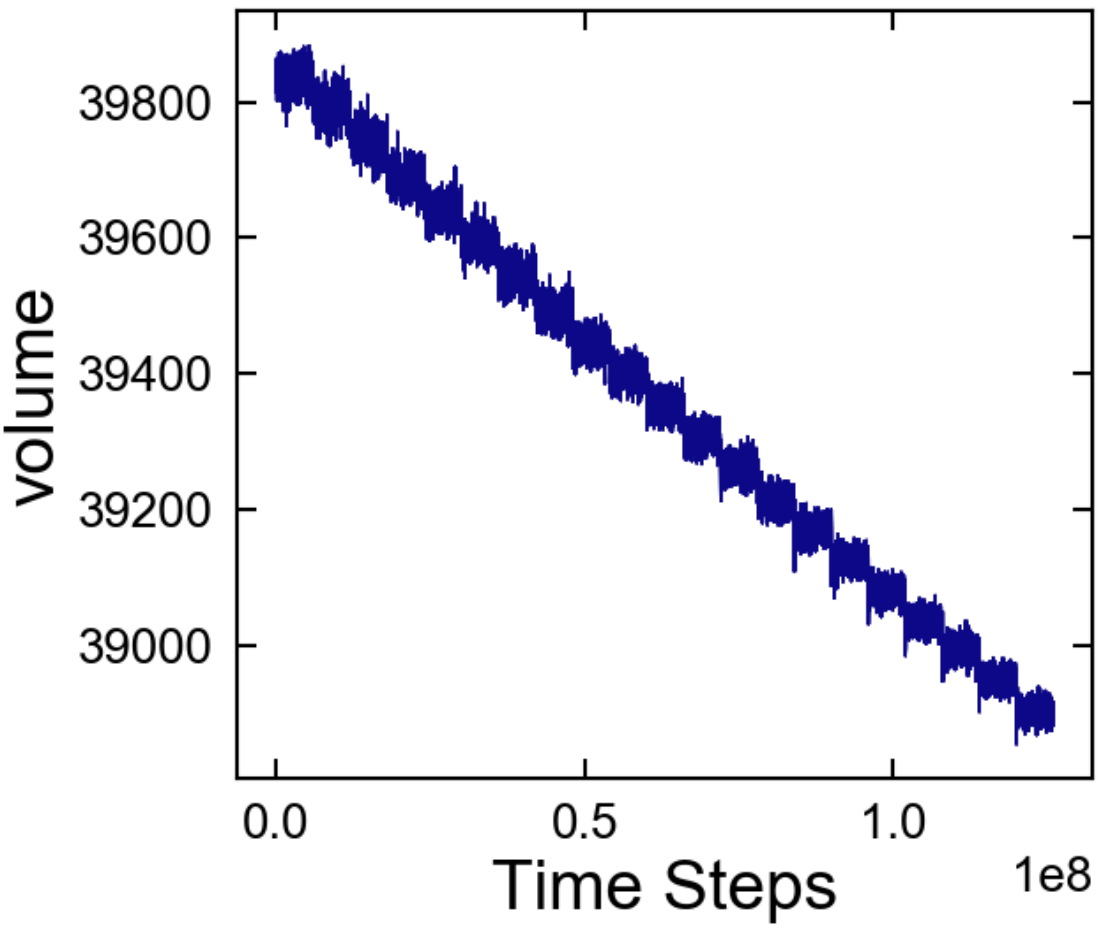
In [4]:
```
df_curing = df[(df.bond==False)&
               (df.quench_time==fine_quench_time)&
               (df.quench_method=='fine_quench')&
               (df.CC_bond_angle==109.5)&
               (df.cooling_method==cooling_method)]#&
                       #(df.stop_after_percent==80.)]
df_curing.quench_temp_prof[-1][-1][0]/df_curing.dcd_write
df_curing.CC_bond_angle_const
#60000000.0/df_curing.dcd_write
```

Out[4]: c88721a9d284845da2754000bb76d46f 25

```
3a03184c70bc3330eb93850126804e46    25
1ccba542af7a2a217511b03c826e604f    25
5c78ffc9384d3e76b797c33f8a92f6f1    25
Name: CC_bond_angle_const, dtype: object
```

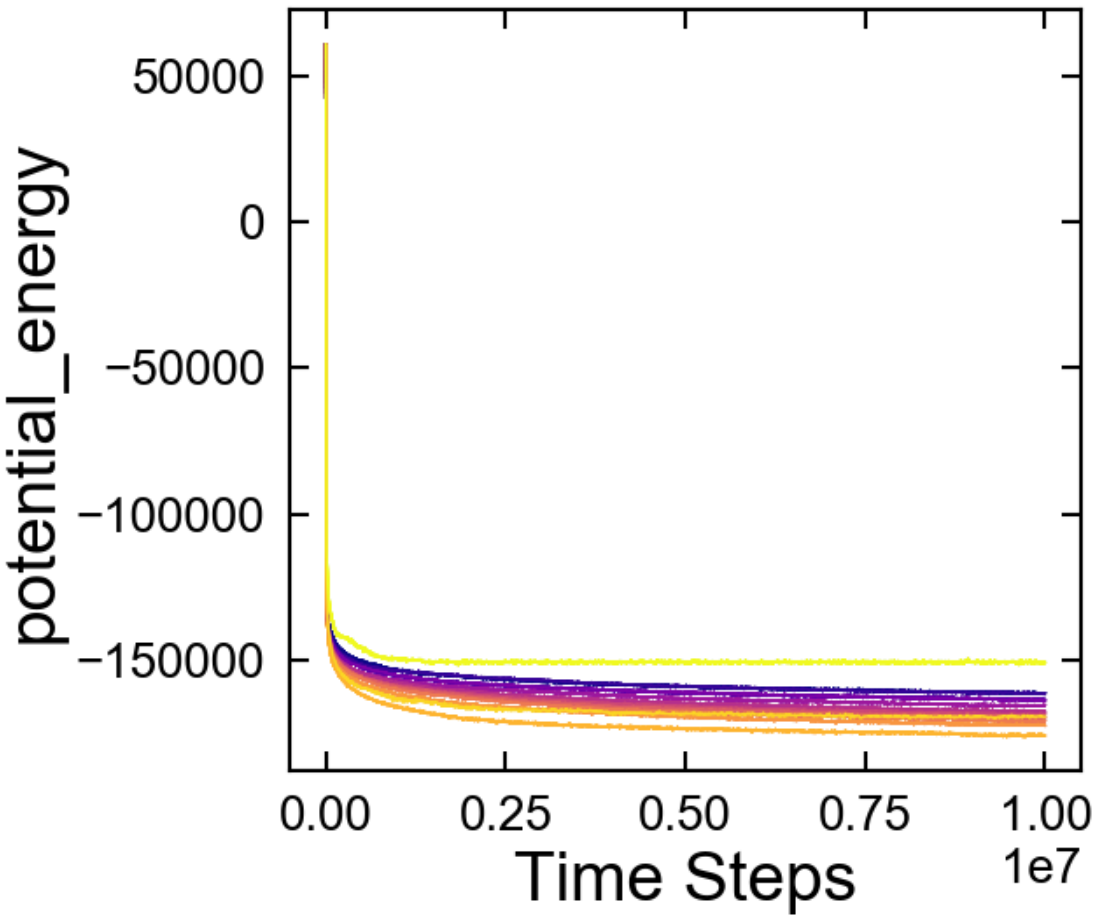
```
In [5]: #stop_after_percents = [50.,80.,100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        df_curing = df[(df.bond==False)&
                       (df.quench_time==fine_quench_time)&
                       #(df.quench_method=='fine_quench')&
                       (df.cooling_method==cooling_method)&
                       (df.CC_bond_angle==109.5)&
                       (df.stop_after_percent==70.)]
        plt.figure()
        for sap in stop_after_percents:
            plot_equilibriation(df_curing,
                                project,
                                PROP_NAME,
                                draw_decorrelated_samples=False,
                                draw_equilibrium_window=False,
                                mean_from_second_half=False)

            #break
        plt.xlabel('Time Steps')
        plt.ylabel(PROP_NAME)
        #plt.legend(fontsize=5)
        #plt.xlim(0,100)
        #plt.ylim(34000,41000)
        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



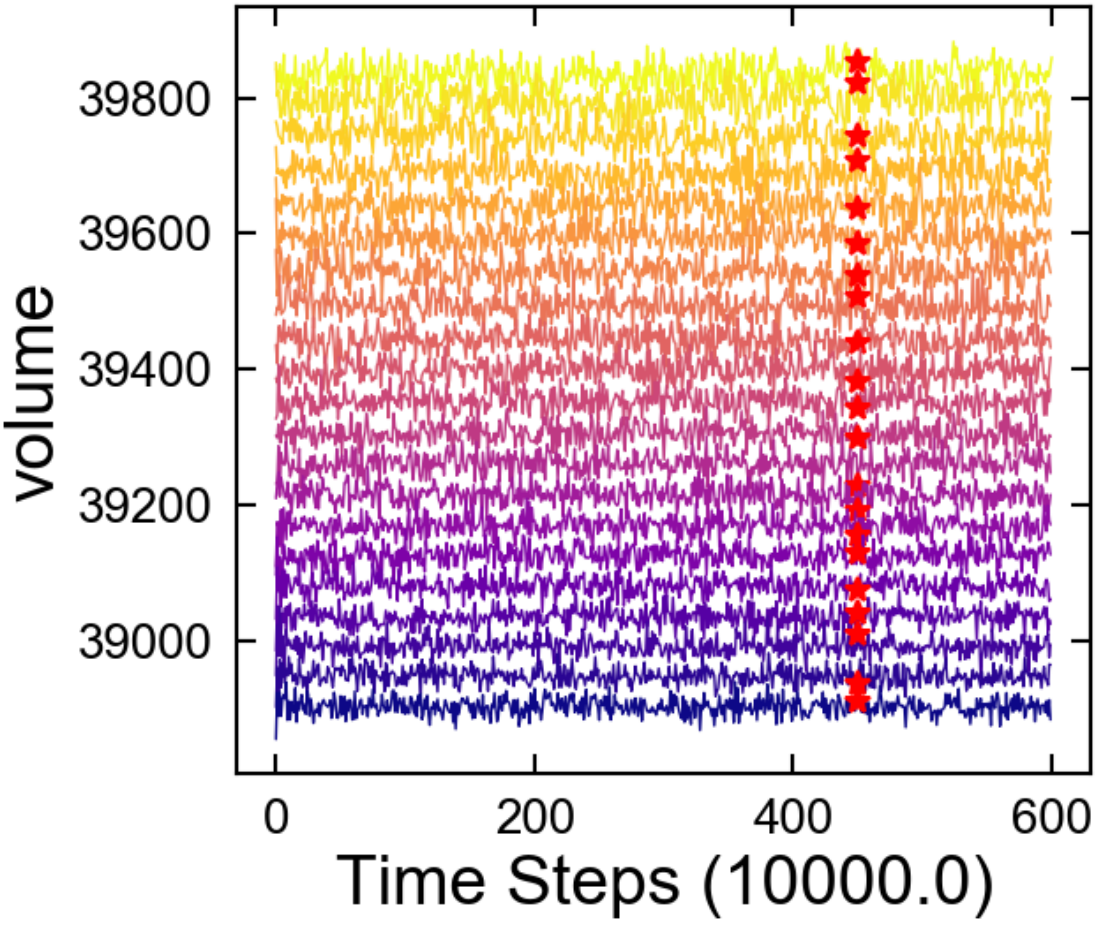
In [6]:
```
#stop_after_percents = [50.,80.,100.]#np.arange(10,105,15,dtype=float)
PROP_NAME
='potential_energy'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
df_curing = df[(df.bond==False)&
               (df.quench_time==1e7)&
               #(df.quench_method=='fine_quench')&
               (df.cooling_method=='quench')&
               (df.CC_bond_angle==109.5)&
               (df.stop_after_percent==70.)]
plt.figure()
for sap in stop_after_percents:
    plot_equilibriation(df_curing,
                        project,
                        PROP_NAME,
                        draw_decorrelated_samples=False,
                        draw_equilibrium_window=False,
                        mean_from_second_half=False)

    #break
plt.xlabel('Time Steps')
plt.ylabel(PROP_NAME)
#plt.legend(fontsize=5)
#plt.xlim(0,100)
#plt.ylim(34000,41000)
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [7]: #stop_after_percents = [100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        for sap in stop_after_percents:
            df_curing = df[(df.bond==False)&
                           (df.quench_time==fine_quench_time)&
                           (df.quench_method=='fine_quench')&
                           (df.CC_bond_angle==109.5)&
                           (df.cooling_method==cooling_method)&
                           (df.stop_after_percent==70.)]
            split_log(df_curing,project,PROP_NAME,filter_temp=1.0,rtol=0.01,show_all=True)
            log_write = df_curing.log_write.values[0]
            break
        plt.xlabel('Time Steps ({})'.format(log_write))
        plt.ylabel(PROP_NAME)
        #plt.legend(fontsize=5)
        #plt.xlim(0,100)
        ##plt.ylim(1,2)
        plt.show()
        quench_time
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



Out[7]: 3000000.0

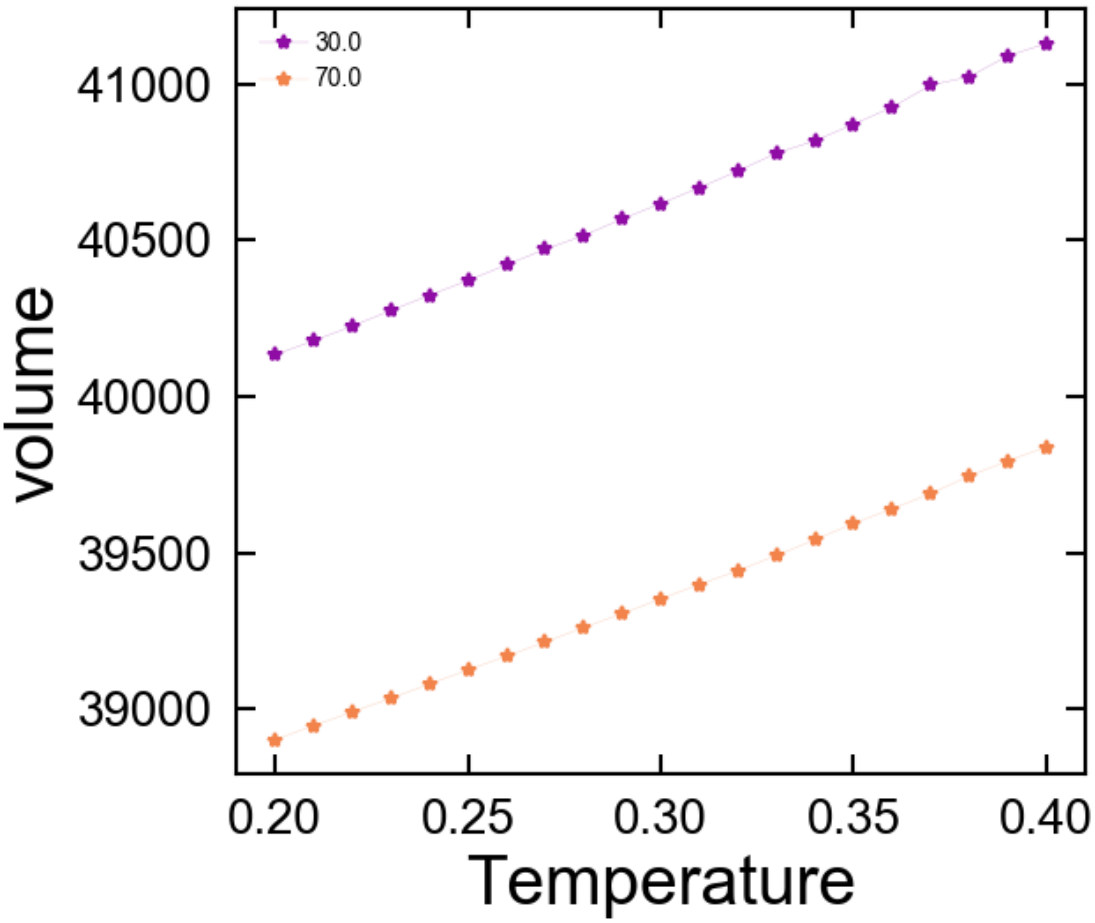
```
In [8]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = [0.,40.,80.,100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
        for i,sap in enumerate(stop_after_percents):
            filter_saps=[30.,70.]#[0.,10.,20.,30.]
            if sap not in filter_saps:
                continue
            df_curing = df[(df.bond==False)&
```

```
                    #(df.tau==tau)&
                    #(df.use_curing_job_P==use_curing_job_P)&
                    #(df.tauP==tauP)&
                    #(df.P==P)&
                    #(df.num_c10==num_c10)&
                    (df.CC_bond_angle==109.5)&
                     (df.quench_time==fine_quench_time)&
                     (df.quench_method=='fine_quench')&
                        (df.cooling_method==cooling_method)&
                        (df.stop_after_percent==sap)]
    for job_id in df_curing.index:
        job = project.open_job(id=job_id)
        #print(job.sp.P,job.sp.stop_after_percent)
        means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
        mean_vols = np.asarray(means)
        density = mean_vols/job.sp.n_particles
        #print(temps,means)

        if False:
            model = piecewise(temps, mean_vols)
            #print(model)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
            else:
                print('WARNING: found more or less than 2 line segments in regression!')

        Tg,Tg_prop_val = find_Tg(temps,mean_vols)
        show_transition = False
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop_val,
                        marker='*',
                        s=100,
                        color='r',
                       zorder=0)#colors[i])
        plt.plot(temps,
                 mean_vols,
                 marker='*',
                 color=colors[i],
                 label=job.sp.stop_after_percent,
                 linewidth=0.1)
plt.xlabel('Temperature')
plt.ylabel(PROP_NAME)
plt.legend(fontsize=10)
```

```
        #plt.xlim(0.11,0.51)
        #plt.ylim(34000,43000)
        plt.show()
```

```
5c78ffc9384d3e76b797c33f8a92f6f1
using line iftting
c88721a9d284845da2754000bb76d46f
using line iftting
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



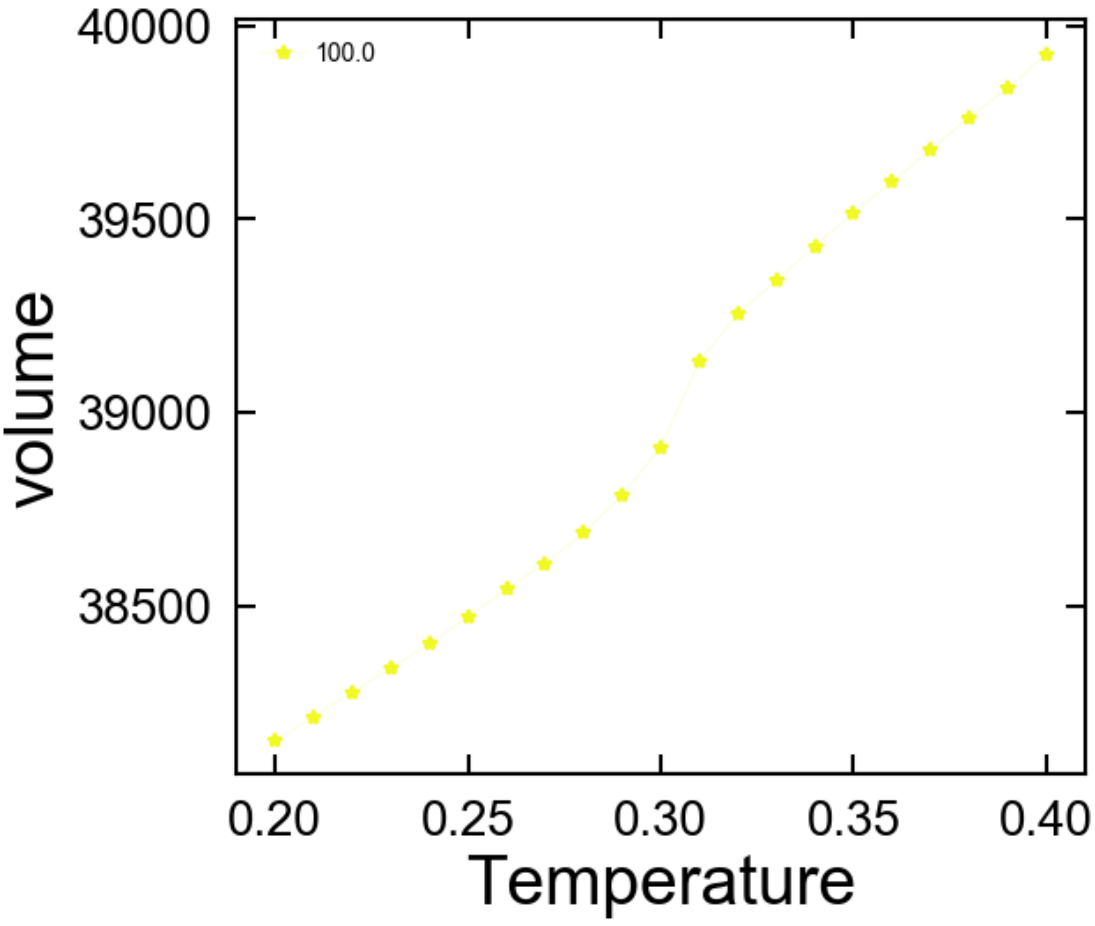

```
In [9]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = [0.,40.,80.,100.]#np.arange(10,105,15,dtype=float)
        PROP_NAME ='volume'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        plt.figure()
        colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
```

```
for i,sap in enumerate(stop_after_percents):
    filter_saps=[100.]
    if sap not in filter_saps:
        continue
    df_curing = df[(df.bond==False)&
                   #(df.tau==tau)&
                   #(df.use_curing_job_P==use_curing_job_P)&
                   #(df.tauP==tauP)&
                   #(df.P==P)&
                   #(df.num_c10==num_c10)&
                    (df.quench_time==fine_quench_time)&
                    (df.quench_method=='fine_quench')&
                   (df.cooling_method==cooling_method)&
                   (df.stop_after_percent==sap)]
    for job_id in df_curing.index:
        job = project.open_job(id=job_id)
        #print(job.sp.P,job.sp.stop_after_percent)
        means,stds,times,temps = get_split_quench_job_property_mean_std(job,PROP_NAME)
        mean_vols = np.asarray(means)
        density = mean_vols/job.sp.n_particles
        #print(temps,means)

        if False:
            model = piecewise(temps, mean_vols)
            #print(model)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
                xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                ys = l2.coeffs[1]*xs+l2.coeffs[0]
                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)
            else:
                print('WARNING: found more or less than 2 line segments in regression!')

        Tg,Tg_prop_val = find_Tg(temps,mean_vols)
        show_transition = False
        if show_transition:
            #plt.axvline(x=Tg,
            #            linewidth=1.0,
            #            color=colors[i])
            plt.scatter(Tg,
                        Tg_prop_val,
                        marker='*',
                        s=100,
                        color='r',
                       zorder=0)#colors[i])
        plt.plot(temps,
                 mean_vols,
                 marker='*',
                 color=colors[i],
                 label=job.sp.stop_after_percent,
```

```
                        linewidth=0.1)
        plt.xlabel('Temperature')
        plt.ylabel(PROP_NAME)
        plt.legend(fontsize=10)
        #plt.xlim(0.11,0.51)
        #plt.ylim(34000,43000)
        plt.show()
```

```
7932520b6d3b660a9f8bc0ddb8a9d0fb
using line iftting
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



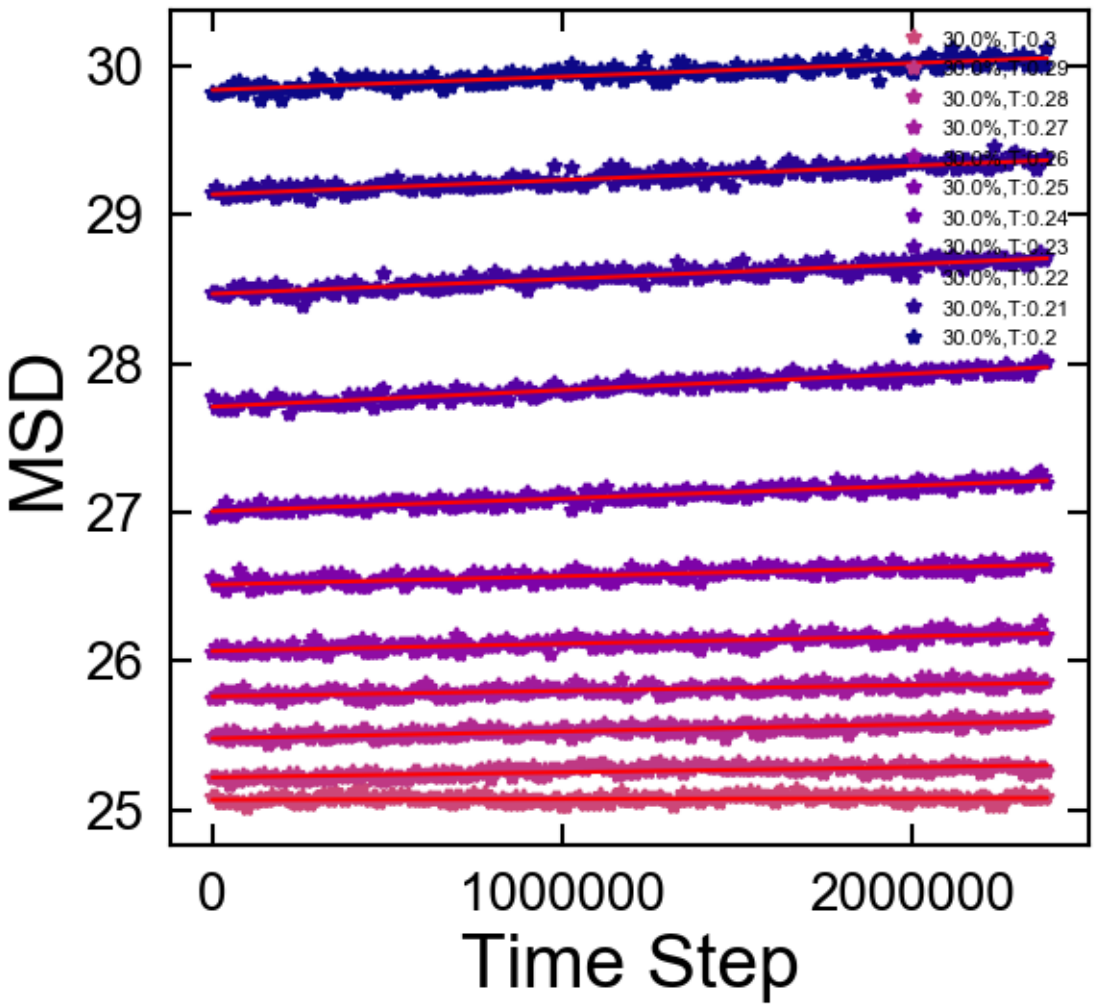

```
In [10]: def get_split_quench_job_msd(job,prop_name):
             times = []
             prop_vals = []
             qTs=[]
             if job.isfile('msd.log'):
```

```
                log_path = job.fn('msd.log')
                data = np.genfromtxt(log_path, names=True)
                PROP_NAME =prop_name
                prop_values = data[PROP_NAME]#'pair_lj_energy']
                time_steps = data['timestep']
                len_prof = len(job.sp.quench_temp_prof)
                for i in range(0,len_prof,2):
                    current_point = job.sp.quench_temp_prof[i]
                    next_point = job.sp.quench_temp_prof[i+1]
                    start_time = current_point[0]
                    end_time = next_point[0]
                    if current_point[1]!=next_point[1]:
                        print('WARNING! Detected a non isothermal step')
                    target_T = current_point[1]
                    #print(time_steps)
                    #print(start_time,end_time)
                    indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
                    start_index = indices[0][0]
                    end_index = indices[0][-1]
                    sliced_ts = time_steps[start_index:end_index+1]
                    sliced_prop_vals = prop_values[start_index:end_index+1]
                    #sliced_pe = pe[start_index:end_index+1]
                    #mean,std = get_mean_and_std(job,sliced_ts,sliced_prop_vals,sliced_pe)
                    #means.append(mean)
                    #stds.append(std)
                    times.append(sliced_ts)
                    prop_vals.append(sliced_prop_vals)
                    qTs.append(target_T)
            return times,prop_vals,qTs

In [11]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()

         for i,sap in enumerate(stop_after_percents):
             filter_saps=[30.0]
             if sap not in filter_saps:
                 continue
             df_curing = df[(df.bond==False)&
                            (df.quench_time==fine_quench_time)&
                            (df.quench_method=='fine_quench')&
                            (df.cooling_method==cooling_method)&
                            (df.CC_bond_angle==109.5)&
                            (df.stop_after_percent==sap)]
             for job_id in df_curing.index:
                 job = project.open_job(id=job_id)
                 #print(job.sp.P,job.sp.stop_after_percent)
                 times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
                 colors = plt.cm.plasma(np.linspace(1,0,len(msds)))
                 for j,msd in enumerate(msds):
                     #print(len(msd))
                     start_index = int(len(times[j])*0.6)
                     time=times[j][start_index:]
                     print('using the last {} timesteps of MSD'.format(len(time)))
                     normalized_time = times[0][:len(time)]
                     eq_msd = msd[start_index:]
                     quench_T = qTs[j]
                     if quench_T<0.3:
                         if True:
                             #print(time,msd,quench_T)
                             popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                                    time,
                                                    eq_msd,
                                                    p0=[0.2,1.0],
```

```
                                        maxfev=2000000,
                                        bounds=([0.0,0.0],[1.0,4.0]))
                        #print(time,eq_msd)
                        #print(time,eq_msd)
                        #print(time[0],time[-1],time[-1]-time[0])
                        popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                        normalized_time,
                                        eq_msd,
                                        p0=[1.,0.0],
                                        bounds=([-1,0.0],[np.infty,np.infty]))

                        # determine the slop using linear regression
                        par = np.polyfit(normalized_time, eq_msd, 1, full=True)
                        drdt_A = popt[0]#par[0][0]#0-slope, 1-intercept
                        x=normalized_time
                        if False:
                            y = popt[0]*x**(popt[1])
                            #y = x**(popt[0])
                        else:
                            y=popt[0]*x+popt[1]
                            #y=par[0][0]*x+par[0][1]
                        #calculate the diffusion coefficient


                        plt.figure(0)

                        if True:
                            #print(x,y)
                            plt.plot(x,
                                     y,
                                     color='r',#colors[i],
                                     linewidth=1.5,
                                     zorder=1)#,
        #label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))

                    #Tg,Tg_prop_val = find_Tg(temps,mean_vols,sap)
                    show_transition = False
                    if show_transition:
                        #plt.axvline(x=Tg,
                        #            linewidth=1.0,
                        #            color=colors[i])
                        plt.scatter(Tg,
                                    Tg_prop_val,
                                    marker='*',
                                    s=100,
                                    color='r',
                                   zorder=0)#colors[i])
                    plt.plot(normalized_time,
                             eq_msd,
                             marker='*',
                             color=colors[j],
        label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                             zorder=0,
                             linewidth=0.0)
        plt.xlabel('Time Step')
        plt.ylabel('MSD')
        #plt.xscale('log')
        #plt.yscale('log')
        plt.legend(fontsize=8)
        #plt.xlim(0.11,0.51)
        #plt.ylim(34000,43000)
        plt.show()
```

```
using the last 241 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
```

```
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD
using the last 240 timesteps of MSD


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



<matplotlib.figure.Figure at 0x11f69fef0>

```
In [12]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         plt.figure()
         colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
         Tgs=[]
         cure_percents = []
         Diffusivities = []
         Cure_Ts=[]
         for i,sap in enumerate(stop_after_percents):
             filter_saps=[0.0,30,50]#,90]
             if sap not in filter_saps:
                 continue
             Ts=[]
             Ds=[]
```

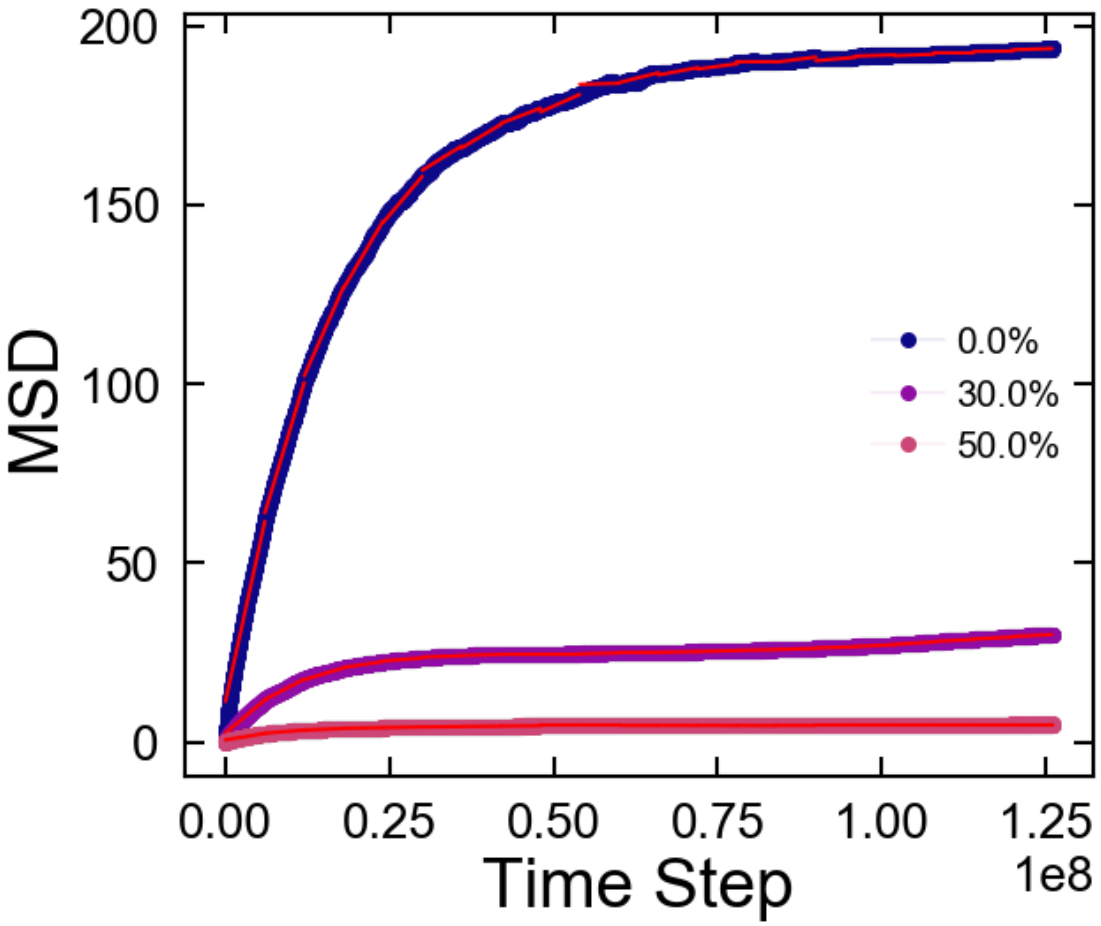
```
df_curing = df[(df.bond==False)&
               (df.quench_time==fine_quench_time)&
               (df.quench_method=='fine_quench')&
               (df.CC_bond_angle==109.5)&
               (df.cooling_method==cooling_method)&
               (df.stop_after_percent==sap)]
cure_percent = df_curing.cure_percent.values[0]
cure_percents.append(cure_percent)
for job_id in df_curing.index:
    job = project.open_job(id=job_id)
    #print(job.sp.P,job.sp.stop_after_percent)
    log_path = job.fn('msd.log')
    data = np.genfromtxt(log_path, names=True)
    prop_values = data[PROP_NAME]#'pair_lj_energy']
    all_time_steps = data['timestep']
    if True:
        plt.figure(0)
        plt.plot(all_time_steps,#times[0][0:len(time)],
                     prop_values,
                     marker='o',
                     markersize=5,
                     color=colors[i],
                     label='{}%'.format(job.sp.stop_after_percent),
                     linewidth=0.1)
    times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
    for j,msd in enumerate(msds):
        start_index = int(len(times[j])*0.6)
        time=times[j]
        quench_T = qTs[j]
        eq_msd = msd[start_index:]
        eq_time = time[start_index:]
        #print(quench_T)
        # determine the slop using linear regression
        fit_method='curve_fit'#'power_law','poly_fit'
        if fit_method=='curve_fit':
            norm_eq_time = (eq_time-eq_time[0])
            #print(norm_eq_time,eq_msd)
            popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                           eq_time,
                                           eq_msd,
                                           p0=[1.,0.0],
                                           bounds=([-1,0.0],[np.infty,np.infty]))
            drdt_A = popt[0]
            m=popt[0]
            b=popt[1]
        elif fit_method=='poly_fit':
            par = np.polyfit(time, msd, 1, full=True)
            drdt_A = par[0][0]#0-slope, 1-intercept
            m=par[0][0]
            b=par[0][1]
        elif fit_method=='power_law':
            popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                   time,
                                   msd,
                                   p0=[0.2,1.0],
                                   #p0=[1.0],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0],[1.0,4.0]))
            raise NotImplementedError('Diffusivity not determined')

        x=time
        y=m*x+b

        #calculate the diffusion coefficient
        dimensions=3
        D_A = drdt_A/(2*dimensions)
```

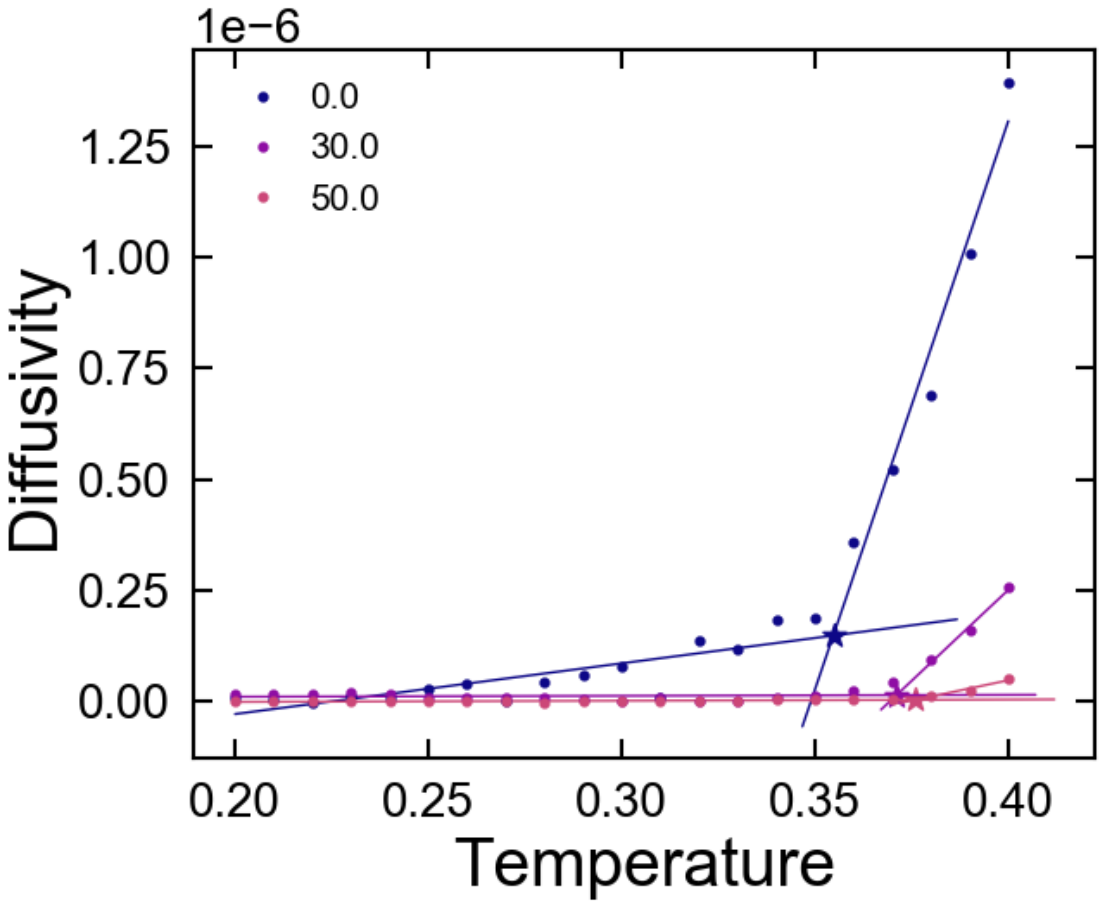
```
            Ts.append(quench_T)
            Ds.append(D_A)
            plt.figure(0)

            if False:
                plt.plot(time,#times[0][0:len(time)],
                         msd,
                         marker='o',
                         markersize=5,
                         color=colors[i],
#label='{}%,T:{}'.format(job.sp.stop_after_percent,round(qTs[j],3)),
                         linewidth=0.1)
            if True:
                #print(x,y)
                plt.plot(x,
                         y,
                         color='r',#colors[i],
                         linewidth=1.5)#,
                         #label='qT:{},x1:{}'.format(job.sp.quench_T,round(popt[1],3)))
            #plt.xlim(0.835e8,0.845e8)

            #break
    plt.xlabel('Time Step')
    plt.ylabel('MSD')
    #plt.xlim(87610000.0, 90000000.0)
    #plt.ylim(44000,45000)
    #plt.xscale('log')
    #plt.yscale('log')
    plt.legend(fontsize=15)
    plt.figure(1)
    #print(Ts,Ds)
    Ts=np.asarray(Ts)
    Ds=np.asarray(Ds)
    Cure_Ts.append(Ts)
    Diffusivities.append(Ds)
    #Ds[Ds<0]=0
    plt.plot(Ts,
             Ds,
             marker='.',
             color=colors[i],
             linewidth=0,
            label=sap)
    D_fit_method='line_fit'#'VLF'#'line_fit','power_law'
    if D_fit_method == 'power_law':
        popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                    Ts,
                                    Ds,
                                    p0=[1.0,1.0],
                                    #p0=[1.0],
                                    #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                    #bounds=([0],[4.0]))
                                    maxfev=2000000,
                                    bounds=([0.0,0.0],[np.infty,4.0]))
        print('popt',popt)
        w=popt[0]
        x1=popt[1]
        xs=Ts
        ys=(w*xs)**x1
        To=ys**(1/x1)
        print('To',To,ys)
        plt.plot(xs,
                 ys,
                 color=colors[i],
                 zorder=0,
                 linewidth=1)
        #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
        Tgs.append(To)
    elif D_fit_method == 'VLF':
```

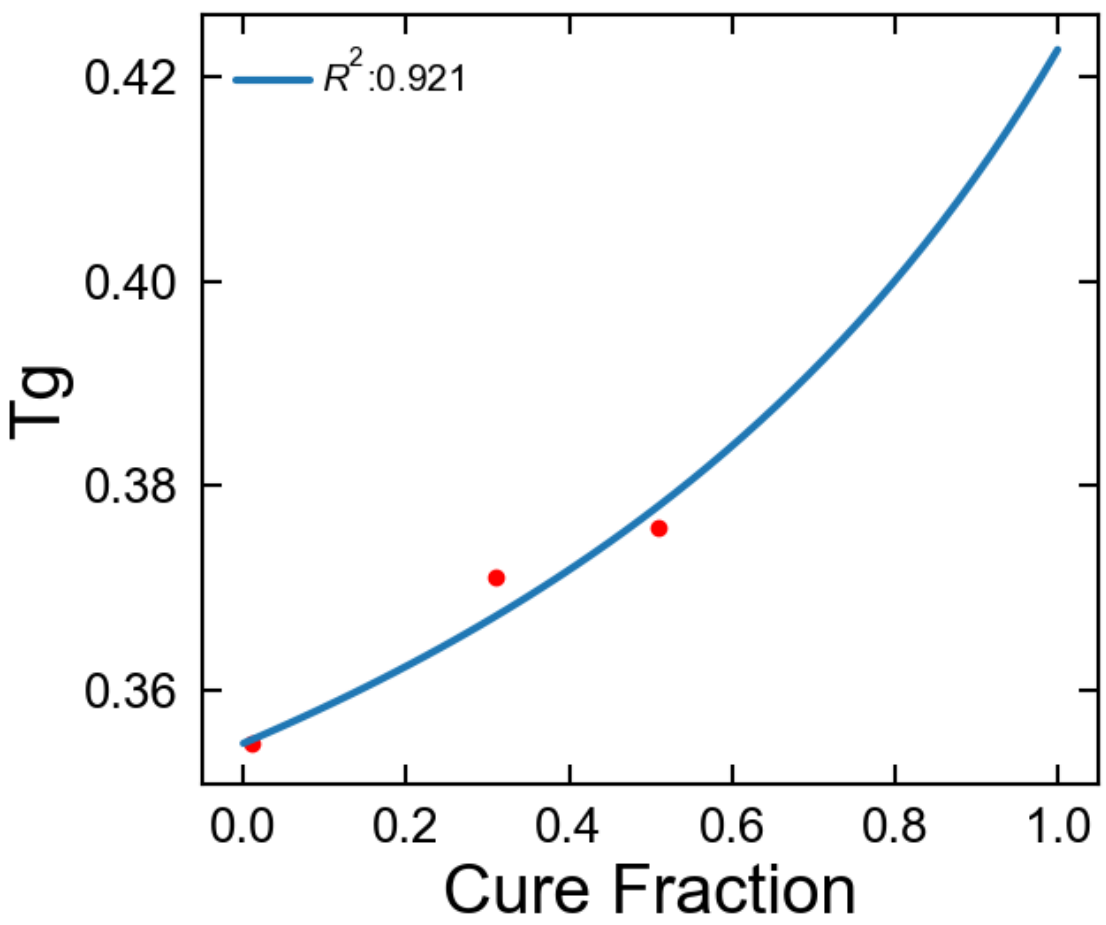
```
            popt, pcov = curve_fit(lambda t,Do,To,B: Do*np.exp(B/(To-t)) ,
                                   Ts,
                                   Ds,
                                   p0=[0.1,1.0,1.0],
                                   #p0=[1.0],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0,0.0],[np.infty,4.0,np.infty]))
            print('popt',popt)
            Do=popt[0]
            To=popt[1]
            B=popt[2]
            xs=Ts
            ys=Do*np.exp(B/(To-xs))
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            #Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            #Tgs.append(To)
        elif D_fit_method == 'line_fit':
            model = piecewise(Ts, Ds)
            #print('len(model.segments)',len(model.segments))
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
                x,y = line_intersect(m1,b1,m2,b2)
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            Tgs.append(Tg)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=100,
                    color=colors[i],
                   zorder=0)#colors[i])
        plt.xlabel('Temperature')
        plt.ylabel('Diffusivity')


        #plt.xscale('log')
        #plt.yscale('log')
        #plt.ylim(-1e-8,1e-8)
        plt.legend(fontsize=15)


    plt.figure(2)
```

```
        cure_percents = np.asarray(cure_percents)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        plt.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
        plt.legend(fontsize=20)
        plt.xlabel('Cure Fraction')
        plt.ylabel('Tg')
        plt.legend(fontsize=15)

        #plt.ylim(34000,43000)
        plt.show()
```

```
using line iftting
using line iftting
using line iftting
T1 0.422695870723 lambda 0.5
[  1.  31.  51.]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


200
150
100
50
0
MSD
0.0%
30.0%
50.0%
0.00
0.25
0.50
0.75
1.00
1.25
1e8
Time Step


1e−6
0.0
30.0
50.0
1.25
1.00
0.75
0.50
0.25
0.00
Diffusivity
0.20
0.25
0.30
0.35
0.40
Temperature




```
In [13]: def Fit_Diffusivity(Ts,Ds,method='use_viscous_region',min_D=1e-8):
             indices = np.where(Ds>min_D)#0.00000095)
             start_index = indices[0][0]
             D_As=Ds[start_index:]
             quenchTs=Ts[start_index:]
             model = piecewise(quenchTs, D_As)
             if len(model.segments) == 2:
                 lines = []
                 l1 = model.segments[0]
                 m1 = l1.coeffs[1]
                 b1 = l1.coeffs[0]
                 l2 = model.segments[1]
                 m2 = l2.coeffs[1]
                 b2 = l2.coeffs[0]
                 x,y = line_intersect(m1,b1,m2,b2)
                 xs1 =
         np.linspace(l1.start_t,l1.end_t)#np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                 ys1 = l1.coeffs[1]*xs1+l1.coeffs[0]
                 xs2 =
         np.linspace(l2.start_t,l2.end_t)#np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
                 ys2 = l2.coeffs[1]*xs2+l2.coeffs[0]
             else:
                 print('WARNING: found {} line segments in
         regression!'.format(len(model.segments)))

             if method=='use_viscous_region':
```

```
            Tg = -b2/m2
            Tg_prop = 0.
        else:
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)

        return Tg,Tg_prop,xs1,ys1,xs2,ys2

    def Calc_Diffusivity(eq_time,
                         eq_msd,
                         fit_method='curve_fit'):
        #fit_method='curve_fit'#'power_law','poly_fit'
        if fit_method=='curve_fit':
            norm_eq_time = (eq_time-eq_time[0])
            #print(norm_eq_time,eq_msd)
            popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                           eq_time,
                                           eq_msd,
                                           p0=[1.,0.0],
                                           bounds=([-1,0.0],[np.infty,np.infty]))
            drdt_A = popt[0]
            m=popt[0]
            b=popt[1]
        elif fit_method=='poly_fit':
            par = np.polyfit(time, msd, 1, full=True)
            drdt_A = par[0][0]#0-slope, 1-intercept
            m=par[0][0]
            b=par[0][1]
        elif fit_method=='power_law':
            popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                   time,
                                   msd,
                                   p0=[0.2,1.0],
                                   #p0=[1.0],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0],[1.0,4.0]))
            raise NotImplementedError('Diffusivity not determined')

        x=time
        y=m*x+b

        #calculate the diffusion coefficient
        dimensions=3
        D = drdt_A/(2*dimensions)
        return D,m,b
```

In [14]:
```
df_curing = df[(df.bond==False)&
                       (df.quench_T<=0.5)&
                       #(df.quench_time==2e7)&
                       (df.CC_bond_angle==109.5)&
                       #(df.cooling_method=='quench')&
                       (df.stop_after_percent==0)]
df_curing.cure_percent
```

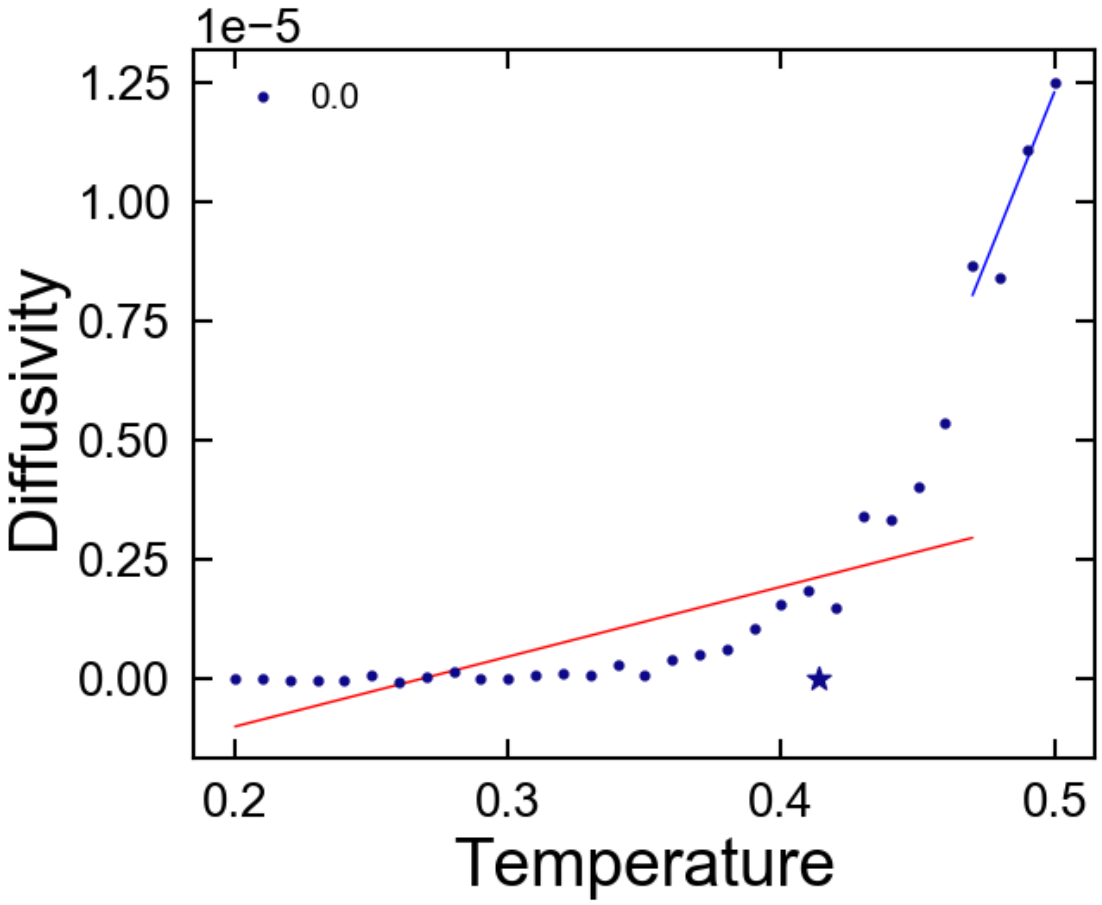
Out[14]:
```
aae3f001ba728dcbaaabee92497e9dff    1.0
9563dae83c6421ee974fe4fc7736b33e    1.0
fb527158ca2e63455aefd6257ca2d8f1    1.0
d20cf362ebe3323f67bf203e193e0e05    1.0
c5b597199afe3a2c56c4ba0e6fb5e763    1.0
de561ba324039652ef569eaa3ce58d76    1.0
dce9af6aa2dbed7fa9a2906cdb5e85cb    1.0
3a03184c70bc3330eb93850126804e46    1.0
025bf7aa039207802c8ad9ffed019895    1.0
```

```
1ab83e3d0583af6336f62e74cf07daeb    1.0
7816f41bd6749105682a886f6353dc74    1.0
4b184d33804a3486677b7824d4116b6e    1.0
Name: cure_percent, dtype: float64
```

In [37]:
```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='aparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
plt.figure()
colors = plt.cm.plasma(np.linspace(0,1,len(stop_after_percents)))
Tgs=[]
cure_percents = []
Diffusivities = []
Cure_Ts=[]
for i,sap in enumerate(stop_after_percents):
    filter_saps=[0]#[0.0,30,50,70]#,90]
    if sap not in filter_saps:
        continue
    Ts=[]
    Ds=[]

    df_curing = df[(df.bond==False)&
                   (df.quench_T<=0.5)&
                   #(df.quench_time==2e7)&
                   (df.CC_bond_angle==109.5)&
                   #(df.cooling_method=='anneal')&
                   (df.stop_after_percent==sap)]
    cure_percent = df_curing.cure_percent.values[0]
    cure_percents.append(cure_percent)
    for key,df_grp in df_curing.groupby('cooling_method'):

        if key=='quench':
            df_filt = df_grp[(df_grp.quench_time==1e7)]
        else:
            df_filt = df_grp

        for job_id in df_filt.index:
            job = project.open_job(id=job_id)
            #print(job)
            log_path = job.fn('msd.log')
            data = np.genfromtxt(log_path, names=True)
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            all_time_steps = data['timestep']
            equilibriated_ts_percentage = 0.9
            if key=='anneal':
                times,msds,qTs = get_split_quench_job_msd(job,PROP_NAME)
                for j,msd in enumerate(msds):
                    start_index = int(len(times[j])*equilibriated_ts_percentage)
                    time=times[j]
                    quench_T = qTs[j]
                    eq_msd = msd[start_index:]
                    eq_time = time[start_index:]
                    D_A,m,b = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                    Ts.append(quench_T)
                    Ds.append(D_A)
            else:
                start_index = int(len(all_time_steps)*equilibriated_ts_percentage)
                time=all_time_steps
                quench_T = job.sp.quench_T
                eq_msd = prop_values[start_index:]
                eq_time = time[start_index:]
                D_A,m,b = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                Ts.append(quench_T)
                Ds.append(D_A)
```

```
            Ts=np.asarray(Ts)
            Ds=np.asarray(Ds)
            Cure_Ts.append(Ts)
            Diffusivities.append(Ds)
            #print(Ds)
            Tg,Tg_prop,xs1,ys1,xs2,ys2 = Fit_Diffusivity(Ts,
                                                         Ds,
                                                         method='use_viscous_region',
                                                         min_D=0)#0)
            Tgs.append(Tg)
            if True:
                plt.plot(xs1,
                         ys1,
                         color='r',#colors[i],
                         zorder=0,
                         linewidth=1)
                plt.plot(xs2,
                         ys2,
                         color='b',#colors[i],
                         zorder=0,
                         linewidth=1)

            #print(Ds)
            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],
                     linewidth=0.0,
                    label=sap)

            plt.scatter(Tg,
                        Tg_prop,
                        marker='*',
                        s=100,
                        color=colors[i],
                       zorder=0)#colors[i])
            plt.legend(fontsize=15)
            plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
            plt.xlabel('Temperature')
            plt.ylabel('Diffusivity')
            #plt.ylim(-0.5e-7,.5e-5)
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



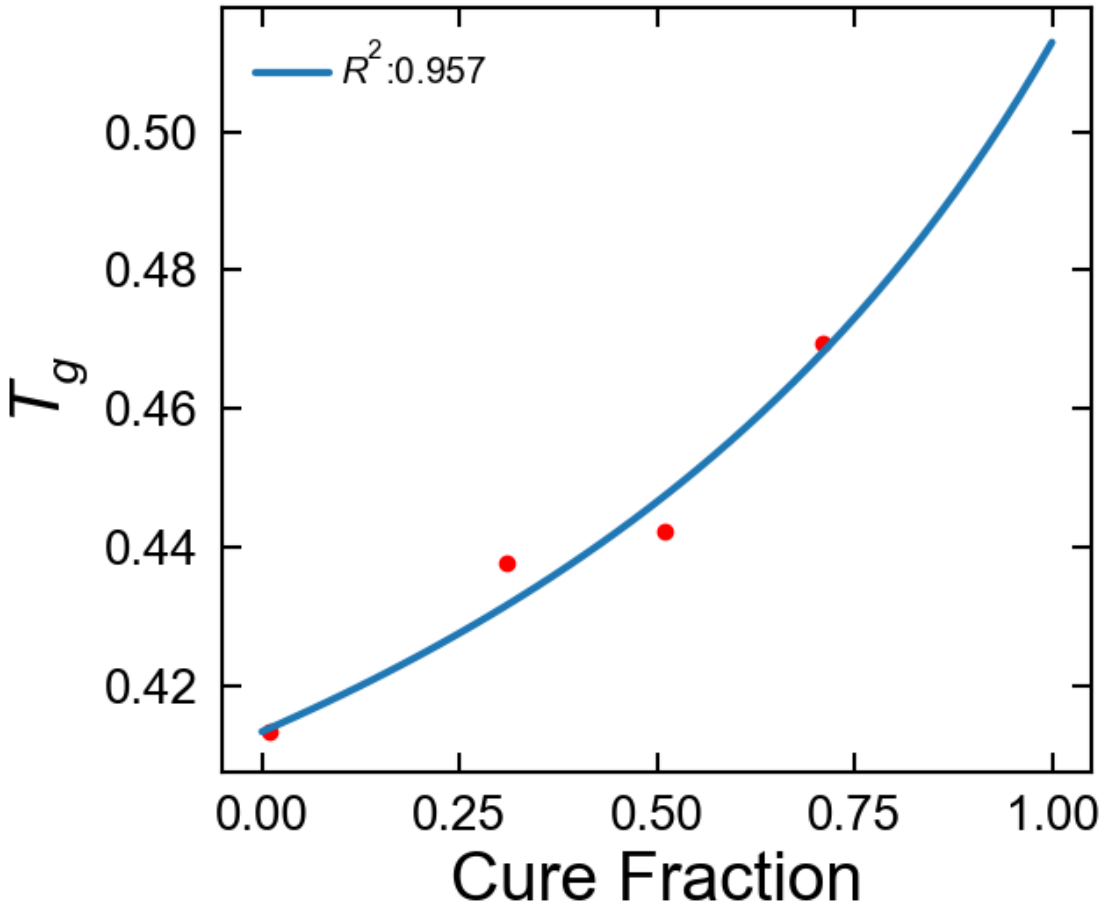
In [27]:
```
cure_percents = np.asarray(cure_percents)
Tgs = np.asarray(Tgs)
Tg_data = np.asarray([cure_percents/100.,Tgs])
np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
print(cure_percents_ss)
print(Tgs_ss)
R2,fit_Tgs,T1,inter_parm =
fit_Tg_to_DiBenedetto(cure_percents_ss/100.,Tgs_ss,T1=None,T0=Tgs_ss[0])
print('T1',T1,'lambda',inter_parm)
#plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
#print(cure_percents_ss)
alphas = np.linspace(0,1)
fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
plt.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
plt.scatter(cure_percents/100.,
            Tgs,
            #label='$E_a$:{}'.format(activation_energy),
            color='r')#colors[i])
plt.legend(fontsize=20)
plt.xlabel('Cure Fraction')
plt.ylabel('$T_g$')
plt.legend(fontsize=15)
```

```
[  1.  31.  51.  71.]
[ 0.41335565  0.43779844  0.44219008  0.46933005]
```

```
T1 0.512882078805 lambda 0.5
```

Out[27]: <matplotlib.legend.Legend at 0x11f0d3400>

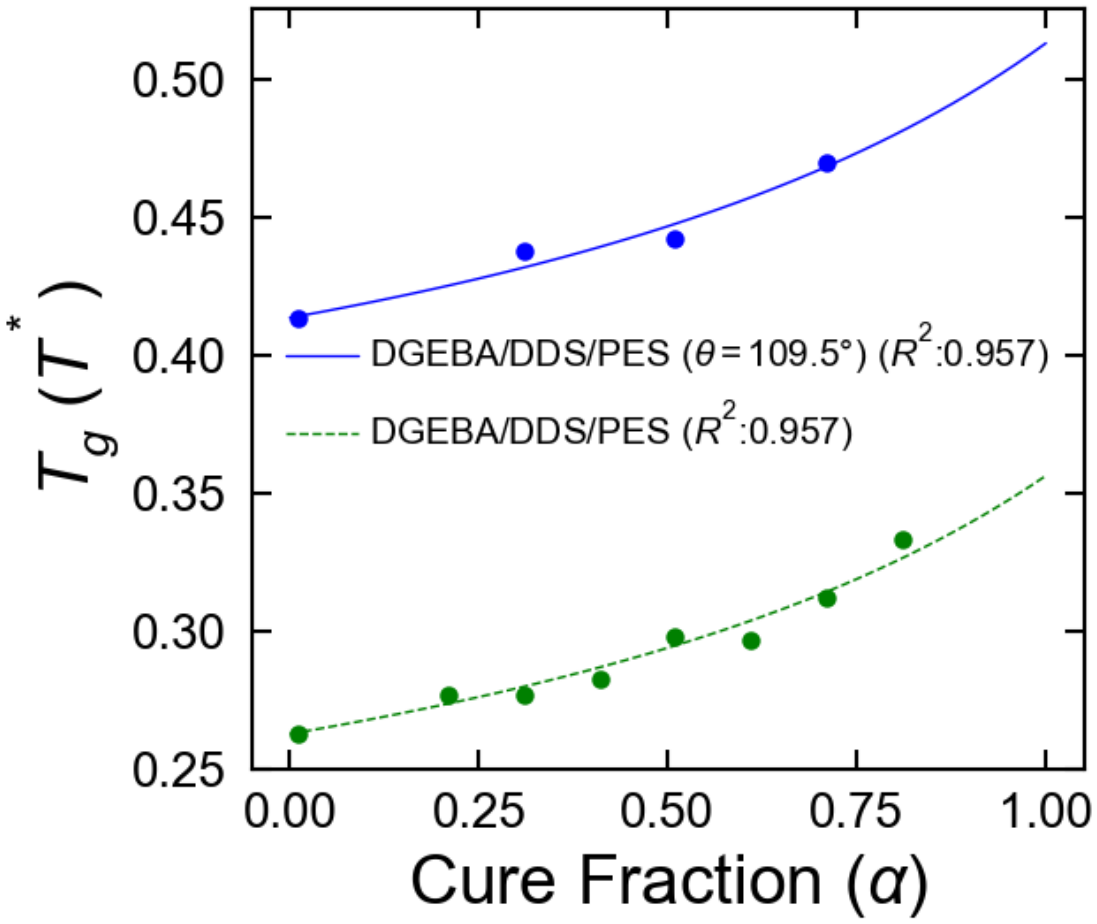
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [28]:
```
data_DGEBA_DDS = np.genfromtxt('DGEBA_DDS__PES_w_angle_Tg.txt')
data_DGEBA_DDS_PES = np.genfromtxt('DGEBA_DDS_PES_Tg.txt')  #
Coarse_Fine_anneal_DGEBA_DDS_PES_6e6

alphas = np.linspace(0,1)
Tgs_ss = data_DGEBA_DDS[1]#[:-1]
cure_fractions = data_DGEBA_DDS[0]
R2,fit_Tgs,T1,inter_parm =
fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=Tgs_ss[0])
print('T1',T1,'lambda',inter_parm)
fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
plt.plot(alphas,
```

```
                 fit_ydata,
                linestyle='-',
                color='b',
                 linewidth=1,
                 label='DGEBA/DDS/PES ($\\theta=109.5\degree$) ($R^2$:{})'.format(round(R2,3)))

        Tgs_ss = data_DGEBA_DDS_PES[1]#[:-1]
        cure_fractions = data_DGEBA_DDS_PES[0]
        R2,fit_Tgs,T1,inter_parm =
        fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=Tgs_ss[0])
        print('T1',T1,'lambda',inter_parm)
        fit_ydata = DiBenedetto(alphas,T1,T0=Tgs_ss[0],inter_param=inter_parm)
        plt.plot(alphas,
                 fit_ydata,
                linestyle='--',
                 linewidth=1,
                color='g',
                 label='DGEBA/DDS/PES ($R^2$:{})'.format(round(R2,3)))


        plt.plot(data_DGEBA_DDS[0],
                 data_DGEBA_DDS[1],
                 linewidth=0.0,
                 color='b',
                 marker='o')
        plt.plot(data_DGEBA_DDS_PES[0],
                 data_DGEBA_DDS_PES[1],
                 linewidth=0.0,
                 color='g',
                 marker='o')
        plt.legend(fontsize=15)
        plt.xlabel('Cure Fraction ($\\alpha$)')
        plt.ylabel('$T_g\ (T^*)$')
        plt.savefig('Tg_with_and_wo_bond_angle.png',transparent=True)
```

```
T1 0.512882078805 lambda 0.5
T1 0.356100677422 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

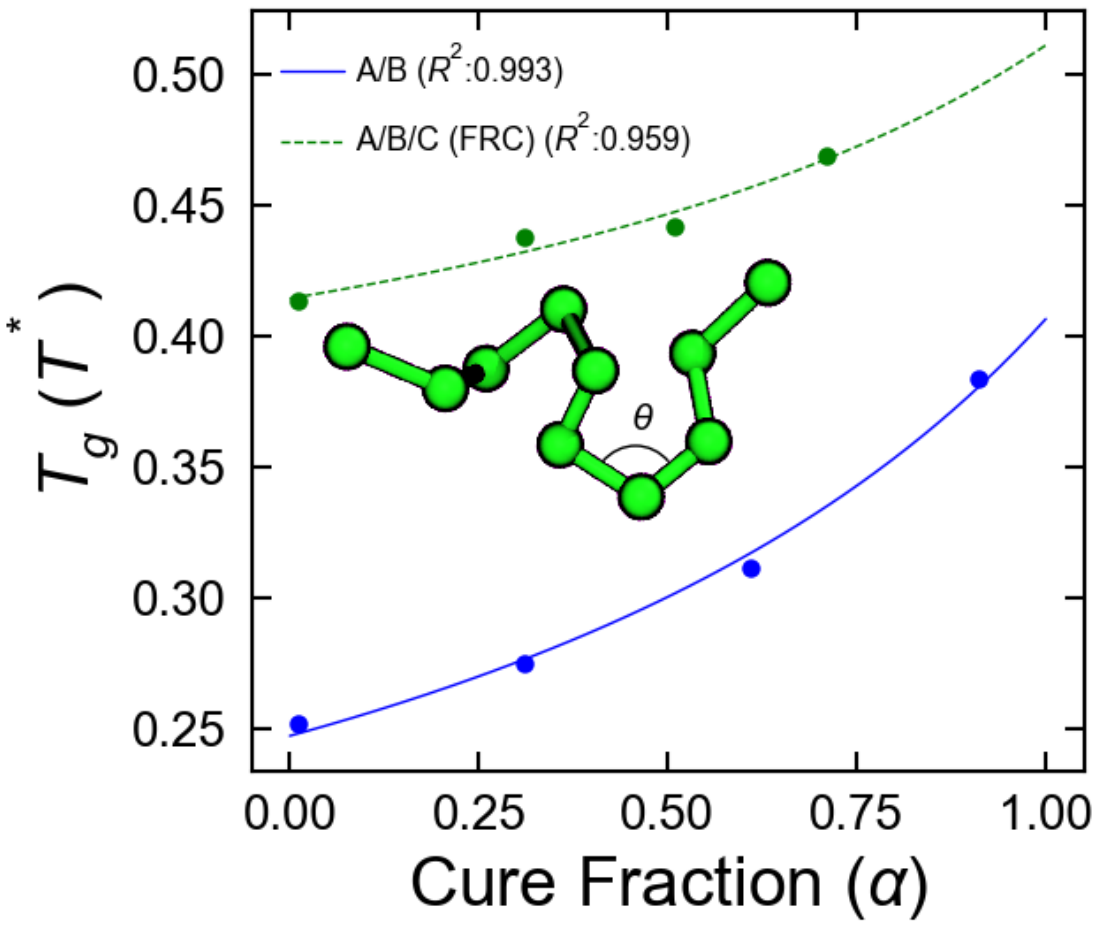



```
In [29]: data_DGEBA_DDS = np.genfromtxt('DGEBA_DDS_Tg_anneal.txt')  #anneal_DGEBA_DDS_LJ.ipynb
         data_DGEBA_DDS_PES = np.genfromtxt('DGEBA_DDS__PES_w_angle_Tg.txt')
         from matplotlib.patches import Arc
         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.3, 0.40, 0.4, 0.4]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.515, 0.49, 0.1, 0.1]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')

         alphas = np.linspace(0,1)
         Tgs_ss = data_DGEBA_DDS[:,1]#[:-1]
         cure_fractions = data_DGEBA_DDS[:,0]
         R2,fit_Tgs,T1,inter_parm,T0 =
         fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
         print('T1',T1,'lambda',inter_parm)
         fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
         ax1.plot(alphas,
                  fit_ydata,
                 linestyle='-',
                 color='b',
                  linewidth=1,
                  label='A/B ($R^2$:{})'.format(round(R2,3)))
```

```
        Tgs_ss = data_DGEBA_DDS_PES[1]#[:-1]
        cure_fractions = data_DGEBA_DDS_PES[0]
        R2,fit_Tgs,T1,inter_parm,T0 =
        fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
        print('T1',T1,'lambda',inter_parm)
        fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
        ax1.plot(alphas,
                 fit_ydata,
                linestyle='--',
                 linewidth=1,
                color='g',
                 label='A/B/C (FRC) ($R^2$:{})'.format(round(R2,3)))


        ax1.plot(data_DGEBA_DDS[:,0],
                 data_DGEBA_DDS[:,1],
                 linewidth=0.0,
                 color='b',
                 marker='o')
        ax1.plot(data_DGEBA_DDS_PES[0],
                 data_DGEBA_DDS_PES[1],
                 linewidth=0.0,
                 color='g',
                 marker='o')
        ax1.legend(fontsize=13,loc='upper left')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g\ (T^*)$')
        im = plt.imread('Coarse_Fine_anneal_DGEBA_DDS_PES_6e6_with_angle/FRC.png')
        #ax2.set_title('FJC')
        ax2.imshow(im)

        ax3.set_xlim(0,1)
        ax3.set_ylim(0,1)
        ax3.annotate('$\\theta$', xy=(0.5, 0.45),fontsize=15)
        ax3.add_patch(Arc((0.55,-0.20),width=0.75,height=1,theta1=45, theta2=135.0))
        savefig(plt,
               'Coarse_Fine_anneal_DGEBA_DDS_PES_6e6_with_angle',
               'Tg_with_and_wo_PES_bond_angle.pdf')
        plt.savefig('Tg_with_and_wo_PES_bond_angle.png',transparent=True)
```

```
T1 0.406699386095 lambda 0.5
T1 0.511279205368 lambda 0.5
```

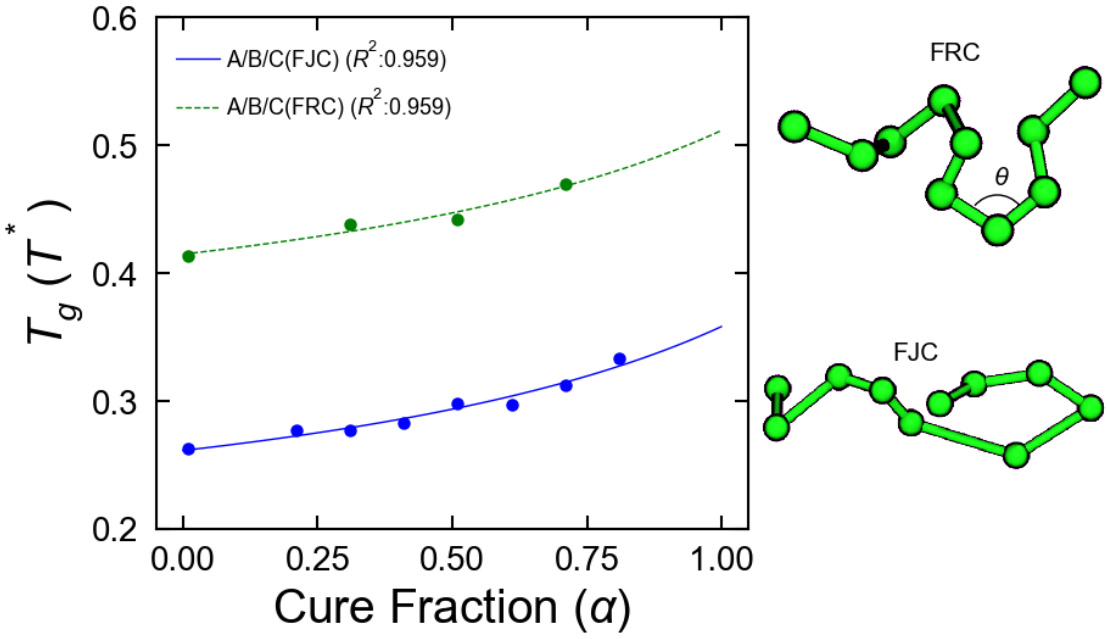
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [19]: data_DGEBA_DDS = np.genfromtxt('DGEBA_DDS_PES_Tg.txt')  #anneal_DGEBA_DDS_LJ.ipynb
         print(data_DGEBA_DDS)
         data_DGEBA_DDS_PES = np.genfromtxt('DGEBA_DDS__PES_w_angle_Tg.txt')
         from matplotlib.patches import Arc
         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.95, 0.55, 0.4, 0.4]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [1.17, 0.640, 0.1, 0.1]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.95, 0.20, 0.4, 0.4]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')

         alphas = np.linspace(0,1)
         Tgs_ss = data_DGEBA_DDS[1]#[:-1]
         cure_fractions = data_DGEBA_DDS[0]
         R2,fit_Tgs,T1,inter_parm,T0 =
         fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
         print('T1',T1,'lambda',inter_parm)
         fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
         ax1.plot(alphas,
                  fit_ydata,
                 linestyle='-',
```

```
                color='b',
                 linewidth=1,
                 label='A/B/C(FJC) ($R^2$:{})'.format(round(R2,3)))
        print(data_DGEBA_DDS[:,0])
        ax1.plot(data_DGEBA_DDS[0],
                 data_DGEBA_DDS[1],
                 linewidth=0.0,
                 color='b',
                 marker='o')

        Tgs_ss = data_DGEBA_DDS_PES[1]#[:-1]
        cure_fractions = data_DGEBA_DDS_PES[0]
        R2,fit_Tgs,T1,inter_parm,T0 =
        fit_Tg_to_DiBenedetto(cure_fractions,Tgs_ss,T1=None,T0=None)
        print('T1',T1,'lambda',inter_parm)
        fit_ydata = DiBenedetto(alphas,T1,T0,inter_param=inter_parm)
        ax1.plot(alphas,
                 fit_ydata,
                linestyle='--',
                 linewidth=1,
                color='g',
                 label='A/B/C(FRC) ($R^2$:{})'.format(round(R2,3)))

        ax1.plot(data_DGEBA_DDS_PES[0],
                 data_DGEBA_DDS_PES[1],
                 linewidth=0.0,
                 color='g',
                 marker='o')
        ax1.legend(fontsize=13,loc='upper left')
        ax1.set_ylim(0.2,0.6)
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('$T_g\ (T^*)$')
        im = plt.imread('Coarse_Fine_anneal_DGEBA_DDS_PES_6e6_with_angle/FRC.png')
        ax2.annotate('FRC',xy=(700, 200),fontsize=15)
        ax2.imshow(im)

        im = plt.imread('Coarse_Fine_anneal_DGEBA_DDS_PES_6e6/FJC.png')
        #ax4.set_xlim(0,1)
        ax4.set_ylim(200,-100)
        ax4.annotate('FJC',xy=(200, 0),fontsize=15)
        ax4.imshow(im)

        #ax1.annotate('', xy=(0.2, 3.7e4), xytext=(0.12,
        3.4e4),arrowprops=dict(facecolor='black', shrink=0.05))
        ax3.set_xlim(0,1)
        ax3.set_ylim(0,1)

        ax3.annotate('$\\theta$', xy=(0.5, 0.45),fontsize=15)
        ax3.add_patch(Arc((.55,-0.20),width=0.75,height=1,theta1=45, theta2=135.0))
        savefig(plt,
               'Coarse_Fine_anneal_DGEBA_DDS_PES_6e6_with_angle',
               'Tg_of_blend_with_and_wo_PES_bond_angle.pdf')
        plt.savefig('Tg_of_blend_with_and_wo_PES_bond_angle.png',transparent=True)
```

```
[[ 0.01        0.21        0.31        0.41        0.51        0.61        0.71
   0.81      ]
 [ 0.26252358  0.27694041  0.27684044  0.2822291   0.29771358  0.29641493
   0.31181878  0.33289583]]
T1 0.357950752712 lambda 0.5
[ 0.01        0.26252358]
T1 0.511279205368 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
```


0.6
0.5
0.4
0.3
0.2
T_g^* (T^*)
A/B/C(FJC) (R^2:0.959)
A/B/C(FRC) (R^2:0.959)
0.00
0.25
0.50
0.75
1.00
Cure Fraction (α)
FRC
θ
FJC

In [2]:
```
from common import *
import matplotlib
%matplotlib inline
```

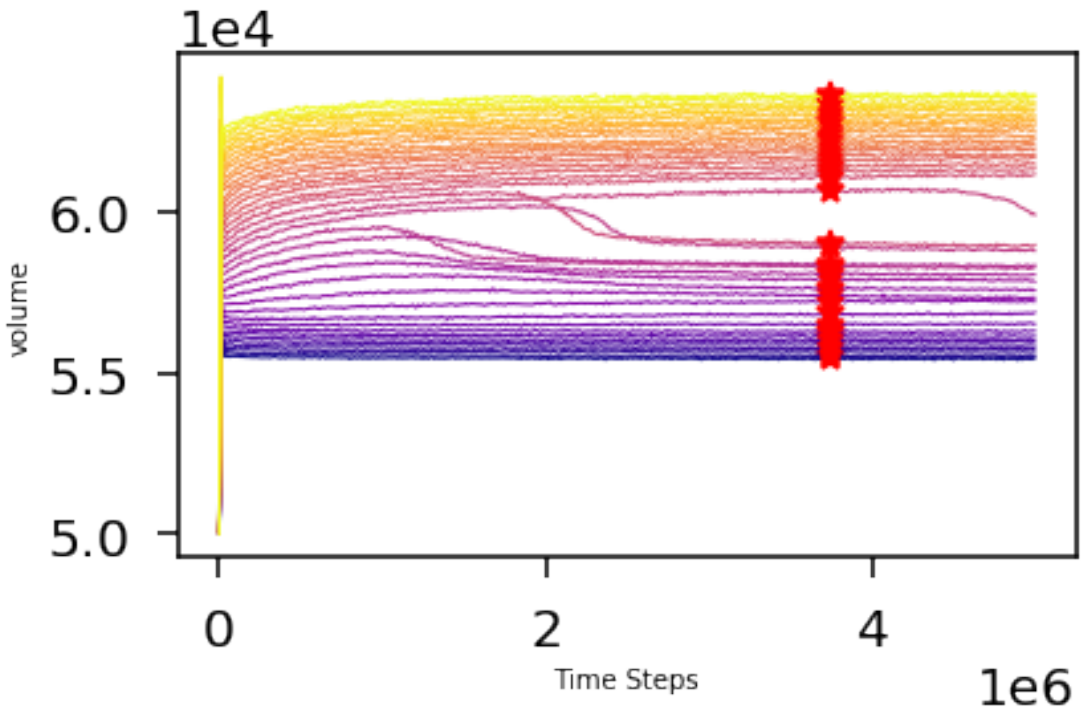
1 Parameters for DGEBA/DDS/PES (LJ)

In [3]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/longer_chains/'
data_path = '/home/sthomas/data/longer_chains/'
tau=1
tauP=10
num_c10=0
kT = 1.7
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=2.0
density=1.0
quench_time=5e6
P = 6.0#[4.5, 6, 8]
stop_after_percents =
[0.,30.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
```

In [4]:
```
#data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_kestrel/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_fry/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_glass_transition/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/new_step_quench/'
#data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Tg/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

```
In [ ]: df_filtered = df[(df.bond==False)&
                                #(df.tau==tau)&
                                #(df.use_curing_job_P==use_curing_job_P)&
                                #(df.tauP==tauP)&
                                #(df.P==P)&
                              (df.quench_T==0.35)&
                            #(df.quench_time==quench_time)&
                                (df.cooling_method=='quench')&
                                (df.stop_after_percent==30)]
        df_filtered

In [ ]: fig,ax = plt.subplots()
        df_filtered = df[(df.bond==True)]
        df_sorted = df_filtered.sort_values('stop_after_percent')
        for pot,gp in df_sorted.groupby('pot'):
            gp.plot(x='cure_percent',y='pressure',ax=ax,label=pot)
        ax.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        ax.set_xlabel('Cure Percent')
        ax.set_ylabel('Pressure')
        #plt.legend(fontsize=10)
        #plt.xlim(start_t+1)
        #ax.set_ylim(34000,42000)
        plt.show()
        df_sorted.groupby('pot',as_index=True)['pressure'].describe()

In [8]: PROP_NAME ='volume'#'temperature'#'kinetic_energy'#'potential_energy'#'volume'


        df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              #(df.quench_method=='fine_quench')&
                              (df.quench_time ==quench_time)&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot==pot)&
                              (df.stop_after_percent==30.)]
        #print(df_filtered.quench_T)
        plot_equilibriation(df_filtered,project,PROP_NAME,mean_from_second_half=True)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        plt.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        plt.xlabel('Time Steps')
        plt.ylabel(PROP_NAME)
        #plt.legend(fontsize=6)
        #plt.xlim(1.5e7)
        #plt.ylim(-2.6e4,0)
        plt.show()
```

```
did not find out.log for 86e98819a5d39b3391c81f5d97178302
did not find out.log for 95e232aae8487475a3226249e36dc027
did not find out.log for 9d929bd1fe1c3a9ad75e5bc31507d4bf


/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/font_manager.py:1331: UserWarning: findfont: Font family ['sans-
serif'] not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/figure.py:2299: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so results might be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```

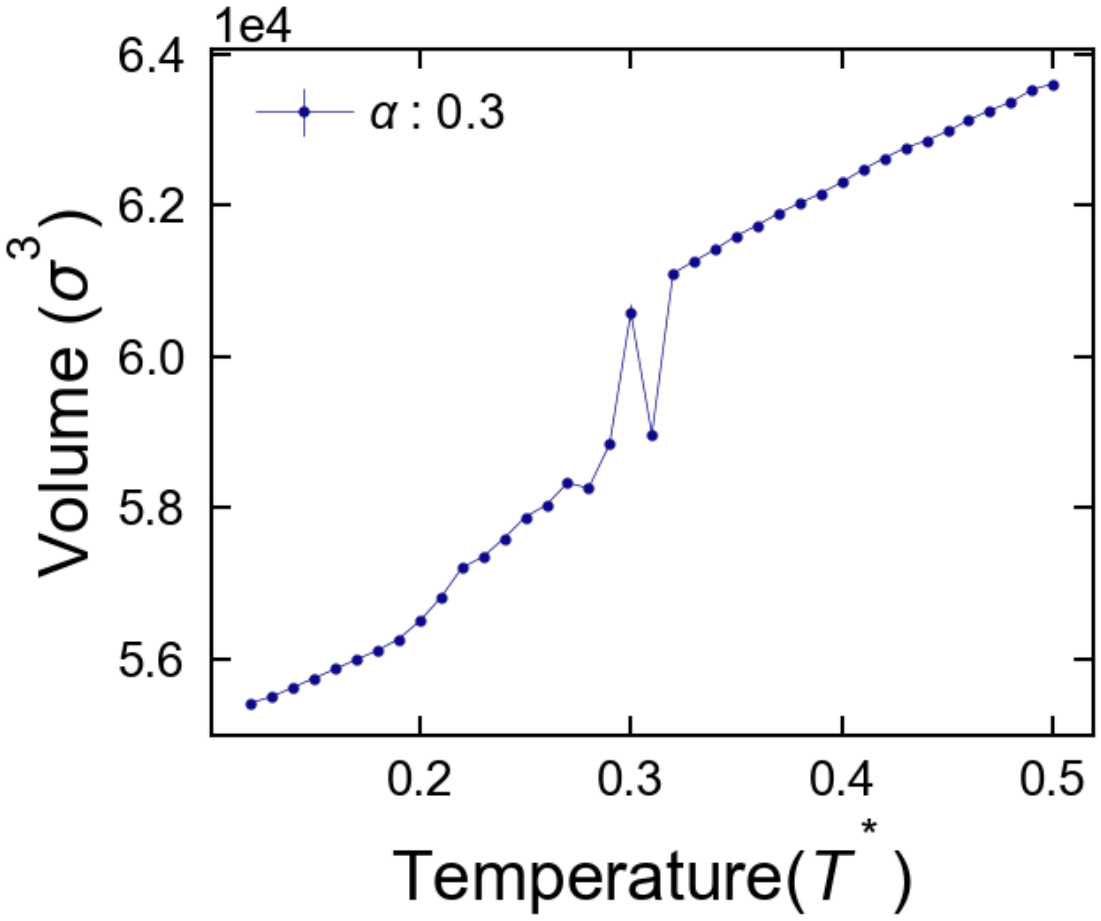



```
In [11]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         #stop_after_percents = [10,100.]
         plt.figure()
         filter_saps = [30.]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             #plt.figure()
             df_filtered = df[(df.integrator==integrator) &
                              (df.kT==kT) &
                              (df.quench_time ==quench_time)&
                              #(df.quench_method=='fine_quench')&
                              (df.tauP == tauP)&
                              (df.n_particles==N)&
                              (df.density==density)&
                              (df.activation_energy==activation_energy)&
                              (df.P==P)&
                              (df.cooling_method==cooling_method)&
                              (df.pot==pot)&
                              (df.stop_after_percent==sap)]
             prop='volume'#'bond_harmonic_energy'#'volume'#'potential_energy'#'pair_lj_energy'#'
             quenchTs,mean_vals,val_stds = get_values_for_quenchTs(df_filtered,
                                                                   project,
                                                                   prop,
                                                                   mean_from_second_half=True)

             #plot_data_with_regression(quenchTs, mean_vals)
             if False:
                 model = piecewise(quenchTs, mean_vals)
                 #print(model)
                 if len(model.segments) == 2:
                     lines = []
                     l1 = model.segments[0]
```

```
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            show_actual_line=True
            if show_actual_line:
                xs = np.linspace(l1.start_t,l1.end_t)
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
            else:
                xs = np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
                ys = l1.coeffs[1]*xs+l1.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
            xs = np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys = l2.coeffs[1]*xs+l2.coeffs[0]
            plt.plot(xs,
                     ys,
                     color=colors[i],
                     zorder=0,
                     linewidth=1)
        else:
            print('WARNING: found more or less than 2 line segments in
regression!',len(model.segments))
            for line in model.segments:
                l1 = line
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                xs = np.linspace(l1.start_t,l1.end_t)
                ys = l1.coeffs[1]*xs+l1.coeffs[0]

                plt.plot(xs,
                         ys,
                         color=colors[i],
                         zorder=0,
                         linewidth=1)

    plt.errorbar(quenchTs,
                 mean_vals,
                 val_stds,
                 marker='.',
                 color=colors[i],
                 label='$\\alpha$ : {}'.format(sap/100),
                 linewidth=0.5,
                zorder=1)#1.5)

    show_transition = False
    if show_transition:
        Tg,Tg_prop = find_Tg(quenchTs,mean_vals,sap)
        print(Tg)
        #plt.axvline(x=Tg,
        #            linewidth=1.0,
        #            color=colors[i])
        plt.scatter(Tg,
                    Tg_prop,
                    marker='*',
                    s=100,
                    color='r',
                   zorder=0)#colors[i])
#plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

plt.xlabel('Temperature($T^*$)')
plt.ylabel('Volume ($\\sigma^3$)')
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
```

```
        plt.legend(fontsize=20)
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
        savefig(plt,
                'quench_DGEBA_DDS_PES_100mer_LJ',
                'V_vs_qT.pdf')
```

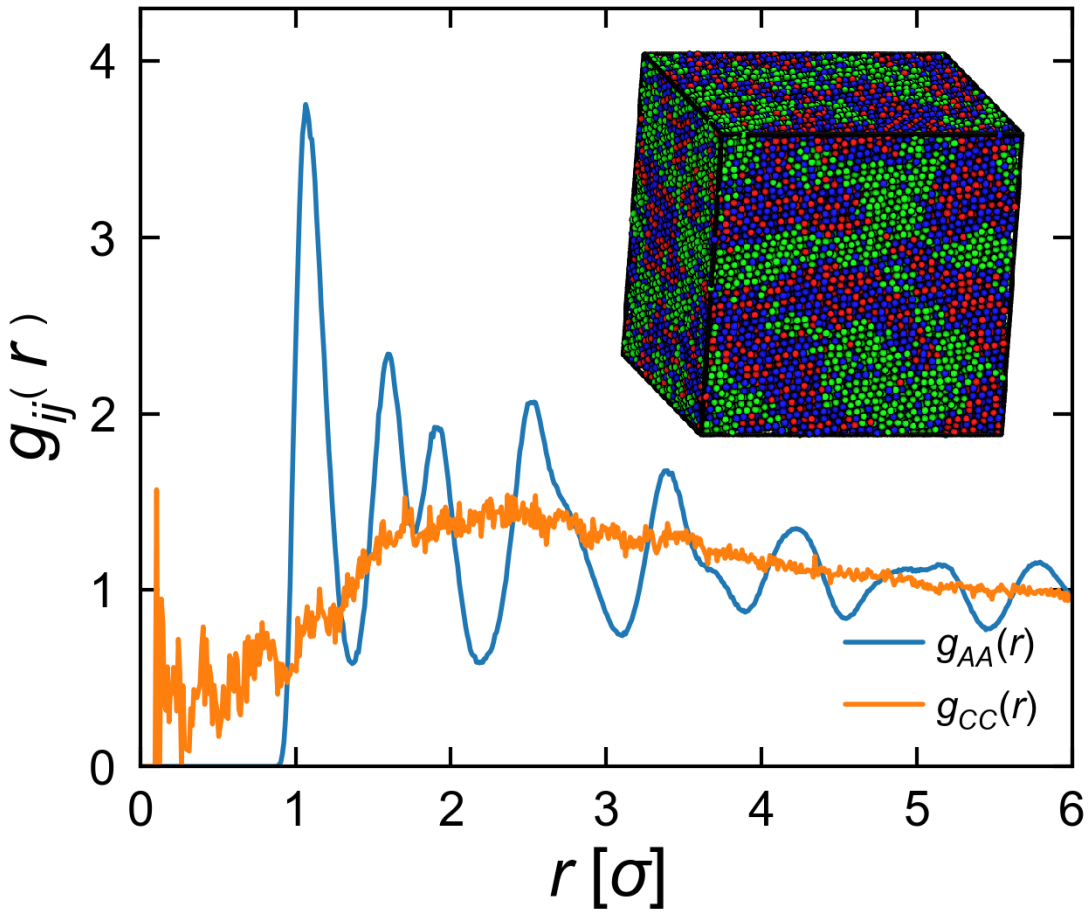
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [12]:
```
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure(dpi=200)
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.54, 0.52, 0.37, 0.37]#max t2
        ax2 = fig.add_axes([left, bottom, width, height])
        ax2.axis('off')
        left, bottom, width, height = [0.505, 0.645, 0.23, 0.23]#max t2
        ax3 = fig.add_axes([left, bottom, width, height])
```

```
        ax3.axis('off')
        left, bottom, width, height = [0.71, 0.645, 0.23, 0.23]#max t2
        ax4 = fig.add_axes([left, bottom, width, height])
        ax4.axis('off')
        figure_axes=[ax2,ax3,ax4]
        rdfs = ['rdf_DDS_DDS.txt','rdf_PES_PES.txt']
        labels = ['$g_{AA}(r)$','$g_{CC}(r)$']
        filter_saps = [30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
        qTs=[0.28]
        colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
        for i,sap in enumerate(filter_saps):
            #plt.figure()
            if sap not in filter_saps:
                continue
            df_filtered = df[(df.bond==False)&
                             #(df.tau==tau)&
                             #(df.use_curing_job_P==use_curing_job_P)&
                             #(df.tauP==tauP)&
                             #(df.P==P)&

                           #(df.quench_time==quench_time)&
                             (df.cooling_method=='quench')&
                             (df.stop_after_percent==sap)]
            df_sorted = df_filtered.sort_values('quench_T')
            for j,qT in enumerate(qTs):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    for k,rdf_file in enumerate(rdfs):
                        if job.isfile(rdf_file):
                            data=np.genfromtxt(job.fn(rdf_file))
                            r=data[:,0]
                            gr=data[:,1]
                        else:
                            continue
                            raise FileNotFoundError('Dont have the rdf file for',job)
                        ax1.plot(r,
                                 gr,
                                 linewidth=2,
                                 #label='(T: {:.2f} kT ($\\alpha$ :
        {})'.format(job.sp.quench_T,sap/100))
                                 label=labels[k])
            im = plt.imread(job.fn('final_snapshot.png'))
            #figure_axes[i].set_title('$\\alpha$ : {}'.format(sap/100),fontsize=9)
            figure_axes[i].imshow(im)

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g_{ij}\left(\ r\ \right)$")
        ax1.set_ylim(0,4.3)
        ax1.set_xlim(0.0,6)
        ax1.legend(fontsize=16,loc='lower right')
        savefig(plt,
                'quench_DGEBA_DDS_PES_100mer_LJ',
                'xstal_quench_0_28.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

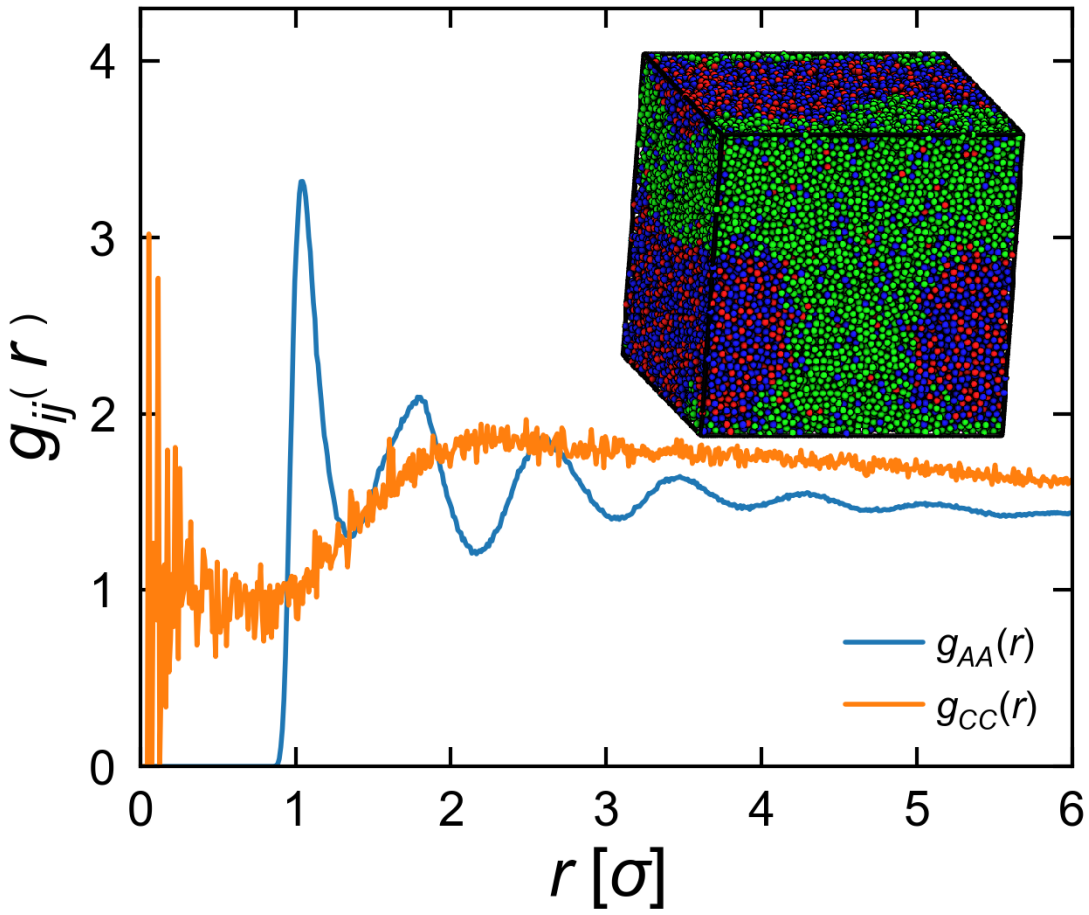
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [13]: from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression

         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.54, 0.52, 0.37, 0.37]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.505, 0.645, 0.23, 0.23]#max t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         left, bottom, width, height = [0.71, 0.645, 0.23, 0.23]#max t2
         ax4 = fig.add_axes([left, bottom, width, height])
         ax4.axis('off')
         figure_axes=[ax2,ax3,ax4]
         rdfs = ['rdf_DDS_DDS.txt','rdf_PES_PES.txt']
         labels = ['$g_{AA}(r)$','$g_{CC}(r)$']
         filter_saps = [30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
         qTs=[0.35]
         colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
         for i,sap in enumerate(filter_saps):
             #plt.figure()
             if sap not in filter_saps:
                 continue
             df_filtered = df[(df.bond==False)&
```

```
                            #(df.tau==tau)&
                            #(df.use_curing_job_P==use_curing_job_P)&
                            #(df.tauP==tauP)&
                            #(df.P==P)&

                        #(df.quench_time==quench_time)&
                            (df.cooling_method=='quench')&
                            (df.stop_after_percent==sap)]
        df_sorted = df_filtered.sort_values('quench_T')
        for j,qT in enumerate(qTs):
            df_xstal = df_sorted[(df_sorted.quench_T==qT)]
            for jobid in df_xstal.index:
                job=project.open_job(id=jobid)
                for k,rdf_file in enumerate(rdfs):
                    if job.isfile(rdf_file):
                        data=np.genfromtxt(job.fn(rdf_file))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        continue
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr,
                             linewidth=2,
                             #label='(T: {:.2f} kT ($\\alpha$ :
{})'.format(job.sp.quench_T,sap/100))
                             label=labels[k])
        im = plt.imread(job.fn('final_snapshot.png'))
        #figure_axes[i].set_title('$\\alpha$ : {}'.format(sap/100),fontsize=9)
        figure_axes[i].imshow(im)

    #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

    ax1.set_xlabel(r"$r\ [\sigma]$")
    ax1.set_ylabel(r"$g_{ij}\left(\ r\ \right)$")
    ax1.set_ylim(0,4.3)
    ax1.set_xlim(0.0,6)
    ax1.legend(fontsize=16,loc='lower right')
    savefig(plt,
            'quench_DGEBA_DDS_PES_100mer_LJ',
            'xstal_quench_0_35.pdf')
    #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
    #plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


$g_{ij}(r)$
$r\ [\sigma]$
$g_{AA}(r)$
$g_{CC}(r)$
0
1
2
3
4
5
6

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'
data_path = '/home/sthomas/data/LB_mixing/'
tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,30.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.
,30.,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
df_filtered=df[df.cooling_method=='anneal']
df_filtered.quench_T
```

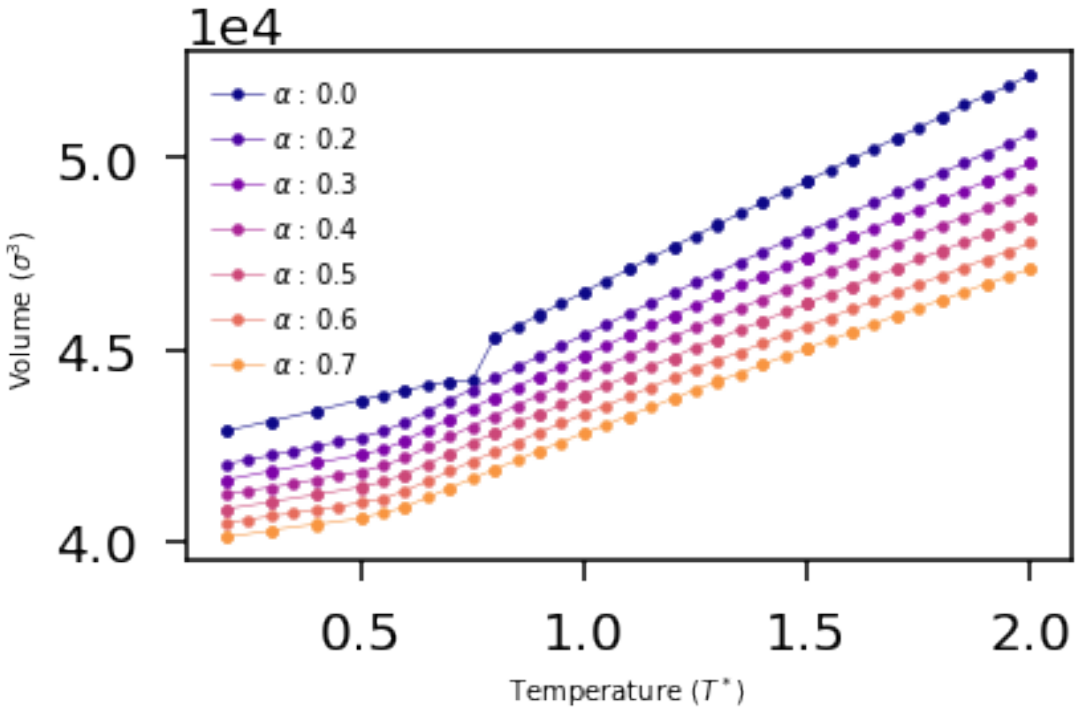
Out[2]:
```
b8be500b426e8327e01b7885e96715f6    0.5
20b93f25713322579e908894cc22a503    0.5
3824222526c14ceef9d02f04dcf325fb    0.5
70a4d1efe9658021d1afbaef2eeade20    0.5
5f8a278914243d9820f14cb07a9f0a34    0.5
Name: quench_T, dtype: object
```

In [3]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0,20,30.,40,50.,60,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
```

```
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=2.0)&
                       (df.quench_T>=0.2)&
                       (df.CC_bond_angle!=109.5)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
                df_curing = df_grp[(df_grp.bond==False)&
                                   (df_grp.calibrationT==305)&
                                   (df_grp.cooling_method==cooling_method)&
                                   (df_grp.stop_after_percent==sap)]
                cure_percent = df_curing.cure_percent.mean()
                cure_percents.append(cure_percent)
                df_sorted = df_curing.sort_values('quench_T')
                #plt.plot(Tg,
                #         Tg_prop,
                #         marker='*',
                #         color=colors[i],
                #         markersize=15)#,

                plt.plot(df_sorted['quench_T'],
                         df_sorted['volume'],
                         marker='.',
                         color=colors[i],#cooling_colors[j],
                         linewidth=0.5,
                         label='$\\alpha$ : {:.1f}'.format(sap/100))



        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        #plt.xlim(0.7,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Volume ($\\sigma^3$)')
        #savefig(plt,'piecewise_regression','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression','all_alphas_volumes.pdf')
        plt.show()
```

```
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/font_manager.py:1331: UserWarning: findfont: Font family ['sans-
serif'] not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/font_manager.py:1331: UserWarning: findfont: Font family
['cursive'] not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/font_manager.py:1331: UserWarning: findfont: Font family ['Arial']
not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/font_manager.py:1331: UserWarning: findfont: Font family ['sans']
not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
/home/sthomas/miniconda3/envs/epoxpy/lib/python3.6/site-
packages/matplotlib/figure.py:2299: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so results might be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```

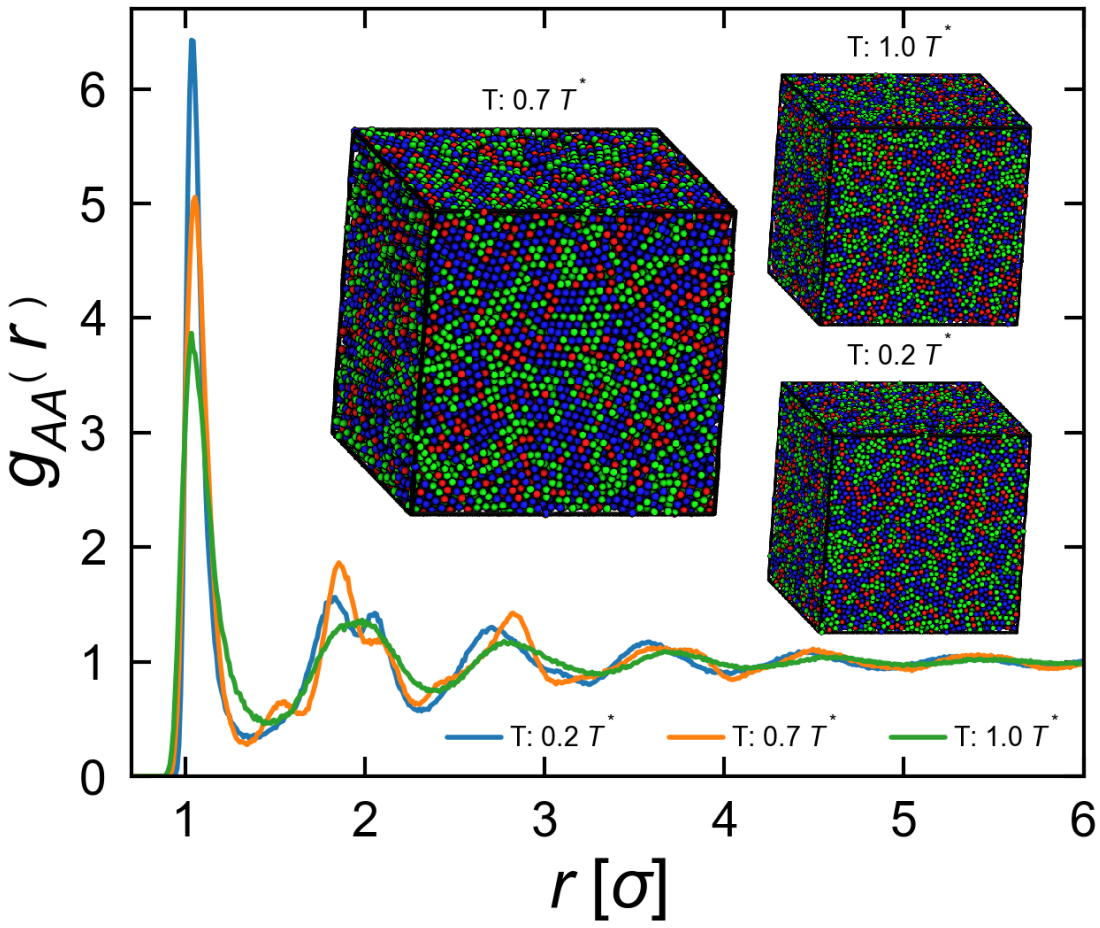



```
In [4]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure(dpi=200)
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.66, 0.34, 0.24, 0.24]#max t2
        ax2 = fig.add_axes([left, bottom, width, height])
        ax2.axis('off')
        left, bottom, width, height = [0.30, 0.45, 0.37, 0.37]#max t2
        ax3 = fig.add_axes([left, bottom, width, height])
        ax3.axis('off')
        left, bottom, width, height = [0.66, 0.63, 0.24, 0.24]#max t2
        ax4 = fig.add_axes([left, bottom, width, height])
        ax4.axis('off')
        figure_axes=[ax2,ax3,ax4]

        filter_saps = [0.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
        qTs=[0.2,0.7,1.0]
        colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
        for i,sap in enumerate(stop_after_percents):
            #plt.figure()
            if sap not in filter_saps:
                continue
            df_filtered = df[(df.CC_bond_angle!=109.5)&
                             (df.stop_after_percent==sap)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
            df_sorted = df_filtered.sort_values('quench_T')
            for j,qT in enumerate(qTs):
```

```
            df_xstal = df_sorted[(df_sorted.quench_T==qT)]
            for jobid in df_xstal.index:
                job=project.open_job(id=jobid)
                print(job.isfile('final.hoomdxml'),job)
                if job.isfile('rdf_DDS_DDS.txt'):
                    data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                    r=data[:,0]
                    gr=data[:,1]
                else:
                    raise FileNotFoundError('Dont have the rdf file for',job)
                ax1.plot(r,
                         gr,
                         linewidth=2.0,
                         label='T: {:.1f} $T^*$'.format(job.sp.quench_T))
                im = plt.imread(job.fn('final_snapshot.png'))
                figure_axes[j].set_title('T: {:.1f}
        $T^*$'.format(job.sp.quench_T),fontsize=11)
                figure_axes[j].imshow(im)
                break

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
        ax1.set_ylim(0,6.7)
        ax1.set_xlim(0.7,6)
        ax1.legend(fontsize=11,loc='lower right',ncol=3)
        savefig(plt,
                'piecewise_regression',
                'xstal_near_transition_0_percent.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False b92076fa673faa46ebd465be5e331c52
False ab8dedac36fddd9457ab655f0553c697
False 35ff7c7e3bddc9e43b4515dd42e35d6e
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

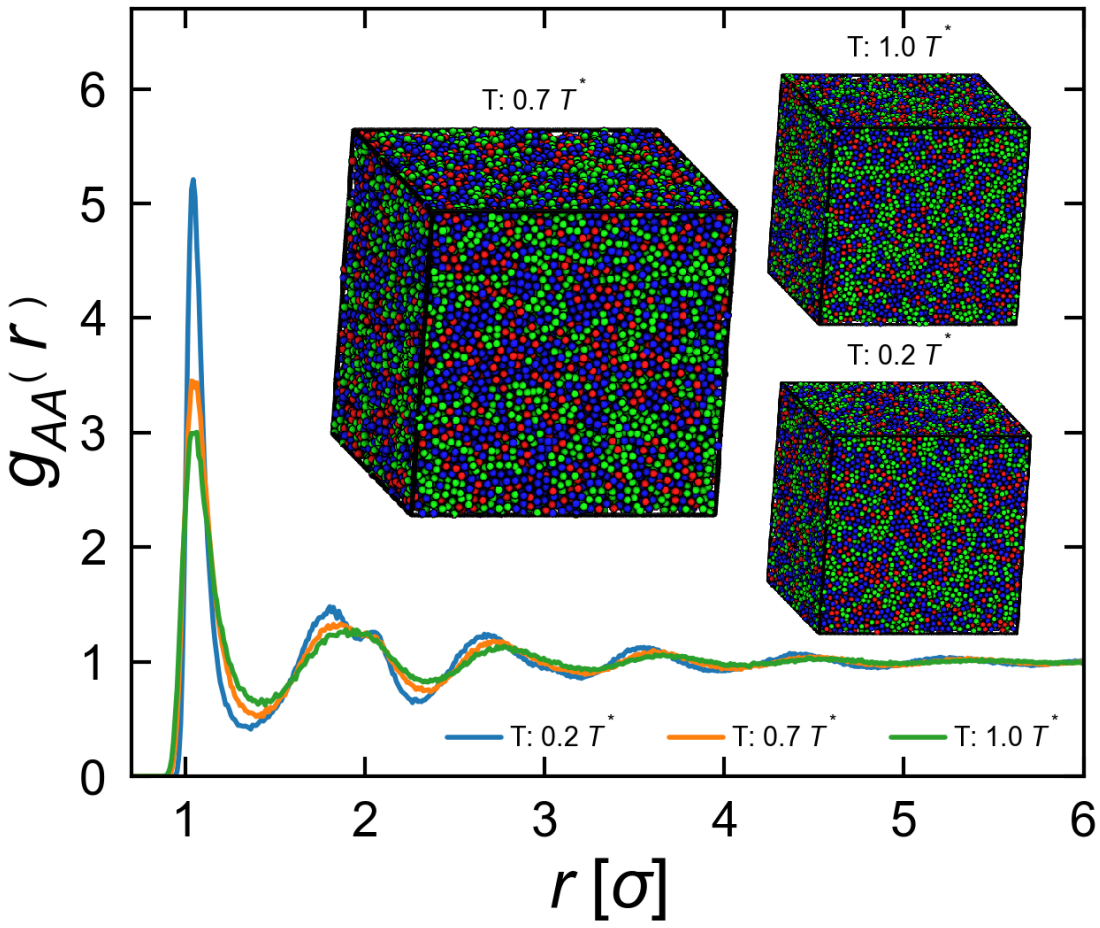



```
In [5]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure(dpi=200)
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.66, 0.34, 0.24, 0.24]#max t2
        ax2 = fig.add_axes([left, bottom, width, height])
        ax2.axis('off')
        left, bottom, width, height = [0.30, 0.45, 0.37, 0.37]#max t2
        ax3 = fig.add_axes([left, bottom, width, height])
        ax3.axis('off')
        left, bottom, width, height = [0.66, 0.63, 0.24, 0.24]#max t2
        ax4 = fig.add_axes([left, bottom, width, height])
        ax4.axis('off')
        figure_axes=[ax2,ax3,ax4]

        filter_saps = [30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
        qTs=[0.2,0.7,1.0]
        colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
        for i,sap in enumerate(filter_saps):
            #plt.figure()
            df_filtered = df[(df.CC_bond_angle!=109.5)&
                             (df.stop_after_percent==sap)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
            df_sorted = df_filtered.sort_values('quench_T')
```

```
            for j,qT in enumerate(qTs):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    print(job.isfile('final.hoomdxml'),job)
                    if job.isfile('rdf_DDS_DDS.txt'):
                        data=np.genfromtxt(job.fn('rdf_DDS_DDS.txt'))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr,
                             linewidth=2.0,
                             label='T: {:.1f} $T^*$'.format(job.sp.quench_T))
                    im = plt.imread(job.fn('final_snapshot.png'))
                    figure_axes[j].set_title('T: {:.1f}
        $T^*$'.format(job.sp.quench_T),fontsize=11)
                    figure_axes[j].imshow(im)
                    break

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g_{AA}\left(\ r\ \right)$")
        ax1.set_ylim(0,6.7)
        ax1.set_xlim(0.7,6)
        ax1.legend(fontsize=11,loc='lower right',ncol=3)
        savefig(plt,
                'piecewise_regression',
                'xstal_near_transition_30_percent.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False 321d6ba084b98282b29a2283722159cc
False 73e373072315913235e58b03ca35ea63
False 76bf7e2a2e0ec57ddf702f544d68e992
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

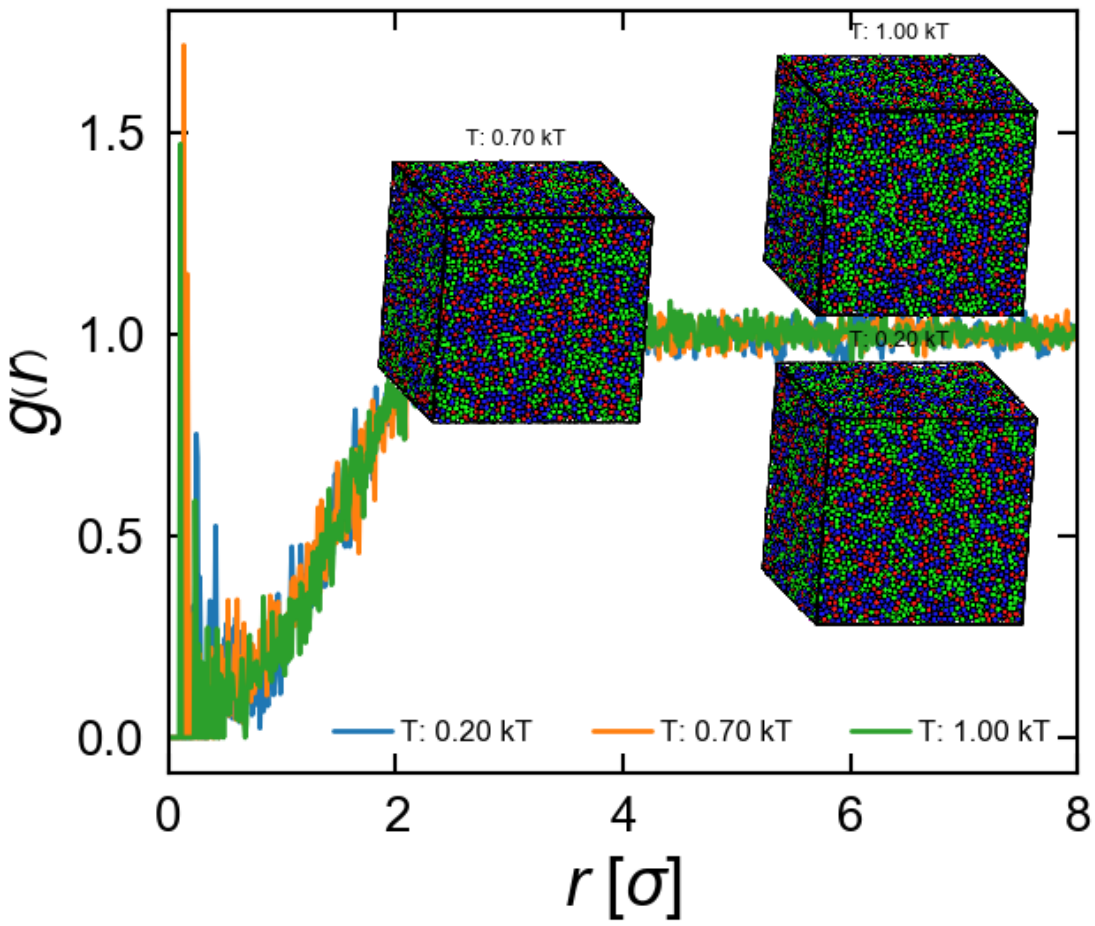



```
In [6]: from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.66, 0.35, 0.25, 0.25]#max t2
        ax2 = fig.add_axes([left, bottom, width, height])
        ax2.axis('off')
        left, bottom, width, height = [0.35, 0.54, 0.25, 0.25]#max t2
        ax3 = fig.add_axes([left, bottom, width, height])
        ax3.axis('off')
        left, bottom, width, height = [0.66, 0.64, 0.25, 0.25]#max t2
        ax4 = fig.add_axes([left, bottom, width, height])
        ax4.axis('off')
        figure_axes=[ax2,ax3,ax4]

        filter_saps = [30.]#[40.,50.,60.]#[60.,70.,80.,90.,100.]
        qTs=[0.2,0.7,1.0]
        colors = plt.cm.plasma(np.linspace(0.75,0,len(filter_saps)))
        for i,sap in enumerate(filter_saps):
            #plt.figure()
            df_filtered = df[(df.CC_bond_angle!=109.5)&
                             (df.stop_after_percent==sap)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
            df_sorted = df_filtered.sort_values('quench_T')
```

```
            for j,qT in enumerate(qTs):
                df_xstal = df_sorted[(df_sorted.quench_T==qT)]
                for jobid in df_xstal.index:
                    job=project.open_job(id=jobid)
                    print(job.isfile('final.hoomdxml'),job)
                    if job.isfile('rdf_PES_PES.txt'):
                        data=np.genfromtxt(job.fn('rdf_PES_PES.txt'))
                        r=data[:,0]
                        gr=data[:,1]
                    else:
                        raise FileNotFoundError('Dont have the rdf file for',job)
                    ax1.plot(r,
                             gr,
                             linewidth=2.0,
                             label='T: {:.2f} kT'.format(job.sp.quench_T))
                    im = plt.imread(job.fn('final_snapshot.png'))
                    figure_axes[j].set_title('T: {:.2f} kT'.format(job.sp.quench_T),fontsize=9)
                    figure_axes[j].imshow(im)
                    break

        #plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

        ax1.set_xlabel(r"$r\ [\sigma]$")
        ax1.set_ylabel(r"$g\left(r\right)$")
        #ax1.set_ylim(0,5.5)
        ax1.set_xlim(0,8)
        ax1.legend(fontsize=12,loc='lower right',ncol=3)
        savefig(plt,
                'piecewise_regression',
                'xstal_near_transition_30_percent_CC.pdf')
        #plt.savefig('volume_cure_{}.png'.format(sap),transparent=False)
        #plt.show()
```

```
False 321d6ba084b98282b29a2283722159cc
False 73e373072315913235e58b03ca35ea63
False 76bf7e2a2e0ec57ddf702f544d68e992
```

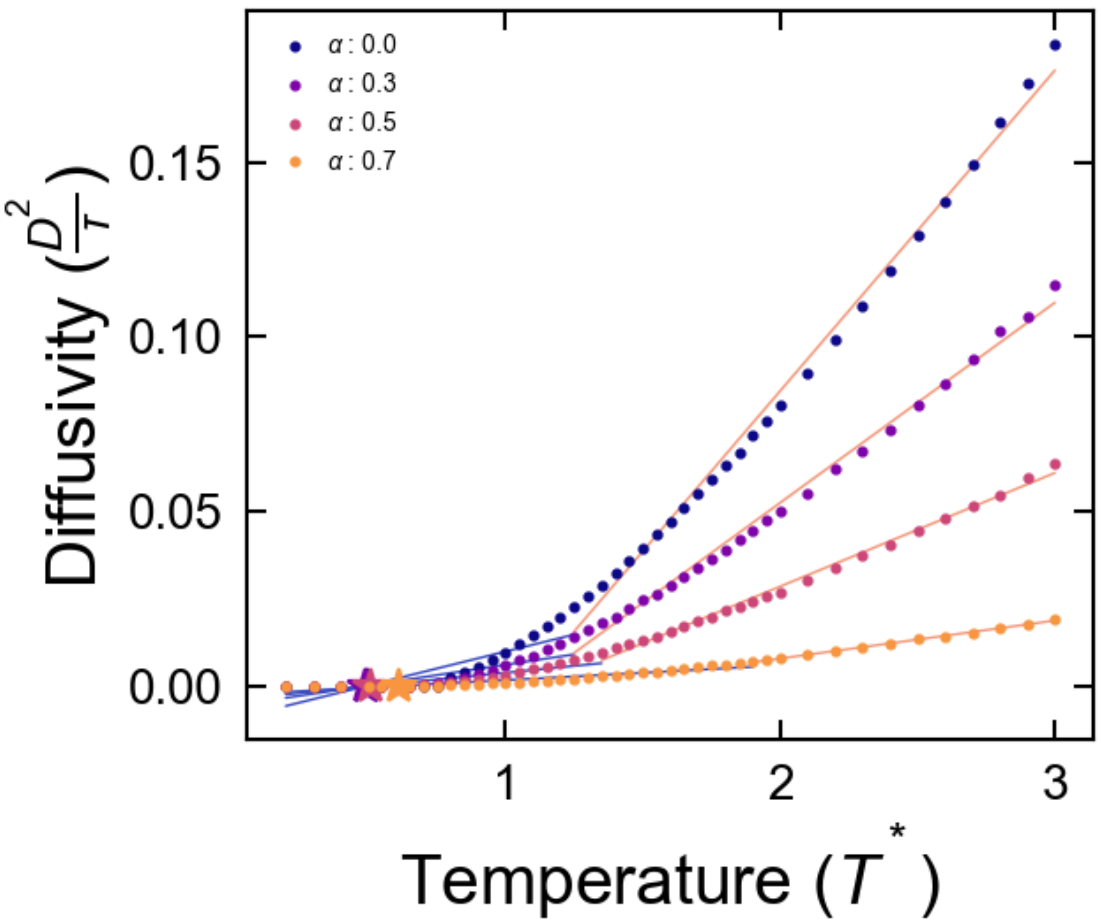
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [7]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.,50.,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=3.0)&
                       (df.quench_T>=0.2)&
                       (df.CC_bond_angle!=109.5)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
```

```
            df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)

            mul_fact=1000000
            Ds_scaled=Ds*mul_fact
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                                   Ds_scaled,
                                                   method='use_viscous_region',
                                                   min_D=0,
                                                   ver=2,
                                                   viscous_line_index=0)

            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop,
                     marker='*',
                     color=colors[i],
                     markersize=15)#,

            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0,
                     label='$\\alpha$ : {:.1f}'.format(sap/100))


            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=l_colors[li],
                         zorder=0,
                         linewidth=1)
            Tgs.append(Tg)
        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        np.savetxt('Tg_{}.txt'.format(cooling_method),np.transpose(Tgs))
        #plt.xlim(0.7,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression','all_alphas.pdf')
        plt.show()
```

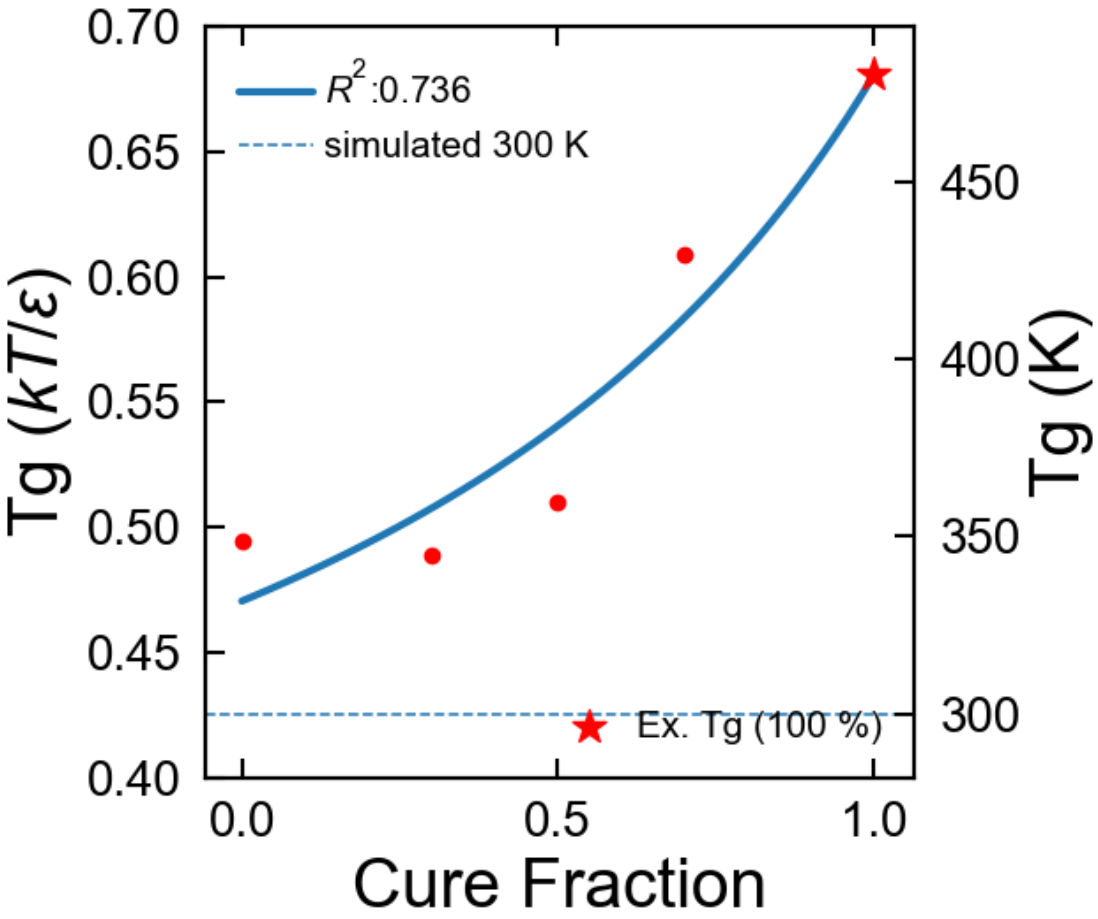
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [8]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        ax2=ax1.twinx()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
```

```
                color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Ex. Tg (100 %)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.4
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 0.7
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction')
        ax1.set_ylabel('Tg ($kT/\\epsilon$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression','dibeneditto.pdf')
```

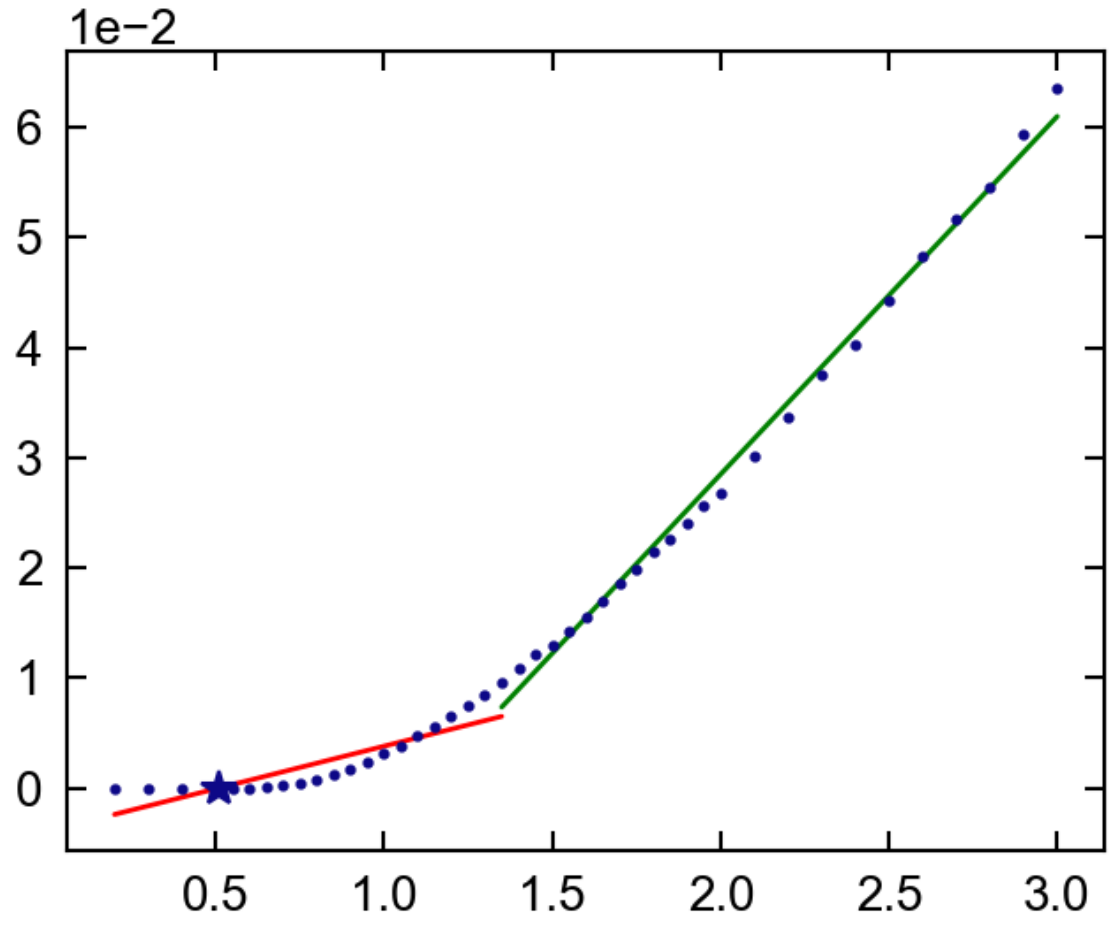
```
[ 0.49468235  0.48854131  0.51008971  0.60857186]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.49468235  0.48854131  0.51008971  0.60857186]
T1 0.680589767749 lambda 0.5
300 K in T*: 0.425368604843


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [9]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=3.0)&
                       (df.quench_T>=0.2)&
                       (df.CC_bond_angle!=109.5)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
            cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
            for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
```

```
            df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)

            mul_fact=1000000
            Ds_scaled=Ds*mul_fact
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                                   Ds_scaled,
                                                   method='use_viscous_region',
                                                   min_D=0,
                                                   ver=2,
                                                   viscous_line_index=0)

            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop,
                     marker='*',
                     color=colors[i],
                     markersize=15)#,

            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0)


            l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=l_colors[li],
                         zorder=0,
                         linewidth=2)
            Tgs.append(Tg)
        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        #plt.xlim(0.7,1.5)
        #plt.ylim(-1e-4,5e-4)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression',
                '50percent.pdf')
        np.savetxt('Tg_{}.txt'.format(cooling_method),np.transpose(Tgs))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−2
6
5
4
3
2
1
0
0.5
1.0
1.5
2.0
2.5
3.0

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

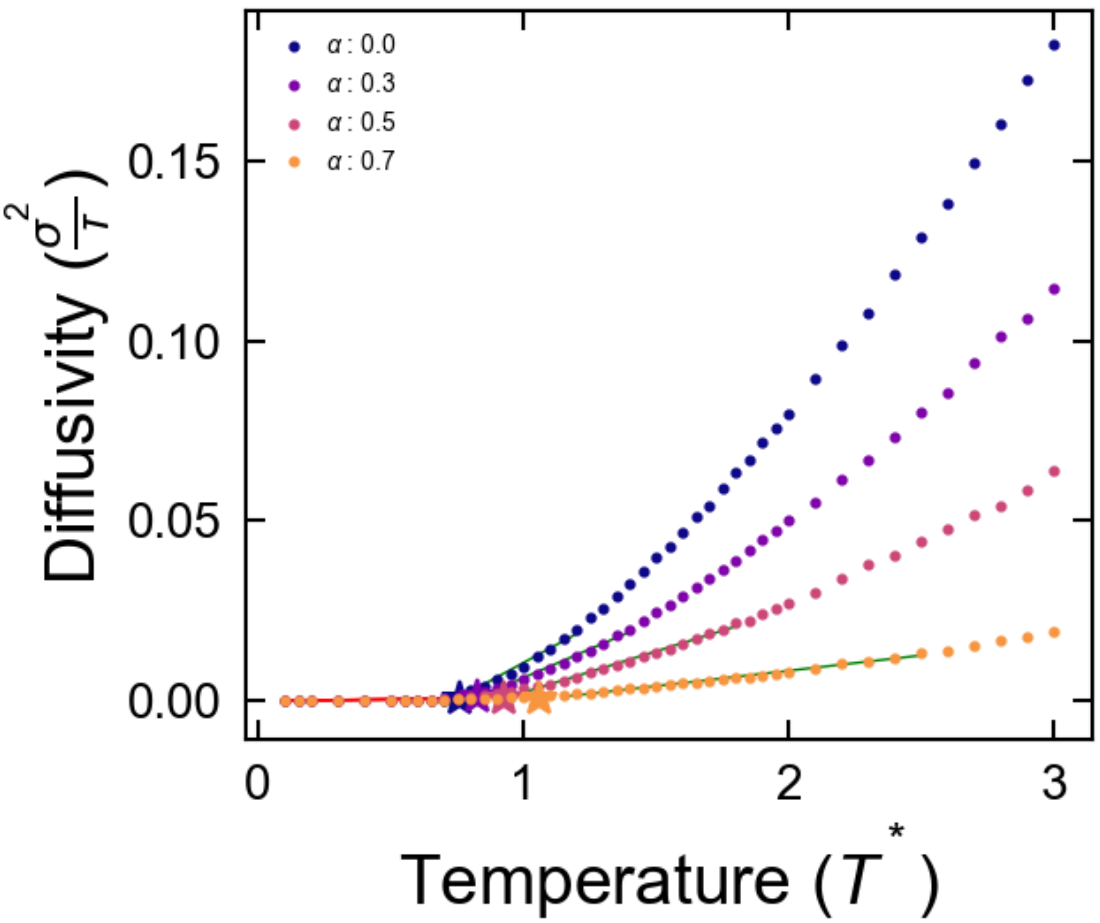
In [2]:
```
def get_custom_ranges(cooling_method):
    if cooling_method=='quench':
        custom_ranges_l1={00.0:[0.1,0.8],
                          30.0:[0.1,0.8],
                          50.0:[0.1,0.8],
                          70.0:[0.1,0.8]}
        custom_ranges_l2={00.0:[0.7,1.2],
                          30.0:[0.85,1.4],
                          50.0:[1.0,1.8],
                          70.0:[1.15,2.5]}
    elif cooling_method=='anneal':
        custom_ranges_l1={00.0:[0.1,0.8],
                          30.0:[0.1,0.8],
                          50.0:[0.1,0.8],
                          70.0:[0.1,0.8]}
        custom_ranges_l2={00.0:[0.7,1.2],
                          30.0:[0.85,1.4],
                          50.0:[1.0,1.8],
                          70.0:[1.15,2.5]}
    else:
        raise ValueError(cooling_method+'is unknown')
    return custom_ranges_l1, custom_ranges_l2
```

In [3]:
```
import matplotlib
#from common import *
```

```
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0,30.,50.,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'

df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.1)&
               (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.calibrationT==305)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        print(cure_percent)
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,
                               df_curing,
                               name=PROP_NAME,
                               quench_time=1e7,
                              equilibriated_ts_percentage=0.50)
        Cure_Ts.append(Ts)

        mul_fact=1000000
        Ds_scaled=Ds*mul_fact
        custom_ranges_l1, custom_ranges_l2 = get_custom_ranges(cooling_method)
        print(custom_ranges_l1[sap])
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                   Ds_scaled,
                                   method='use_viscous_region',
                                   min_D=0,
                                   ver=4,
                                   viscous_line_index=0,
                                   l1_T_bounds=custom_ranges_l1[sap],
                                   l2_T_bounds=custom_ranges_l2[sap])
        xs = Ts#np.linspace(0.1,4)
        plt.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
```

```
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=l_colors[li],
                         zorder=0,
                         linewidth=1)
            Tgs.append(Tg)
        #break
    plt.legend(fontsize=10)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    #plt.xlim(0.3,1.0)
    #plt.ylim(-1e-6,7e-6)
    plt.xlabel('Temperature ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
    savefig(plt,'piecewise_regression_custom_range','all_alphas.pdf')
    print(data)
    np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.trans
    pose(data))

    plt.show()
```

```
0.002499999944
msd file not present in eec7ef6b3b99153c1fcafb4d737e82f9
[0.1, 0.8]
30.0
msd file not present in f7526565cc30f84a17cfda88c1aca65c
[0.1, 0.8]
50.0
[0.1, 0.8]
70.0
[0.1, 0.8]
[array([  2.49999994e-03,   3.00000000e+01,   5.00000000e+01,
         7.00000000e+01]), array([ 0.75650914,  0.82738406,  0.92692524,
1.06032263])]


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

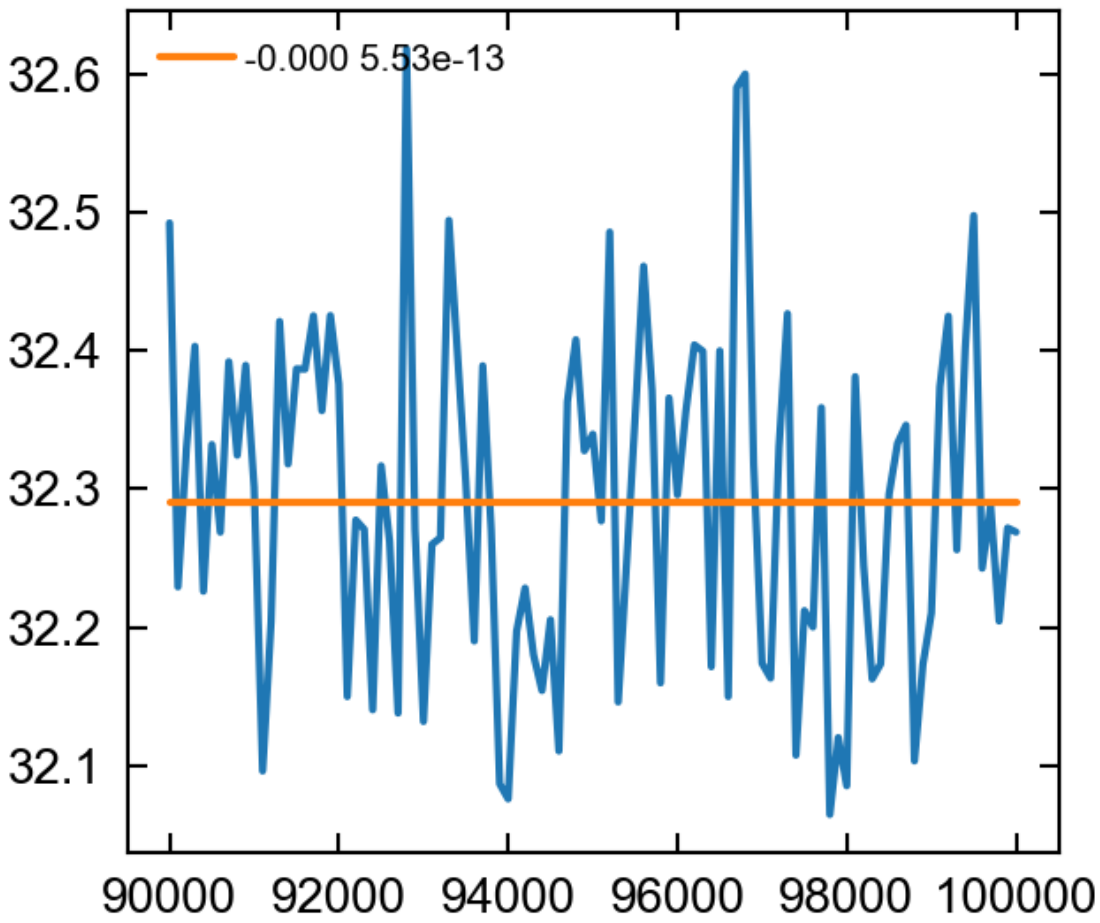



```
In [4]: job = project.open_job(id='3d87a227fbf1bde17b1bd8db5f81bf0c')
        name='bparticles'
        equilibriated_ts_percentage=0.90
        Ts=[]
        Ds=[]
        msd=True
        if msd:
            if job.isfile('msd.log'):
                print(job.sp.stop_after_percent,job.sp.quench_T)
                log_path = job.fn('msd.log')
                data = np.genfromtxt(log_path, names=True)
                prop_values = data[name]#'pair_lj_energy']
                if False:#key=='anneal':
                    times,msds,qTs = get_split_quench_job_msd(job,name)
                    for j,msd in enumerate(msds):
                        start_index = int(len(times[j])*equilibriated_ts_percentage)
                        time=times[j]*job.sp.md_dt
                        quench_T = qTs[j]
                        eq_msd = msd[start_index:]
                        eq_time = time[start_index:]
                        D_A,m,b = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                        Ts.append(quench_T)
                        Ds.append(D_A)
                else:
                    all_time_steps = data['timestep']
                    start_index = int(len(all_time_steps)*equilibriated_ts_percentage)
```

```
                time=all_time_steps*job.sp.md_dt
                quench_T = job.sp.quench_T
                eq_msd = prop_values[start_index:]
                eq_time = time[start_index:]
                #print(job)
                D_A,m,b,r_2 = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                if r_2 <0.9:
                    print(r_2,job)
                Ts.append(quench_T)
                Ds.append(D_A)
                plt.plot(eq_time,eq_msd)
                plt.plot(eq_time,m*eq_time+b,label='{:.3f} {:.2e}'.format(r_2,m))
                #plt.xscale('log')
                plt.legend(fontsize=15)
        else:
            print('msd file not present in',job)
    else:
        if job.isfile('out.log'):
            print(job.sp.stop_after_percent,job.sp.quench_T)
            log_path = job.fn('out.log')
            data = np.genfromtxt(log_path, names=True)
            prop_values = data['volume']#'pair_lj_energy']

            all_time_steps = data['timestep']
            start_index = int(len(all_time_steps)*equilibriated_ts_percentage)
            time=all_time_steps*job.sp.md_dt
            quench_T = job.sp.quench_T
            eq_prop = prop_values[start_index:]
            eq_time = time[start_index:]
            plt.plot(eq_time,eq_prop)
            #plt.xscale('log')
            plt.legend(fontsize=15)
```

```
70.0 0.5
-3.02927927187e-09 3d87a227fbf1bde17b1bd8db5f81bf0c
```

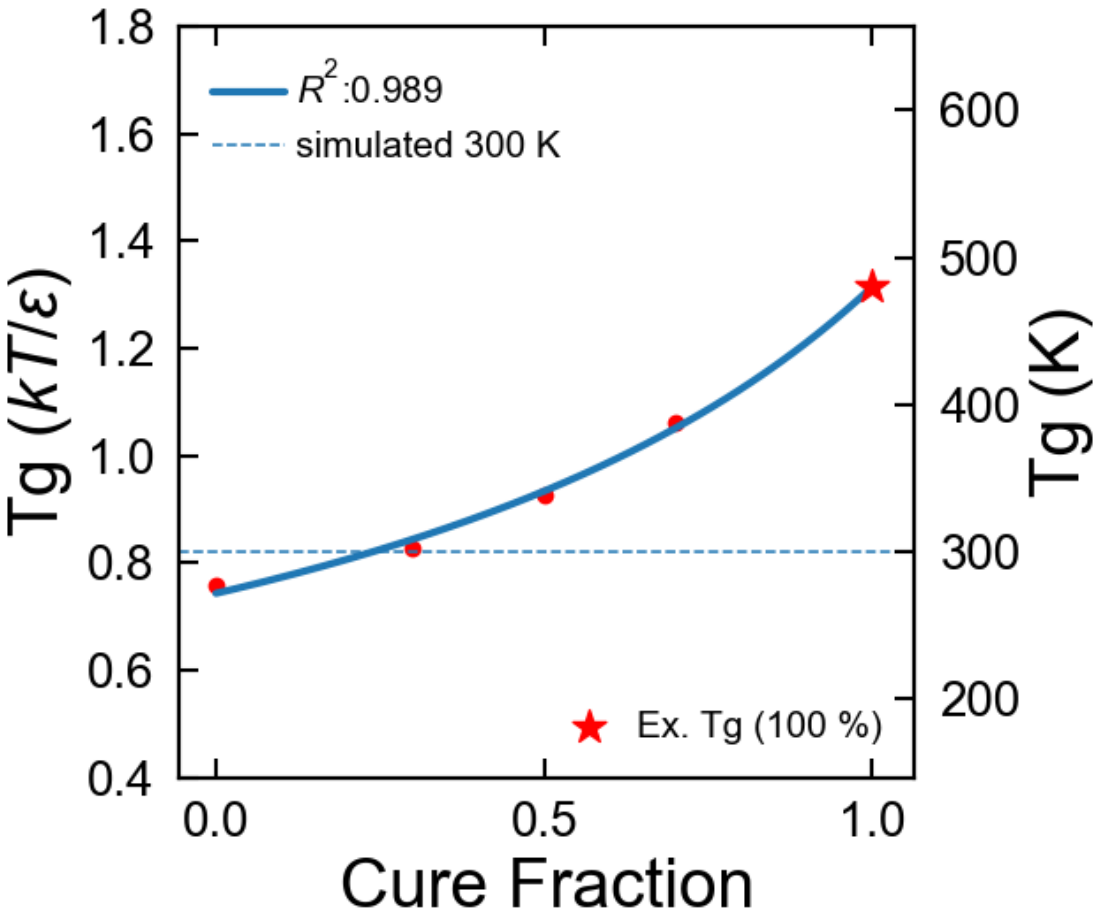
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [5]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        ax2=ax1.twinx()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
```

```
                color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Ex. Tg (100 %)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.4
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction')
        ax1.set_ylabel('Tg ($kT/\\epsilon$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

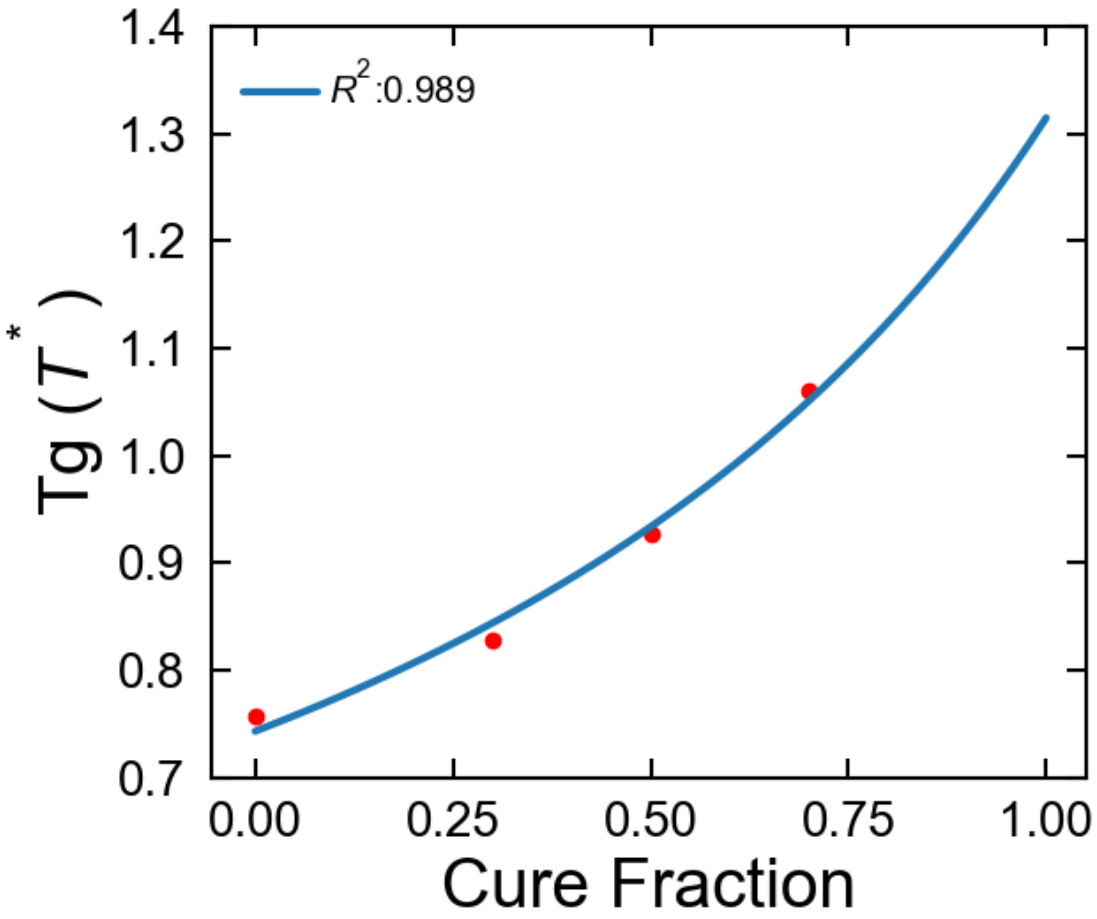
```
[ 0.75650914  0.82738406  0.92692524  1.06032263]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.75650914  0.82738406  0.92692524  1.06032263]
T1 1.31476557963 lambda 0.5
300 K in T*: 0.821728487266


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [6]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
        #print(cure_percents_ss)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])
```

```
        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)

        sim_low_lim = 0.7
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.4
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=False
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range','dibeneditto_dimless.pdf')
```

```
[ 0.75650914  0.82738406  0.92692524  1.06032263]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.75650914  0.82738406  0.92692524  1.06032263]
T1 1.31476557963 lambda 0.5
300 K in T*: 0.821728487266


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

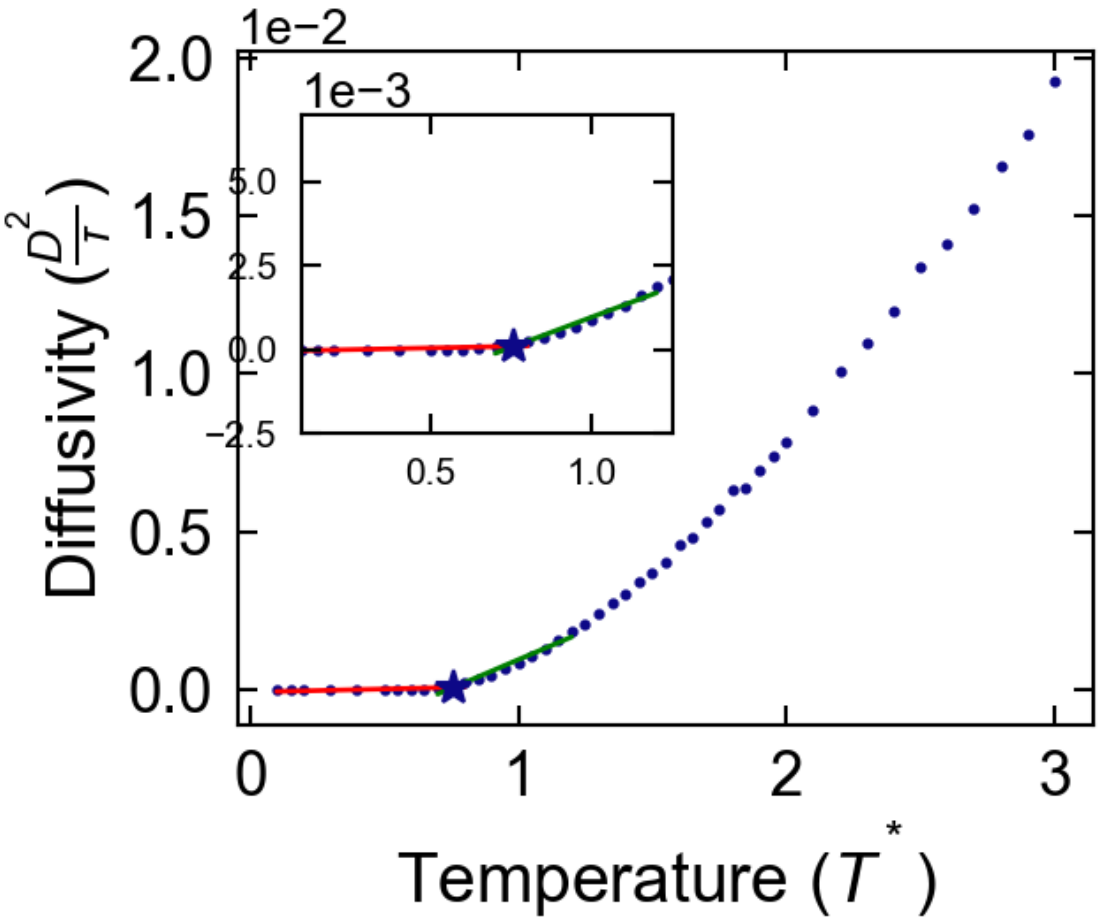



```
In [7]: import matplotlib
        #from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        custom_ranges_l1=[[0.1,0.8],
                          [0.1,0.8],
                          [0.1,0.8],
```

```
                  [0.1,0.8]]
custom_ranges_l2=[[0.7,1.2],
                  [0.85,1.4],
                  [1.0,1.8],
                  [1.15,2.5]]
df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.1)&
               (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=1000000
        Ds_scaled=Ds*mul_fact
        custom_ranges_l1, custom_ranges_l2 = get_custom_ranges(cooling_method)
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                   Ds_scaled,
                                   method='use_viscous_region',
                                   min_D=0,
                                   ver=4,
                                   viscous_line_index=0,
                                   l1_T_bounds=custom_ranges_l1[i],
                                   l2_T_bounds=custom_ranges_l2[i])
        xs = Ts#np.linspace(0.1,4)
        ax1.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,
        ax2.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=0)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            ax1.plot(xs,
                     ys,
                     color=l_colors[li],
```

```
                        zorder=1,
                        linewidth=2)
                ax2.plot(xs,
                        ys,
                        color=l_colors[li],
                        zorder=1,
                        linewidth=2)
            Tgs.append(Tg)
        #break
    ax1.legend(fontsize=10,loc='lower right')
    ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
    ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
    ax1.tick_params(axis = 'both', which = 'major',labelsize=25)
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    ax2.set_xlim(0.1,1.25)
    ax2.set_ylim(-2.5e-3,7e-3)
    ax1.set_xlabel('Temperature ($T^*$)')
    ax1.set_ylabel('Diffusivity ($\\frac{D^2}{\\tau}$)')
    #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
    savefig(plt,'piecewise_regression_custom_range',
            '50percent.pdf')
    data=[cure_percents,Tgs]
    np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.trans
    pose(data))

    plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



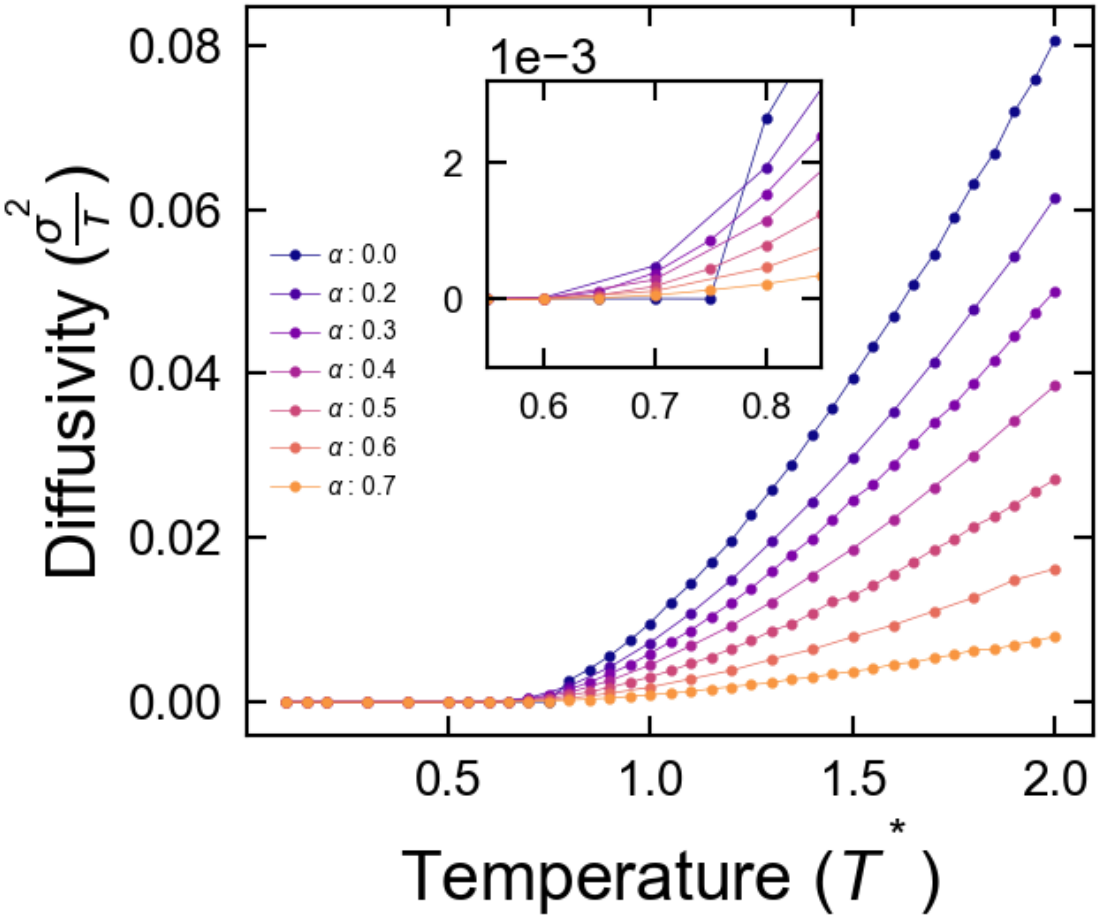

```
In [22]: import matplotlib
         #from common import *
         %matplotlib inline
         from piecewise.regressor import piecewise
         #https://www.datadoghq.com/blog/engineering/piecewise-regression/
         from piecewise.plotter import plot_data_with_regression
         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.45, 0.59, 0.27, 0.27]
         ax2 = fig.add_axes([left, bottom, width, height])
         #stop_after_percents = np.arange(10,105,15,dtype=float)
         PROP_NAME
         ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
         #plt.figure()
         filter_saps=[0.,20.,30.,40.,50.,60.,70.,]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
         colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
         Tgs=[]
         Tgs_tangent=[]
         cure_percents = []
         Cure_Ts=[]
         markers=['+','.']
         markersize=[10,10]
         cooling_method='quench'
         custom_ranges_l1=[[0.1,0.8],
                           [0.1,0.8],
                           [0.1,0.8],
```

```
                  [0.1,0.8]]
custom_ranges_l2=[[0.7,1.2],
                  [0.85,1.4],
                  [1.0,1.8],
                  [1.15,2.5]]
df_filtered=df[(df.quench_T<=2.0)&
               (df.quench_T>=0.1)&
               (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=1000000
        Ds_scaled=Ds*mul_fact


        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.5,
                 zorder=0,
                label='$\\alpha$ : {}'.format(sap/100))
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.5,
                 zorder=0)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            #ax1.plot(xs,
            #         ys,
            #         color=l_colors[li],
            #         zorder=1,
            #         linewidth=2)
            ##ax2.plot(xs,
            #         ys,
            #         color=l_colors[li],
            #         zorder=1,
            #         linewidth=2)
        Tgs.append(Tg)
    #break
ax1.legend(fontsize=10,loc='center left')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
ax1.tick_params(axis = 'both', which = 'major',labelsize=20)
Tgs = np.asarray(Tgs)
cure_percents = np.asarray(cure_percents)
data=[cure_percents,Tgs]
ax2.set_xlim(0.55,0.85)
ax2.set_ylim(-1.0e-3,3.2e-3)
#ax1.set_ylim(-0.01,0.09)
```

```
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range',
                'D_vs_qT.pdf')
        #data=[cure_percents,Tgs]
        #np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.tran
        spose(data))

        plt.show()
```

```
msd file not present in eec7ef6b3b99153c1fcafb4d737e82f9
msd file not present in 163fc4fd74d2f91abd24d5c1fd77087e
msd file not present in 856e6a966fe8909ce5b475cb68a88e8e
msd file not present in c9d3463db5bc0e09516026e9c3076365
```

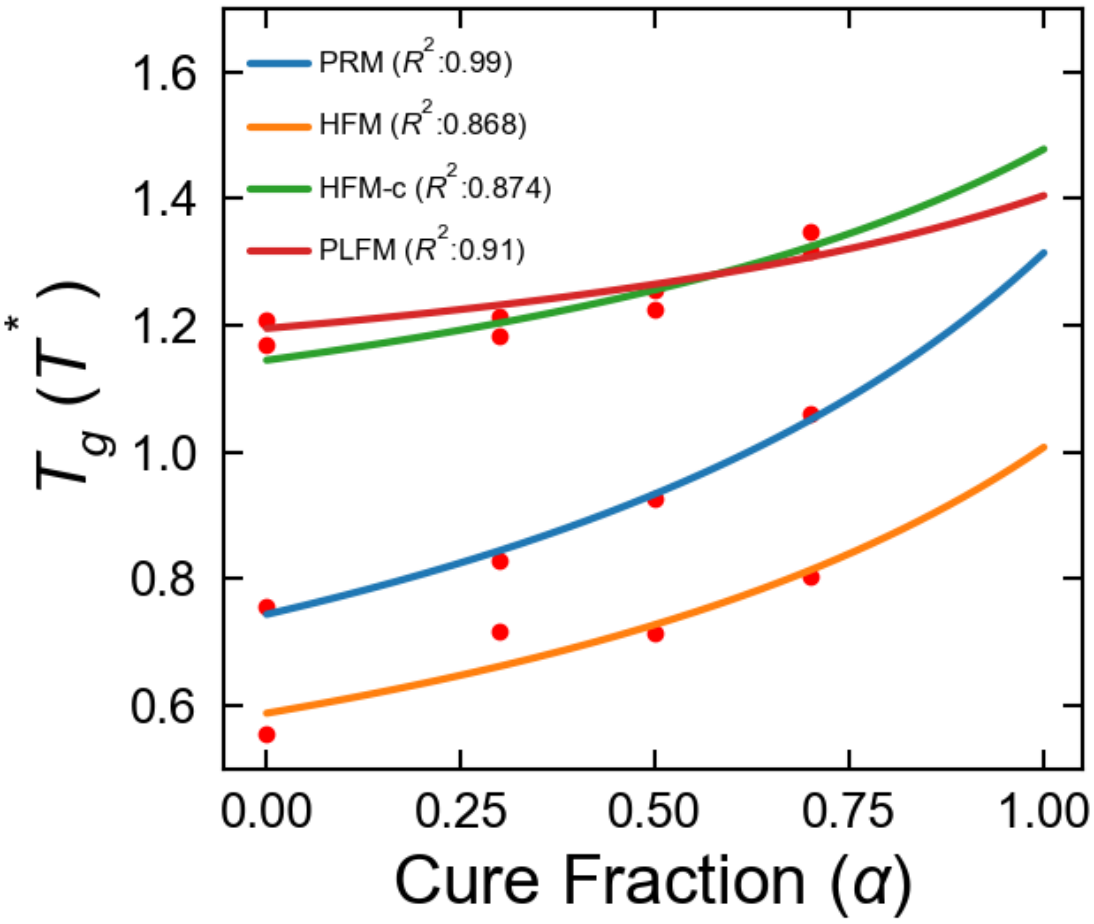
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [13]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *
fig, ax1 = plt.subplots()

fitting_methods = [('PRM','piecewise_regression_custom_range'),
             ('HFM','hyperbola_fitting_unbounded'),
             ('HFM-c','hyperbola_fitting_bounded'),
             ('PLFM','custom_fn_fitting')]

for fitting_method in fitting_methods:
    path = fitting_method[1]+'/asym_Tg_quench.txt'
    data = np.genfromtxt(path)
    cure_percents=data[:,0]
    Tgs=data[:,1]
    cure_percents = np.asarray(cure_percents)
    #Tgs=[0.65,0.75,0.9,2.1]
    Tgs = np.asarray(Tgs)
    Tg_data = np.asarray([cure_percents/100.,Tgs])
    #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
    cure_percents_ss = cure_percents#[:-1]
    Tgs_ss = Tgs#[:-1]
    print(cure_percents_ss)
    print(Tgs_ss)
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                        Tgs_ss,
                                                        T1=None,
                                                        T0=None)
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    ax1.plot(alphas,fit_ydata,label='{}
($R^2$:{})'.format(fitting_method[0],round(R2,3)))
    ax1.scatter(cure_percents/100.,
                Tgs,
                #label='$E_a$:{}'.format(activation_energy),
                color='r')#colors[i])

ax1.set_xlabel('Cure Fraction ($\\alpha$)')
ax1.set_ylabel('$T_g\ (T^*)$')
ax1.legend(fontsize=12,loc='upper left')
ax1.set_ylim(0.5,1.7)
plt.ticklabel_format(axis='y',style='plain')
savefig(plt,'asym_lj_comparing_fitting_methods','dibeneditto.pdf')
```

```
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.7560558   0.82877034  0.92730615  1.05985229]
T1 1.31449512722 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.55596761  0.71575049  0.71337006  0.80419833]
T1 1.00730194738 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 1.16932472  1.18443329  1.22566569  1.34937034]
T1 1.47814845226 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 1.20961687  1.21480038  1.25698223  1.31850432]
T1 1.40474553022 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
```

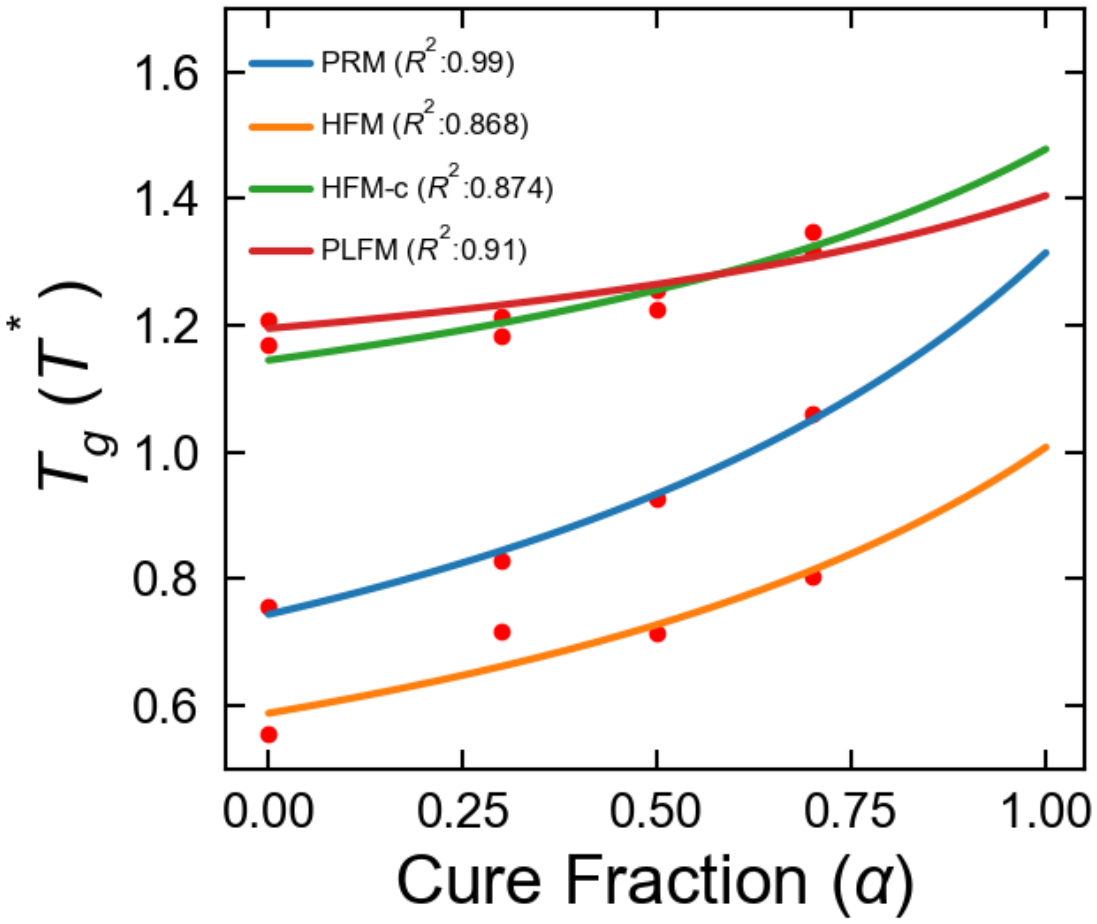
```
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [13]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *
fig, ax1 = plt.subplots()

fitting_methods = [('PRM','piecewise_regression_custom_range'),
             ('HFM','hyperbola_fitting_unbounded'),
             ('HFM-c','hyperbola_fitting_bounded'),
             ('PLFM','custom_fn_fitting')]

for fitting_method in fitting_methods:
    path = fitting_method[1]+'/asym_Tg_quench.txt'
    data = np.genfromtxt(path)
    cure_percents=data[:,0]
    Tgs=data[:,1]
    cure_percents = np.asarray(cure_percents)
    #Tgs=[0.65,0.75,0.9,2.1]
    Tgs = np.asarray(Tgs)
    Tg_data = np.asarray([cure_percents/100.,Tgs])
    #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
    cure_percents_ss = cure_percents#[:-1]
    Tgs_ss = Tgs#[:-1]
    print(cure_percents_ss)
    print(Tgs_ss)
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                        Tgs_ss,
                                                        T1=None,
                                                        T0=None)
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    ax1.plot(alphas,fit_ydata,label='{}
($R^2$:{})'.format(fitting_method[0],round(R2,3)))
    ax1.scatter(cure_percents/100.,
                Tgs,
                #label='$E_a$:{}'.format(activation_energy),
                color='r')#colors[i])

ax1.set_xlabel('Cure Fraction ($\\alpha$)')
ax1.set_ylabel('$T_g\ (T^*)$')
ax1.legend(fontsize=12,loc='upper left')
ax1.set_ylim(0.5,1.7)
plt.ticklabel_format(axis='y',style='plain')
savefig(plt,'asym_lj_comparing_fitting_methods','dibeneditto.pdf')
```

```
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.7560558   0.82877034  0.92730615  1.05985229]
T1 1.31449512722 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.55596761  0.71575049  0.71337006  0.80419833]
T1 1.00730194738 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 1.16932472  1.18443329  1.22566569  1.34937034]
T1 1.47814845226 lambda 0.5
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 1.20961687  1.21480038  1.25698223  1.31850432]
T1 1.40474553022 lambda 0.5


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
```

```
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



F.3 Code for Chapter 5

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

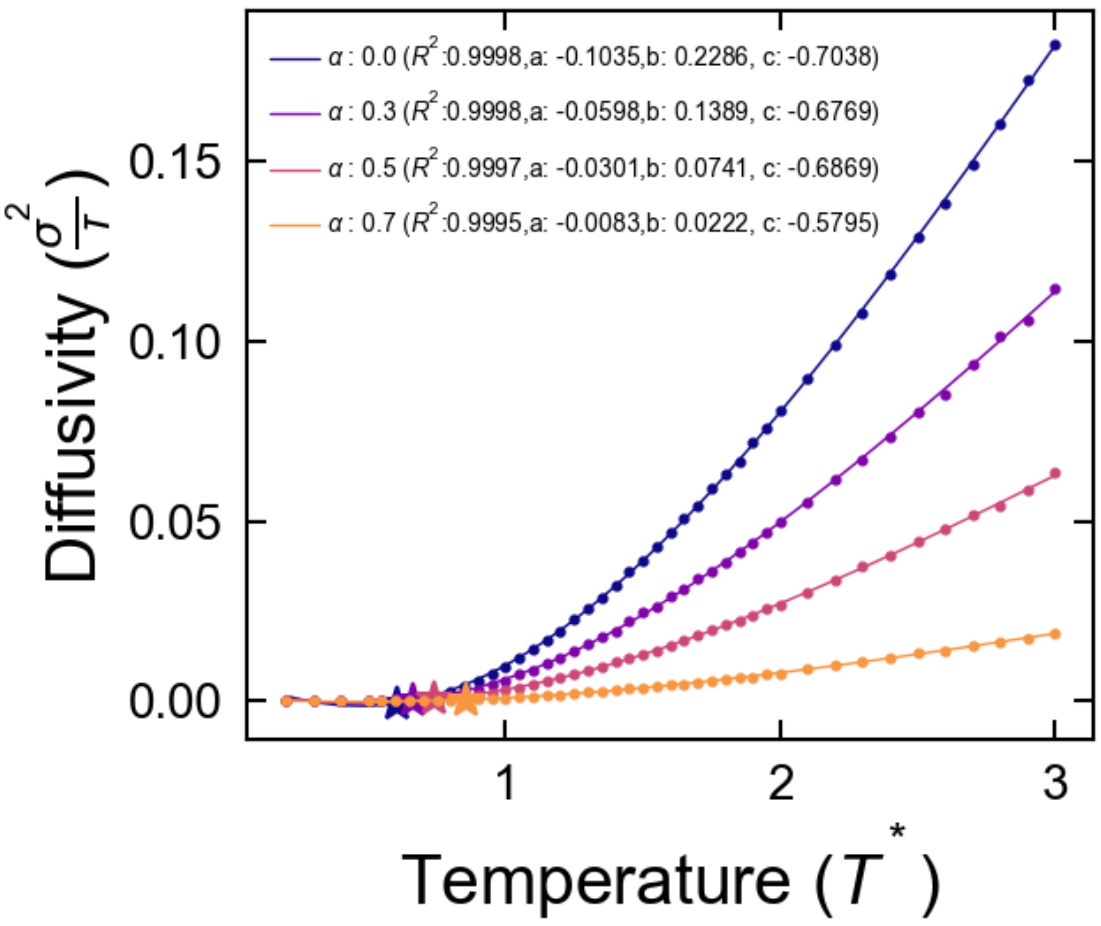
In [2]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *

def H0(x,x0,c):
        return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

def hyper(x,x0,y0,a,b,c):
    #a=0.0
    return y0+a*(x-x0)+b*H0(x,x0,c)

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,x0,a,b,c):
    #a=0.0
    return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

def get_tangent(x,x0,y0,a,b,c):
    m=slope_of_tangent(x,x0,a,b,c)
    y=hyper(x,x0,y0,a,b,c)
    b=y-(m*x)
```

```
        return m,b
```

In [6]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
PROP_NAME ='bparticles'
filter_saps=[0.0,30.,50.,70.]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.2)&
               (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=100
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        #print('To',T0)
        #print('a',a,'b',b)
        #print('Ps',Ps[0],Ps[-1])
        xs = Ts
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plt.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
```

```
                    Ds,
                    marker='.',
                    color=colors[i],#cooling_colors[j],
                    linewidth=0.0)

            m1,b1=get_tangent(Ts[0],*popt)
            m2,b2=get_tangent(Ts[-1],*popt)

            tgx,tgy=line_intersect(m1,b1,m2,b2)
            l1x=np.linspace(Ts[0],tgx+0.1)
            l1y=(m1*l1x)+b1
            #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            plt.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                    color=colors[i],#cooling_colors[j],
                     label='$\\alpha$ : {:.1f} ($R^2$:{:.4f},a: {:.4f},b: {:.4f}, c:
    {:.4f})'.format(sap/100,
          r_squared,
          a,
          b,
          c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        savefig(plt,'hyperbola_fitting_unbounded',
                'all_alphas.pdf')
        np.savetxt('hyperbola_fitting_unbounded/Tg_{}.txt'.format(cooling_method),np.transpose(d
        ata))

        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

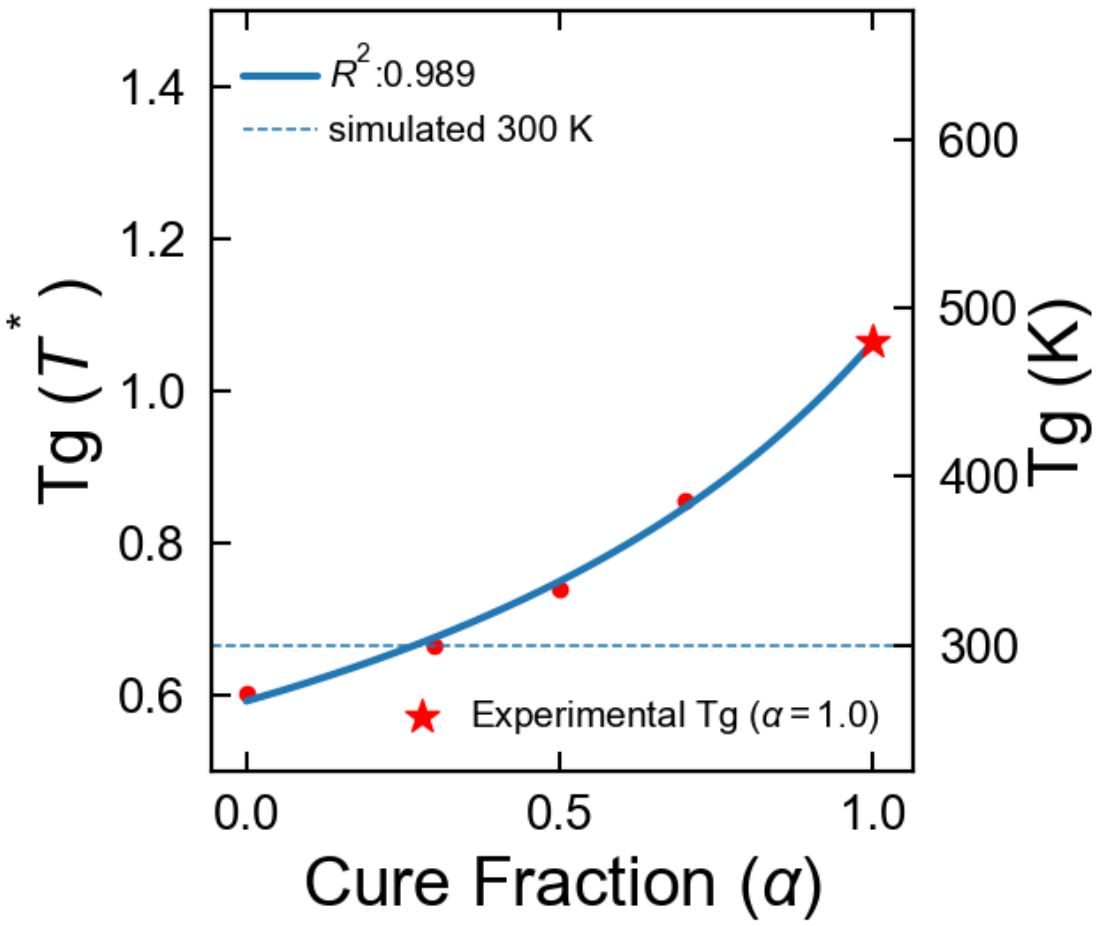


In [7]:
```
cure_percents = np.asarray(cure_percents)
fig, ax1 = plt.subplots()
ax2=ax1.twinx()
Tgs = np.asarray(Tgs)
Tgs_tangent = np.asarray(Tgs_tangent)
#print(Tgs)
Tg_data = np.asarray([cure_percents/100.,Tgs])
#np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
cure_percents_ss = cure_percents#[:-1]
Tgs_ss = Tgs#[:-1]
R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                    Tgs_ss,
                                                    T1=None,
                                                    T0=None)
print('T1',T1,'lambda',inter_parm)
alphas = np.linspace(0,1)
fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
ax1.scatter(cure_percents/100.,
            Tgs,
            color='r')#colors[i])

Tg_sim = T1#0.851796418313
Tg_exp = 480
roomT_exp = 300
Tex_toTsim = Tg_exp/Tg_sim
```

```
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
        ($\\alpha=1.0$)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.5
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.5
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'hyperbola_fitting_unbounded','dibeneditto.pdf')
```

```
T1 1.06455665325 lambda 0.5
300 K in T*: 0.665347908279
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

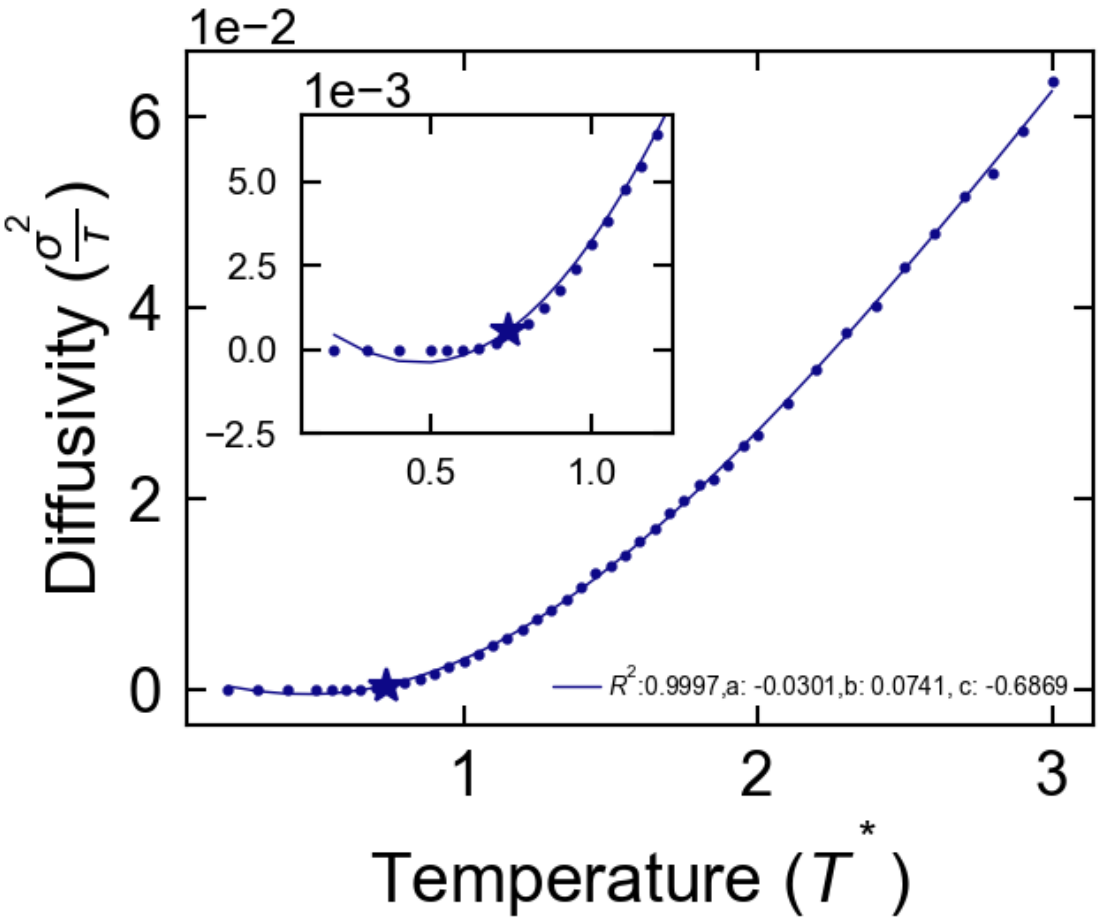



```
In [5]: import matplotlib
        from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression

        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])
        PROP_NAME ='bparticles'
        filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        df_filtered=df[(df.quench_T<=3.0)&
                       (df.quench_T>=0.2)&
                       (df.CC_bond_angle!=109.5)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
```

```
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.calibrationT==305)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        ax1.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
        ax2.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0)

        m1,b1=get_tangent(Ts[0],*popt)
        m2,b2=get_tangent(Ts[-1],*popt)

        tgx,tgy=line_intersect(m1,b1,m2,b2)
        l1x=np.linspace(Ts[0],tgx+0.1)
        l1y=(m1*l1x)+b1
        #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
```

```
            l2x=np.linspace(tgx-0.1,Ts[-1])
            l2y=(m2*l2x)+b2
            #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

            Tgs_tangent.append(tgx)
            #print(yHYP/mul_fact)
            ax1.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                    color=colors[i],#cooling_colors[j],
                     label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                               a,
                                                                               b,
                                                                               c))
            ax2.plot(xs,
                     yHYP/mul_fact,
                     linewidth=1.0,
                     color=colors[i],#cooling_colors[j],
                     label='$R^2$:{:.4f},a: {:.4f},b: {:.4f}, c: {:.4f}'.format(r_squared,
                                                                               a,
                                                                               b,
                                                                               c))

            xvals = np.linspace(-3,4)
            yfits = hyper(xvals, *popt)
            if True:
                Tg=popt[0]
                Tgs.append(Tg)
            else:
                Tgs.append(tgx)
        ax1.legend(fontsize=10,loc='lower right')
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
        ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
        ax1.tick_params(axis = 'both', which = 'major',labelsize=25)
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        ax2.set_xlim(0.1,1.25)
        ax2.set_ylim(-2.5e-3,7e-3)
        ax1.set_xlabel('Temperature ($T^*$)')
        ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        savefig(plt,'hyperbola_fitting_unbounded',
                '50percent.pdf')

        plt.show()
```

```
To 0.738858168024
a -0.0300698093094 b 0.0741400268776
Ps 0.322456700855 0.923541090659


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


1e−2
6
4
2
0
Diffusivity ($\frac{\sigma}{T}^2$)
1e−3
5.0
2.5
0.0
−2.5
0.5
1.0
R^2:0.9997,a: -0.0301,b: 0.0741, c: -0.6869
1
2
3
Temperature (T^*)

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
def get_custom_ranges(cooling_method):
    if cooling_method=='quench':
        custom_ranges_l1={00.0:[0.1,0.8],
                          30.0:[0.1,0.8],
                          50.0:[0.1,0.8],
                          70.0:[0.1,0.8]}
        custom_ranges_l2={00.0:[0.7,1.2],
                          30.0:[0.85,1.4],
                          50.0:[1.0,1.8],
                          70.0:[1.15,2.5]}
    elif cooling_method=='anneal':
        custom_ranges_l1={00.0:[0.1,0.8],
                          30.0:[0.1,0.8],
                          50.0:[0.1,0.8],
                          70.0:[0.1,0.8]}
        custom_ranges_l2={00.0:[0.7,1.2],
                          30.0:[0.85,1.4],
                          50.0:[1.0,1.8],
                          70.0:[1.15,2.5]}
    else:
        raise ValueError(cooling_method+'is unknown')
    return custom_ranges_l1, custom_ranges_l2
```

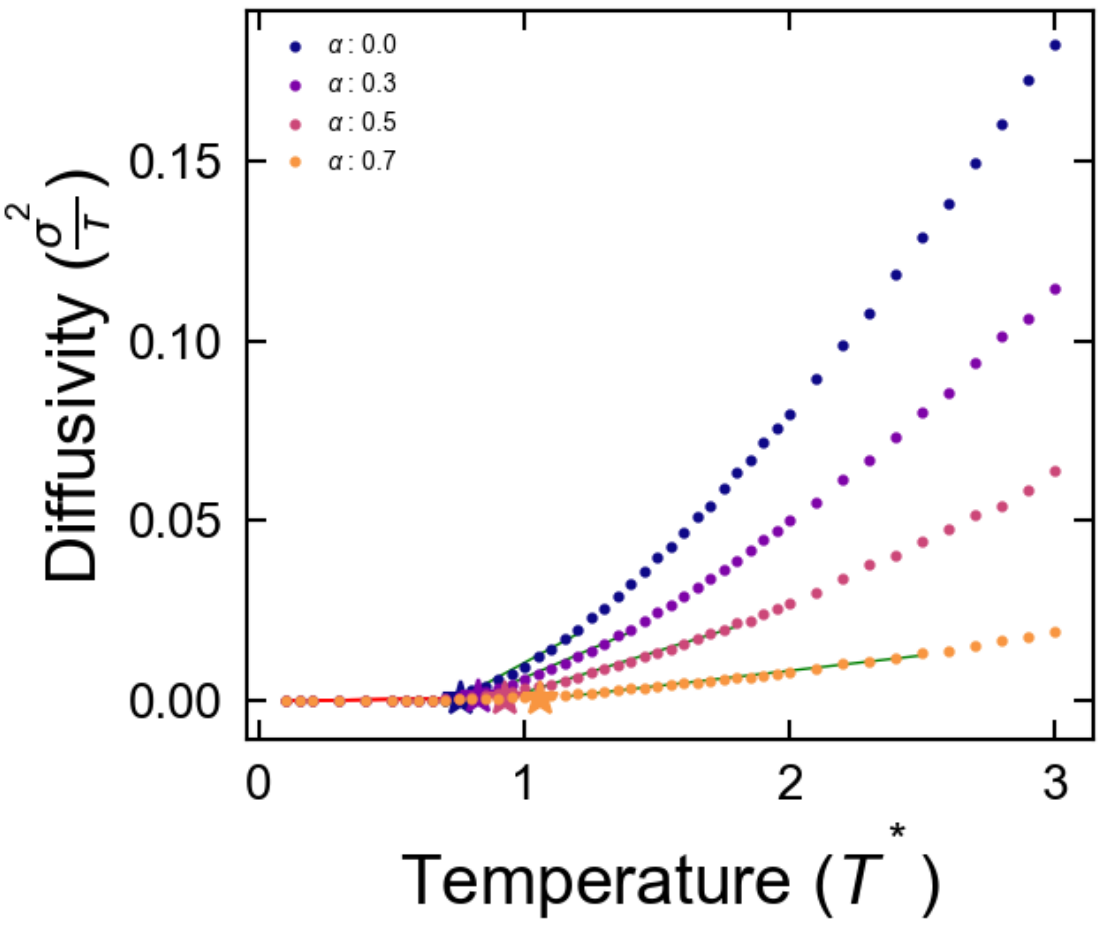
In [8]:
```
import matplotlib
#from common import *
%matplotlib inline
```

```
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
#stop_after_percents = np.arange(10,105,15,dtype=float)
PROP_NAME
='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
#plt.figure()
filter_saps=[0.0,30.,50.,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'

df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.1)&
               (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.calibrationT==305)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=1000000
        Ds_scaled=Ds*mul_fact
        custom_ranges_l1, custom_ranges_l2 = get_custom_ranges(cooling_method)
        print(custom_ranges_l1[sap])
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                Ds_scaled,
                                method='use_viscous_region',
                                min_D=0,
                                ver=4,
                                viscous_line_index=0,
                                l1_T_bounds=custom_ranges_l1[sap],
                                l2_T_bounds=custom_ranges_l2[sap])
        xs = Ts#np.linspace(0.1,4)
        plt.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,

        plt.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 label='$\\alpha$ : {:.1f}'.format(sap/100))


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            plt.plot(xs,
                     ys,
                     color=l_colors[li],
```

```
                        zorder=0,
                        linewidth=1)
                Tgs.append(Tg)
            #break
        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        #plt.xlim(0.5,1.5)
        #plt.ylim(-1e-4,5e-4)
        plt.xlabel('Temperature ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        #savefig(plt,'piecewise_regression_custom_range','all_alphas_zoomed.pdf')
        savefig(plt,'piecewise_regression_custom_range','all_alphas.pdf')
        np.savetxt('piecewise_regression_custom_range/Tg_{}.txt'.format(cooling_method),np.trans
        pose(data))

        plt.show()
```

```
[0.1, 0.8]
[0.1, 0.8]
[0.1, 0.8]
[0.1, 0.8]
```

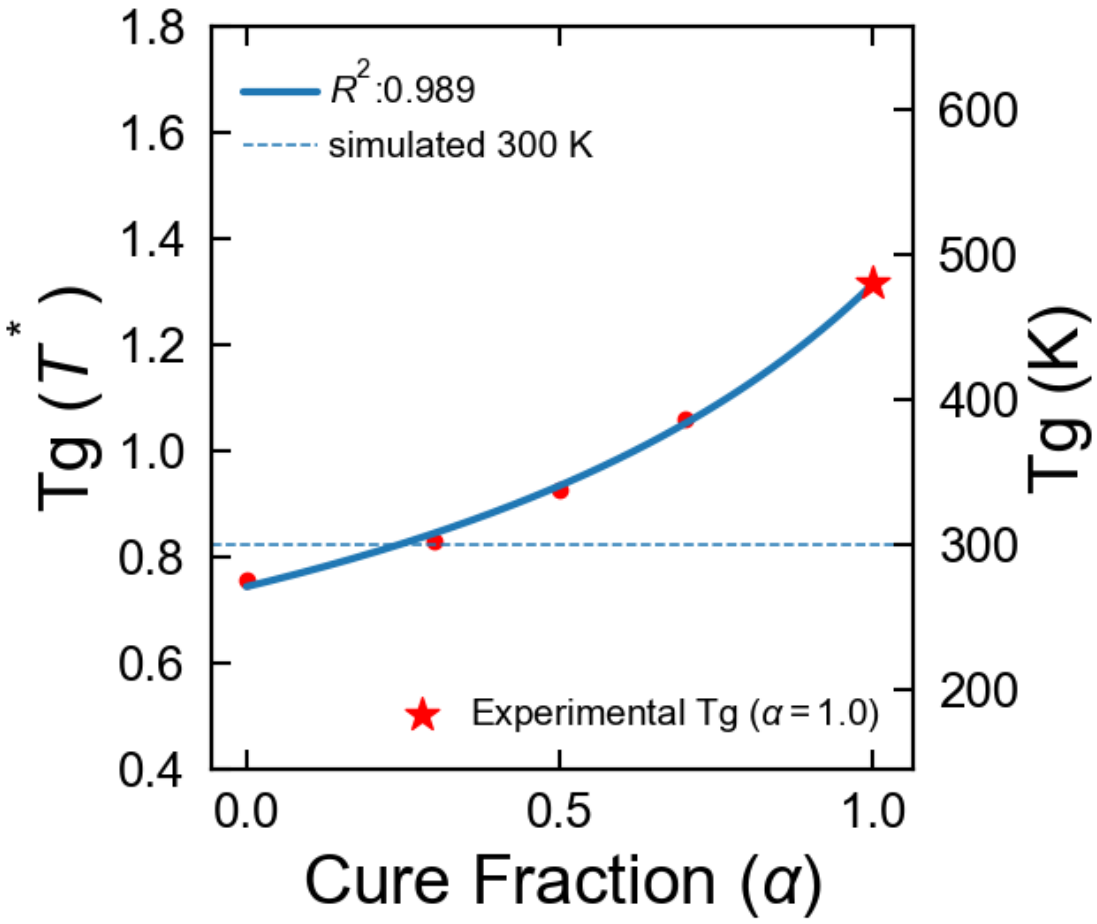
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [9]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        ax2=ax1.twinx()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    color='r')

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
```

```
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
        ($\\alpha=1.0$)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.4
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range','dibeneditto.pdf')
```

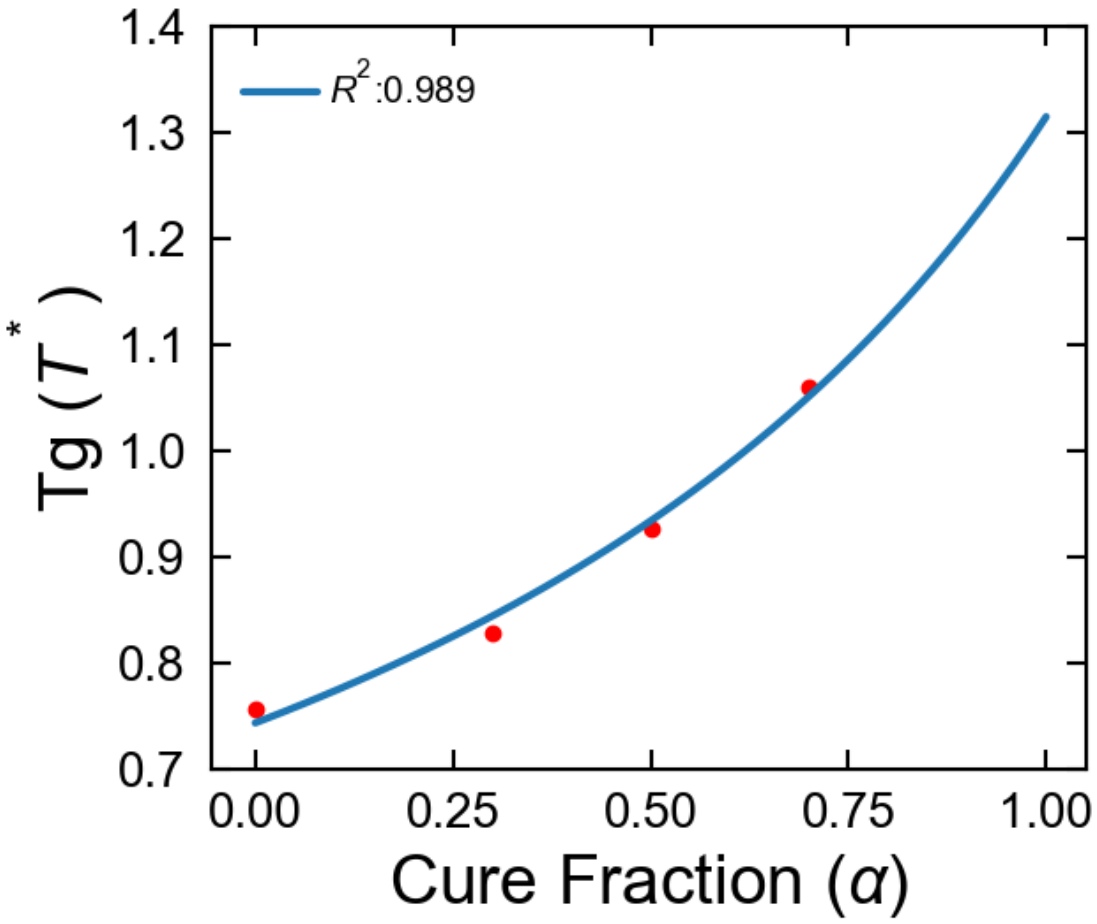
```
[ 0.75650915  0.82738406  0.9269257   1.06032262]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.75650915  0.82738406  0.9269257   1.06032262]
T1 1.31476583141 lambda 0.5
300 K in T*: 0.821728644629


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [5]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    color='r')

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
```

```
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)

        sim_low_lim = 0.7
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.4
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=False
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range','dibeneditto_dimless.pdf')
```

```
[ 0.75650915  0.82738406  0.9269257   1.06032262]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.75650915  0.82738406  0.9269257   1.06032262]
T1 1.31476583141 lambda 0.5
300 K in T*: 0.821728644629


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

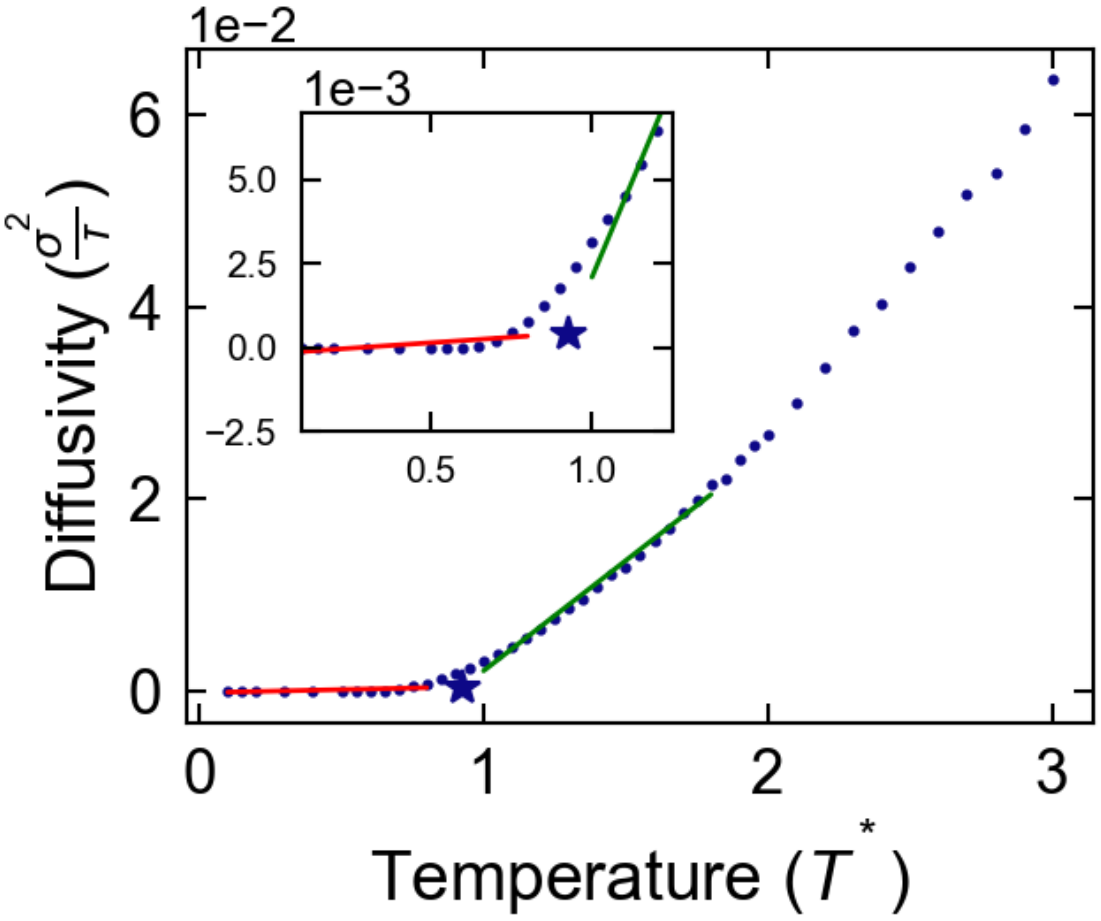



```
In [6]: import matplotlib
        #from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.3, 0.53, 0.30, 0.30]
        ax2 = fig.add_axes([left, bottom, width, height])
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[50.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'

        df_filtered=df[(df.quench_T<=3.0)&
                       (df.quench_T>=0.1)&
```

```
            (df.CC_bond_angle!=109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                       (df_grp.calibrationT==305)&
                       (df_grp.cooling_method==cooling_method)&
                       (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=1000000
        Ds_scaled=Ds*mul_fact
        custom_ranges_l1, custom_ranges_l2 = get_custom_ranges(cooling_method)
        Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                Ds_scaled,
                                method='use_viscous_region',
                                min_D=0,
                                ver=4,
                                viscous_line_index=0,
                                l1_T_bounds=custom_ranges_l1[sap],
                                l2_T_bounds=custom_ranges_l2[sap])
        xs = Ts#np.linspace(0.1,4)
        ax1.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,
        ax2.plot(Tg,
                 Tg_prop/mul_fact,
                 marker='*',
                 color=colors[i],
                 zorder=3,
                 markersize=15)#,

        ax1.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=0)
        ax2.plot(Ts,
                 Ds,
                 marker='.',
                 color=colors[i],#cooling_colors[j],
                 linewidth=0.0,
                 zorder=0)


        #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
        l_colors=['r','g']
        for li,line_val in enumerate(line_vals):
            xs=line_val[0]
            ys=line_val[1]/mul_fact
            ax1.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=1,
                     linewidth=2)
            ax2.plot(xs,
                     ys,
                     color=l_colors[li],
                     zorder=1,
                     linewidth=2)
```

```
            Tgs.append(Tg)
        #break
    ax1.legend(fontsize=10,loc='lower right')
    ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),size=1)
    ax2.tick_params(axis = 'both', which = 'major',labelsize=15)
    ax1.tick_params(axis = 'both', which = 'major',labelsize=25)
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    ax2.set_xlim(0.1,1.25)
    ax2.set_ylim(-2.5e-3,7e-3)
    ax1.set_xlabel('Temperature ($T^*$)')
    ax1.set_ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    savefig(plt,'piecewise_regression_custom_range',
            '50percent.pdf')

    plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
import numpy as np
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
from common import *

def H0(x,x0,c):
        return 1/2*(x-x0)+((x-x0)**2/4+math.exp(c))**0.5

def hyper(x,x0,y0,a,b,c):
    #a=0.0
    return y0+a*(x-x0)+b*H0(x,x0,c)

def Pt(x,x0,c):
    return 0.5+ (x-x0)/(2*((x-x0)**2+4*math.exp(c))**0.5)

def slope_of_tangent(x,x0,a,b,c):
    #a=0.0
    return a+(0.5*b)+(b*(x-x0)/(2*(4*math.exp(c)+(x-x0)**2)**0.5))

def get_tangent(x,x0,y0,a,b,c):
    m=slope_of_tangent(x,x0,a,b,c)
    y=hyper(x,x0,y0,a,b,c)
    b=y-(m*x)
```

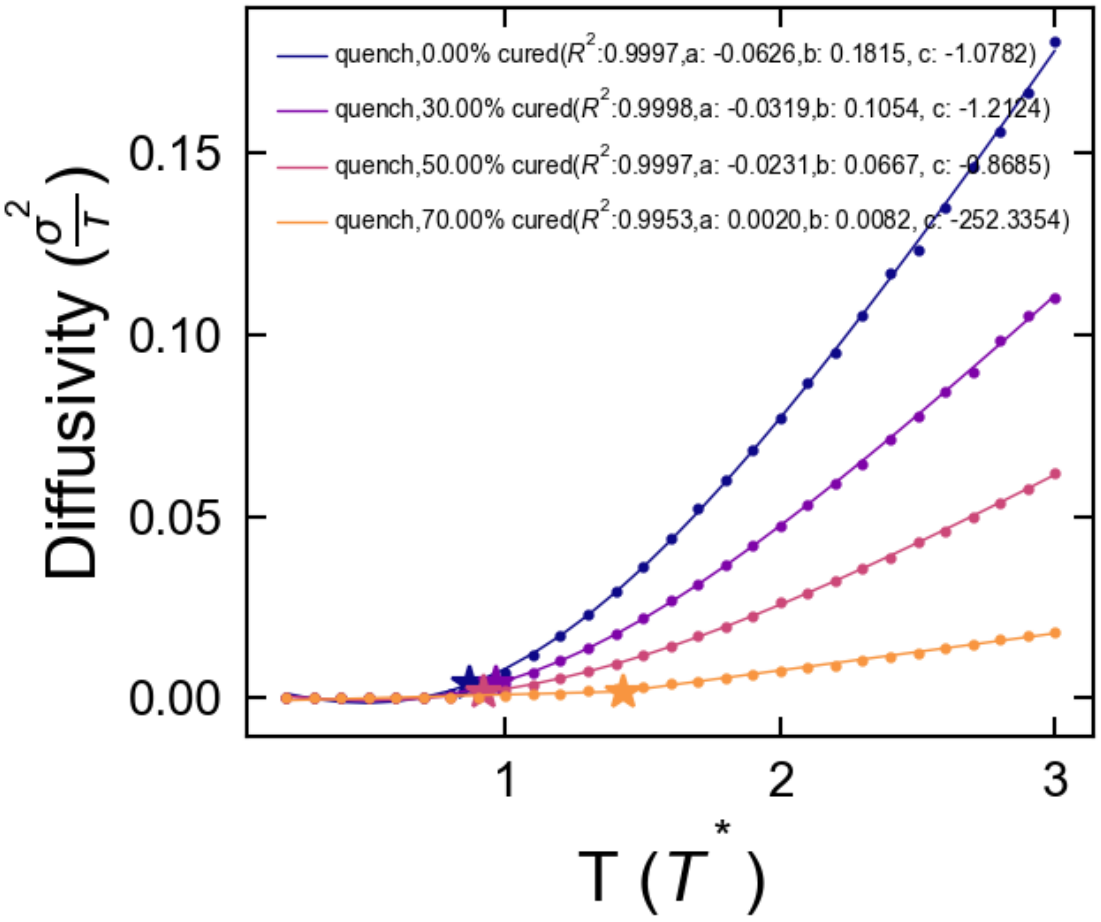
```
        return m,b
```

In [3]:
```
import matplotlib
from common import *
%matplotlib inline
from piecewise.regressor import piecewise
#https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression
PROP_NAME ='bparticles'
filter_saps=[0.,30.,50,70]#[0.0,30.,50.,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,
90]
colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
Tgs=[]
Tgs_tangent=[]
cure_percents = []
Cure_Ts=[]
markers=['+','.']
markersize=[10,10]
cooling_method='quench'
df_filtered=df[(df.quench_T<=3.0)&
               (df.quench_T>=0.20)&
               (df.CC_bond_angle==109.5)&
(df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
for i,sap in enumerate(filter_saps):
    cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
    for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
        df_curing = df_grp[(df_grp.bond==False)&
                           (df_grp.calibrationT==305)&
                           (df_grp.cooling_method==cooling_method)&
                           (df_grp.stop_after_percent==sap)]
        cure_percent = df_curing.cure_percent.mean()
        cure_percents.append(cure_percent)
        Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
        Cure_Ts.append(Ts)

        mul_fact=10000
        Ds_scaled=Ds*mul_fact
        popt, pcov = curve_fit(hyper,
                               Ts,
                               Ds_scaled,
                              maxfev=200000)

        T0=popt[0]#x0
        D0=popt[1]/mul_fact#y0
        a=popt[2]/mul_fact
        b=popt[3]/mul_fact
        c=popt[4]
        Ps=Pt(Ts,T0,popt[4])
        print('To',T0)
        print('a',a,'b',b)
        print('Ps',Ps[0],Ps[-1])
        xs = Ts#np.linspace(0.1,4)
        yHYP = hyper(xs, *popt)
        residuals = Ds_scaled - yHYP
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Ds_scaled-np.mean(Ds_scaled))**2)
        if ss_tot == 0:
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)

        d=hyper(T0,*popt)
        plt.plot(T0,
                 d/mul_fact,
                 marker='*',
                 color=colors[i],
                 markersize=15)#,
```

```
                plt.plot(Ts,
                         Ds,
                         marker='.',
                         color=colors[i],#cooling_colors[j],
                         linewidth=0.0)

                m1,b1=get_tangent(Ts[0],*popt)
                m2,b2=get_tangent(Ts[-1],*popt)

                tgx,tgy=line_intersect(m1,b1,m2,b2)
                l1x=np.linspace(Ts[0],tgx+0.1)
                l1y=(m1*l1x)+b1
                #plt.plot(l1x,l1y/mul_fact,linewidth=1.0)
                l2x=np.linspace(tgx-0.1,Ts[-1])
                l2y=(m2*l2x)+b2
                #plt.plot(l2x,l2y/mul_fact,linewidth=1.0)

                Tgs_tangent.append(tgx)
                #print(yHYP/mul_fact)
                plt.plot(xs,
                         yHYP/mul_fact,
                         linewidth=1.0,
                        color=colors[i],#cooling_colors[j],
                         label='{},{:.2f}% cured($R^2$:{:.4f},a: {:.4f},b: {:.4f}, c:
        {:.4f})'.format(cooling_method,
              sap,
              r_squared,
              a,
              b,
              c))

                xvals = np.linspace(-3,4)
                yfits = hyper(xvals, *popt)
                if True:
                    Tg=popt[0]
                    Tgs.append(Tg)
                else:
                    Tgs.append(tgx)
        plt.legend(fontsize=10)
        plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        Tgs = np.asarray(Tgs)
        cure_percents = np.asarray(cure_percents)
        data=[cure_percents,Tgs]
        plt.xlabel('T ($T^*$)')
        plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
        savefig(plt,'hyperbola_fitting_unbounded_w_bond_angle','all_alphas.pdf')
        np.savetxt('hyperbola_fitting_unbounded_w_bond_angle/Tg_{}.txt'.format(cooling_method),n
        p.transpose(data))

        plt.show()
```

```
To 0.871864932895
a -0.0626145329434 b 0.181472418145
Ps 0.250456038732 0.938449881882
To 0.962093200711
a -0.0318593177601 b 0.105393932144
Ps 0.213646151083 0.940820490982
To 0.921241451531
a -0.0230603562989 b 0.0666647768404
Ps 0.256784868997 0.924341925123


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/scipy/optimize/minpack.py:785: OptimizeWarning: Covariance of the parameters
could not be estimated
```

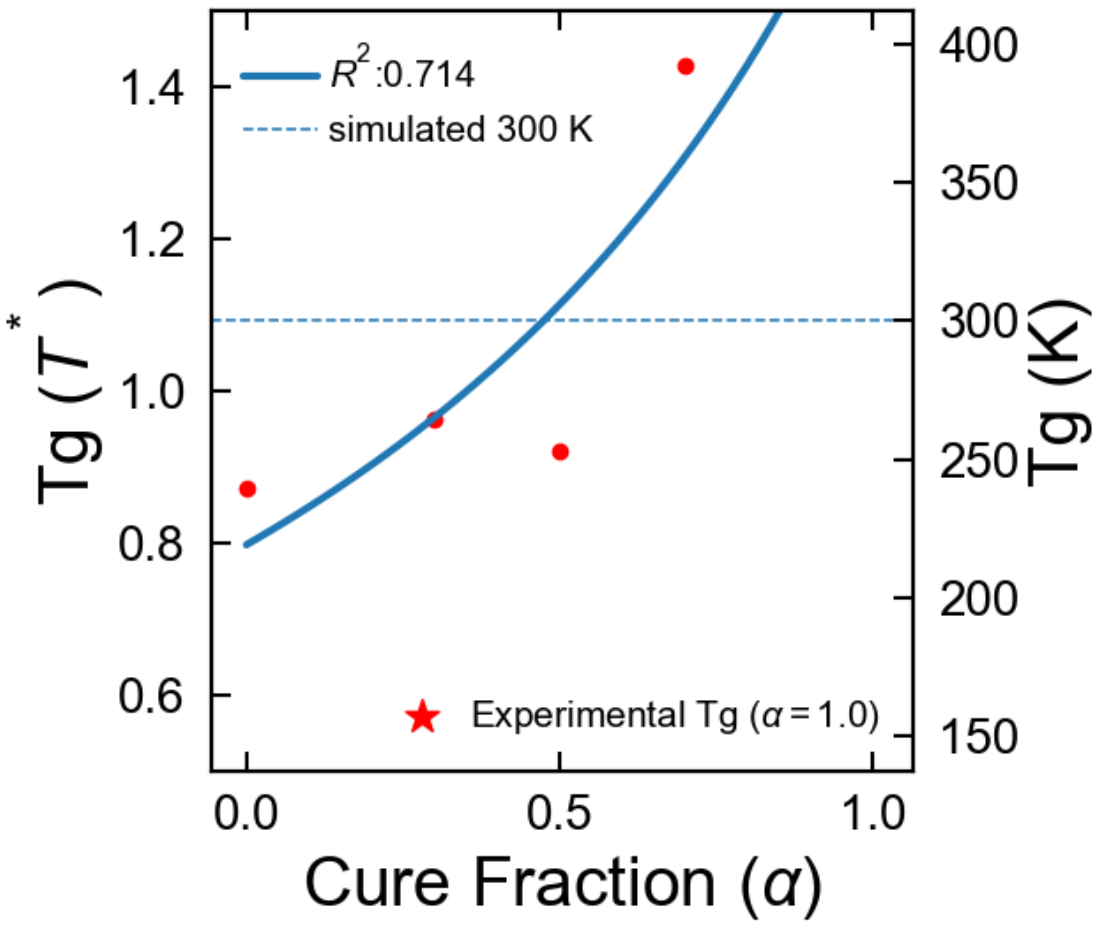
```
  category=OptimizeWarning)


To 1.42801507173
a 0.00197736162304 b 0.00819128372516
Ps 0.0 1.0


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [4]: cure_percents = np.asarray(cure_percents)
        #Tgs=[0.65,0.75,0.9,2.1]
        fig, ax1 = plt.subplots()
        ax2=ax1.twinx()
        Tgs = np.asarray(Tgs)
        Tgs_tangent = np.asarray(Tgs_tangent)
        print(Tgs)
        Tg_data = np.asarray([cure_percents/100.,Tgs])
        #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
```

```
        cure_percents_ss = cure_percents#[:-1]
        Tgs_ss = Tgs#[:-1]
        print(cure_percents_ss)
        print(Tgs_ss)
        R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                            Tgs_ss,
                                                            T1=None,
                                                            T0=None)
        print('T1',T1,'lambda',inter_parm)
        alphas = np.linspace(0,1)
        fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
        ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
        ($\\alpha=1.0$)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.5
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.5
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'hyperbola_fitting_unbounded_w_bond_angle','dibeneditto.pdf')
```

```
[ 0.87186493  0.9620932   0.92124145  1.42801507]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.87186493  0.9620932   0.92124145  1.42801507]
T1 1.74642651184 lambda 0.5
300 K in T*: 1.0915165699


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

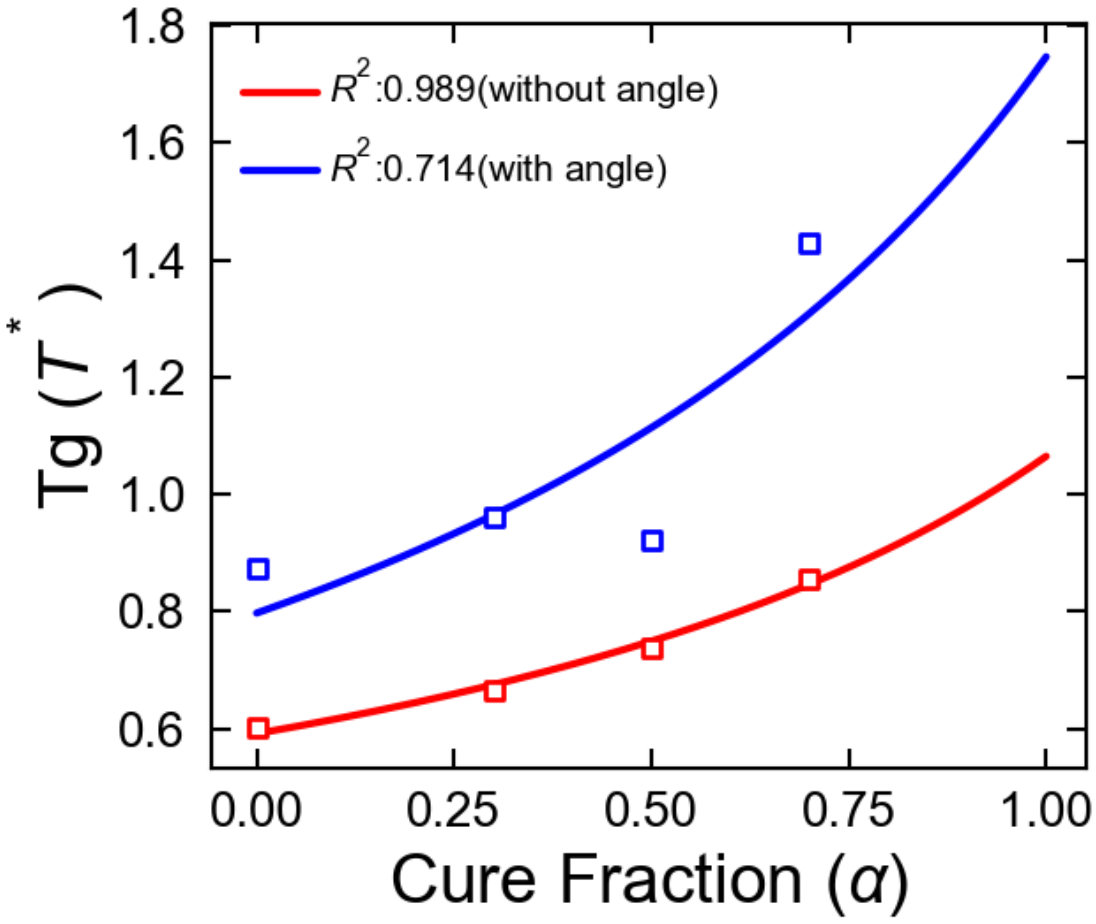



```
In [5]: paths=['hyperbola_fitting_unbounded','hyperbola_fitting_unbounded_w_bond_angle']
        labels=['without angle','with angle']
        colors=['r','b']
        for i,path in enumerate(paths):
            full_path=path+'/Tg_quench.txt'
            data=np.genfromtxt(full_path)
            print(data)
            cp=data[:,0]
            Tg_data=data[:,1]
            R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cp/100.,
                                                                Tg_data,
                                                                T1=None,
                                                                T0=None)
            print('T1',T1,'lambda',inter_parm)
            alphas = np.linspace(0,1)
            fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
            plt.plot(alphas,
                     fit_ydata,
                     colors[i],
                     zorder=0,
                     label='$R^2$:{}({})'.format(round(R2,3),labels[i]))
            plt.scatter(cp/100.,
                        Tg_data,
                        marker='s',
                        facecolor='w',
                        linewidth=2,
```

```
                    edgecolor=colors[i],
                    s=60,
                    color=colors[i],
                   zorder=1)
        plt.legend(fontsize=15)
        plt.xlabel('Cure Fraction ($\\alpha$)')
        plt.ylabel('Tg ($T^*$)')
        savefig(plt,'hyperbola_fitting_unbounded_w_bond_angle','dibeneditto_angle_effect.pdf')
```

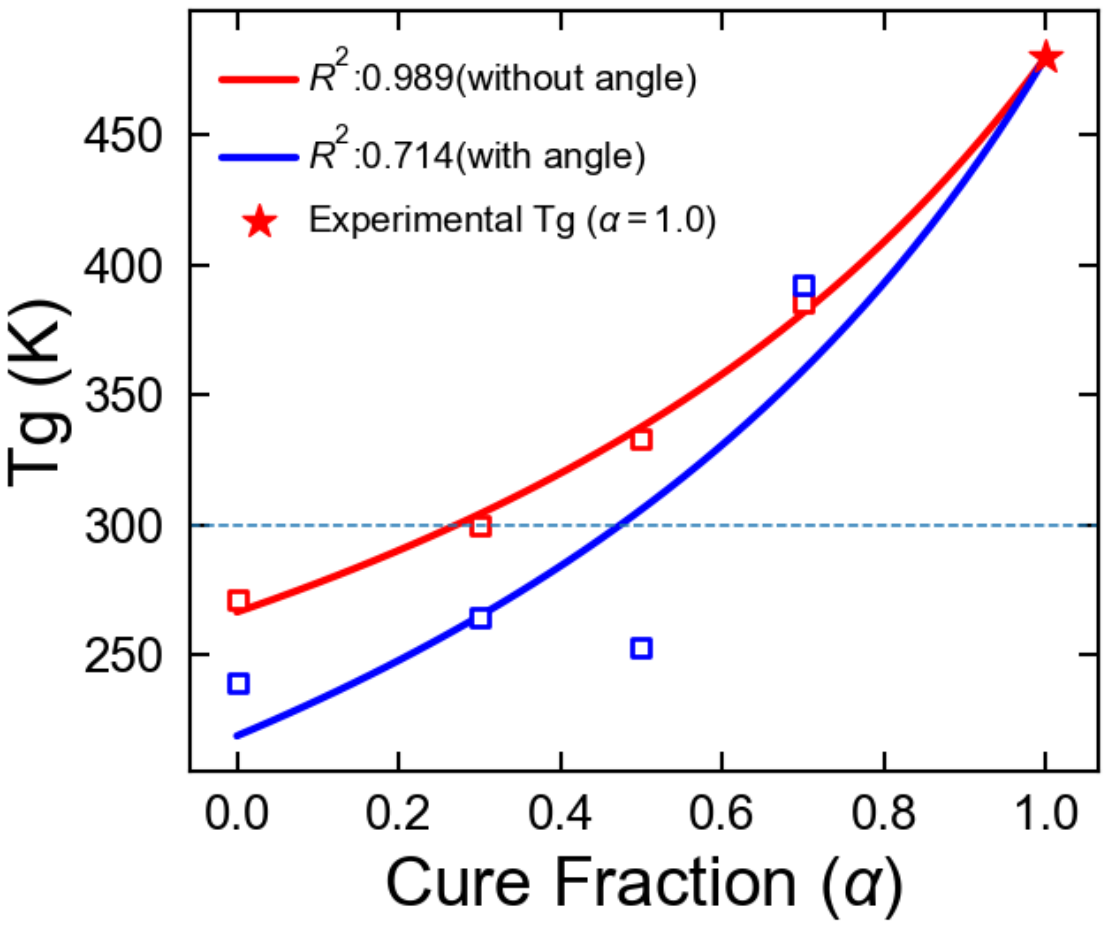
```
[[  2.49999994e-03   6.01926220e-01]
 [  3.00000000e+01   6.64786594e-01]
 [  5.00000000e+01   7.38851124e-01]
 [  7.00000000e+01   8.55726782e-01]]
T1 1.06455665325 lambda 0.5
[[  2.49999994e-03   8.71864933e-01]
 [  3.00000000e+01   9.62093201e-01]
 [  5.00000000e+01   9.21241452e-01]
 [  7.00000000e+01   1.42801507e+00]]
T1 1.74642651184 lambda 0.5
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [6]:
```
paths=['hyperbola_fitting_unbounded','hyperbola_fitting_unbounded_w_bond_angle']
labels=['without angle','with angle']
colors=['r','b']
for i,path in enumerate(paths):
    full_path=path+'/Tg_quench.txt'
    data=np.genfromtxt(full_path)
    cp=data[:,0]
    Tg_data=data[:,1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cp/100.,
                                                        Tg_data,
                                                        T1=None,
                                                        T0=None)
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    fit_ydata=np.asarray(fit_ydata)

    Tg_sim = T1#0.851796418313
    Tg_exp = 480
    roomT_exp = 300
    Tex_toTsim = Tg_exp/Tg_sim
    Tsimtoexp = Tg_exp/Tg_sim
    roomT_sim =  Tg_sim*roomT_exp/Tg_exp
    Tg0_exp = Tg_exp*T0/Tg_sim
    print('300 K in T*:',roomT_sim)
    #print(Tg_data)
    #print(fit_ydata*Tsimtoexp)
    plt.plot(alphas,
             fit_ydata*Tsimtoexp,
             colors[i],
             zorder=0,
             label='$R^2$:{}({})'.format(round(R2,3),labels[i]))
    plt.scatter(cp/100.,
                Tg_data*Tsimtoexp,
                marker='s',
                facecolor='w',
                linewidth=2,
                edgecolor=colors[i],
                s=60,
                #label='$E_a$:{}'.format(activation_energy),
                color=colors[i],
               zorder=1)
plt.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
($\\alpha=1.0$)')
plt.axhline(y=300,linewidth=1.1,linestyle='--')
plt.legend(fontsize=15)
plt.xlabel('Cure Fraction ($\\alpha$)')
plt.ylabel('Tg (K)')
savefig(plt,'hyperbola_fitting_unbounded_w_bond_angle','dibeneditto_angle_effect_SI.pdf'
)
```

```
T1 1.06455665325 lambda 0.5
300 K in T*: 0.665347908279
T1 1.74642651184 lambda 0.5
300 K in T*: 1.0915165699


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


R^2:0.989(without angle)
R^2:0.714(with angle)
Experimental Tg (α = 1.0)
Tg (K)
450
400
350
300
250
0.0
0.2
0.4
0.6
0.8
1.0
Cure Fraction (α)

In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LB_mixing/'

tau=1
tauP=10
num_c10=0
kT = 3.0
N=50000
use_curing_job_P=False
cooling_method = 'quench'
integrator='NPT'
pot='LangH'
activation_energy=3.0
density=1.0
quench_time=1e7
P = 10.0#[4.5, 6, 8]
stop_after_percents = [0.,50.0,70.,100.]#np.arange(0,110,10,dtype=float)#[0.,10.,20.,30.
,40.,50.]#[60.,70.,80.,90.,100.]#[40,50,60,70,80,90,100]#[0.,10.,20.,30.]
#np.arange(0,110,10,dtype=float)#[0.,10.]#,55.,100.]#np.arange(10,105,15,dtype=float)
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
#print(statepoints)
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
def Fit_Diffusivity1(Ts,
                     Ds,
                     method='use_viscous_region',
                     min_D=1e-8,
                     ver=1,
                     viscous_line_index=1,
                     l1_T_bounds=[0,1],
                     l2_T_bounds=[0,1]):
    indices = np.where(Ds>min_D)#0.00000095)
    start_index = indices[0][0]
    D_As=Ds[start_index:]
    quenchTs=Ts[start_index:]
    #print('quenchTs',quenchTs)
    model = piecewise(quenchTs, D_As)
    #print(ver)
    if ver==4:
        #print('ver 4')
        line_vals=[]
        Ts_low_i = np.where(Ts>=l1_T_bounds[0])[0]
        if len(Ts_low_i)==0:
            raise ValueError('lower bound for T fitting of line 1 too low. Use a higher
T')
        l1_low_i = Ts_low_i[0]
        Ts_low_i = np.where(Ts>=l2_T_bounds[0])[0]
```

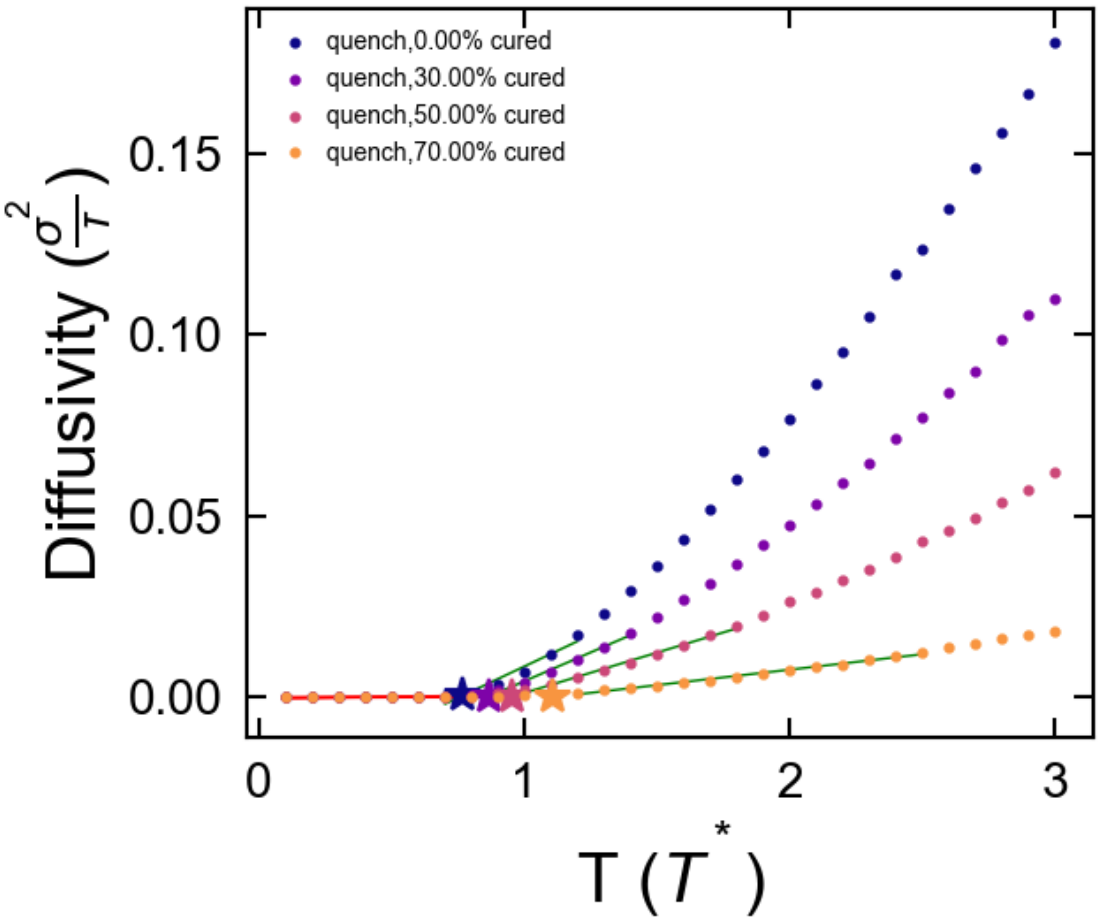
```
        if len(Ts_low_i)==0:
            raise ValueError('lower bound for T fitting of line 2 too low. Use a higher
T')
        l2_low_i = Ts_low_i[0]

        Ts_high_i = np.where(Ts<=l1_T_bounds[1])[0]
        if len(Ts_high_i)==0:
            raise ValueError('upper bound for T fitting of line 1 too high. Use a lower
T')
        l1_high_i = Ts_high_i[-1]
        Ts_high_i = np.where(Ts<=l2_T_bounds[1])[0]
        if len(Ts_high_i)==0:
            raise ValueError('upper bound for T fitting of line 2 too high. Use a lower
T')
        l2_high_i = Ts_high_i[-1]
        #print('Ts_high_i',Ts_high_i)
        l1Ts=Ts[l1_low_i:l1_high_i+1]
        l1Ds=Ds[l1_low_i:l1_high_i+1]
        #print(l1_low_i,l1_high_i,l1Ts)
        l2Ts=Ts[l2_low_i:l2_high_i+1]
        l2Ds=Ds[l2_low_i:l2_high_i+1]
        #print(l2_low_i,l2_high_i,l2Ts,'Ts',Ts)
        par = np.polyfit(l1Ts, l1Ds, 1, full=True)
        m1 = par[0][0]#0-slope, 1-intercept
        b1 = par[0][1]
        xs = np.linspace(l1Ts[0],l1Ts[-1])
        ys = m1*xs+b1
        line_vals.append((xs,ys))

        par = np.polyfit(l2Ts, l2Ds, 1, full=True)
        m2 = par[0][0]#0-slope, 1-intercept
        b2 = par[0][1]
        xs = np.linspace(l2Ts[0],l2Ts[-1])
        ys = m2*xs+b2
        line_vals.append((xs,ys))

        x,y = line_intersect(m1,b1,m2,b2)
        Tg=x
        Tg_prop = y

        return Tg,Tg_prop,line_vals
    if ver==3:
        line_vals=[]
        Ts_low_i = np.where(Ts>=l1_T_bounds[0])[0]
        if len(Ts_low_i)==0:
            raise ValueError('lower bound for T fitting of line 1 too low. Use a higher
T')
        l1_low_i = Ts_low_i[0]
        Ts_low_i = np.where(Ts>=l2_T_bounds[0])[0]
        if len(Ts_low_i)==0:
            raise ValueError('lower bound for T fitting of line 2 too low. Use a higher
T')
        l2_low_i = Ts_low_i[0]

        Ts_high_i = np.where(Ts<=l1_T_bounds[1])[0]
        if len(Ts_high_i)==0:
            raise ValueError('upper bound for T fitting of line 1 too high. Use a lower
T')
        l1_high_i = Ts_high_i[-1]
        Ts_high_i = np.where(Ts<=l2_T_bounds[1])[0]
        if len(Ts_high_i)==0:
            raise ValueError('upper bound for T fitting of line 2 too high. Use a lower
T')
        l2_high_i = Ts_high_i[-1]
        #print('Ts_high_i',Ts_high_i)
        l1Ts=Ts[l1_low_i:l1_high_i+1]
        l1Ds=Ds[l1_low_i:l1_high_i+1]
        #print(l1_low_i,l1_high_i,l1Ts)
```

```
        l2Ts=Ts[l2_low_i:l2_high_i+1]
        l2Ds=Ds[l2_low_i:l2_high_i+1]
        #print(l2_low_i,l2_high_i,l2Ts,'Ts',Ts)
        par = np.polyfit(l1Ts, l1Ds, 1, full=True)
        m1 = par[0][0]#0-slope, 1-intercept
        b1 = par[0][1]
        xs = np.linspace(l1Ts[0],l1Ts[-1])
        ys = m1*xs+b1
        line_vals.append((xs,ys))

        par = np.polyfit(l2Ts, l2Ds, 1, full=True)
        m2 = par[0][0]#0-slope, 1-intercept
        b2 = par[0][1]
        xs = np.linspace(l2Ts[0],l2Ts[-1])
        ys = m2*xs+b2
        line_vals.append((xs,ys))
        if viscous_line_index==0:
            Tg = -b1/m1
        elif viscous_line_index==1:
            Tg = -b2/m2
        Tg_prop = 0.
        return Tg,Tg_prop,line_vals
    elif ver==2:
        n_lines=len(model.segments)
        if n_lines == 0:
            raise ValueError('Found zero lines in piecewise fitting')
        lines=[]
        line_vals=[]
        for i in range(n_lines):
            line = model.segments[i]
            lines.append(line)
            xs = np.linspace(line.start_t,line.end_t)
            ys = line.coeffs[1]*xs+line.coeffs[0]
            line_vals.append((xs,ys))

        if method=='use_viscous_region':
            if n_lines>1:
                l2=lines[viscous_line_index]
            else:
                l2=lines[0]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            Tg = -b2/m2
            Tg_prop = 0.
        else:
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
        return Tg,Tg_prop,line_vals
    else:
        if len(model.segments) == 2:
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs1 =
np.linspace(l1.start_t,l1.end_t)#np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys1 = l1.coeffs[1]*xs1+l1.coeffs[0]
            xs2 =
np.linspace(l2.start_t,l2.end_t)#np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys2 = l2.coeffs[1]*xs2+l2.coeffs[0]
        else:
            print('WARNING: found {} line segments in
regression!'.format(len(model.segments)))
        if method=='use_viscous_region':
            Tg = -b2/m2
            Tg_prop = 0.
```

```
            else:
                Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
            return Tg,Tg_prop,xs1,ys1,xs2,ys2

In [3]: def get_custom_ranges(cooling_method):
            if cooling_method=='quench':
                custom_ranges_l1={00.0:[0.1,0.8],
                                  30.0:[0.1,0.8],
                                  50.0:[0.1,0.8],
                                  70.0:[0.1,0.8]}
                custom_ranges_l2={00.0:[0.7,1.2],
                                  30.0:[0.85,1.4],
                                  50.0:[1.0,1.8],
                                  70.0:[1.15,2.5]}
            elif cooling_method=='anneal':
                custom_ranges_l1={00.0:[0.1,0.8],
                                  30.0:[0.1,0.8],
                                  50.0:[0.1,0.8],
                                  70.0:[0.1,0.8]}
                custom_ranges_l2={00.0:[0.7,1.2],
                                  30.0:[0.85,1.4],
                                  50.0:[1.0,1.8],
                                  70.0:[1.15,2.5]}
            else:
                raise ValueError(cooling_method+'is unknown')
            return custom_ranges_l1, custom_ranges_l2

In [9]: import matplotlib
        #from common import *
        %matplotlib inline
        from piecewise.regressor import piecewise
        #https://www.datadoghq.com/blog/engineering/piecewise-regression/
        from piecewise.plotter import plot_data_with_regression
        #stop_after_percents = np.arange(10,105,15,dtype=float)
        PROP_NAME
        ='bparticles'#'volume'#'pair_lj_energy','bond_harmonic_energy'#'potential_energy'
        #plt.figure()
        filter_saps=[0.0,30.,50.,70.]#,100.]#,100.]#[0.0,50.0,100.0]#,30,50,70]#,90]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(filter_saps)))
        Tgs=[]
        Tgs_tangent=[]
        cure_percents = []
        Cure_Ts=[]
        markers=['+','.']
        markersize=[10,10]
        cooling_method='quench'
        custom_ranges_l1=[[0.1,0.5],
                          [0.1,0.8],
                          [0.1,0.8],
                          [0.1,0.8]]
        custom_ranges_l2=[[0.8,1.2],
                          [0.85,1.4],
                          [1.0,1.8],
                          [1.15,2.5]]
        custom_ranges_l1=[[0.1,0.8],
                          [0.1,0.8],
                          [0.1,0.8],
                          [0.1,0.8]]
        custom_ranges_l2=[[0.7,1.2],
                          [0.9,1.2],
                          [1.0,1.8],
                          [1.15,2.5]]

        df_filtered=df[(df.quench_T<=3.0)&
                       (df.quench_T>=0.1)&
                       (df.CC_bond_angle==109.5)&
        (df.cooling_method==cooling_method)]#(df.quench_T<=3.0)&(df.quench_T>=0.05)&
        for i,sap in enumerate(filter_saps):
```

```
        cooling_colors = plt.cm.plasma(np.linspace(0,0.75,2))
        for j,(cooling_method,df_grp) in enumerate(df_filtered.groupby('cooling_method')):
            df_curing = df_grp[(df_grp.bond==False)&
                               (df_grp.calibrationT==305)&
                               (df_grp.cooling_method==cooling_method)&
                               (df_grp.stop_after_percent==sap)]
            cure_percent = df_curing.cure_percent.mean()
            cure_percents.append(cure_percent)
            Ts,Ds=getDiffusivities(project,df_curing,name=PROP_NAME)
            Cure_Ts.append(Ts)

            mul_fact=1000000
            Ds_scaled=Ds*mul_fact
            custom_ranges_l1,custom_ranges_l2=get_custom_ranges(cooling_method)
            Tg,Tg_prop,line_vals = Fit_Diffusivity1(Ts,
                                        Ds_scaled,
                                        ver=4,
                                        l1_T_bounds=custom_ranges_l1[sap],
                                        l2_T_bounds=custom_ranges_l2[sap])
            xs = Ts#np.linspace(0.1,4)
            plt.plot(Tg,
                     Tg_prop/mul_fact,
                     marker='*',
                     color=colors[i],
                     markersize=15)#,

            plt.plot(Ts,
                     Ds,
                     marker='.',
                     color=colors[i],#cooling_colors[j],
                     linewidth=0.0,
                    label='{},{:.2f}% cured'.format(cooling_method,
                                                                    sap))


            #l_colors = plt.cm.coolwarm(np.linspace(0,0.75,len(line_vals)))
            l_colors=['r','g']
            for li,line_val in enumerate(line_vals):
                xs=line_val[0]
                ys=line_val[1]/mul_fact
                plt.plot(xs,
                         ys,
                         color=l_colors[li],
                         zorder=0,
                         linewidth=1)
            Tgs.append(Tg)
        #break
    plt.legend(fontsize=10)
    plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
    Tgs = np.asarray(Tgs)
    cure_percents = np.asarray(cure_percents)
    data=[cure_percents,Tgs]
    #plt.xlim(0.2,2.5)
    #plt.ylim(-1e-4,1e-3)
    plt.xlabel('T ($T^*$)')
    plt.ylabel('Diffusivity ($\\frac{\\sigma^2}{\\tau}$)')
    savefig(plt,'piecewise_regression_custom_range_bond_angle','all_alphas_zoomed.pdf')
    savefig(plt,'piecewise_regression_custom_range_bond_angle','all_alphas.pdf')
    np.savetxt('piecewise_regression_custom_range_bond_angle/Tg_{}.txt'.format(cooling_metho
    d),np.transpose(data))

    plt.show()
```

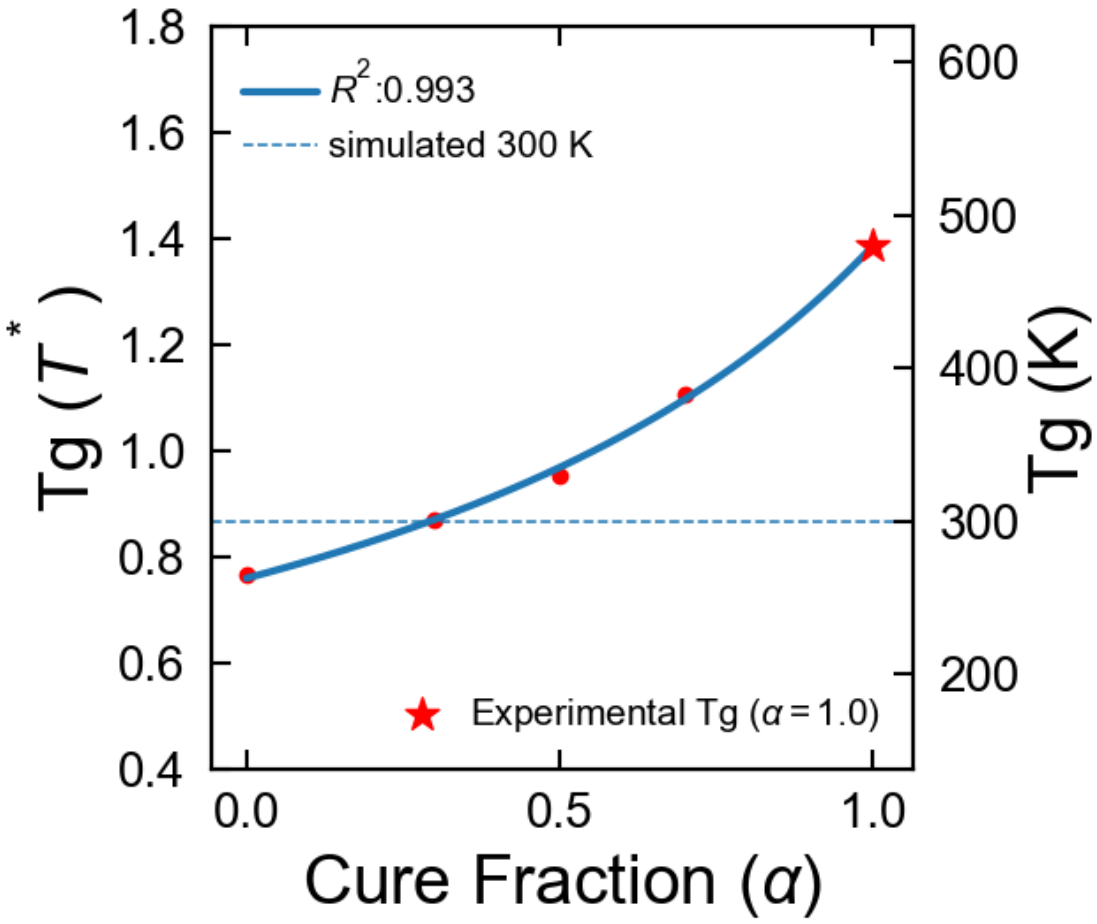
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [10]: cure_percents = np.asarray(cure_percents)
         #Tgs=[0.65,0.75,0.9,2.1]
         fig, ax1 = plt.subplots()
         ax2=ax1.twinx()
         Tgs = np.asarray(Tgs)
         Tgs_tangent = np.asarray(Tgs_tangent)
         print(Tgs)
         Tg_data = np.asarray([cure_percents/100.,Tgs])
         #np.savetxt('DGEBA_DDS__PES_w_angle_Tg.txt',Tg_data)
         cure_percents_ss = cure_percents#[:-1]
         Tgs_ss = Tgs#[:-1]
         print(cure_percents_ss)
         print(Tgs_ss)
         R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cure_percents_ss/100.,
                                                             Tgs_ss,
                                                             T1=None,
                                                             T0=None)
         print('T1',T1,'lambda',inter_parm)
         #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
         #print(cure_percents_ss)
         alphas = np.linspace(0,1)
         fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
         ax1.plot(alphas,fit_ydata,label='$R^2$:{}'.format(round(R2,3)))
```

```
        ax1.scatter(cure_percents/100.,
                    Tgs,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='r')#colors[i])

        Tg_sim = T1#0.851796418313
        Tg_exp = 480
        roomT_exp = 300
        Tex_toTsim = Tg_exp/Tg_sim
        roomT_sim =  Tg_sim*roomT_exp/Tg_exp
        Tg0_exp = Tg_exp*T0/Tg_sim
        print('300 K in T*:',roomT_sim)
        ax2.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
        ($\\alpha=1.0$)')
        ax2.set_ylabel('Tg (K)')

        sim_low_lim = 0.4
        ex_low_lim = sim_low_lim*Tex_toTsim
        sim_up_lim = 1.8
        ex_up_lim = sim_up_lim*Tex_toTsim
        ax2.set_ylim(ex_low_lim,ex_up_lim)
        ax1.set_ylim(sim_low_lim,sim_up_lim)
        #ax1.set_ylim(0,3)
        show_roomT=True
        if show_roomT:
            ax1.axhline(y=roomT_sim,linewidth=1.1,linestyle='--',label='simulated 300 K')
        ax1.set_xlabel('Cure Fraction ($\\alpha$)')
        ax1.set_ylabel('Tg ($T^*$)')
        ax1.legend(fontsize=15,loc='upper left')
        ax2.legend(fontsize=15,loc='lower right')
        plt.ticklabel_format(axis='y',style='plain')
        savefig(plt,'piecewise_regression_custom_range_bond_angle','dibeneditto.pdf')
```

```
[ 0.76575988  0.86836421  0.95199585  1.10718064]
[  2.49999994e-03   3.00000000e+01   5.00000000e+01   7.00000000e+01]
[ 0.76575988  0.86836421  0.95199585  1.10718064]
T1 1.38657003106 lambda 0.5
300 K in T*: 0.866606269412


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

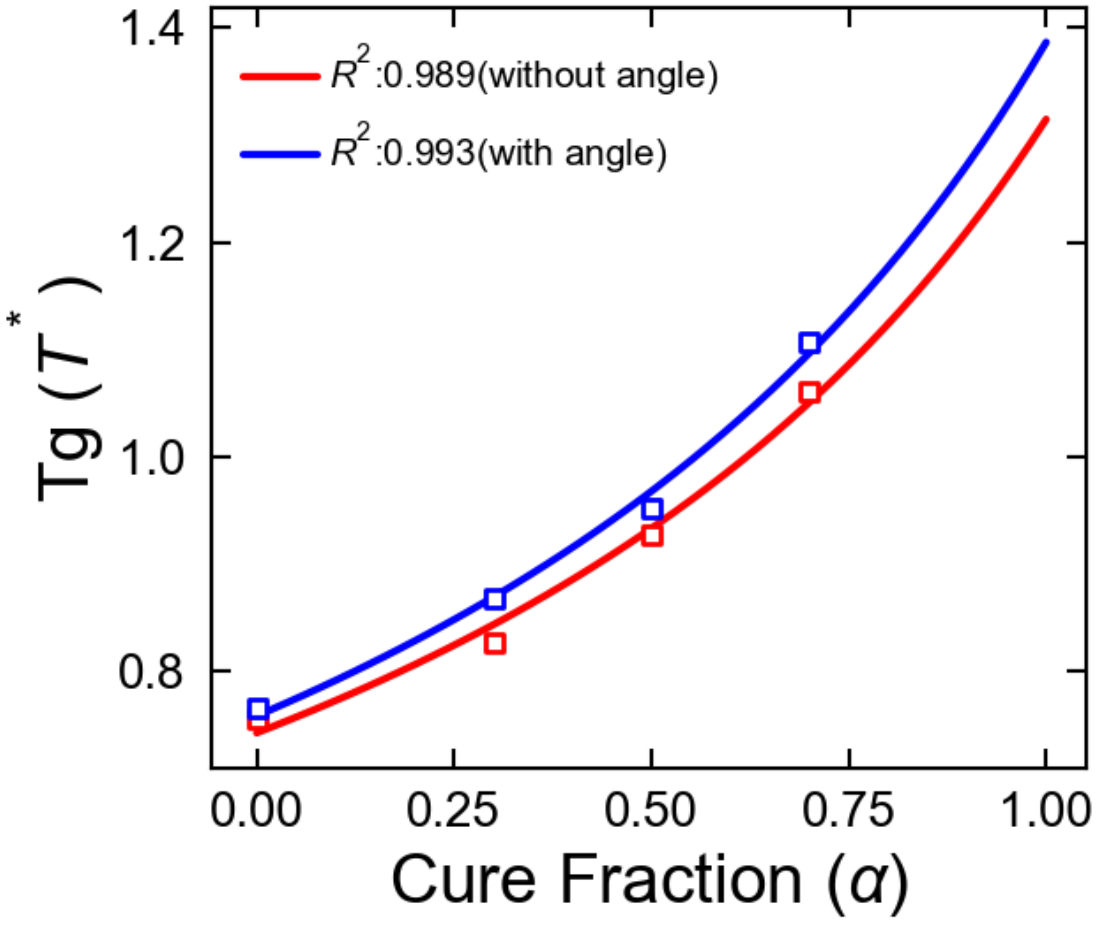


In [6]:
```
paths=['piecewise_regression_custom_range','piecewise_regression_custom_range_bond_angle
']
labels=['without angle','with angle']
colors=['r','b']
for i,path in enumerate(paths):
    full_path=path+'/Tg_quench.txt'
    data=np.genfromtxt(full_path)
    cp=data[:,0]
    Tg_data=data[:,1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cp/100.,
                                                        Tg_data,
                                                        T1=None,
                                                        T0=None)
    print(labels[i],'T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    plt.plot(alphas,
             fit_ydata,
             colors[i],
             zorder=0,
             label='$R^2$:{}({})'.format(round(R2,3),labels[i]))
    plt.scatter(cp/100.,
                Tg_data,
                marker='s',
```

```
                    facecolor='w',
                    linewidth=2,
                    edgecolor=colors[i],
                    s=60,
                    #label='$E_a$:{}'.format(activation_energy),
                    color=colors[i],
                   zorder=1)
        plt.legend(fontsize=15)
        plt.xlabel('Cure Fraction ($\\alpha$)')
        plt.ylabel('Tg ($T^*$)')
        savefig(plt,'piecewise_regression_custom_range_bond_angle','dibeneditto_angle_effect.pdf
        ')
```

```
without angle T1 1.31476583141 lambda 0.5
with angle T1 1.38657003106 lambda 0.5
```

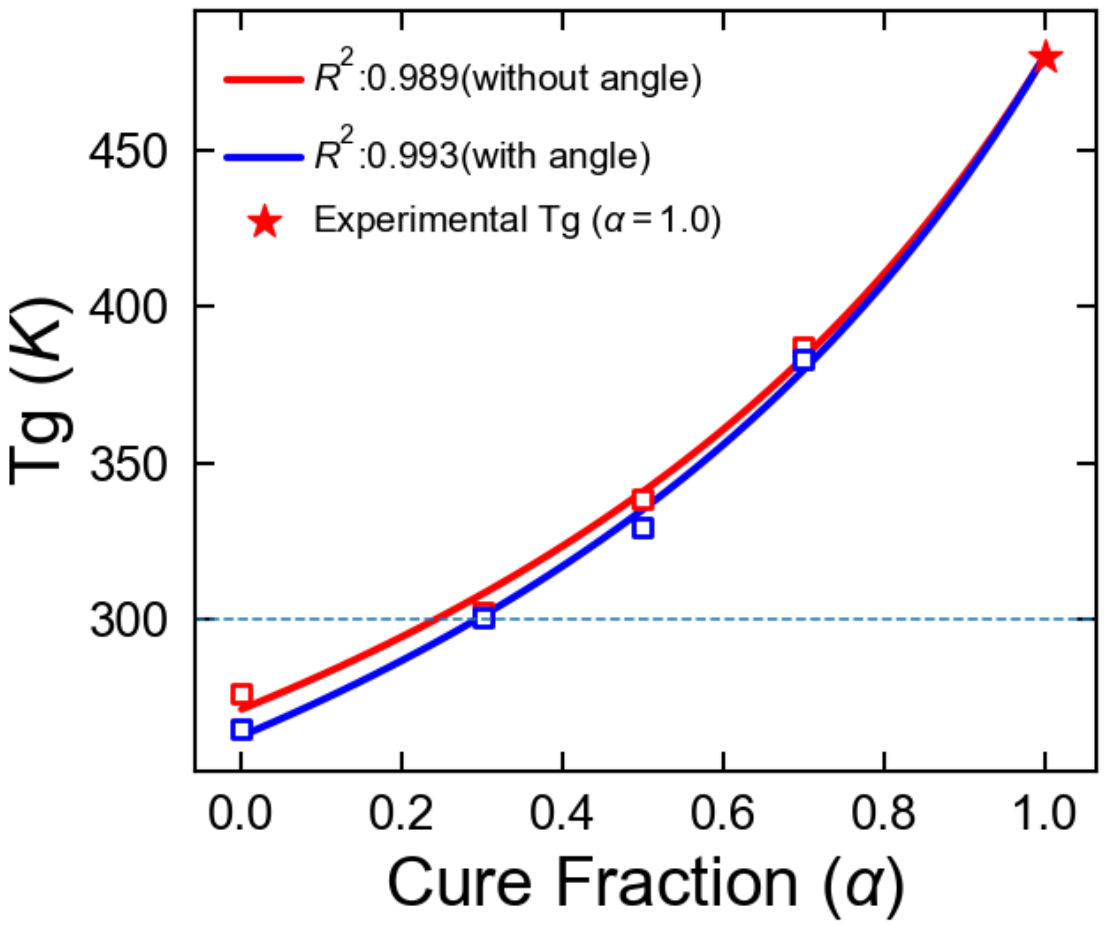
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [7]:
```
paths=['piecewise_regression_custom_range','piecewise_regression_custom_range_bond_angle
']
labels=['without angle','with angle']
colors=['r','b']
for i,path in enumerate(paths):
    full_path=path+'/Tg_quench.txt'
    data=np.genfromtxt(full_path)
    cp=data[:,0]
    Tg_data=data[:,1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cp/100.,
                                                        Tg_data,
                                                        T1=None,
                                                        T0=None)
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    fit_ydata=np.asarray(fit_ydata)

    Tg_sim = T1#0.851796418313
    Tg_exp = 480
    roomT_exp = 300
    Tex_toTsim = Tg_exp/Tg_sim
    Tsimtoexp = Tg_exp/Tg_sim
    roomT_sim =  Tg_sim*roomT_exp/Tg_exp
    Tg0_exp = Tg_exp*T0/Tg_sim
    print('300 K in T*:',roomT_sim)
    #print(Tg_data)
    #print(fit_ydata*Tsimtoexp)
    plt.plot(alphas,
             fit_ydata*Tsimtoexp,
             colors[i],
             zorder=0,
             label='$R^2$:{}({})'.format(round(R2,3),labels[i]))
    plt.scatter(cp/100.,
                Tg_data*Tsimtoexp,
                marker='s',
                facecolor='w',
                linewidth=2,
                edgecolor=colors[i],
                s=60,
                #label='$E_a$:{}'.format(activation_energy),
                color=colors[i],
               zorder=1)
plt.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Experimental Tg
($\\alpha=1.0$)')
plt.axhline(y=300,linewidth=1.1,linestyle='--')
plt.legend(fontsize=15)
plt.xlabel('Cure Fraction ($\\alpha$)')
plt.ylabel('Tg ($K$)')
savefig(plt,'piecewise_regression_custom_range_bond_angle','dibeneditto_angle_effect_SI.
pdf')
```

```
T1 1.31476583141 lambda 0.5
300 K in T*: 0.821728644629
T1 1.38657003106 lambda 0.5
300 K in T*: 0.866606269412
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



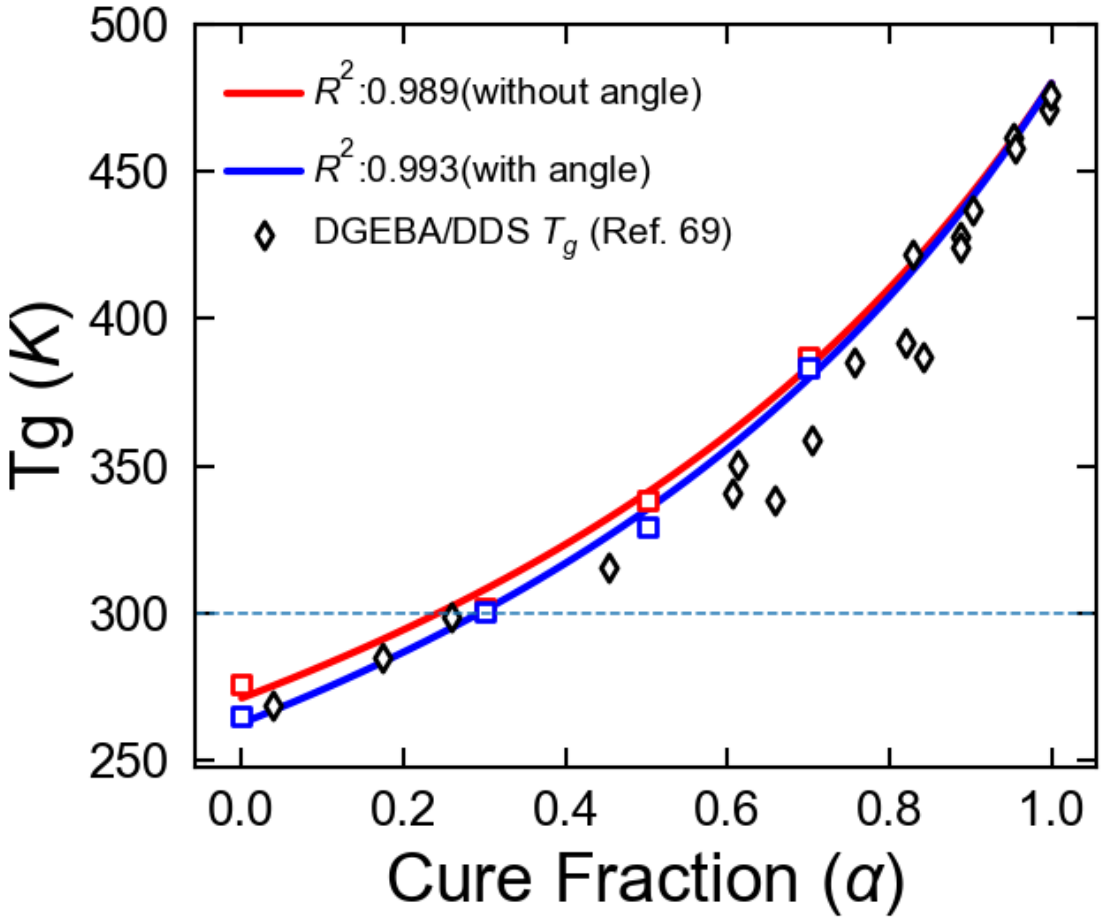
In [8]:
```
paths=['piecewise_regression_custom_range','piecewise_regression_custom_range_bond_angle
']
labels=['without angle','with angle']
colors=['r','b']
for i,path in enumerate(paths):
    full_path=path+'/Tg_quench.txt'
    data=np.genfromtxt(full_path)
    cp=data[:,0]
    Tg_data=data[:,1]
    R2,fit_Tgs,T1,inter_parm,T0 = fit_Tg_to_DiBenedetto(cp/100.,
                                                        Tg_data,
                                                        T1=None,
                                                        T0=None)
    print('T1',T1,'lambda',inter_parm)
    #plt.plot(cure_percents,fit_Tgs,label='$R^2$:{}'.format(round(R2,3)))
    #print(cure_percents_ss)
    alphas = np.linspace(0,1)
    fit_ydata = DiBenedetto(alphas,T1,T0=T0,inter_param=inter_parm)
    fit_ydata=np.asarray(fit_ydata)

    Tg_sim = T1#0.851796418313
    Tg_exp = 480
    roomT_exp = 300
    Tex_toTsim = Tg_exp/Tg_sim
    Tsimtoexp = Tg_exp/Tg_sim
    roomT_sim =  Tg_sim*roomT_exp/Tg_exp
```

```
    Tg0_exp = Tg_exp*T0/Tg_sim
    print('300 K in T*:',roomT_sim)
    #print(Tg_data)
    #print(fit_ydata*Tsimtoexp)
    plt.plot(alphas,
             fit_ydata*Tsimtoexp,
             colors[i],
             zorder=0,
             label='$R^2$:{}({})'.format(round(R2,3),labels[i]))
    plt.scatter(cp/100.,
                Tg_data*Tsimtoexp,
                marker='s',
                facecolor='w',
                linewidth=2,
                edgecolor=colors[i],
                s=60,
                #label='$E_a$:{}'.format(activation_energy),
                color=colors[i],
               zorder=1)
#plt.scatter(1.00,Tg_exp,marker='*',color='r',s=200,label='Ex. Tg (100 %)')
plt.axhline(y=300,linewidth=1.1,linestyle='--')

exp1_data = np.genfromtxt('Min1993.txt',delimiter=',')
#print(exp1_data)
plt.scatter(exp1_data[:,0],
                exp1_data[:,1],
                marker='d',
                facecolor='w',
                linewidth=2,
                edgecolor='k',
                s=60,
                #label='$E_a$:{}'.format(activation_energy),
                color='k',
               zorder=1,
               label='DGEBA/DDS $T_g$ (Ref. 69)')
if False:
    exp1_data = np.genfromtxt('Jenninger2000.txt',delimiter=',')
    #print(exp1_data)
    plt.scatter(exp1_data[:,0],
                    exp1_data[:,1],
                    marker='d',
                    facecolor='w',
                    linewidth=2,
                    edgecolor='c',
                    s=60,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='c',
                   zorder=1,
                   label='DGEBA/DDS/PES $T_g$ (Ref. 79)')

if False:
    exp1_data = np.genfromtxt('Schawe2017.txt',delimiter=',')
    print(exp1_data)
    plt.scatter(exp1_data[:,0],
                    exp1_data[:,1],
                    marker='d',
                    facecolor='w',
                    linewidth=2,
                    edgecolor='y',
                    s=60,
                    #label='$E_a$:{}'.format(activation_energy),
                    color='y',
                   zorder=1,
                   label='DGEBA/DDM $T_g$ (Ref. 89)')

plt.ylim(248,500)
plt.legend(fontsize=15)
plt.xlabel('Cure Fraction ($\\alpha$)')
```

```
        plt.ylabel('Tg ($K$)')
        savefig(plt,'piecewise_regression_custom_range_bond_angle','dibeneditto_angle_effect_SI_
        compare_exp.pdf')
```

```
T1 1.31476583141 lambda 0.5
300 K in T*: 0.821728644629
T1 1.38657003106 lambda 0.5
300 K in T*: 0.866606269412
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [2]:
```
from common import *
```

In [3]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LJ_System_Size/'
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools



names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [28]:
```
import os
#import structure_factor as sf
import math
from scipy import interpolate
import gsd
import gsd.fl
import gsd.hoomd

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111)
left, bottom, width, height = [0.95, 0.3, 0.5, 0.5]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
#ax3 = ax1.twinx()
typeId=2
n_views=40
grid_size=512
gammas = [4.5]#,18,36,72]#[4.5,9,18,36,45,52,72]
colors = itertools.cycle(cm.rainbow(np.linspace(0, 1, len(gammas))))
for gamma in gammas:
    df_filtered = df[(df.n_particles==1000000)&
                     (df.t_Final==1e7)&
                       (df.trial==0)]
                     # (df.t_Final==6e6)]


    #print(df_filtered)
    #grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
    times_for_all_trials=[]
    qs_for_all_trials=[]
    Is_for_all_trials=[]

    qms_all=[] #qmax
    Is_all=[]
    Qs_all=[]
    times_all=[]
    cure_all=[]
```

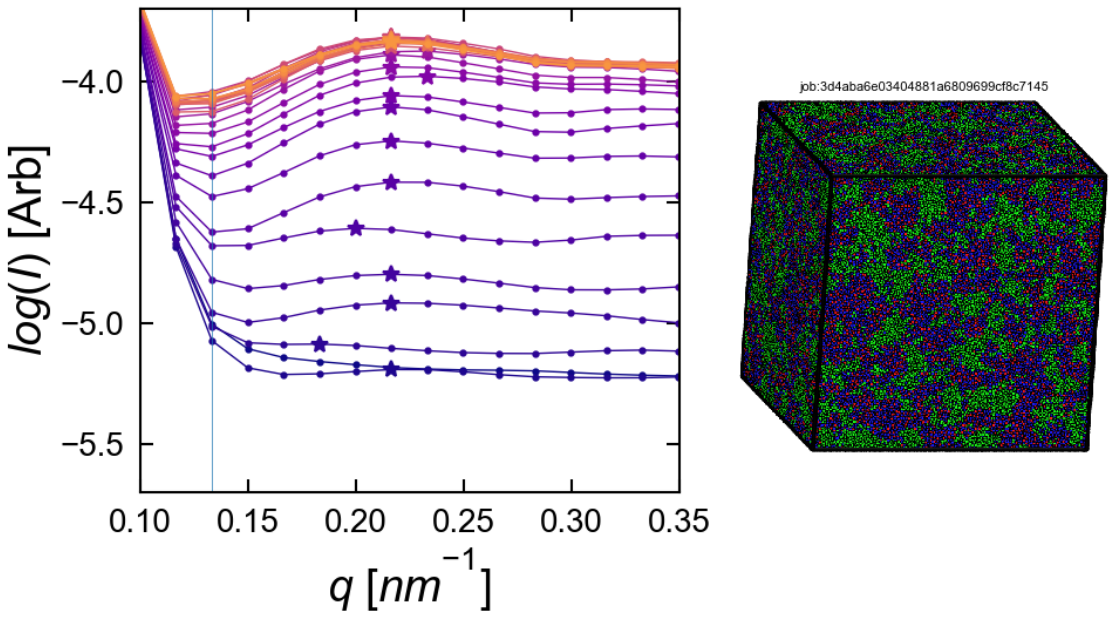
```
    color=next(colors)
    for signac_id in df_filtered['signac_id']:
        job = project.open_job(id=signac_id)
        #render_frame(job,filename='snap_1.png')
        if job.isfile('final_snapshot.png'):
            im = plt.imread(job.fn('final_snapshot.png'))
            ax2.set_title('job:{}'.format(job),fontsize=8)
            ax2.imshow(im,zorder=1)
        #print(job.workspace())
        if 'Lx' in job.document:
            half_box_length = job.document['Lx']/2
        else:
            print('Lx not found in',job)

        q_half_length = 2*math.pi/(half_box_length/1.06)
        diffract_dir_pattern
='diffract_type_{}_n_views_{}_grid_size_{}_frame'.format(typeId,
n_views,
grid_size)
        directories = os.listdir(job.workspace())
        directories = [d for d in os.listdir(job.workspace()) if
d.startswith(diffract_dir_pattern)]
        directories.sort(key = lambda x: int(x.split('_')[-1]))
        #print(len(directories))
        #print(directories)
        num_frames = len(directories)

        qs_for_all_times=[]
        Is_for_all_times=[]
        times_for_all_times=[]
        qs_list = []
        times_list = []
        Is_list = []
        Qs_list=[]
        for i,diffract_dir in enumerate(directories):
            print("Progress {:2.1%}".format(i / num_frames), end="\r")
            if diffract_dir.startswith(diffract_dir_pattern):
                frame = int(diffract_dir.split('_')[-1])
                if frame%1==0 and frame >0e6/job.sp.dcd_write:#==119 or frame==123:#%100
== 0:#num_frames/30:
                    if job.isfile('{}/asq.txt'.format(diffract_dir)):
                        #print(job.fn('{}/asq.txt'.format(diffract_dir)))
                        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                        time = round(frame*job.sp.dcd_write)
                        legend = '{} $\\Delta t(\Gamma:{})$'.format(time,job.sp.gamma)
                        qs = data[:,0]
                        Is = data[:,1]
                        #print(qs.shape)
                        qs_for_all_times.append(qs)
                        Is_for_all_times.append(Is)
                        times_for_all_times.append(time)

                        dq=qs[1]-qs[0]
                        Is_exp = np.exp(Is)
                        q_sq = qs**2
                        Q = np.sum(Is_exp*qs*dq)
                        Qs_list.append(Q)
                        first_peak_q,first_peak_i =
get_highest_maxima(job.document['Lx'],qs,Is)
                        #print(first_peak_q,first_peak_i)
                        if first_peak_q is None:
                            #print(q_half_length)
                            fn = interpolate.interp1d(qs,Is,kind='cubic')
                            first_peak_q=q_half_length
                            first_peak_i=fn(first_peak_q)
                        if first_peak_q >0.8:# and time > 2.0e5:
                            fn = interpolate.interp1d(qs,Is,kind='cubic')
                            first_peak_q=q_half_length
```

```
                                first_peak_i=0#fn(first_peak_q)
                            qs_list.append(first_peak_q)
                            times_list.append(time)
                            Is_list.append(first_peak_i)
                        else:
                            print(job,'did not contain diffraction data in ',diffract_dir)

                qs_for_all_trials.append(qs_for_all_times)#this is to plot the S(q)
                Is_for_all_trials.append(Is_for_all_times)
                times_for_all_trials.append(times_for_all_times)

                qms_all.append(np.asarray(qs_list)) #this is to plot q_max
                Is_all.append(np.asarray(Is_list))
                Qs_all.append(np.asarray(Qs_list))
                log_data = np.genfromtxt(job.fn('out.log'))
                times = log_data[:,0]#/(job.sp.dt*job.sp.dcd_write)
                cure = log_data[:,9]
                times_all.append(times)
                cure_all.append(cure)

            #print(qs_all)
            qs_for_all_trials=np.asarray(qs_for_all_trials)
            Is_for_all_trials=np.asarray(Is_for_all_trials)
            times_for_all_trials=np.asarray(times_for_all_trials)
            q_mean = np.mean(qs_for_all_trials,axis=0)
            I_mean = np.mean(Is_for_all_trials,axis=0)
            time_mean= np.mean(times_for_all_trials,axis=0)

            qms_all = np.asarray(qms_all)
            Is_all = np.asarray(Is_all)
            Qs_all = np.asarray(Qs_all)
            times_all = np.asarray(times_all)
            cure_all = np.asarray(cure_all)
            Qs_av = np.mean(Qs_all,axis=0)
            Qs_std = np.std(Qs_all,axis=0)
            #print(Is_all)
            qs_av = np.mean(qms_all,axis=0)
            qs_std = np.std(qms_all,axis=0)
            Is_av = np.mean(Is_all,axis=0)
            times_av = np.mean(times_all,axis=0)
            cure_av = np.mean(cure_all,axis=0)
            cure_std = np.std(cure_all,axis=0)
            #print(len(times_list),len(qs_av))
        t_colors = colors = plt.cm.plasma(np.linspace(0,0.75,len(q_mean)))
        for i,q in enumerate(q_mean):
            I=I_mean[i]
            ax.plot(q*1.06,
                    I,
                    marker='.',
                    linewidth=1.0,
                   color=t_colors[i])
            ax.plot(qs_av[i]*1.06,
                    Is_av[i],
                    marker='*',
                    markersize=10,
                   color=t_colors[i])

        ax.axvline(x=q_half_length,linewidth=0.5)
        ax.set_xlim(0.10,.35)
        ax.set_ylim(-5.7,-3.7)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel(r"$log(I)$ [Arb]")
        #plt.xlim(0.1,1.25)
        #ax.ylim(-4.3,-3.6)
```

```
Progress 96.7%
```

```
Out[28]: <matplotlib.text.Text at 0x1282a8278>
```

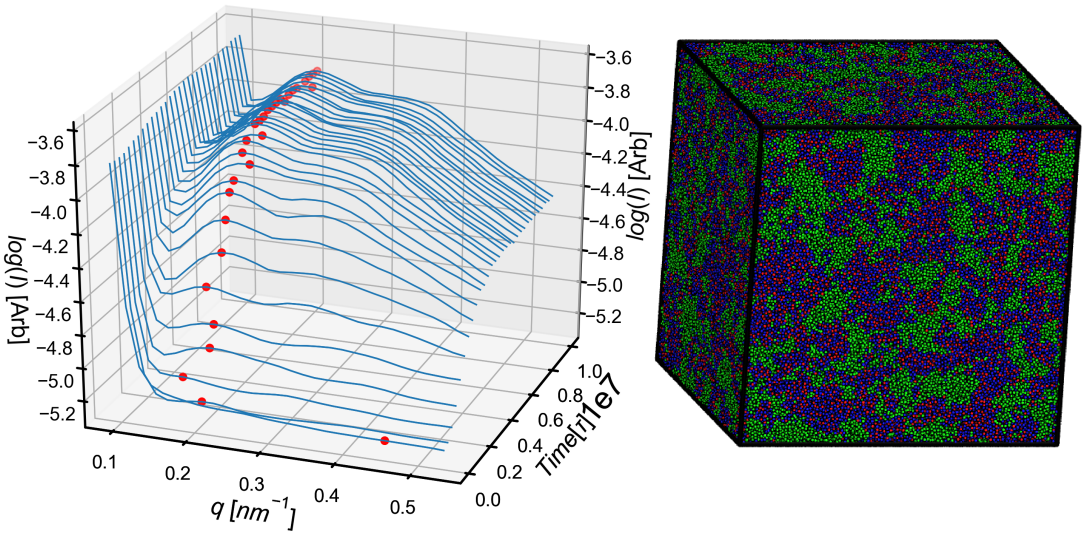
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [29]:
```
fig = plt.figure(dpi=300)
ax = fig.add_subplot(111, projection='3d')
#ax.set_title('z-axis left side')
ax = fig.add_axes(MyAxes3D(ax, 'l'))
left, bottom, width, height = [0.88, 0.2, 0.6, 0.6]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
qlim_max=35
qlim_min=6
Ilim=20
q = q_mean[0][qlim_min:qlim_max]
I=I_mean[:,qlim_min:qlim_max]
#print(q)
#print(I)
#X,Y = np.meshgrid(q_mean[0],time_mean)
X,Y = np.meshgrid(q,time_mean)

Z=I
#matplotlib.rcParams['xtick.labelsize'] = 15
#matplotlib.rcParams['ytick.labelsize'] = 15
#matplotlib.rcParams['xtick.major.pad'] = 2
#matplotlib.rcParams['ytick.major.pad'] = 2
#matplotlib.rcParams['ztick.major.pad'] = 1
#surf = ax.plot_surface(X, Y, Z, cmap=cm.plasma,)#,rstride=1, cstride=1,linewidth=1,
antialiased=True)
surf = ax.plot_wireframe(X, Y, Z, linewidth=1.0,zorder=0.2,rstride=1, cstride=0,
antialiased=True)

view_1 = (0, 180)#back
view_2 = (25, -70)#angled
view_3 = (25, 0)#front from top
```

```
        view_4 = (-5, 90)#right
        view_5 = (10, -90)#angled
        init_view = view_2
        ax.view_init(*init_view)
        ax.set_xlabel(r"$q$ [$nm^{-1}$]", fontsize=15,rotation=150)
        ax.set_ylabel(r"$Time[\tau]$",fontsize=15, rotation=150)
        ax.set_zlabel(r"$log(I)$ [Arb]", fontsize=15,rotation=90)
        ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0),
        useMathText=True,rotation=0,labelsize=1)
        ax.tick_params(axis = 'both',labelsize=12,pad=3)
        #print(q_half_length,Is_av[:101])
        ax.scatter3D(qs_av, time_mean,
        Is_av,color='r',zorder=0.5,marker='o',antialiased=True,s=20)
        if job.isfile('final_snapshot.png'):
            im = plt.imread(job.fn('final_snapshot.png'))
            #ax2.set_title('job:{}'.format(job),fontsize=8)
            ax2.imshow(im,zorder=1)
        #plt.savefig('3D_q_plot_2e6_particles.png',transparent=True)
        savefig(plt,
                '3D_plot_for_q_LJ',
                 'morphology_evolution_1e6.pdf')
        plt.show()
```

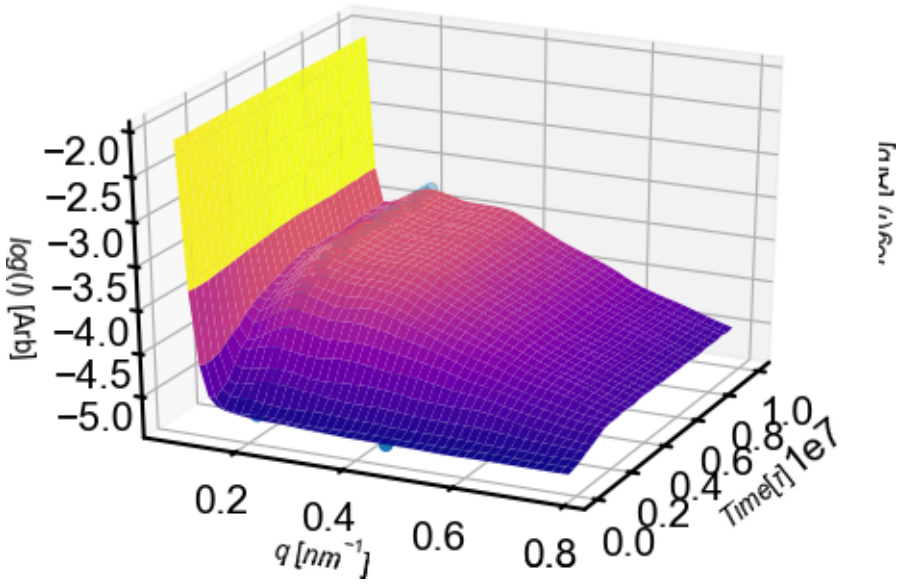
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [30]:
```
        fig = plt.figure(figsize=(15, 4))
        ax = fig.add_subplot(131, projection='3d')
        #ax.set_title('z-axis left side')
        ax = fig.add_axes(MyAxes3D(ax, 'l'))

        q = q_mean[0][5:51]
        I=I_mean[:,5:51]
        #print(I)
        #X,Y = np.meshgrid(q_mean[0],time_mean)
```

```
X,Y = np.meshgrid(q,time_mean)
#print(X.shape)
#Z=I_mean
Z=I
matplotlib.rcParams['xtick.labelsize'] = 15
matplotlib.rcParams['ytick.labelsize'] = 15
matplotlib.rcParams['xtick.major.pad'] = 2
matplotlib.rcParams['ytick.major.pad'] = 2
#matplotlib.rcParams['ztick.major.pad'] = 1
surf = ax.plot_surface(X, Y, Z, cmap=cm.plasma,)#,rstride=1, cstride=1,linewidth=1,
antialiased=True)
ax.scatter3D(qs_av, time_mean, Is_av,zorder=1)
view_1 = (25, -135)
view_2 = (25, -65)
init_view = view_2
ax.view_init(*init_view)
ax.set_xlabel(r"$q$ [$nm^{-1}$]", fontsize=10, rotation=150)
ax.set_ylabel(r"$Time[\tau]$", fontsize=10, rotation=150)
ax.set_zlabel(r"$log(I)$ [Arb]", fontsize=10, rotation=60)
ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
ax.xaxis._axinfo['label']['space_factor'] = 1.0
ax.yaxis._axinfo['label']['space_factor'] = 1.0
ax.zaxis._axinfo['label']['space_factor'] = 1.0
ax.scatter3D
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [6]: import os
        #import structure_factor as sf
        import math
        from scipy import interpolate
```

```
import gsd
import gsd.fl
import gsd.hoomd

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111)
left, bottom, width, height = [0.95, 0.3, 0.5, 0.5]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
#ax3 = ax1.twinx()
typeId=2
n_views=40
grid_size=512
gammas = [4.5]#,18,36,72]#[4.5,9,18,36,45,52,72]
colors = itertools.cycle(cm.rainbow(np.linspace(0, 1, len(gammas))))
for gamma in gammas:
    df_filtered = df[(df.n_particles==80000)&
                   (df.t_Final==1e7)&
                     (df.trial==0)]
                   # (df.t_Final==6e6)]


    #print(df_filtered)
    #grpedByGamma = df_filtered.groupby('profile')#.apply(lambda x: x.sort_values('T'))
    times_for_all_trials=[]
    qs_for_all_trials=[]
    Is_for_all_trials=[]

    qms_all=[] #qmax
    Is_all=[]
    Qs_all=[]
    times_all=[]
    cure_all=[]
    color=next(colors)
    for signac_id in df_filtered['signac_id']:
        job = project.open_job(id=signac_id)
        #render_frame(job,filename='snap_1.png')
        if job.isfile('final_snapshot.png'):
            im = plt.imread(job.fn('final_snapshot.png'))
            ax2.set_title('job:{}'.format(job),fontsize=8)
            ax2.imshow(im,zorder=1)
        #print(job.workspace())
        if 'Lx' in job.document:
            half_box_length = job.document['Lx']/2
        else:
            print('Lx not found in',job)

        q_half_length = 2*math.pi/(half_box_length/1.06)
        diffract_dir_pattern
='diffract_type_{}_n_views_{}_grid_size_{}_frame'.format(typeId,
n_views,
grid_size)
        directories = os.listdir(job.workspace())
        directories = [d for d in os.listdir(job.workspace()) if
d.startswith(diffract_dir_pattern)]
        directories.sort(key = lambda x: int(x.split('_')[-1]))
        #print(len(directories))
        #print(directories)
        num_frames = len(directories)

        qs_for_all_times=[]
        Is_for_all_times=[]
        times_for_all_times=[]
        qs_list = []
        times_list = []
        Is_list = []
```

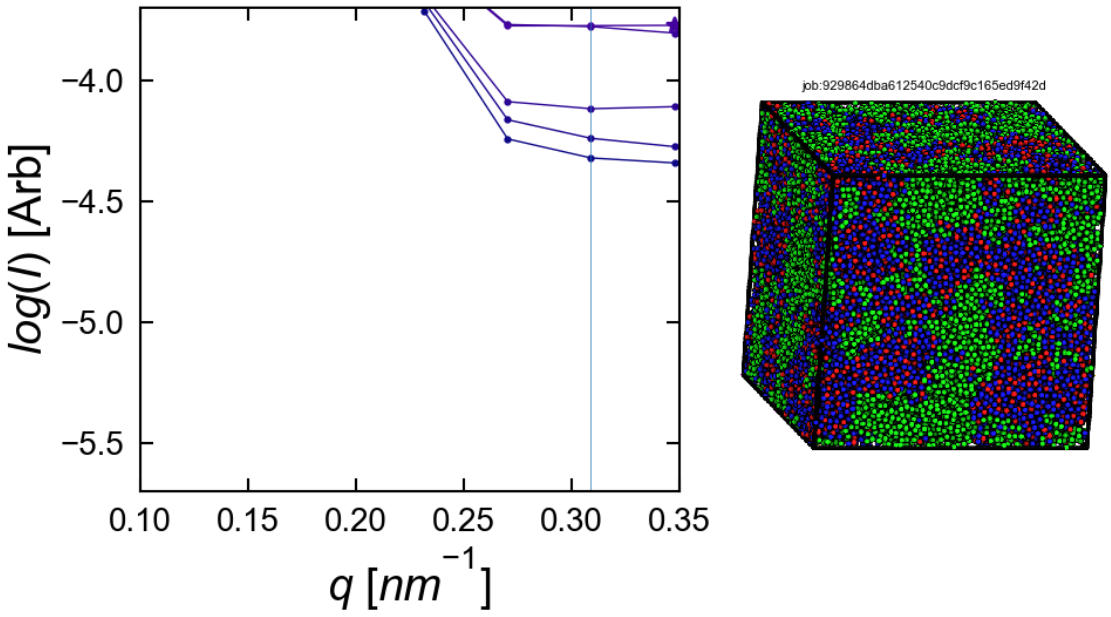
```
        Qs_list=[]
        for i,diffract_dir in enumerate(directories):
            print("Progress {:2.1%}".format(i / num_frames), end="\r")
            if diffract_dir.startswith(diffract_dir_pattern):
                frame = int(diffract_dir.split('_')[-1])
                if frame%1==0 and frame >0e6/job.sp.dcd_write:#==119 or frame==123:#%100
== 0:#num_frames/30:
                    if job.isfile('{}/asq.txt'.format(diffract_dir)):
                        #print(job.fn('{}/asq.txt'.format(diffract_dir)))
                        data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                        time = round(frame*job.sp.dcd_write)
                        legend = '{} $\\Delta t(\Gamma:{})$'.format(time,job.sp.gamma)
                        qs = data[:,0]
                        Is = data[:,1]
                        #print(qs.shape)
                        qs_for_all_times.append(qs)
                        Is_for_all_times.append(Is)
                        times_for_all_times.append(time)

                        dq=qs[1]-qs[0]
                        Is_exp = np.exp(Is)
                        q_sq = qs**2
                        Q = np.sum(Is_exp*qs*dq)
                        Qs_list.append(Q)
                        first_peak_q,first_peak_i =
get_highest_maxima(job.document['Lx'],qs,Is)
                        #print(first_peak_q,first_peak_i)
                        if first_peak_q is None:
                            #print(q_half_length)
                            fn = interpolate.interp1d(qs,Is,kind='cubic')
                            first_peak_q=q_half_length
                            first_peak_i=fn(first_peak_q)
                        if first_peak_q >0.8:# and time > 2.0e5:
                            fn = interpolate.interp1d(qs,Is,kind='cubic')
                            first_peak_q=q_half_length
                            first_peak_i=0#fn(first_peak_q)
                        qs_list.append(first_peak_q)
                        times_list.append(time)
                        Is_list.append(first_peak_i)
                    else:
                        print(job,'did not contain diffraction data in ',diffract_dir)

        qs_for_all_trials.append(qs_for_all_times)#this is to plot the S(q)
        Is_for_all_trials.append(Is_for_all_times)
        times_for_all_trials.append(times_for_all_times)

        qms_all.append(np.asarray(qs_list)) #this is to plot q_max
        Is_all.append(np.asarray(Is_list))
        Qs_all.append(np.asarray(Qs_list))
        log_data = np.genfromtxt(job.fn('out.log'))
        times = log_data[:,0]#/(job.sp.dt*job.sp.dcd_write)
        cure = log_data[:,9]
        times_all.append(times)
        cure_all.append(cure)

    #print(qs_all)
    qs_for_all_trials=np.asarray(qs_for_all_trials)
    Is_for_all_trials=np.asarray(Is_for_all_trials)
    times_for_all_trials=np.asarray(times_for_all_trials)
    q_mean = np.mean(qs_for_all_trials,axis=0)
    I_mean = np.mean(Is_for_all_trials,axis=0)
    time_mean= np.mean(times_for_all_trials,axis=0)

    qms_all = np.asarray(qms_all)
    Is_all = np.asarray(Is_all)
    Qs_all = np.asarray(Qs_all)
    times_all = np.asarray(times_all)
    cure_all = np.asarray(cure_all)
```

```
        Qs_av = np.mean(Qs_all,axis=0)
        Qs_std = np.std(Qs_all,axis=0)
        #print(Is_all)
        qs_av = np.mean(qms_all,axis=0)
        qs_std = np.std(qms_all,axis=0)
        Is_av = np.mean(Is_all,axis=0)
        times_av = np.mean(times_all,axis=0)
        cure_av = np.mean(cure_all,axis=0)
        cure_std = np.std(cure_all,axis=0)
        #print(len(times_list),len(qs_av))
    t_colors = colors = plt.cm.plasma(np.linspace(0,0.75,len(q_mean)))
    for i,q in enumerate(q_mean):
        I=I_mean[i]
        ax.plot(q*1.06,
                I,
                marker='.',
                linewidth=1.0,
               color=t_colors[i])
        ax.plot(qs_av[i]*1.06,
                Is_av[i],
                marker='*',
                markersize=10,
               color=t_colors[i])

    ax.axvline(x=q_half_length,linewidth=0.5)
    ax.set_xlim(0.10,.35)
    ax.set_ylim(-5.7,-3.7)
    ax.set_xlabel(r"$q$ [$nm^{-1}$]")
    ax.set_ylabel(r"$log(I)$ [Arb]")
    #plt.xlim(0.1,1.25)
    #ax.ylim(-4.3,-3.6)
```

```
Progress 96.7%
```

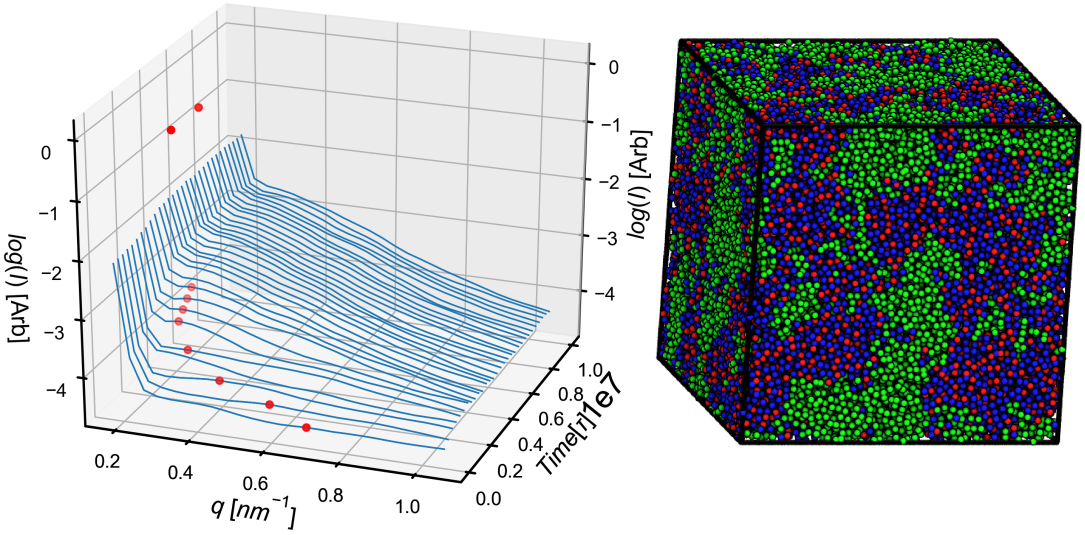
Out[6]: <matplotlib.text.Text at 0x11a2c15f8>

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [28]:
```
fig = plt.figure(dpi=300)
ax = fig.add_subplot(111, projection='3d')
#ax.set_title('z-axis left side')
ax = fig.add_axes(MyAxes3D(ax, 'l'))
left, bottom, width, height = [0.88, 0.2, 0.6, 0.6]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
qlim_max=30
qlim_min=5
Ilim=20
q = q_mean[0][qlim_min:qlim_max]
I=I_mean[:,qlim_min:qlim_max]
#print(q)
#print(I)
#X,Y = np.meshgrid(q_mean[0],time_mean)
X,Y = np.meshgrid(q,time_mean)

Z=I
#matplotlib.rcParams['xtick.labelsize'] = 15
#matplotlib.rcParams['ytick.labelsize'] = 15
#matplotlib.rcParams['xtick.major.pad'] = 2
#matplotlib.rcParams['ytick.major.pad'] = 2
#matplotlib.rcParams['ztick.major.pad'] = 1
#surf = ax.plot_surface(X, Y, Z, cmap=cm.plasma,)#,rstride=1, cstride=1,linewidth=1,
antialiased=True)
surf = ax.plot_wireframe(X, Y, Z, linewidth=1.0,zorder=0.2,rstride=1, cstride=0,
antialiased=True)

view_1 = (0, 180)#back
view_2 = (25, -70)#angled
view_3 = (25, 0)#front from top
view_4 = (-5, 90)#right
view_5 = (10, -90)#angled
init_view = view_2
ax.view_init(*init_view)
ax.set_xlabel(r"$q$ [$nm^{-1}$]", fontsize=15,rotation=150)
ax.set_ylabel(r"$Time[\tau]$",fontsize=15, rotation=150)
ax.set_zlabel(r"$log(I)$ [Arb]", fontsize=15,rotation=90)
ax.ticklabel_format(style='sci', axis='y', scilimits=(0,0),
useMathText=True,rotation=0,labelsize=1)
ax.tick_params(axis = 'both',labelsize=12,pad=3)
#print(q_half_length,Is_av[:101])
last_index = -20
ax.scatter3D(qs_av[:last_index], time_mean[:last_index],
Is_av[:last_index],color='r',zorder=0.5,marker='o',antialiased=True,s=20)
if job.isfile('final_snapshot.png'):
    im = plt.imread(job.fn('final_snapshot.png'))
    #ax2.set_title('job:{}'.format(job),fontsize=8)
    ax2.imshow(im,zorder=1)
#plt.savefig('3D_q_plot_2e6_particles.png',transparent=True)
savefig(plt,
       '3D_plot_for_q_LJ',
        'morphology_evolution_8e4.pdf')
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


log(I) [Arb]
0
−1
−2
−3
−4
0.2
0.4
0.6
0.8
1.0
q [nm⁻¹]
0.0
0.2
0.4
0.6
0.8
1.0
Time[τ]1e7
log(I) [Arb]

In [1]:
```
from common import *
from common import MyAxes3D
import signac
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec
import pandas as pd
import matplotlib
import numpy as np
%matplotlib inline
```

In [2]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/LJ_System_Size/'



names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
#df.head()
```

In [3]:
```
from matplotlib import cm
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import interpolate


import os
import math
import gsd
import gsd.fl
import gsd.hoomd


df_filtered = df[(df.t_Final==1e7)&
                 ((df.n_particles>=1e5)|
                  (df.n_particles==5e4)|
                  (df.n_particles==8e4))]

df_sorted=df_filtered.sort_values('n_particles')
#Ns=df_sorted.n_particles.unique()
#print(Ns)
Ns=[]
mean_qs_for_Ns = []
mean_Is_for_Ns = []
mean_times_for_Ns = []
std_qs_for_Ns = []
std_Is_for_Ns = []
q_hbls=[]
USE_INDE_FRAMES=False
for N,df_grp in df_sorted.groupby('n_particles'):
    times_for_all_trials=[]
    qs_for_all_trials=[]
    Is_for_all_trials=[]

    qms_all=[] #qmax
    Is_all=[]
    Qs_all=[]
    times_all=[]
    cure_all=[]
```

```
            mean_qs=[]
            mean_Is=[]
            std_qs=[]
            std_Is=[]
            mean_qms=[]
            mean_Is_of_qm=[]

            for signac_id in df_grp.index:
                job = project.open_job(id=signac_id)
                #print(job)
                if 'Lx' in job.document:#checking if the job completed
                    #print(job)
                    if USE_INDE_FRAMES:
                        mqfif,stdqfif,mifif,stdifif =
        get_mean_sf_from_independent_frames(job,equilibrated_percent=90)
                        #print(N,mqfif)
                        mean_qs.append(mqfif)
                        std_qs.append(stdqfif)
                        mean_Is.append(mifif)
                        std_Is.append(stdifif)
                    else:
                        diffract_dir ='diffract_type_2'
                        n_particles=job.sp.n_particles
                        if job.isfile('{}/asq.txt'.format(diffract_dir)):
                            data=np.genfromtxt(job.fn('{}/asq.txt'.format(diffract_dir)))
                            #print(np.shape(data[:,0]))
                            q=data[:,0]
                            I=data[:,1]
                            mean_qs.append(q)
                            #std_qs.append(stdqfif)
                            mean_Is.append(I)
                            #std_Is.append(stdifif)

                        else:
                            print(job)
                            print('Final frame is not diffracted')
                else:
                    print('Lx not found for',job)
            if len(mean_qs)>0:
                print('q calculated from',len(mean_qs),'trials for N=',N)
                mean_q_for_N = np.mean(mean_qs,axis=0)
                mean_qs_for_Ns.append(mean_q_for_N)
                std_q_for_N = stats.sem(mean_qs,axis=0)
                std_qs_for_Ns.append(std_q_for_N)
                mean_I_for_N = np.mean(mean_Is,axis=0)
                mean_Is_for_Ns.append(mean_I_for_N)
                std_I_for_N = stats.sem(mean_Is,axis=0)
                std_Is_for_Ns.append(std_I_for_N)
                Ns.append(N)
        #print((mean_qs_for_Ns))
        #print(std_Is_for_Ns)
        #print(Ns)
```

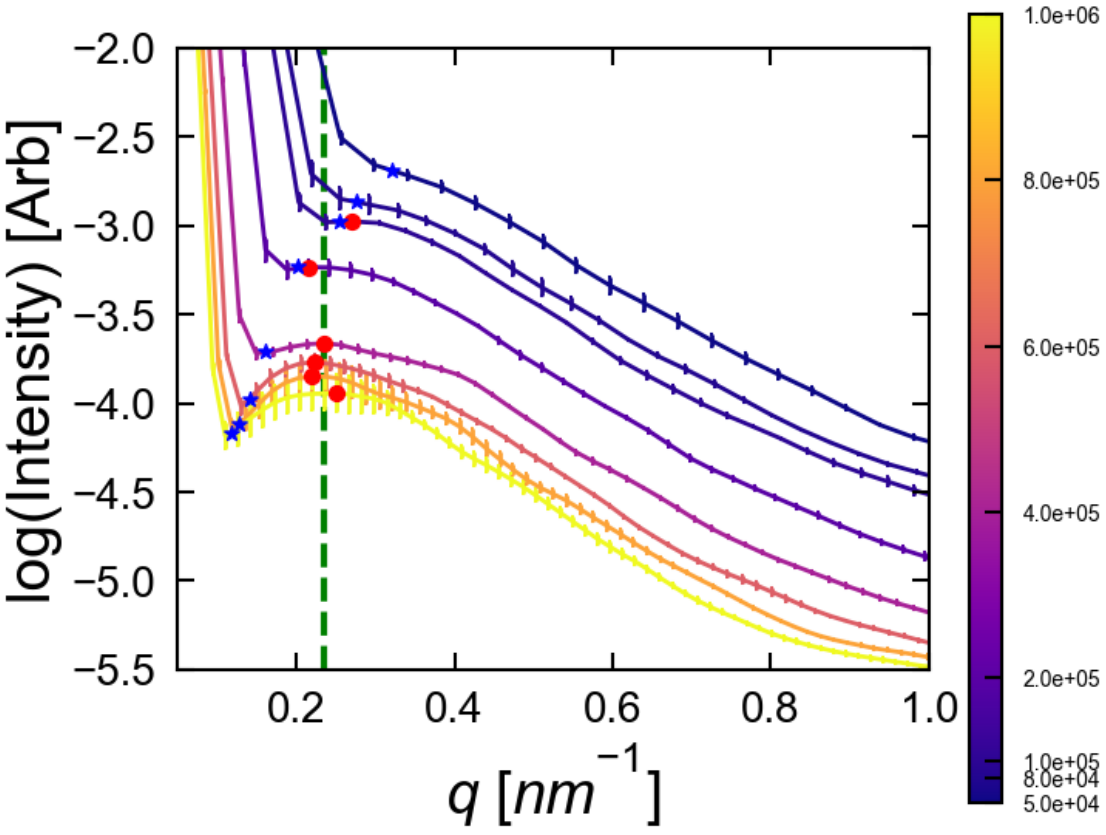
```
q calculated from 3 trials for N= 50000.0
q calculated from 3 trials for N= 80000.0
q calculated from 3 trials for N= 100000.0
q calculated from 3 trials for N= 200000.0
q calculated from 3 trials for N= 400000.0
q calculated from 3 trials for N= 600000.0
q calculated from 3 trials for N= 800000.0
q calculated from 3 trials for N= 1000000.0
Lx not found for b2aea2daba3d84311a6e664fbb4ad3f1
Lx not found for d02d14b7995d6a2866ba7a2c4c9597f6
Lx not found for b0bc66fa9ae800b457ccbc67f8debeaa
```

```
In [4]: fig = plt.figure()
```

```
ax = fig.gca()
ax1 = fig.add_axes([ 0.95, 0.10,0.03, 0.85])

ticks=Ns

cmap = mpl.cm.plasma
norm = mpl.colors.Normalize(vmin=np.min(Ns), vmax=np.max(Ns))
cb1 = mpl.colorbar.ColorbarBase(ax1, cmap=cmap,
                                norm=norm,
                                ticks=ticks,
                                format = "%.1e",
                                spacing='uniform',
                                orientation='vertical')

for l in ax1.yaxis.get_ticklabels():
    #l.set_weight("bold")
    l.set_fontsize(10)
print(Ns)
cmap_indices = []
maxN = np.max(Ns)
minN = np.min(Ns)
for N in Ns:
    cmap_i = (N-minN)/(maxN-minN)
    cmap_indices.append(cmap_i)
#cmap = [plt.cm.plasma(i) for i in np.linspace(0, 1, len(Ns))]
cmap = [plt.cm.plasma(i) for i in cmap_indices]
ax.set_color_cycle(cmap)
offsets = np.linspace(0,4,num=len(Ns))
first_peak_qs=[]
for i,N in enumerate(Ns):
    #print('N',N)
    #i=l-i_temp-1
    q=mean_qs_for_Ns[i]
    I=mean_Is_for_Ns[i]
    I_std=std_Is_for_Ns[i]
    #print('I_std',I_std)
    #print(len(I))

    offset = 0#offsets[i]
    #time=time_mean[i]
    legend='{:.1e}'.format(N)
    fn = interpolate.interp1d(q,I,kind='cubic')
    ax.errorbar(q,
                I+offset,
                I_std,
                #marker='.',
                markersize=5,
                linewidth=2,
                capsize=1,
                label=legend,
                zorder=1)
    #df_filtered= df[(df.t_Final==6e6)&
    #               (df.n_particles==N)]
    df_filt = df_filtered[df_filtered.n_particles==N]
    sids = df_filt.signac_id
    job = project.open_job(id=sids[0])
    if 'Lx' not in job.document:
        print(job,'does not contain Lx')
        continue
    first_peak_q,first_peak_i = get_highest_maxima(job.document['Lx'],q,I)
    #print(first_peak_q,first_peak_i)
    if first_peak_q is not None:
        if first_peak_q <0.8:
            first_peak_qs.append(first_peak_q)
            ax.scatter(first_peak_q,
                   first_peak_i+offset,
                   color='r',s=50,zorder=2)
    half_box_length = job.document['Lx']/2
```

```
            q_hbl = 2*math.pi/(half_box_length*1.06)
            ax.scatter(q_hbl,
                       fn(q_hbl)+offset,
                       marker='*',
                       s=50,
                       color='b',
                       zorder=2)#,s=10)

        print(first_peak_qs)
        mean_q=np.mean(first_peak_qs)
        print('mean first peak',mean_q)
        ax.axvline(x=mean_q,linestyle='--',color='g',zorder=0)#,label='$\\langle
        q_{max}\\rangle$')

        ax.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax.set_ylabel("log(Intensity) [Arb]")
        #ax.legend(fontsize=10)
        ax.set_xlim(0.05,1.0)
        ax.set_ylim(-5.5,-2.0)
        savefig(plt,
               'lj_model_min_sys_size',
               'lj_finite_size_effect.pdf')
        plt.show()
```

```
[50000.0, 80000.0, 100000.0, 200000.0, 400000.0, 600000.0, 800000.0, 1000000.0]
[0.27067864743204978, 0.21483777258951239, 0.23446066894915649, 0.22344028944357619,
0.21992640103854047, 0.25127578526617805]
mean first peak 0.235769927453


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/cbook.py:136: MatplotlibDeprecationWarning: The set_color_cycle
attribute was deprecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

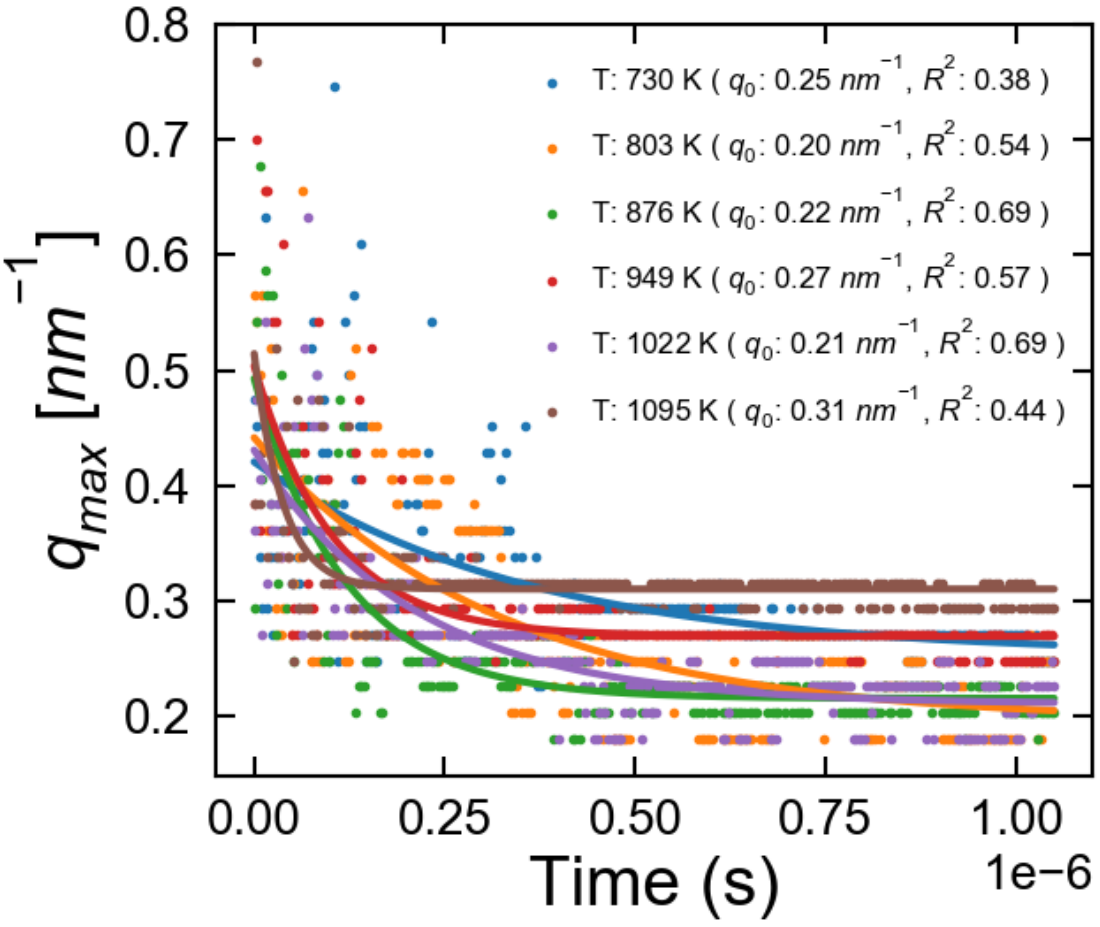

log(Intensity) [Arb]
−2.0
−2.5
−3.0
−3.5
−4.0
−4.5
−5.0
−5.5
0.2
0.4
0.6
0.8
1.0
q [nm^−1]
1.0e+06
8.0e+05
6.0e+05
4.0e+05
2.0e+05
1.0e+05
8.0e+04
5.0e+04

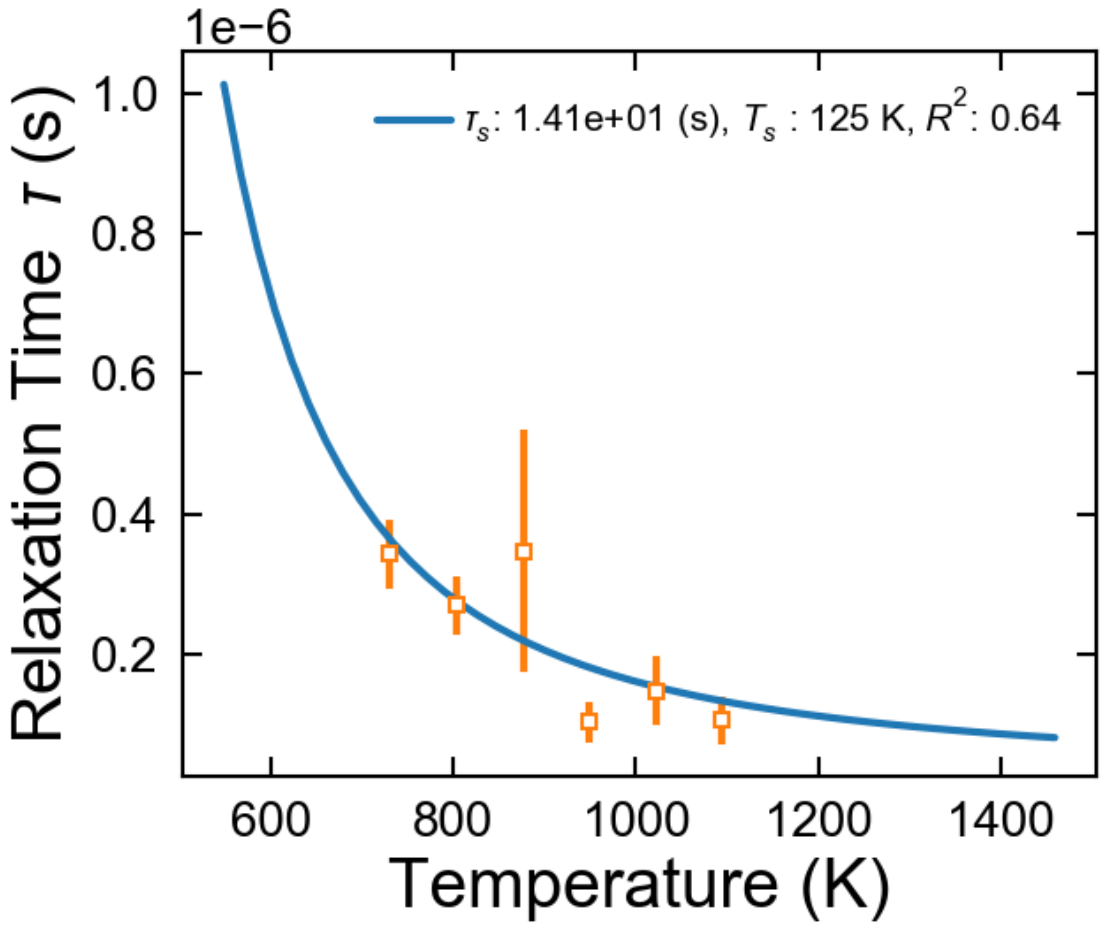
In [1]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_kestrel/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_fry/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/lj_tau_new/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
from scipy import stats
dcd_write = 1000
md_dt = 1e-2
time_conversion = 1.05e-11 #s
distance_conversion = 1.06 #nm
T_conversion = 365.01 #K
df_filtered = df[(df.n_particles==400000)&
                 (df.dcd_write==dcd_write)&
                 (df.md_dt==md_dt)&
                 (df.t_Final==1e7)&
                 #(df.trial!=3)&
                 (df.kT<=3.0)&
                 (df.kT>=2.0)]
taus=[]
taus_sem=[]
kTs=[]

for kT,df_kts in df_filtered.groupby('kT'):
    taus_for_this_kt=[]
    for trial,df_trial in df_kts.groupby('trial'):
        times,qms = get_qms(project,df_trial,trial)
        times=times*time_conversion
        qms = qms*distance_conversion
        #print('times',times)
        if len(times)>0:
            mul_fact = 1
            scales_qms = qms*mul_fact
            tau,A0,q0,r_squared = get_tau(times,scales_qms)
            #print('tau for kT:{},trial:{} = {}'.format(kT,trial,tau))
            taus_for_this_kt.append(tau)
```

```
                A0=A0/mul_fact
                q0=q0/mul_fact
                plot_tau=True
                if plot_tau:
                    if trial ==2:
                        plt.figure(0)
                        #print(qms)
                        dist_unit='$nm^{-1}$'
                        plt.scatter(times,
                                qms,
                                marker='.',
                                label='T: {:.0f} K ( $q_0$: {:.2f} {}, $R^2$: {:.2f}
        )'.format(kT*T_conversion,
            q0,
            dist_unit,
            r_squared))
                        plt.plot(times,
                                f_t(times,tau,A0,q0))
                        plt.xlim(-5e-8,1.1e-6)

                        plt.xlabel("Time (s)")
                        plt.ylabel(r"$q_{max}$ [$nm^{-1}$]")
                        plt.ticklabel_format(style='sci',
                                             axis='x',
                                             scilimits=(0,0),
                                             useMathText=True)
                        plt.legend(fontsize=12,loc='best')

                    if len(taus_for_this_kt)>0:
                        taus.append(np.mean(taus_for_this_kt))
                        taus_sem.append(stats.sem(taus_for_this_kt))
                        kTs.append(kT)
        savefig(plt,
               'relaxation_time_LJ_av_tau',
               'trial2_tau_fit.pdf')
        if len(taus)>1:
            taus=np.asarray(taus)
            taus_sem=np.asarray(taus_sem)
            kTs=np.asarray(kTs)*T_conversion
            plt.figure(1)
            tau_s,T_s,r_squared = fit_wlf(kTs,taus)
            fitkTs = np.linspace(1.5,4.0)*T_conversion
            fit_taus = wlf(fitkTs,tau_s,T_s)
            plt.plot(fitkTs,
                    fit_taus,
                    label='$\\tau_s$: {:.2e} (s), $T_s$ : {:.0f} K, $R^2$: {:.2f}'.format(tau_s,
                                                                                          T_s,
                                                                                          r_squared))
            plt.errorbar(kTs,
                         taus,
                         taus_sem,
                          markerfacecolor='w',
                         fmt='s')
            plt.xlabel("Temperature (K)")
            plt.ylabel('Relaxation Time  $\\tau$ (s)')
            plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
            plt.legend(fontsize=15,loc='best')
            savefig(plt,
               'relaxation_time_LJ_av_tau',
               'wlf_fit.pdf')


        plt.show()
```

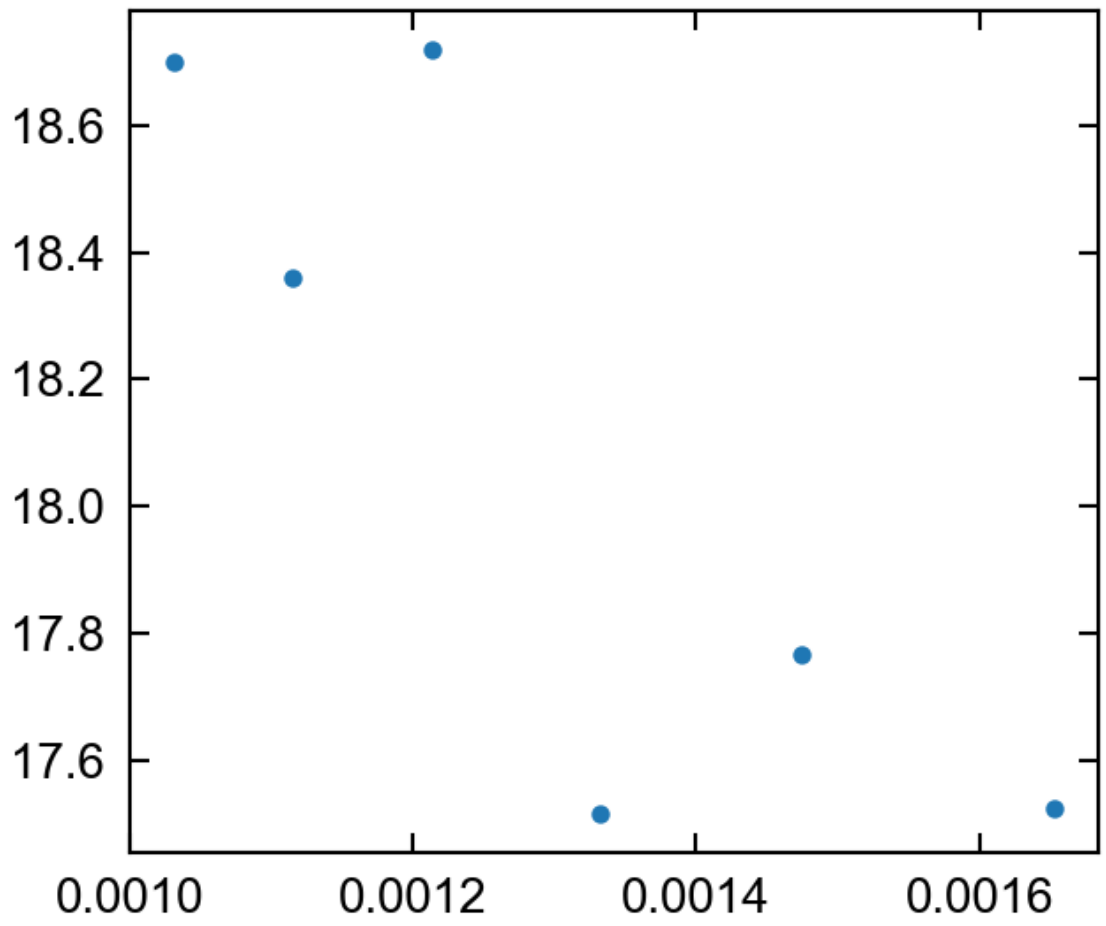
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


0.8
0.7
0.6
0.5
0.4
0.3
0.2
q_{max} [nm^{-1}]
T: 730 K (q_0: 0.25 nm^{-1}, R^2: 0.38)
T: 803 K (q_0: 0.20 nm^{-1}, R^2: 0.54)
T: 876 K (q_0: 0.22 nm^{-1}, R^2: 0.69)
T: 949 K (q_0: 0.27 nm^{-1}, R^2: 0.57)
T: 1022 K (q_0: 0.21 nm^{-1}, R^2: 0.69)
T: 1095 K (q_0: 0.31 nm^{-1}, R^2: 0.44)
0.00
0.25
0.50
0.75
1.00
Time (s)
1e−6




```
In [3]: plt.plot(1/(kTs-T_s),
                 np.log(1/(taus/tau_s)),
                marker='o',
                linewidth=0.0)
        plt.show()
        np.log(1/(taus/tau_s))
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
Out[3]: array([ 17.52521223,  17.76624017,  17.51520357,  18.72006425,
                18.3611811 ,  18.69871572])
```

```
In [4]: def plot_wlf(ax,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s, label,color,normalize=False):
            fit_taus = wlf(exp_fitKts,tau_s=exp_tau_s,Ts=exp_Ts)
            if normalize:
                fit_taus=fit_taus/exp_tau_s
                normalized_exp_tau_s = exp_tau_s/exp_tau_s
            else:
                normalized_exp_tau_s = exp_tau_s
            ln_at = np.log(fit_taus/exp_tau_s)
            ax.plot(exp_fitKts,#-exp_Ts,
                    fit_taus,
                     zorder=0,
                    color=color,
                    linestyle='--',
                    #marker='.',
                    label='WLF ({})'.format(label))
            if False:
                ax.scatter(exp_Ts,
                           normalized_exp_tau_s,
                           marker='s',
                           zorder=1,
                           #color='r',
                          label='$\\tau_s(T_s)$ ({})'.format(label))
```

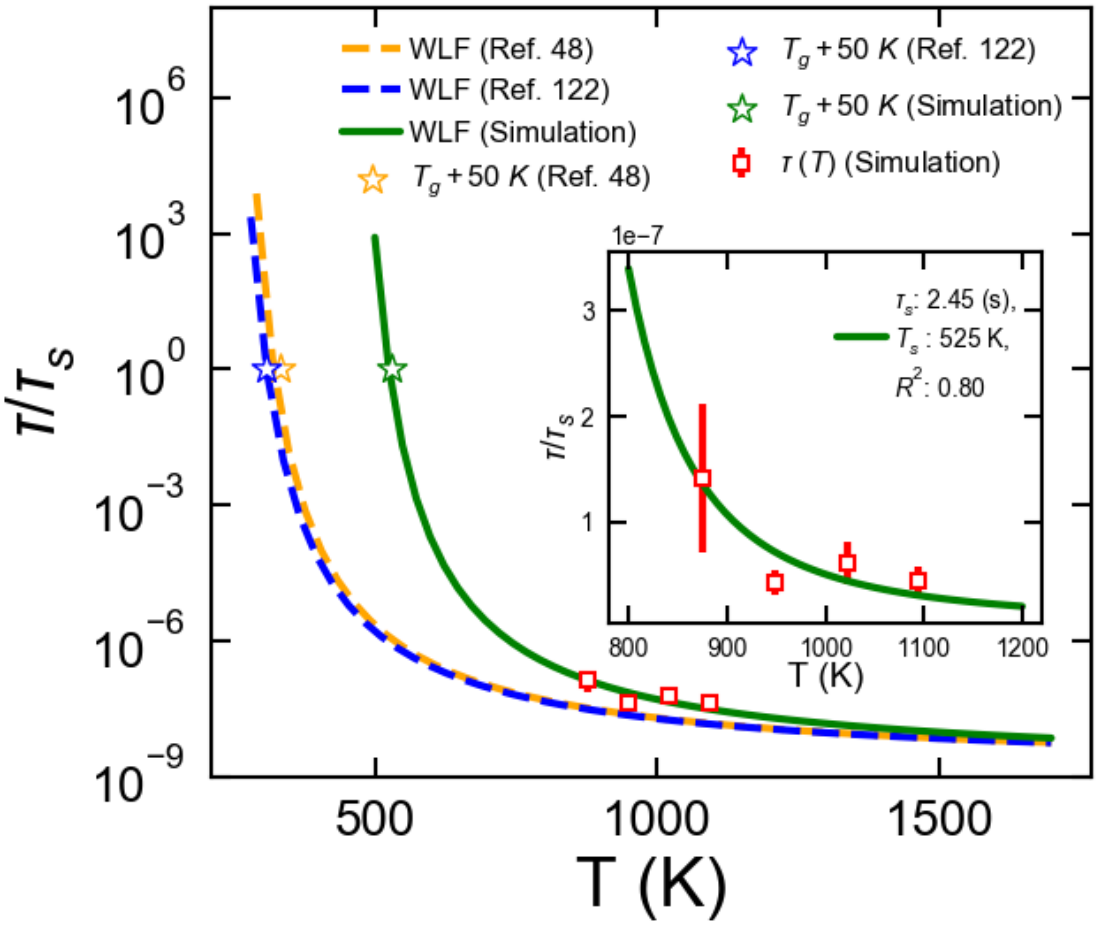
```
    ax.scatter(exp_Tg+50,
                normalized_exp_tau_s,
                marker='*',
               s=150,
                color=color,
               facecolor='w',
                #color='g',
                zorder=1,
               label='$T_g+50\ K$ ({})'.format(label))


    ax.legend(fontsize=10)
    ax.set_xlabel('$T\ (K)$')
    ax.set_ylabel('Relaxation Time (s)')
    ax.set_yscale('log')
```

In [15]:
```
#fit_taus = wlf(fitkTs,2au_s,T_s)
normalize=True
Tg=480
Ts=Tg+50
fitkTs = np.linspace(Tg-50,1700)
#print(fitkTs)
exp_Tg = 282
exp_Ts = 321
exp_fitKts = np.linspace(290,1700)
exp_tau_s = 1.74e6
fig = plt.figure()
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.55, 0.35, 0.35, 0.35]#max t2
ax2 = fig.add_axes([left, bottom, width, height])

plot_wlf(ax1,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s,'Ref.
48','orange',normalize=normalize)#Jyotishkumar et. al, 12.9%
ABS/DGEBA/DDS',normalize=normalize)

exp_Tg = 308-50
exp_Ts = 308
exp_fitKts = np.linspace(280,1700)
exp_tau_s = 1.63e6
plot_wlf(ax1,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s,'Ref. 122','b',normalize=normalize)#'Yu,
et. al, 20% PES/DGEBA/MTHPA',normalize=normalize)

#print(taus)
tau_s,T_s,r_squared = fit_wlf(kTs[2:],taus[2:],Ts_fixed=None)#480+50)
#print('taus',tau_s,T_s)
sim_Tg = 480
sim_Ts = T_s
sim_fitKts = np.linspace(500,1700)
sim_tau_s = tau_s
#fit_taus = wlf(fitkTs,tau_s,T_s)
fitkTs = np.linspace(75,1200)
sim_fit_taus = wlf(sim_fitKts,sim_tau_s,sim_Ts)
if normalize:
    sim_fit_taus=sim_fit_taus/sim_tau_s
    normalized_taus = taus/sim_tau_s
    normalized_tau_sems = taus_sem/sim_tau_s
    normalized_sim_tau_s = sim_tau_s/sim_tau_s
else:
    sim_fit_taus=sim_fit_taus
    normalized_taus = taus
    normalized_tau_sems = taus_sem
    normalized_sim_tau_s = sim_tau_s
ln_at = np.log(fit_taus/tau_s)
ax1.plot(sim_fitKts,
        sim_fit_taus,
         color='g',
         zorder=0,
        label='WLF (Simulation)')
```

```
ax1.errorbar(kTs[2:],
             normalized_taus[2:],
             normalized_tau_sems[2:],
             markerfacecolor='w',
             zorder=0,
             color='r',
             fmt='s',
            label='$\\tau\ (T)$ (Simulation)')
if False:
    ax1.scatter(sim_Ts,
                normalized_sim_tau_s,
                marker='s',
                facecolor='w',
                zorder=1,
                color='b',
               label='$\\tau_s\ (T-s)$ (Simulation)')
ax1.scatter(sim_Tg+50,
            normalized_sim_tau_s,
            marker='*',
            s=150,
            facecolor='w',
            color='g',
            zorder=1,
           label='$T_g+50\ K$ (Simulation)')

#ax1.set_yscale('log')
ax1.legend(fontsize=12,ncol=2)
ax1.set_xlabel('T (K)')
ax1.set_ylabel('$\\tau/\\tau_s$')
ax1.set_ylim(1e-9,1e8)
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)

sim_fitKts = np.linspace(800,1200)
sim_tau_s = tau_s
#fit_taus = wlf(fitkTs,tau_s,T_s)
fitkTs = np.linspace(600,1200)
sim_fit_taus = wlf(sim_fitKts,sim_tau_s,sim_Ts)
if normalize:
    sim_fit_taus=sim_fit_taus/sim_tau_s
    normalized_taus = taus/sim_tau_s
    normalized_tau_sems = taus_sem/sim_tau_s
    normalized_sim_tau_s = sim_tau_s/sim_tau_s
else:
    sim_fit_taus=sim_fit_taus
    normalized_taus = taus
    normalized_tau_sems = taus_sem
    normalized_sim_tau_s = sim_tau_s

ln_at = np.log(fit_taus/tau_s)
ax2.plot(sim_fitKts,
        sim_fit_taus,
         zorder=0,
         color='g',
        label='$\\tau_s$: {:.2f} (s), \n$T_s$ : {:.0f} K, \n$R^2$: {:.2f}'.format(tau_s,
                                                                                  T_s,
                                                                                  r_squared))
ax2.errorbar(kTs[2:],
             normalized_taus[2:],
             normalized_tau_sems[2:],
             markerfacecolor='w',
             zorder=0,
             color='r',
             fmt='s')
ax2.set_xlabel("T (K)",fontsize=14,labelpad=0)
ax2.set_ylabel(r"$\tau/\tau_s$",fontsize=14,labelpad=0)
ax2.legend(fontsize=10)
#ax2.set_yscale('log')
#ax2.set_ylim(0,3.4)
```

```
        #ax2.set_xlim(0.7,6)
        ax2.tick_params(axis='both', labelsize=10,length=7,pad=6)
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)
        ax2.yaxis.offsetText.set_fontsize(10)
        #plt.xlim(0,3000)
        savefig(plt,
                'relaxation_time_LJ_av_tau',
               'wlf_normalized.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

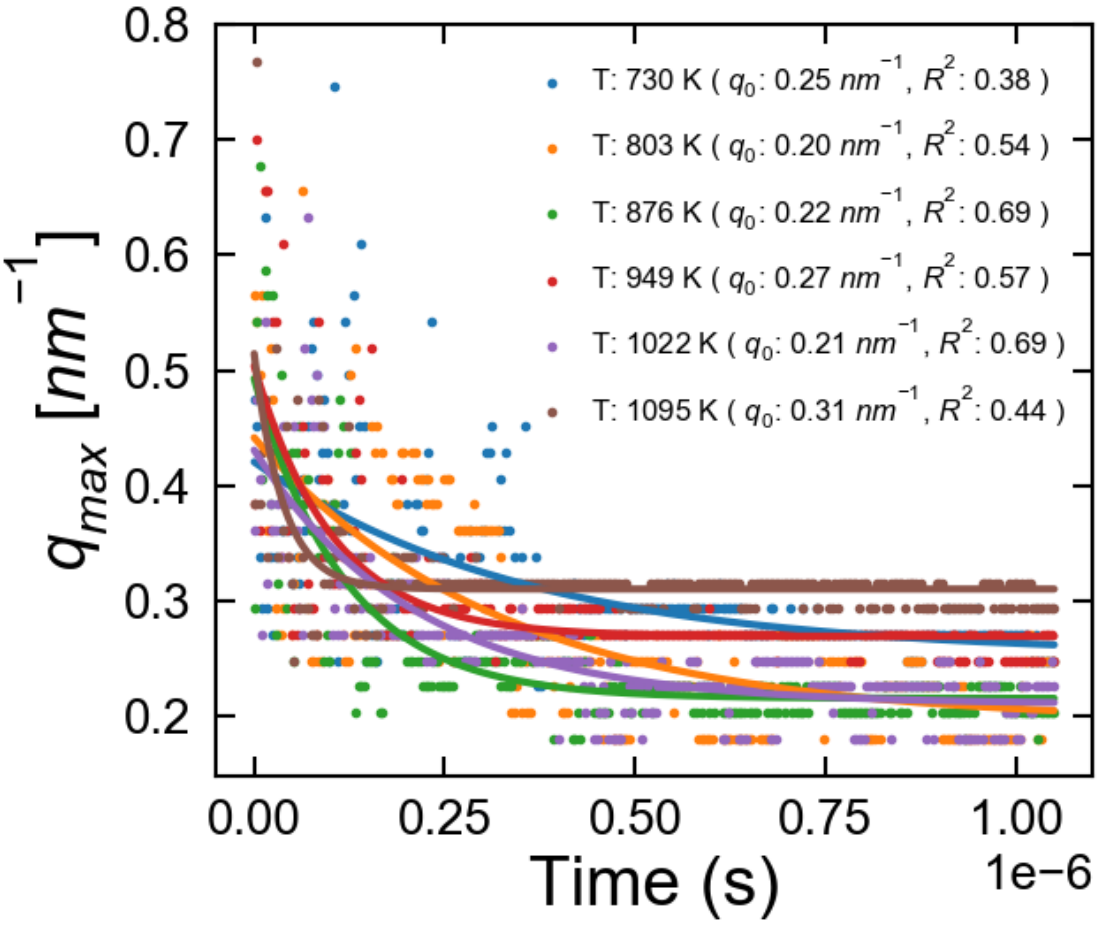


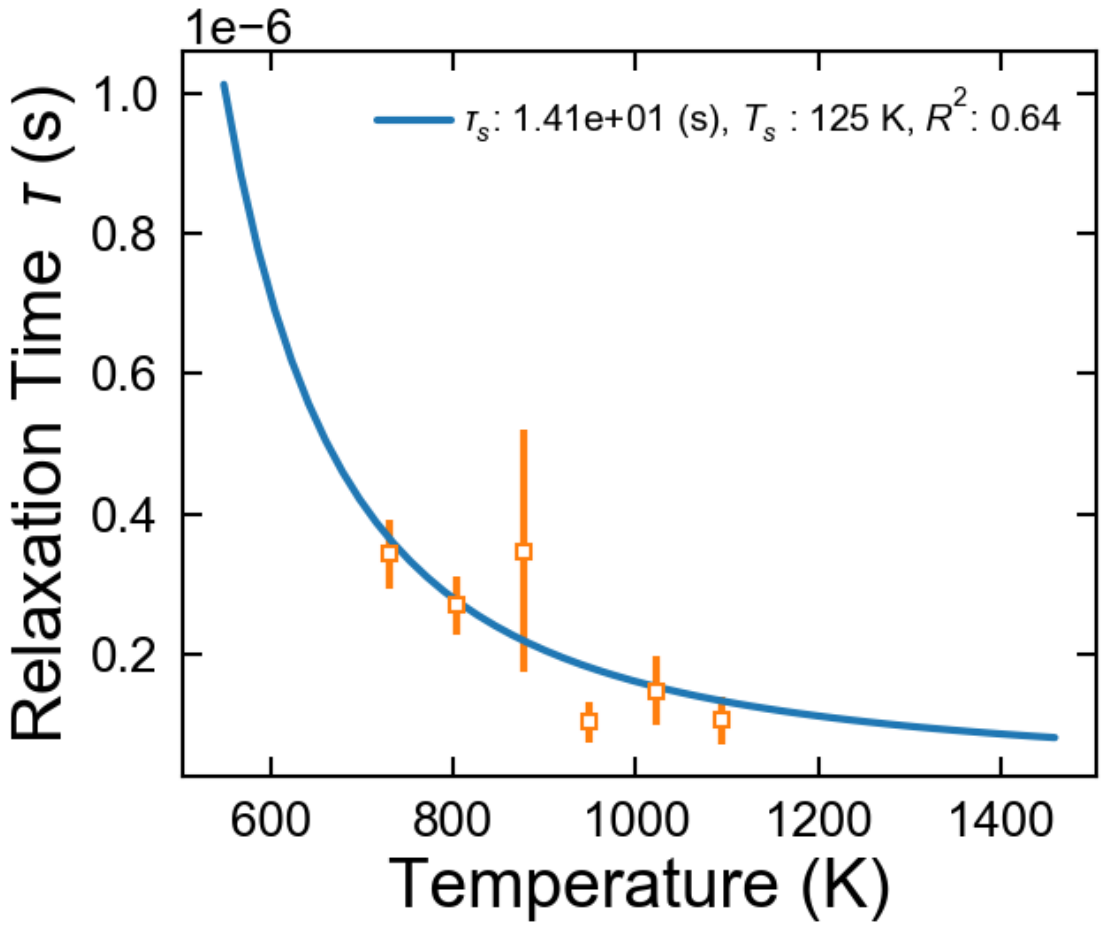
In [1]:
```
data_path='/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/Diffusivity/large_system/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_kestrel/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology_fitting_fry/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/morphology/'
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/lj_tau_new/'

import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *


names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

project = signac.get_project(data_path)
df_index = pd.DataFrame(project.index())
df_index = df_index.set_index(['_id'])
statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
df = pd.DataFrame(statepoints).T.join(df_index)
df = df.sort_values('T')
#df.head()
```

In [2]:
```
from scipy import stats
dcd_write = 1000
md_dt = 1e-2
time_conversion = 1.05e-11 #s
distance_conversion = 1.06 #nm
T_conversion = 365.01 #K
df_filtered = df[(df.n_particles==400000)&
                 (df.dcd_write==dcd_write)&
                 (df.md_dt==md_dt)&
                 (df.t_Final==1e7)&
                 #(df.trial!=3)&
                 (df.kT<=3.0)&
                 (df.kT>=2.0)]
taus=[]
taus_sem=[]
kTs=[]

for kT,df_kts in df_filtered.groupby('kT'):
    taus_for_this_kt=[]
    for trial,df_trial in df_kts.groupby('trial'):
        times,qms = get_qms(project,df_trial,trial)
        times=times*time_conversion
        qms = qms*distance_conversion
        #print('times',times)
        if len(times)>0:
            mul_fact = 1
            scales_qms = qms*mul_fact
            tau,A0,q0,r_squared = get_tau(times,scales_qms)
            #print('tau for kT:{},trial:{} = {}'.format(kT,trial,tau))
            taus_for_this_kt.append(tau)
```

```
                A0=A0/mul_fact
                q0=q0/mul_fact
                plot_tau=True
                if plot_tau:
                    if trial ==2:
                        plt.figure(0)
                        #print(qms)
                        dist_unit='$nm^{-1}$'
                        plt.scatter(times,
                                qms,
                                marker='.',
                                label='T: {:.0f} K ( $q_0$: {:.2f} {}, $R^2$: {:.2f}
        )'.format(kT*T_conversion,
            q0,
            dist_unit,
            r_squared))
                        plt.plot(times,
                                f_t(times,tau,A0,q0))
                        plt.xlim(-5e-8,1.1e-6)

                        plt.xlabel("Time (s)")
                        plt.ylabel(r"$q_{max}$ [$nm^{-1}$]")
                        plt.ticklabel_format(style='sci',
                                             axis='x',
                                             scilimits=(0,0),
                                             useMathText=True)
                        plt.legend(fontsize=12,loc='best')

            if len(taus_for_this_kt)>0:
                taus.append(np.mean(taus_for_this_kt))
                taus_sem.append(stats.sem(taus_for_this_kt))
                kTs.append(kT)
        savefig(plt,
               'relaxation_time_LJ_av_tau',
               'trial2_tau_fit.pdf')
        if len(taus)>1:
            taus=np.asarray(taus)
            taus_sem=np.asarray(taus_sem)
            kTs=np.asarray(kTs)*T_conversion
            plt.figure(1)
            tau_s,T_s,r_squared = fit_wlf(kTs,taus)
            fitkTs = np.linspace(1.5,4.0)*T_conversion
            fit_taus = wlf(fitkTs,tau_s,T_s)
            plt.plot(fitkTs,
                    fit_taus,
                    label='$\\tau_s$: {:.2e} (s), $T_s$ : {:.0f} K, $R^2$: {:.2f}'.format(tau_s,
                                                                                          T_s,
                                                                                          r_squared))
            plt.errorbar(kTs,
                         taus,
                         taus_sem,
                          markerfacecolor='w',
                         fmt='s')
            plt.xlabel("Temperature (K)")
            plt.ylabel('Relaxation Time  $\\tau$ (s)')
            plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0), useMathText=True)
            plt.legend(fontsize=15,loc='best')
            savefig(plt,
               'relaxation_time_LJ_av_tau',
               'wlf_fit.pdf')


        plt.show()
```

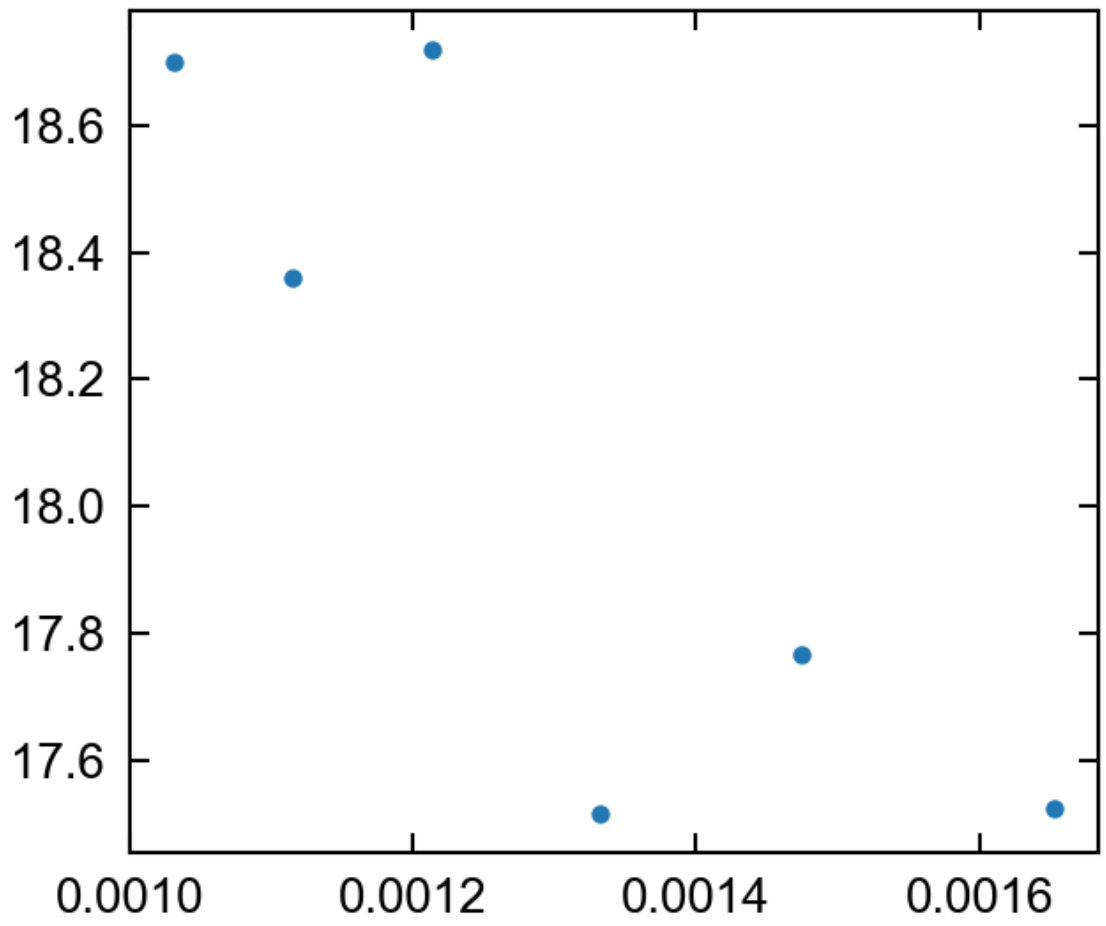
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```


0.8
0.7
0.6
0.5
0.4
0.3
0.2
q_{max} [nm^{-1}]
T: 730 K (q_0: 0.25 nm^{-1}, R^2: 0.38)
T: 803 K (q_0: 0.20 nm^{-1}, R^2: 0.54)
T: 876 K (q_0: 0.22 nm^{-1}, R^2: 0.69)
T: 949 K (q_0: 0.27 nm^{-1}, R^2: 0.57)
T: 1022 K (q_0: 0.21 nm^{-1}, R^2: 0.69)
T: 1095 K (q_0: 0.31 nm^{-1}, R^2: 0.44)
0.00
0.25
0.50
0.75
1.00
Time (s)
1e−6




```
In [3]: plt.plot(1/(kTs-T_s),
                 np.log(1/(taus/tau_s)),
                marker='o',
                linewidth=0.0)
        plt.show()
        np.log(1/(taus/tau_s))
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
Out[3]: array([ 17.52521223,  17.76624017,  17.51520357,  18.72006425,
                18.3611811 ,  18.69871572])
```

```
In [4]: def plot_wlf(ax,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s, label,color,normalize=False):
            fit_taus = wlf(exp_fitKts,tau_s=exp_tau_s,Ts=exp_Ts)
            if normalize:
                fit_taus=fit_taus/exp_tau_s
                normalized_exp_tau_s = exp_tau_s/exp_tau_s
            else:
                normalized_exp_tau_s = exp_tau_s
            ln_at = np.log(fit_taus/exp_tau_s)
            ax.plot(exp_fitKts,#-exp_Ts,
                    fit_taus,
                     zorder=0,
                    color=color,
                    linestyle='--',
                    #marker='.',
                    label='WLF ({})'.format(label))
            if False:
                ax.scatter(exp_Ts,
                           normalized_exp_tau_s,
                           marker='s',
                           zorder=1,
                           #color='r',
                          label='$\\tau_s(T_s)$ ({})'.format(label))
```

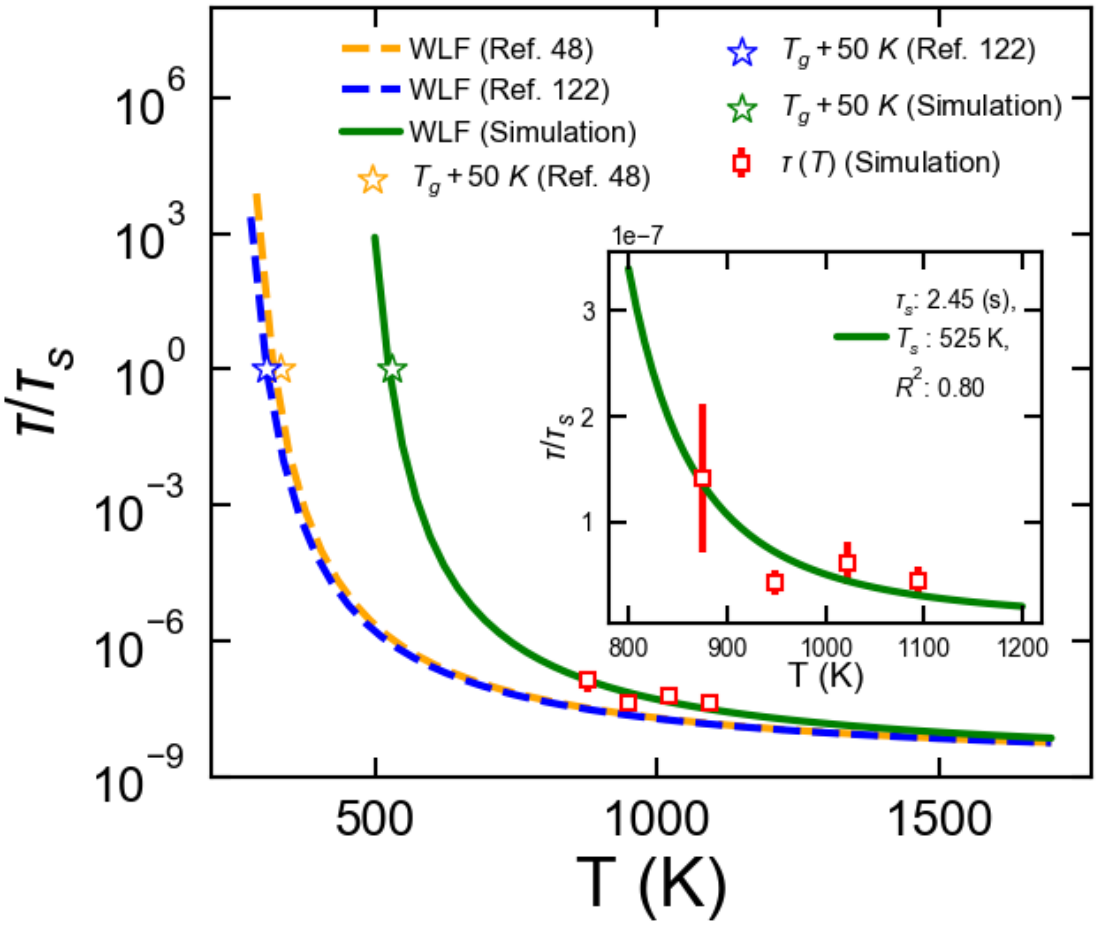
```
            ax.scatter(exp_Tg+50,
                       normalized_exp_tau_s,
                       marker='*',
                      s=150,
                       color=color,
                      facecolor='w',
                       #color='g',
                       zorder=1,
                      label='$T_g+50\ K$ ({})'.format(label))


            ax.legend(fontsize=10)
            ax.set_xlabel('$T\ (K)$')
            ax.set_ylabel('Relaxation Time (s)')
            ax.set_yscale('log')
```

In [15]:
```
        #fit_taus = wlf(fitkTs,2au_s,T_s)
        normalize=True
        Tg=480
        Ts=Tg+50
        fitkTs = np.linspace(Tg-50,1700)
        #print(fitkTs)
        exp_Tg = 282
        exp_Ts = 321
        exp_fitKts = np.linspace(290,1700)
        exp_tau_s = 1.74e6
        fig = plt.figure()
        ax1 = fig.add_subplot(111)
        left, bottom, width, height = [0.55, 0.35, 0.35, 0.35]#max t2
        ax2 = fig.add_axes([left, bottom, width, height])

        plot_wlf(ax1,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s,'Ref.
        48','orange',normalize=normalize)#Jyotishkumar et. al, 12.9%
        ABS/DGEBA/DDS',normalize=normalize)

        exp_Tg = 308-50
        exp_Ts = 308
        exp_fitKts = np.linspace(280,1700)
        exp_tau_s = 1.63e6
        plot_wlf(ax1,exp_Tg,exp_Ts,exp_fitKts,exp_tau_s,'Ref. 122','b',normalize=normalize)#'Yu,
        et. al, 20% PES/DGEBA/MTHPA',normalize=normalize)

        #print(taus)
        tau_s,T_s,r_squared = fit_wlf(kTs[2:],taus[2:],Ts_fixed=None)#480+50)
        #print('taus',tau_s,T_s)
        sim_Tg = 480
        sim_Ts = T_s
        sim_fitKts = np.linspace(500,1700)
        sim_tau_s = tau_s
        #fit_taus = wlf(fitkTs,tau_s,T_s)
        fitkTs = np.linspace(75,1200)
        sim_fit_taus = wlf(sim_fitKts,sim_tau_s,sim_Ts)
        if normalize:
            sim_fit_taus=sim_fit_taus/sim_tau_s
            normalized_taus = taus/sim_tau_s
            normalized_tau_sems = taus_sem/sim_tau_s
            normalized_sim_tau_s = sim_tau_s/sim_tau_s
        else:
            sim_fit_taus=sim_fit_taus
            normalized_taus = taus
            normalized_tau_sems = taus_sem
            normalized_sim_tau_s = sim_tau_s
        ln_at = np.log(fit_taus/tau_s)
        ax1.plot(sim_fitKts,
                 sim_fit_taus,
                  color='g',
                  zorder=0,
                 label='WLF (Simulation)')
```

```
ax1.errorbar(kTs[2:],
             normalized_taus[2:],
             normalized_tau_sems[2:],
             markerfacecolor='w',
             zorder=0,
             color='r',
             fmt='s',
            label='$\\tau\ (T)$ (Simulation)')
if False:
    ax1.scatter(sim_Ts,
                normalized_sim_tau_s,
                marker='s',
                facecolor='w',
                zorder=1,
                color='b',
               label='$\\tau_s\ (T-s)$ (Simulation)')
ax1.scatter(sim_Tg+50,
            normalized_sim_tau_s,
            marker='*',
            s=150,
            facecolor='w',
            color='g',
            zorder=1,
           label='$T_g+50\ K$ (Simulation)')

#ax1.set_yscale('log')
ax1.legend(fontsize=12,ncol=2)
ax1.set_xlabel('T (K)')
ax1.set_ylabel('$\\tau/\\tau_s$')
ax1.set_ylim(1e-9,1e8)
ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)

sim_fitKts = np.linspace(800,1200)
sim_tau_s = tau_s
#fit_taus = wlf(fitkTs,tau_s,T_s)
fitkTs = np.linspace(600,1200)
sim_fit_taus = wlf(sim_fitKts,sim_tau_s,sim_Ts)
if normalize:
    sim_fit_taus=sim_fit_taus/sim_tau_s
    normalized_taus = taus/sim_tau_s
    normalized_tau_sems = taus_sem/sim_tau_s
    normalized_sim_tau_s = sim_tau_s/sim_tau_s
else:
    sim_fit_taus=sim_fit_taus
    normalized_taus = taus
    normalized_tau_sems = taus_sem
    normalized_sim_tau_s = sim_tau_s

ln_at = np.log(fit_taus/tau_s)
ax2.plot(sim_fitKts,
        sim_fit_taus,
         zorder=0,
         color='g',
        label='$\\tau_s$: {:.2f} (s), \n$T_s$ : {:.0f} K, \n$R^2$: {:.2f}'.format(tau_s,
                                                                                  T_s,
                                                                                  r_squared))
ax2.errorbar(kTs[2:],
             normalized_taus[2:],
             normalized_tau_sems[2:],
             markerfacecolor='w',
             zorder=0,
             color='r',
             fmt='s')
ax2.set_xlabel("T (K)",fontsize=14,labelpad=0)
ax2.set_ylabel(r"$\tau/\tau_s$",fontsize=14,labelpad=0)
ax2.legend(fontsize=10)
#ax2.set_yscale('log')
#ax2.set_ylim(0,3.4)
```

```
        #ax2.set_xlim(0.7,6)
        ax2.tick_params(axis='both', labelsize=10,length=7,pad=6)
        ax2.ticklabel_format(axis='y', style='sci', scilimits=(-2,2),labelsize=2)
        ax2.yaxis.offsetText.set_fontsize(10)
        #plt.xlim(0,3000)
        savefig(plt,
                'relaxation_time_LJ_av_tau',
               'wlf_normalized.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [1]:
```
data_path = '/Users/stephenthomas/Google
Drive/Research/2017/Papers/Epoxpy_methods_paper/data/sensitivity_analysis'
```

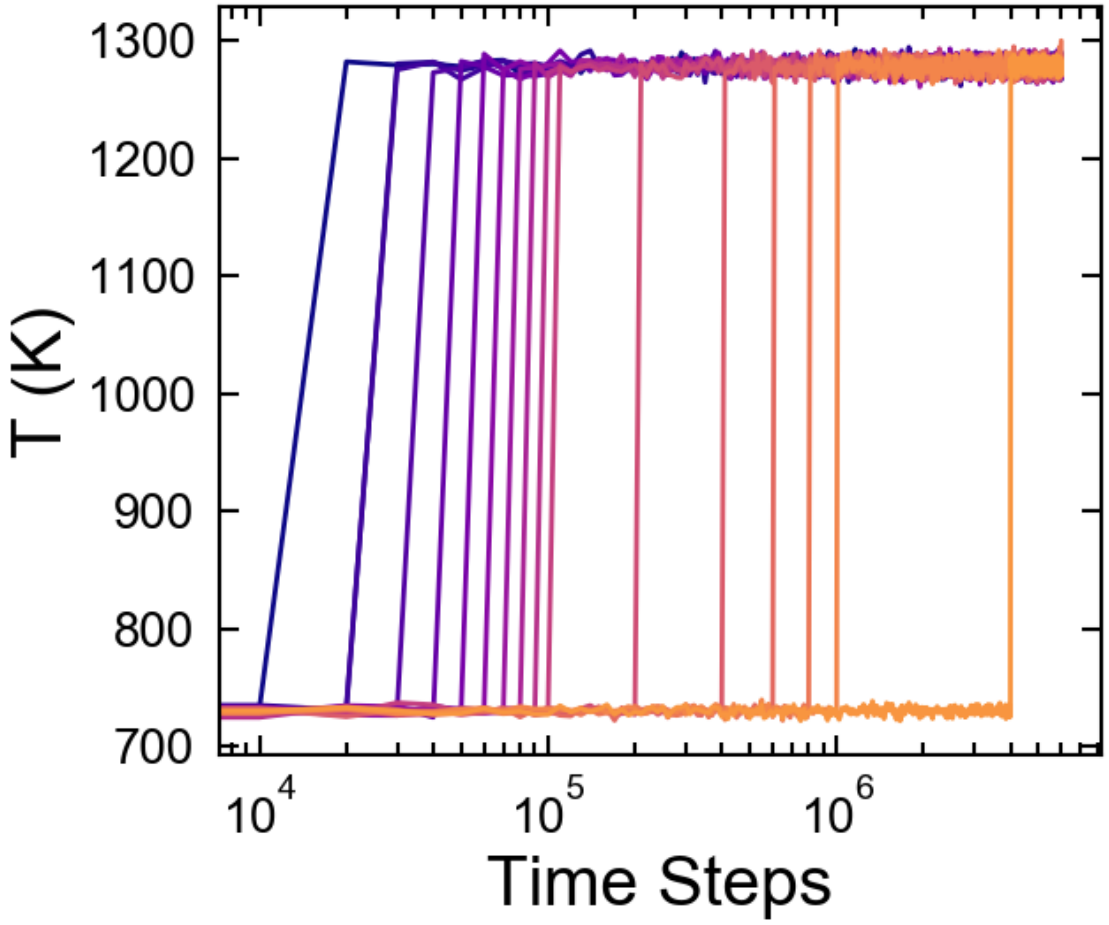
In [2]:
```
import signac
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
%matplotlib inline
from scipy.signal import argrelextrema as argex
import matplotlib.cm as cm
import itertools
from common import *

names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
markers={'iso':'s','lin_ramp':'P','step':'>'}
linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

def init_project():
    df_index = pd.DataFrame(project.index())
    df_index = df_index.set_index(['_id'])
    statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
    #print(statepoints)
    df = pd.DataFrame(statepoints).T.join(df_index)
    df = df.sort_values('T')
    return df
project = signac.get_project(data_path)
df = init_project()
```

In [3]:
```
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                (df.profile=='ramp_up_and_down')]
df['t1'] = df_filtered['temp_prof'].str[1].str[0]
df['t2'] = df_filtered['temp_prof'].str[3].str[0]
```

In [4]:
```
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                (df.profile=='step_SA')]
df['t1'] = df_filtered['temp_prof'].str[1].str[0]
df_sorted = df.sort_values('t1')
```

In [5]:
```
fig = plt.figure()
ax1 = fig.add_subplot(111)
lines=[]
time_conversion = 1.05e-11 #s
distance_conversion = 1.06 #nm
TemperatureConversion = 365.01 #K
df_filtered = df[(df.activation_energy==3.0)&
                (df.stop_after_percent==100.0)&
                (df.profile=='step_SA')]
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_filtered.groupby('t1'))))
for i,(key,dfgrp) in enumerate(df_filtered.groupby('t1')):
    for jobid in dfgrp.index:
        job=project.open_job(id=jobid)
        data = np.genfromtxt(job.fn('out.log'),names=True)
        ax1.plot(data['timestep'],
                 data['temperature']*TemperatureConversion,
                 linewidth=2.0,
                color=colors[i])
ax1.set_xlabel('Time Steps')
ax1.set_ylabel('T (K)')
plt.legend(fontsize=15)
plt.xscale('log')
savefig(plt,
       'sensitivity_analysis_t1',
       'temperature_profiles.pdf')
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use
label='...' kwarg on individual plots.
  warnings.warn("No labelled objects found. "
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



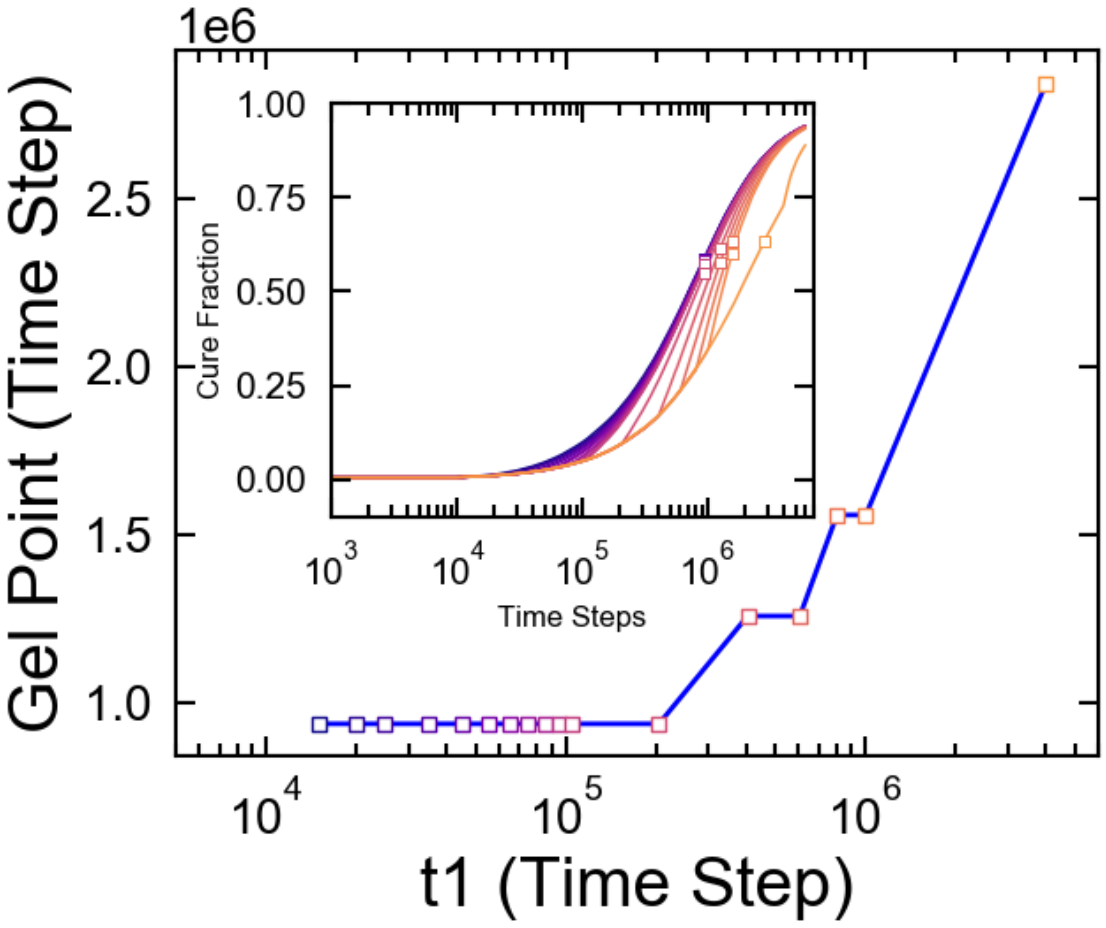

```
In [6]: fig = plt.figure()
        ax1 = fig.add_subplot(111)
        gel_points=[]
        ramp_up_times=[]
        # These are in unitless percentages of the figure size. (0,0 is bottom left)

        df_filtered = df[(df.activation_energy==3.0)&
                         (df.stop_after_percent==100.0)&
                         (df.profile=='step_SA')]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(df_filtered.groupby('t1'))))
        for i,(key,dfgrp) in enumerate(df_filtered.groupby('t1')):
            for jobid in dfgrp.index:
                job=project.open_job(id=jobid)
                x=job.document['gel_point']
```

```
            gel_points.append(x)
            ramp_up_times.append(key)
            print(key,job.document['gel_point'])
            ax1.scatter(key,
                        job.document['gel_point'],
                       marker='s',
                        facecolor='w',
                        linewidth=1.0,
                        color=colors[i],
                       zorder=1)
        ax1.plot(ramp_up_times,
                 gel_points,
                 color='b',
                 linewidth=2.0,
                label='Gel Point',
                zorder=0)
        #print(ramp_up_times)
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax1.set_xlabel('t1 (Time Step)')
        ax1.set_ylabel('Gel Point (Time Step)')
        ax1.set_xscale('log')
        ax1.set_xlim(5e3,6e6)

        left, bottom, width, height = [0.32, 0.45, 0.39, 0.39]
        ax2 = fig.add_axes([left, bottom, width, height])
        colors = plt.cm.plasma(np.linspace(0,0.75,len(df_filtered.groupby('t1'))))
        for i,(key,dfgrp) in enumerate(df_filtered.groupby('t1')):
            for jobid in dfgrp.index:
                job=project.open_job(id=jobid)
                data = np.genfromtxt(job.fn('out.log'),names=True)
                ax2.plot(data['timestep'],
                         data['bond_percentAB']/100,
                         label='t1: {:.1e}'.format(key),
                         color=colors[i],
                         linewidth=1.0,
                        zorder=0)
                ax2.scatter(job.document['gel_point'],
                           job.document['curing_at_gel_point']/100,
                            marker='s',
                            facecolor='w',
                            linewidth=0.7,
                            s=20,
                            color=colors[i],
                           zorder=1)
        ax2.tick_params(axis = 'both', which = 'major', labelsize = 15)
        ax2.set_ylim(-0.1,1)
        ax2.set_xlim(1e3,7e6)
        ax2.set_xlabel('Time Steps',fontsize=12.0)
        ax2.set_ylabel('Cure Fraction',fontsize=12.0)
        ax2.set_xscale('log')
        savefig(plt,
               'sensitivity_analysis_t1',
               'gel_points.pdf')
        plt.show()
```

```
15000.0 940000.0
20000.0 940000.0
25000.0 940000.0
35000.0 940000.0
45000.0 940000.0
55000.0 940000.0
65000.0 940000.0
75000.0 940000.0
85000.0 940000.0
95000.0 940000.0
105000.0 940000.0
```

```
205000.0 940000.0
405000.0 1260000.0
605000.0 1260000.0
805000.0 1560000.0
1005000.0 1560000.0
4005000.0 2840000.0
```

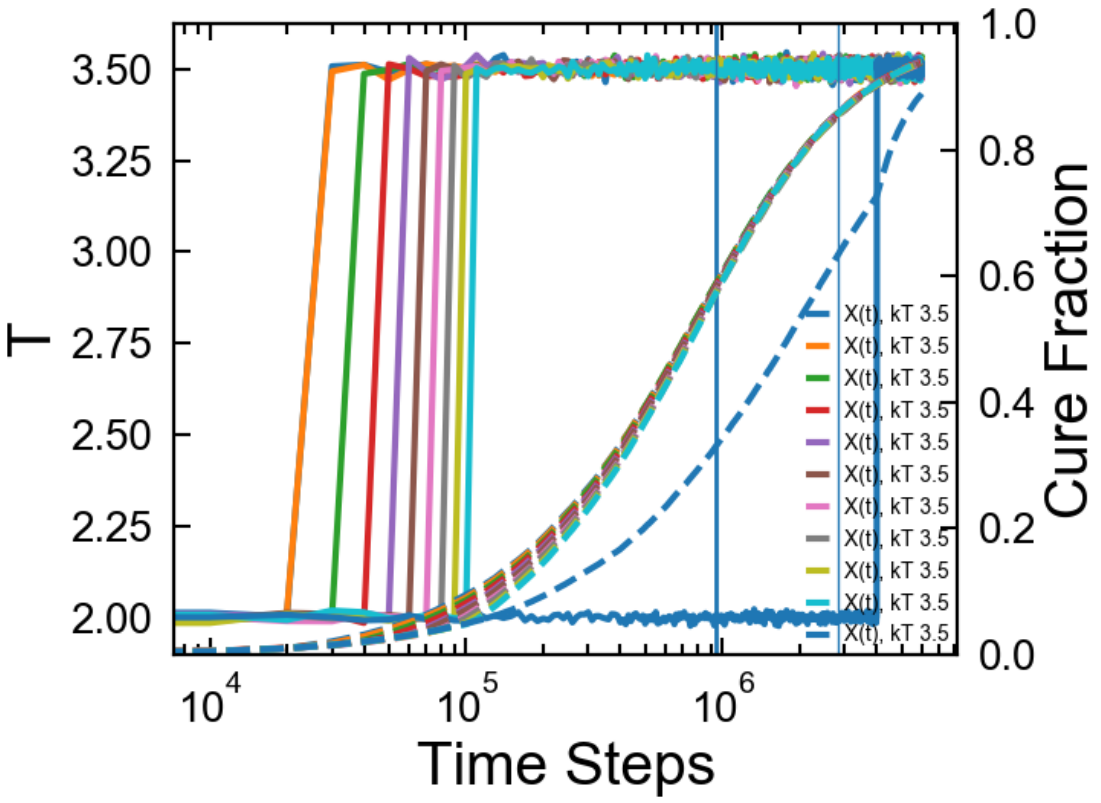
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



In [7]:
```
fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(111)
ax2 = ax1.twinx()
lines=[]
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                (df.t_SetT>10000)&
                ((df.t_SetT<=1e5)|(df.t_SetT==4e6))&
               (df.profile=='step_SA')]
```

```
        df_sorted = df_filtered.sort_values('t_SetT')
        for jobid in df_sorted.index:
            job=project.open_job(id=jobid)
            data = np.genfromtxt(job.fn('out.log'),names=True)
            ln1 = ax2.plot(data['timestep'],
                           data['bond_percentAB']/100,
                           linestyle='--',
                           label='X(t), kT {}'.format(job.sp.kT))
            ln2 = ax1.plot(data['timestep'],
                           data['temperature'])
            ax1.axvline(x=job.document['gel_point'],linewidth=1.0)
            lines.append(ln1)
            ax1.set_xlabel('Time Steps')
            ax2.set_ylabel('Cure Fraction')
            ax2.set_ylim(0,1)
            ax1.set_ylabel('T')


        for i,line in enumerate(lines):
            if i==0:
                lns=line
            else:
                lns=lns+line
        labs = [l.get_label() for l in lns]
        ax1.legend(lns, labs, loc=0)
        legend = ax1.legend(lns,
                            labs,
                            loc='lower right',
                            shadow=False,
                            prop={'size':10},
                            handlelength=1.5,
                            borderaxespad = 0)
        plt.xscale('log')
        plt.show()
```

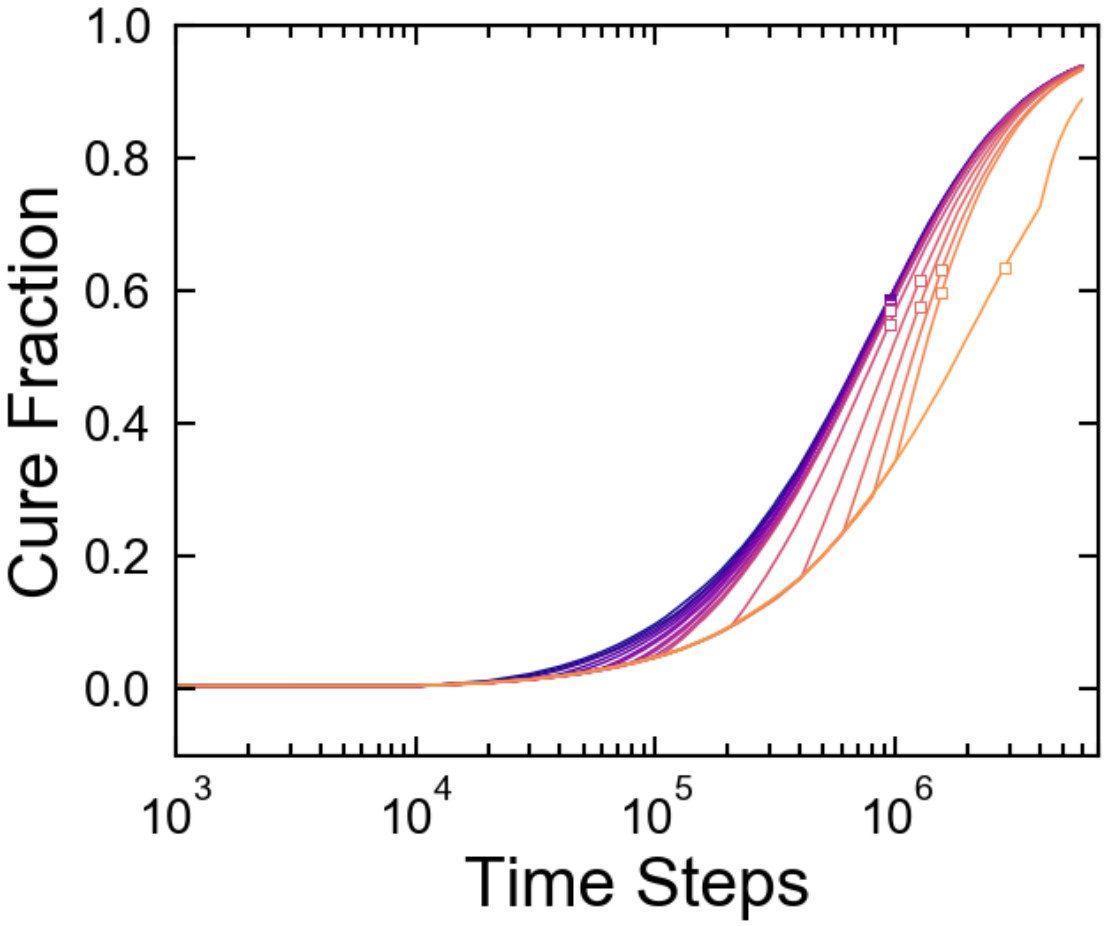
```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```




```
In [9]: fig = plt.figure()
        ax1 = fig.add_subplot(111)
        df_filtered = df[(df.activation_energy==3.0)&
                         (df.stop_after_percent==100.0)&
                         (df.profile=='step_SA')]
        colors = plt.cm.plasma(np.linspace(0,0.75,len(df_filtered.groupby('t1'))))
        for i,(key,dfgrp) in enumerate(df_filtered.groupby('t1')):
            for jobid in dfgrp.index:
                job=project.open_job(id=jobid)
                data = np.genfromtxt(job.fn('out.log'),names=True)
                ax1.plot(data['timestep'],
                         data['bond_percentAB']/100,
                         label='t1: {:.1e}'.format(key),
                         color=colors[i],
                         linewidth=1.0,
                        zorder=0)
                ax1.scatter(job.document['gel_point'],
                            job.document['curing_at_gel_point']/100,
                             marker='s',
                             facecolor='w',
                             linewidth=0.7,
                             s=20,
                             color=colors[i],
                            zorder=1)
        ax1.set_ylim(-0.1,1)
        ax1.set_xlim(1e3,7e6)
        ax1.set_xlabel('Time Steps')
        ax1.set_ylabel('Cure Fraction')
        ax1.set_xscale('log')
        savefig(plt,
```

```
            'sensitivity_analysis_t1',
            'cure_fractions_w_gel_points.pdf')
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

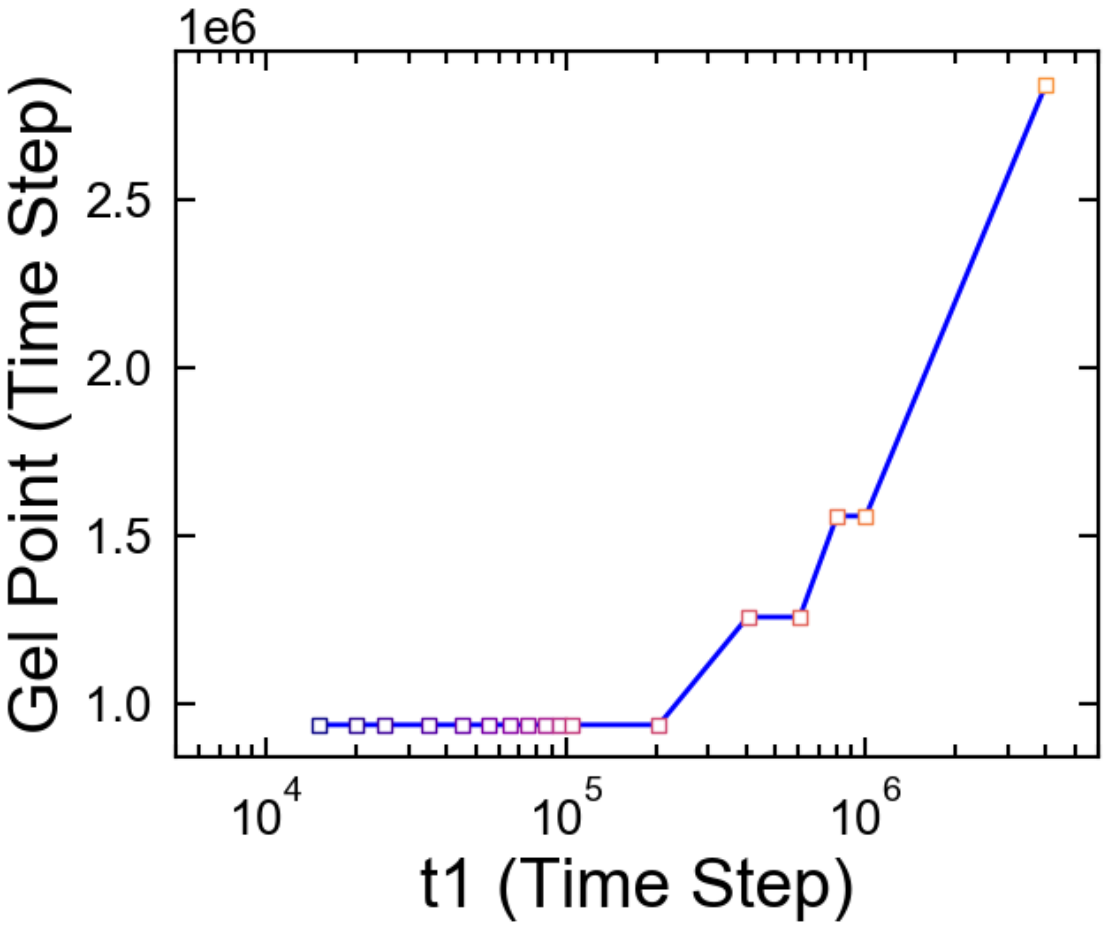


In [11]:
```
fig = plt.figure()
ax1 = fig.add_subplot(111)
gel_points=[]
ramp_up_times=[]
# These are in unitless percentages of the figure size. (0,0 is bottom left)

df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                 (df.profile=='step_SA')]
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_filtered.groupby('t1'))))
for i,(key,dfgrp) in enumerate(df_filtered.groupby('t1')):
    for jobid in dfgrp.index:
        job=project.open_job(id=jobid)
        x=job.document['gel_point']
        gel_points.append(x)
```

```
                ramp_up_times.append(key)
                print(key,job.document['gel_point'])
                ax1.scatter(key,
                            job.document['gel_point'],
                           marker='s',
                            facecolor='w',
                            linewidth=1.0,
                            color=colors[i],
                           zorder=1)
        ax1.plot(ramp_up_times,
                 gel_points,
                 color='b',
                 linewidth=2.0,
                label='Gel Point',
                zorder=0)
        #print(ramp_up_times)
        ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))
        ax1.set_xlabel('t1 (Time Step)')
        ax1.set_ylabel('Gel Point (Time Step)')
        ax1.set_xscale('log')
        ax1.set_xlim(5e3,6e6)
        savefig(plt,
               'sensitivity_analysis_t1',
               'gel_points_only.pdf')
```

```
15000.0 940000.0
20000.0 940000.0
25000.0 940000.0
35000.0 940000.0
45000.0 940000.0
55000.0 940000.0
65000.0 940000.0
75000.0 940000.0
85000.0 940000.0
95000.0 940000.0
105000.0 940000.0
205000.0 940000.0
405000.0 1260000.0
605000.0 1260000.0
805000.0 1560000.0
1005000.0 1560000.0
4005000.0 2840000.0
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

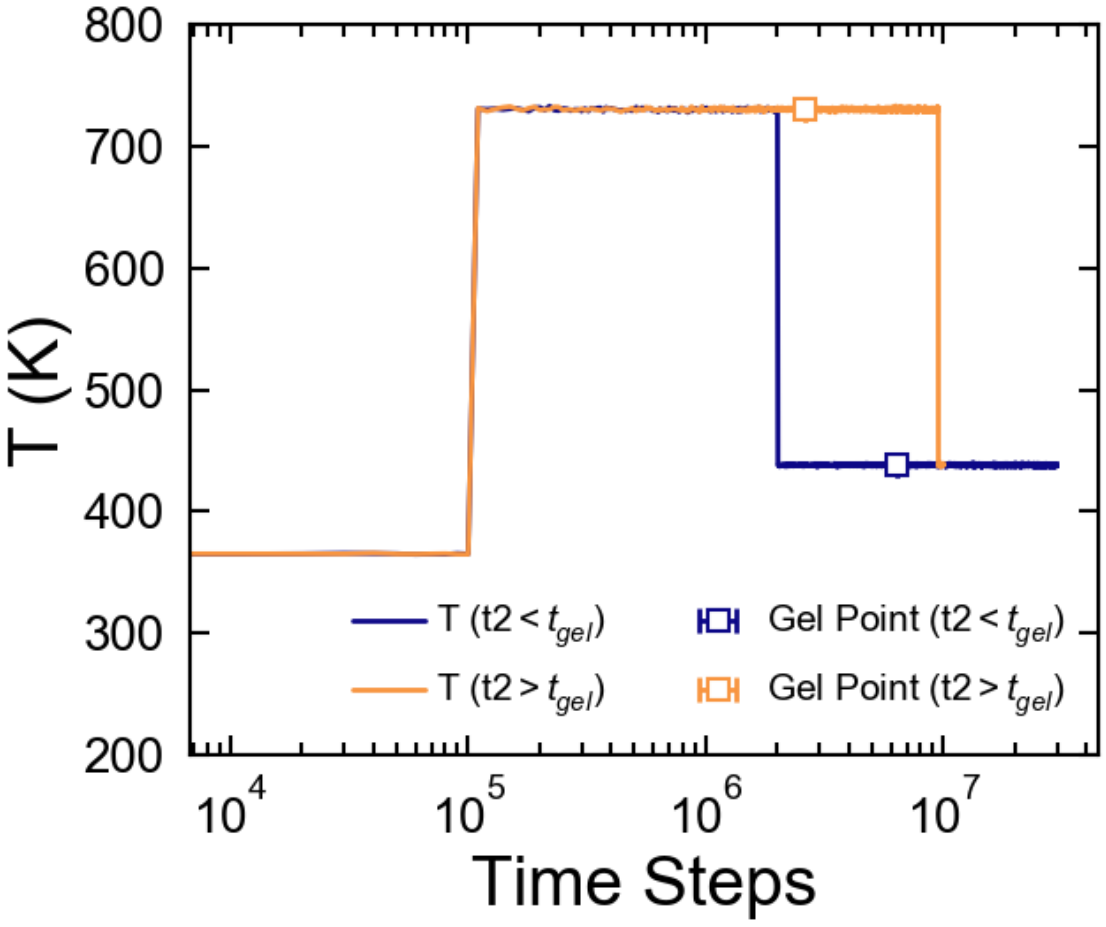

1e6
2.5
2.0
1.5
1.0
Gel Point (Time Step)
10^4
10^5
10^6
t1 (Time Step)

```
In [9]: data_path = '/Users/stephenthomas/Google
        Drive/Research/2017/Papers/Epoxpy_methods_paper/data/sensitivity_analysis'

In [10]: import signac
         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
         import matplotlib
         %matplotlib inline
         from scipy.signal import argrelextrema as argex
         import matplotlib.cm as cm
         import itertools
         from common import *

         names={'iso':'Isothermal','lin_ramp':'Linear Ramp','step':'Step'}
         colors={'iso':'C0','lin_ramp':'C1','step':'C2'}
         markers={'iso':'s','lin_ramp':'P','step':'>'}
         linestyles={'iso':'-','lin_ramp':'--','step':'-.'}

         def init_project():
             df_index = pd.DataFrame(project.index())
             df_index = df_index.set_index(['_id'])
             statepoints = {doc['_id']: doc['statepoint'] for doc in project.index()}
             #print(statepoints)
             df = pd.DataFrame(statepoints).T.join(df_index)
             df = df.sort_values('T')
             return df
         project = signac.get_project(data_path)
         df = init_project()

In [11]: df_filtered = df[(df.activation_energy==3.0)&
                           (df.stop_after_percent==100.0)&
                          (df.profile=='step_SA')]
         df['t1'] = df_filtered['temp_prof'].str[1].str[0]

In [12]: df_filtered = df[(df.stop_after_percent==100.0)&
                          (df.profile=='ramp_up_and_down')]
         df['t1'] = df_filtered['temp_prof'].str[1].str[0]
         df['t2'] = df_filtered['temp_prof'].str[3].str[0]
         df['T2'] = df_filtered['temp_prof'].str[4].str[1]
         df['t_SetT'] = df['t2']

In [55]: import matplotlib.patheffects as pe
         fig = plt.figure()
         ax1 = fig.add_subplot(111)
         lines=[]
         time_conversion = 1.05e-11 #s
         distance_conversion = 1.06 #nm
         TemperatureConversion = 365.01 #K
         df_filtered = df[(df.activation_energy==3.0)&
                          (df.stop_after_percent==100.0)&
                          (df.t1==1.05e5)&
                          (df.E_factor==1.0)&
                          (df.T2==1.2)&
                          (((df.t2==2005001)&(df.t_Final==3e7))|
                           ((df.t2==9505001)&(df.t_Final==1.0e7)))&
                          (df.n_particles==4e5)&
                          (df.profile=='ramp_up_and_down')]
         df_sorted = df_filtered.sort_values('t2')
         colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('t2'))))
         for i,(key,dfgrp) in enumerate(df_sorted.groupby('t2')):
             for jobid in dfgrp.index:
                 job=project.open_job(id=jobid)
                 if job.isfile('out.log') and ('gel_point' in job.document) :
                     #print(job,key)
                     data = np.genfromtxt(job.fn('out.log'),names=True)
                     timesteps = data['timestep']
```

```
            temperature = data['temperature']
            pops=[]
            pops = [ temp_i+1 for temp_i, (x, y) in
enumerate(zip(timesteps,timesteps[1:])) if x>=y]
            if len(pops)>0:
                #print(len(pops),pops)
                timesteps = np.asarray([x for i,x in enumerate(timesteps) if i not in
pops])
                temperature   = np.asarray([x for i,x in enumerate(temperature) if i not
in pops])
            temperature = temperature*TemperatureConversion
            gel_point = job.document['gel_point']
            T_at_gel = temperature[np.isclose(timesteps,gel_point)]
            #print(T_at_gel)
            if job.sp.trial==1:
                if key<gel_point:
                    label='T (t2'+'$< t_{gel}$)'
                else:
                    label='T (t2'+'$> t_{gel}$)'
                ax1.plot(timesteps,
                         temperature,
                         linewidth=2.0,
                        color=colors[i],
                         label=label)
                        #label='T (t2 : {:.1e})'.format(key))

            if False:
                ax1.axvline(x=job.document['gel_point'],
                           color=colors[i],
                           linestyle='--',
                           linewidth=1.0,
                           label='Gel Point')
    if True:
        if key<gel_point:
            label='Gel Point (t2'+'$< t_{gel}$)'
        else:
            label='Gel Point (t2'+'$> t_{gel}$)'
        ax1.errorbar(x=dfgrp.gel_point.mean(),
                    y=T_at_gel,#400,
                    xerr=dfgrp.gel_point.std(),
                     label=label,#'$t_{gel}$'+' (t2 : {:.1e})'.format(key),
                     color=colors[i],#'r',
                     fmt='s',
                     markerfacecolor='w',
                     lw=3,
                     ms=8,
                     capthick=2,
                     elinewidth=2,
                     capsize=5,
                    zorder=2)
if False:
    ax1.axvline(x=df_filtered.gel_point.mean(),
                color='r',
                linestyle='--',
                linewidth=4.0,
                path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
    ax1.errorbar(x=df_filtered.gel_point.mean(),
                y=300,
                xerr=df_filtered.gel_point.std(),
                 label='Gel Point',
                 color='r',
                 fmt='o',
                 markerfacecolor='w',
                 markeredgecolor='r',
                 lw=3,
                 ms=8,
                 capthick=2,
                 elinewidth=3,
```

```
                    capsize=5,
                   zorder=2)
        ax1.set_xlabel('Time Steps')
        ax1.set_ylabel('T (K)')
        ax1.set_ylim(200,800)
        #ax1.set_xlim(5e4,7e6)
        plt.legend(fontsize=15,loc='lower right',ncol=2)
        plt.xscale('log')
        savefig(plt,
               'sensitivity_analysis_t2_N_4e5',
               'temperature_profiles.pdf')
        plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```

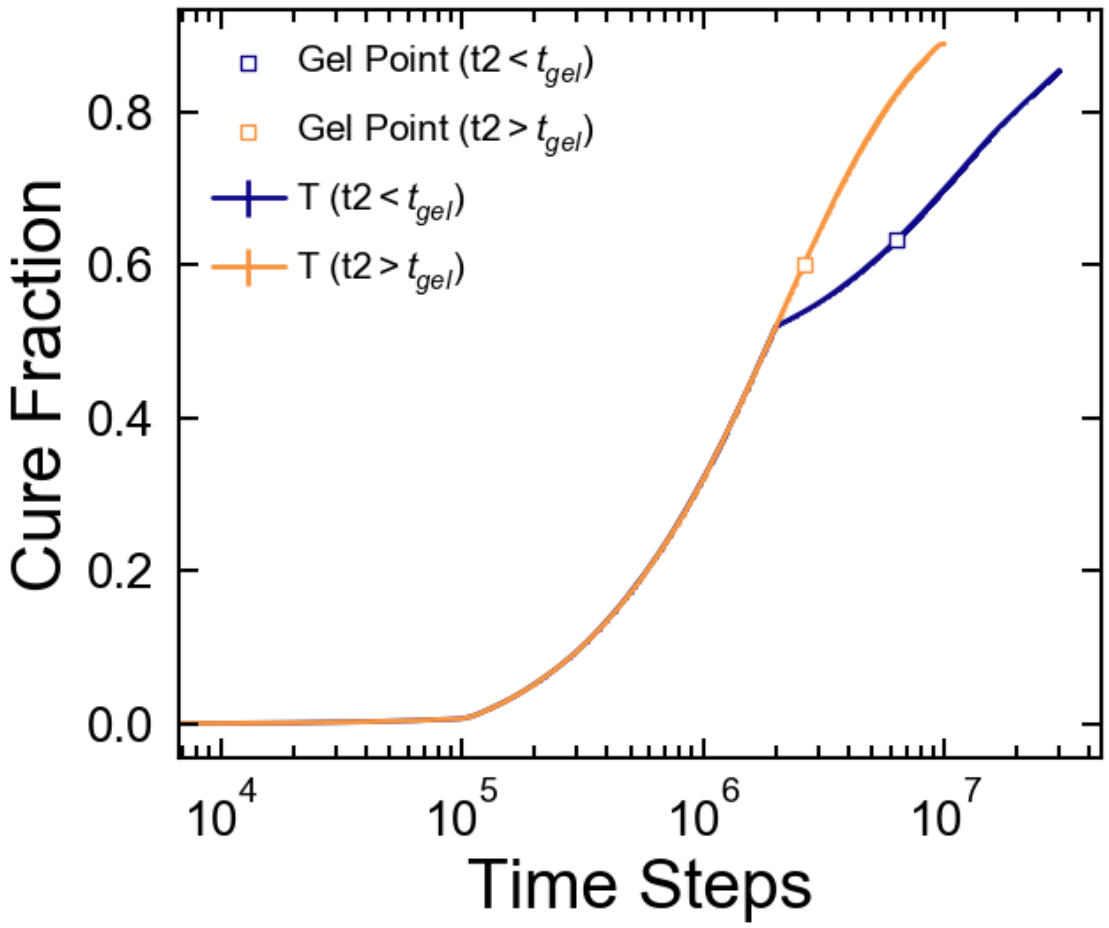


In [56]:
```
fig = plt.figure()
ax1 = fig.add_subplot(111)
lines=[]
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
```

```
                    (df.t1==1.05e5)&
                    (df.E_factor==1.0)&
                    (df.T2==1.2)&
                    (((df.t2==2005001)&(df.t_Final==3e7))|
                     ((df.t2==9505001)&(df.t_Final==1.0e7)))&
                    (df.n_particles==4e5)&
                    (df.profile=='ramp_up_and_down')]
df_sorted = df_filtered.sort_values('t2')
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('t2'))))
for i,(key,dfgrp) in enumerate(df_sorted.groupby('t2')):
    times=[]
    cure_fractions=[]
    gel_points=[]
    cure_at_gel=[]
    #print(dfgrp.t_Final)
    for jobid in dfgrp.index:
        job=project.open_job(id=jobid)
        if job.isfile('out.log') and ('gel_point' in job.document) :
            data = np.genfromtxt(job.fn('out.log'),names=True)
            timesteps = data['timestep']
            cure_fraction = data['bond_percentAB']/100
            pops = [ i for i, (x, y) in enumerate(zip(timesteps[:-1],timesteps[1:])) if
x>=y]
            if len(pops)>0:
                popi=pops[0]+1
                timesteps = np.asarray([x for i,x in enumerate(timesteps) if i!=popi])
                cure_fraction = np.asarray([x for i,x in enumerate(cure_fraction) if
i!=popi])

            times.append(timesteps)
            cure_fractions.append(cure_fraction)
            gel_points.append(job.document['gel_point'])
            cure_at_gel.append(job.document['curing_at_gel_point']/100)
    if key<gel_point:
        label='T (t2'+'$< t_{gel}$)'
    else:
        label='T (t2'+'$> t_{gel}$)'
    ax1.errorbar(np.mean(times,axis=0),
                 np.mean(cure_fractions,axis=0),
                 np.std(cure_fractions,axis=0),
                 label=label,#'t2: {:.1e}'.format(key),
                 color=colors[i],
                 linewidth=2.0,
                 zorder=0)
    if key<gel_point:
        label='Gel Point (t2'+'$< t_{gel}$)'
    else:
        label='Gel Point (t2'+'$> t_{gel}$)'
    ax1.scatter(np.mean(gel_points),
                np.mean(cure_at_gel),
                 marker='s',
                 facecolor='w',
                 linewidth=1.0,
                 color=colors[i],
                zorder=1,
                label=label)#'Gel Point({:.1e})'.format(key))
ax1.set_xlabel('Time Steps')
ax1.set_ylabel('Cure Fraction')
plt.legend(fontsize=15)
plt.xscale('log')
savefig(plt,
        'sensitivity_analysis_t2_N_4e5',
        'cure_profiles.pdf')
plt.show()
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
```

```
warnings.warn("This figure includes Axes that are not "
```

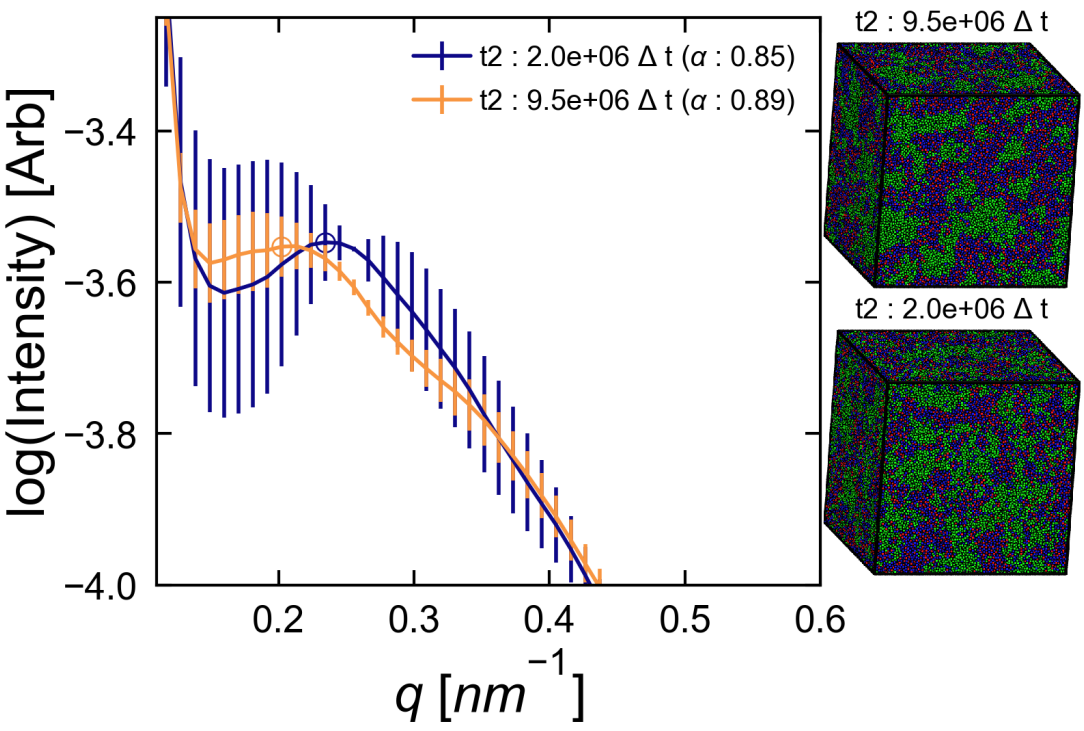


In [27]:
```
from scipy import stats
from scipy import interpolate
fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.90, 0.60, 0.3, 0.3]#[0.6, 0.57, 0.3, 0.3]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.90, 0.25, 0.3, 0.3]#[0.23, 0.27, 0.3, 0.3]#min t2
ax3 = fig.add_axes([left, bottom, width, height])
ax3.axis('off')
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                 (df.t1==1.05e5)&
                 (df.E_factor==1.0)&
                 (df.T2==1.2)&
                 (((df.t2==2005001)&(df.t_Final==3e7))|
                  ((df.t2==9505001)&(df.t_Final==1.0e7)))&
                 (df.n_particles==4e5)&
                 (df.profile=='ramp_up_and_down')]
#print(df_filtered.profile)
df_sorted = df_filtered.sort_values('t2')
print(df_sorted.t2.values)
```

```
print('min t2',df_sorted.t2[3])#.min())
print('max t2',df_sorted.t2.max())
t2_min = 2005001#df_sorted.t2[3]#.min()
t2_max = 9505001#df_sorted.t2.max()#df_sorted.t2.max()
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('t2'))))

df_gp = df_sorted.groupby('t2')
for i,(key,df_grp) in enumerate(df_gp):
    qs_all_trials = []
    Is_all_trials = []
    box_len = df_grp.Lx.mean()
    cure_percent = df_grp.cure_percent.mean()
    #print('cure_percent',cure_percent)
    if key ==t2_max  or key==t2_min:
        #print('key',key)
        for jobid in df_grp.index:
            job=project.open_job(id=jobid)
            if job.isfile('out.log') and ('gel_point' in job.document) :
                #print(job,key)
A=job.sp.num_a*job.sp.n_mul*4*job.sp.percent_bonds_per_step/100/job.sp.bond_period
                if job.isfile('diffract_type_2/asq.txt'):
                    data = np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
                    qs=data[:,0]
                    Is=data[:,1]
                    qs_all_trials.append(qs)
                    Is_all_trials.append(Is)
                    if job.isfile('final_snapshot.png'):
                        im = plt.imread(job.fn('final_snapshot.png'))
                        if key ==t2_max:
                            ax2.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax2.imshow(im)
                        if key ==t2_min:
                            ax3.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax3.imshow(im)
                    #print('Is',Is)
                else:
                    raise FileNotFoundError('Cannot find diffract_type_2/asq.txt
for'+job)
            else:
                print(job,' not completed!')
    if len(Is_all_trials)>0:
        mean_qs = np.mean(qs_all_trials,axis=0)
        mean_Is = np.mean(Is_all_trials,axis=0)
        ax1.errorbar(mean_qs,
                     mean_Is,
                     yerr=stats.sem(Is_all_trials,axis=0),
                     linewidth=2.0,
                     zorder=1,
                     color=colors[i],
                 label='t2 : {:.1e} $\\Delta$ t ($\\alpha$ : {:.2f})'.format(key,
cure_percent/100))
        if True:
            qms=[]
            Ims=[]
            for qs_t,Is_t in zip(qs_all_trials,Is_all_trials):
                first_peak_q,first_peak_i = get_highest_maxima(box_len,qs_t,Is_t)
                qms.append(first_peak_q)
                Ims.append(first_peak_i)
            #print(qms)
            fn = interpolate.interp1d(mean_qs,mean_Is,kind='cubic')
            first_peak_q=np.mean(qms)
            first_peak_i=fn(first_peak_q)
        else:
            first_peak_q,first_peak_i = get_highest_maxima(box_len,
                                                          mean_qs,
                                                          nmean_Is)
```

```
                print(first_peak_q,first_peak_i)
                if first_peak_q is None:
                    #print(q_half_length)
                    fn = interpolate.interp1d(qs,Is,kind='cubic')
                    first_peak_q=q_half_length
                    first_peak_i=fn(first_peak_q)
                if first_peak_q >0.8:
                    fn = interpolate.interp1d(qs,Is,kind='cubic')
                    first_peak_q=q_half_length
                    first_peak_i=0#fn(first_peak_q)
                ax1.scatter(first_peak_q,
                       first_peak_i,
                       color=colors[i],
                        zorder=0,
                            s=100,
                            facecolor='w',
                        marker='o')

        ax1.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax1.set_ylabel("log(Intensity) [Arb]")
        ax1.set_xlim(0.11,0.6)
        ax1.set_ylim(-4.0,-3.25)
        ax1.legend(fontsize=15,loc='upper right')
        savefig(plt,
               'sensitivity_analysis_t2_N_4e5',
               'sf.pdf')
```

```
[ 2005001.  2005001.  2005001.  9505001.  9505001.  9505001.]
min t2 9505001.0
max t2 9505001.0
a52a2aacf366f4bfdb526c3c90a6e553  not completed!
0.234460668949 -3.5471132590244476
0.202488759547 -3.5527970868063505


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



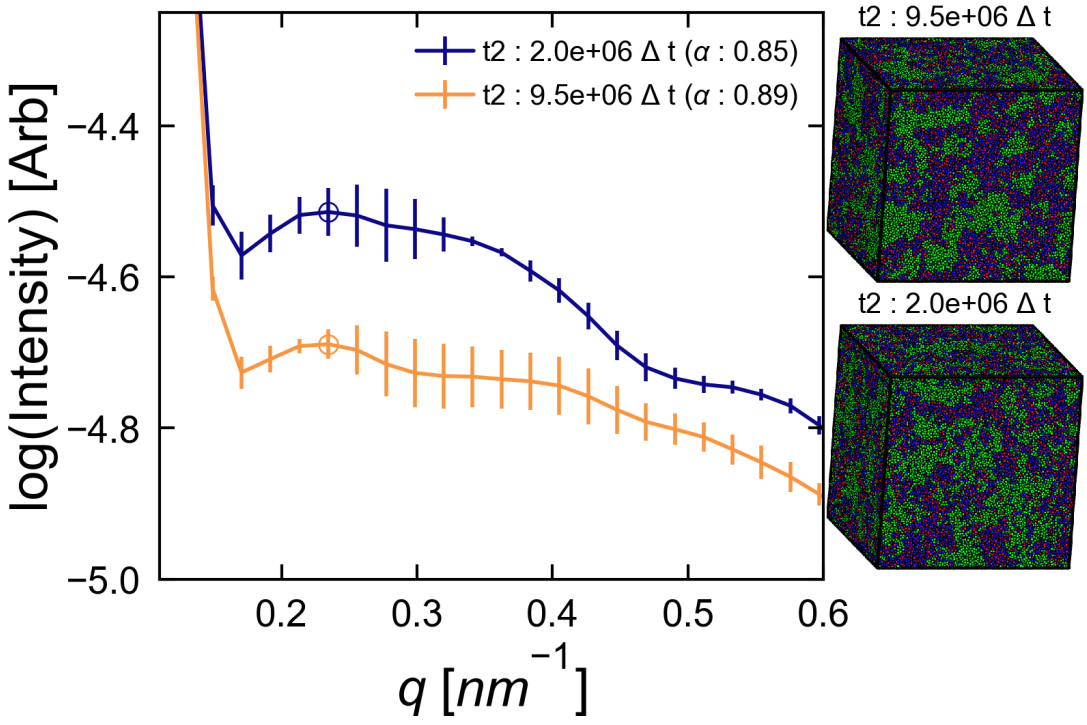

```
In [31]: from scipy import stats
         from scipy import interpolate
         fig = plt.figure(dpi=200)
         ax1 = fig.add_subplot(111)
         left, bottom, width, height = [0.90, 0.60, 0.3, 0.3]#[0.6, 0.57, 0.3, 0.3]#max t2
         ax2 = fig.add_axes([left, bottom, width, height])
         ax2.axis('off')
         left, bottom, width, height = [0.90, 0.25, 0.3, 0.3]#[0.23, 0.27, 0.3, 0.3]#min t2
         ax3 = fig.add_axes([left, bottom, width, height])
         ax3.axis('off')
         df_filtered = df[(df.activation_energy==3.0)&
                          (df.stop_after_percent==100.0)&
                          (df.t1==1.05e5)&
                          (df.E_factor==1.0)&
                          (df.T2==1.2)&
                          (((df.t2==2005001)&(df.t_Final==3e7))|
                           ((df.t2==9505001)&(df.t_Final==1.0e7)))&
                          (df.n_particles==4e5)&
                          (df.profile=='ramp_up_and_down')]
         #print(df_filtered.profile)
         df_sorted = df_filtered.sort_values('t2')
         print(df_sorted.t2.values)
         print('min t2',df_sorted.t2[3])#.min())
         print('max t2',df_sorted.t2.max())
         t2_min = 2005001#df_sorted.t2[3]#.min()
         t2_max = 9505001#df_sorted.t2.max()#df_sorted.t2.max()
         colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('t2'))))

         df_gp = df_sorted.groupby('t2')
         for i,(key,df_grp) in enumerate(df_gp):
             qs_all_trials = []
             Is_all_trials = []
             box_len = df_grp.Lx.mean()
             cure_percent = df_grp.cure_percent.mean()
```

```
    #print('cure_percent',cure_percent)
    if key ==t2_max  or key==t2_min:
        #print('key',key)
        for jobid in df_grp.index:
            job=project.open_job(id=jobid)
            if job.isfile('out.log'):# and ('gel_point' in job.document) :
                #print(job,key)
A=job.sp.num_a*job.sp.n_mul*4*job.sp.percent_bonds_per_step/100/job.sp.bond_period
                if job.isfile('diffract_at_gel/asq.txt'):
                    data = np.genfromtxt(job.fn('diffract_at_gel/asq.txt'))
                    qs=data[:,0]
                    Is=data[:,1]
                    qs_all_trials.append(qs)
                    Is_all_trials.append(Is)
                    if job.isfile('final_snapshot.png'):
                        im = plt.imread(job.fn('final_snapshot.png'))
                        if key ==t2_max:
                            ax2.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax2.imshow(im)
                        if key ==t2_min:
                            ax3.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax3.imshow(im)
                    #print('Is',Is)
                #else:
                    #raise FileNotFoundError('Cannot find diffract_type_2/asq.txt
for'+str(job))
            else:
                print(job,' not completed!')
    if len(Is_all_trials)>0:
        mean_qs = np.mean(qs_all_trials,axis=0)
        mean_Is = np.mean(Is_all_trials,axis=0)
        ax1.errorbar(mean_qs,
                     mean_Is,
                     yerr=stats.sem(Is_all_trials,axis=0),
                     linewidth=2.0,
                     zorder=1,
                     color=colors[i],
                 label='t2 : {:.1e} $\\Delta$ t ($\\alpha$ : {:.2f})'.format(key,
cure_percent/100))
        if True:
            qms=[]
            Ims=[]
            for qs_t,Is_t in zip(qs_all_trials,Is_all_trials):
                first_peak_q,first_peak_i = get_highest_maxima(box_len,qs_t,Is_t)
                qms.append(first_peak_q)
                Ims.append(first_peak_i)
            #print(qms)
            fn = interpolate.interp1d(mean_qs,mean_Is,kind='cubic')
            first_peak_q=np.mean(qms)
            first_peak_i=fn(first_peak_q)
        else:
            first_peak_q,first_peak_i = get_highest_maxima(box_len,
                                                          mean_qs,
                                                          nmean_Is)
        print(first_peak_q,first_peak_i)
        if first_peak_q is None:
            #print(q_half_length)
            fn = interpolate.interp1d(qs,Is,kind='cubic')
            first_peak_q=q_half_length
            first_peak_i=fn(first_peak_q)
        if first_peak_q >0.8:
            fn = interpolate.interp1d(qs,Is,kind='cubic')
            first_peak_q=q_half_length
            first_peak_i=0#fn(first_peak_q)
        ax1.scatter(first_peak_q,
               first_peak_i,
```

```
                        color=colors[i],
                         zorder=0,
                             s=100,
                             facecolor='w',
                         marker='o')

        ax1.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax1.set_ylabel("log(Intensity) [Arb]")
        ax1.set_xlim(0.11,0.6)
        ax1.set_ylim(-5.0,-4.25)
        ax1.legend(fontsize=15,loc='upper right')
        savefig(plt,
               'sensitivity_analysis_t2_N_4e5',
               'sf_at_gel.pdf')
```

```
[ 2005001.  2005001.  2005001.  9505001.  9505001.  9505001.]
min t2 9505001.0
max t2 9505001.0
0.234460668949 -4.514021700778694
0.234460668949 -4.688957286555088
```

```
/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



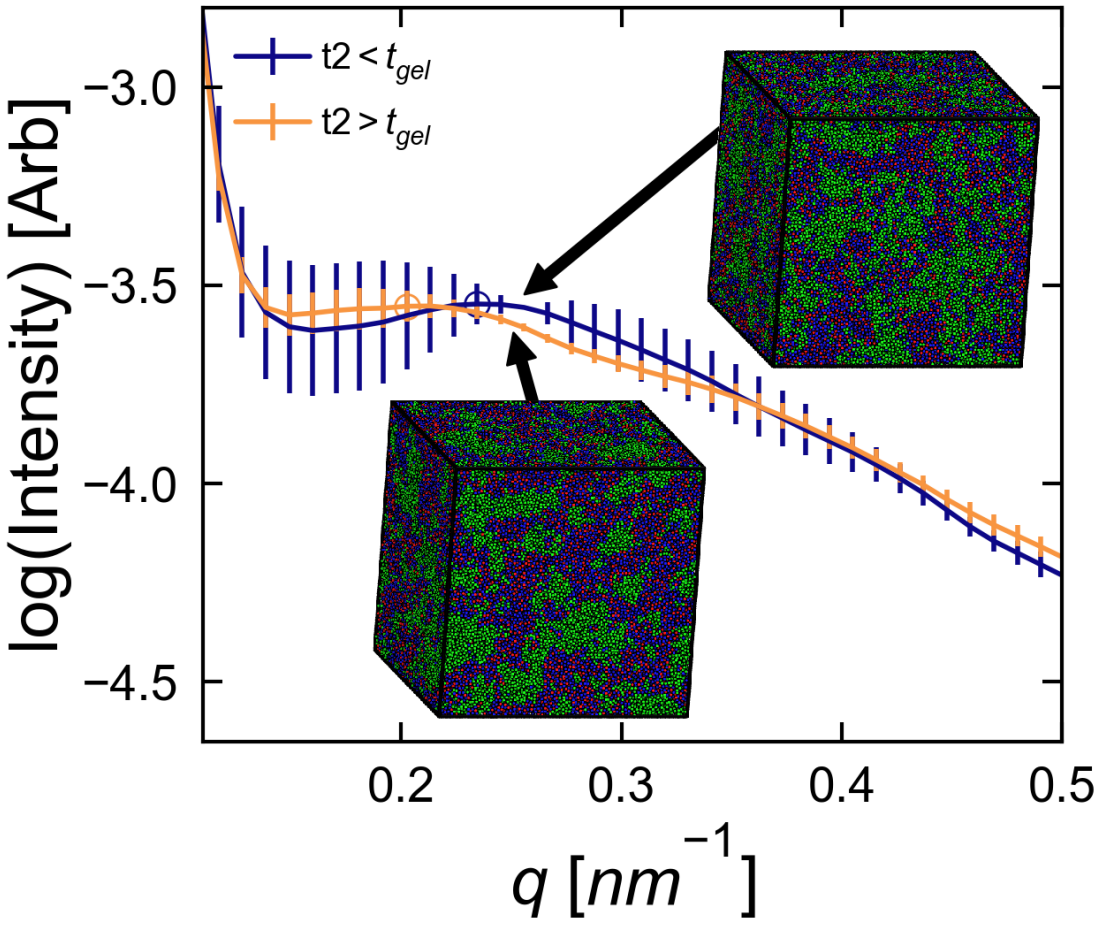

```
In [61]: from scipy import stats
         from scipy import interpolate
         fig = plt.figure(dpi=200)
```

```
ax1 = fig.add_subplot(111)
left, bottom, width, height = [0.34, 0.26, 0.3, 0.3]#[0.6, 0.57, 0.3, 0.3]#max t2
ax2 = fig.add_axes([left, bottom, width, height])
ax2.axis('off')
left, bottom, width, height = [0.61, 0.59, 0.3, 0.3]#[0.23, 0.27, 0.3, 0.3]#min t2
ax3 = fig.add_axes([left, bottom, width, height])
ax3.axis('off')
df_filtered = df[(df.activation_energy==3.0)&
                 (df.stop_after_percent==100.0)&
                 (df.t1==1.05e5)&
                 (df.E_factor==1.0)&
                 (df.T2==1.2)&
                 (((df.t2==2005001)&(df.t_Final==3e7))|
                  ((df.t2==9505001)&(df.t_Final==1.0e7)))&
                 (df.n_particles==4e5)&
                 (df.profile=='ramp_up_and_down')]
#print(df_filtered.profile)
df_sorted = df_filtered.sort_values('t2')
print(df_sorted.t2.values)
print('min t2',df_sorted.t2[3])#.min())
print('max t2',df_sorted.t2.max())
t2_min = 2005001#df_sorted.t2[3]#.min()
t2_max = 9505001#df_sorted.t2.max()#df_sorted.t2.max()
colors = plt.cm.plasma(np.linspace(0,0.75,len(df_sorted.groupby('t2'))))

df_gp = df_sorted.groupby('t2')
for i,(key,df_grp) in enumerate(df_gp):
    qs_all_trials = []
    Is_all_trials = []
    box_len = df_grp.Lx.mean()
    cure_percent = df_grp.cure_percent.mean()
    #print('cure_percent',cure_percent)
    if key ==t2_max  or key==t2_min:
        #print('key',key)
        for jobid in df_grp.index:
            job=project.open_job(id=jobid)
            if job.isfile('out.log') and ('gel_point' in job.document) :
                #print(job,key)
                gel_point = job.document['gel_point']
A=job.sp.num_a*job.sp.n_mul*4*job.sp.percent_bonds_per_step/100/job.sp.bond_period
                if job.isfile('diffract_type_2/asq.txt'):
                    data = np.genfromtxt(job.fn('diffract_type_2/asq.txt'))
                    qs=data[:,0]
                    Is=data[:,1]
                    qs_all_trials.append(qs)
                    Is_all_trials.append(Is)
                    if job.isfile('final_snapshot.png'):
                        im = plt.imread(job.fn('final_snapshot.png'))
                        if key ==t2_max:
                            #ax2.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax2.imshow(im)
                        if key ==t2_min:
                            #ax3.set_title('t2 : {:.1e} $\\Delta$
t'.format(key),fontsize=15)
                            ax3.imshow(im)
                    #print('Is',Is)
                else:
                    raise FileNotFoundError('Cannot find diffract_type_2/asq.txt
for'+job)
            else:
                print(job,' not completed!')
    if len(Is_all_trials)>0:
        mean_qs = np.mean(qs_all_trials,axis=0)
        mean_Is = np.mean(Is_all_trials,axis=0)
        if key<gel_point:
            label='t2'+'$< t_{gel}$'
        else:
```

```
                label='t2'+'$> t_{gel}$'
            ax1.errorbar(mean_qs,
                         mean_Is,
                         yerr=stats.sem(Is_all_trials,axis=0),
                         linewidth=2.0,
                         zorder=1,
                         color=colors[i],
                        label=label)#'t2 : {:.1e} $\\Delta$ t'.format(key,
                          #
        cure_percent/100))
            if True:
                qms=[]
                Ims=[]
                for qs_t,Is_t in zip(qs_all_trials,Is_all_trials):
                    first_peak_q,first_peak_i = get_highest_maxima(box_len,qs_t,Is_t)
                    qms.append(first_peak_q)
                    Ims.append(first_peak_i)
                #print(qms)
                fn = interpolate.interp1d(mean_qs,mean_Is,kind='cubic')
                first_peak_q=np.mean(qms)
                first_peak_i=fn(first_peak_q)
            else:
                first_peak_q,first_peak_i = get_highest_maxima(box_len,
                                                               mean_qs,
                                                               nmean_Is)
            print(first_peak_q,first_peak_i)
            if first_peak_q is None:
                #print(q_half_length)
                fn = interpolate.interp1d(qs,Is,kind='cubic')
                first_peak_q=q_half_length
                first_peak_i=fn(first_peak_q)
            if first_peak_q >0.8:
                fn = interpolate.interp1d(qs,Is,kind='cubic')
                first_peak_q=q_half_length
                first_peak_i=0#fn(first_peak_q)
            ax1.scatter(first_peak_q,
                   first_peak_i,
                   color=colors[i],
                    zorder=0,
                        s=100,
                        facecolor='w',
                    marker='o')

        ax1.set_xlabel(r"$q$ [$nm^{-1}$]")
        ax1.set_ylabel("log(Intensity) [Arb]")
        ax1.set_xlim(0.11,0.5)
        ax1.set_ylim(-4.65,-2.8)
        ax1.legend(fontsize=15,loc='upper left')
        ax1.annotate('', xy=(0.25, -3.6), xytext=(0.27, -4.0),arrowprops=dict(facecolor='black',
        shrink=0.05))
        ax1.annotate('', xy=(0.25, -3.55), xytext=(0.37, -3.),arrowprops=dict(facecolor='black',
        shrink=0.05))
        savefig(plt,
               'sensitivity_analysis_t2_N_4e5',
               'sf_vertical.pdf')
```

```
[ 2005001.  2005001.  2005001.  9505001.  9505001.  9505001.]
min t2 9505001.0
max t2 9505001.0
a52a2aacf366f4bfdb526c3c90a6e553  not completed!
0.234460668949 -3.5471132590244476
0.202488759547 -3.5527970868063505


/Users/stephenthomas/miniconda3/envs/dybond/lib/python3.5/site-
packages/matplotlib/figure.py:1743: UserWarning: This figure includes Axes that are
```

```
not compatible with tight_layout, so its results might be incorrect.
  warnings.warn("This figure includes Axes that are not "
```



F.4 Common Code for All Chapters

```
import cme_utils
from cme_utils.analyze import autocorr
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


def get_split_quench_job_msd(job,prop_name):
    times = []
    prop_vals = []
    qTs=[]
    if job.isfile('msd.log'):
        log_path = job.fn('msd.log')
        data = np.genfromtxt(log_path, names=True)
        PROP_NAME =prop_name
        prop_values = data[PROP_NAME]#'pair_lj_energy']
        time_steps = data['timestep']
        len_prof = len(job.sp.quench_temp_prof)
        for i in range(0,len_prof,2):
            current_point = job.sp.quench_temp_prof[i]
            next_point = job.sp.quench_temp_prof[i+1]
            start_time = current_point[0]
            end_time = next_point[0]
            if current_point[1]!=next_point[1]:
                print('WARNING!_Detected_a_non_isothermal_step')
            target_T = current_point[1]
            indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
            start_index = indices[0][0]
            end_index = indices[0][-1]
            sliced_ts = time_steps[start_index:end_index+1]
            sliced_prop_vals = prop_values[start_index:end_index+1]
            times.append(sliced_ts)
            prop_vals.append(sliced_prop_vals)
            qTs.append(target_T)
    return times,prop_vals,qTs

def _get_decorrelation_time(prop_values,
                            time_steps):
    t = time_steps - time_steps[0]
    dt = t[1] - t[0]
    acorr = autocorr.autocorr1D(prop_values)
    for acorr_i in range(len(acorr)):
        if acorr[acorr_i]<0:
            break
    lags = [i*dt for i in range(len(acorr))]

    decorrelation_time = int(lags[acorr_i])
    if decorrelation_time == 0:
```

```
        decorrelation_time = 1
    decorrelation_stride = int(decorrelation_time/dt)
    nsamples = (int(t[-1])-t[0])/decorrelation_time
    temps = "There are %.5e steps, (" % t[-1]
    temps = temps + "%d" % int(t[-1])
    temps = temps + " frames)\n"
    temps = temps + "You can start sampling at t=%.5e" % t[0]
    temps = temps + " (frame %d)" % int(t[0] )
    temps = temps + " for %d samples\n" % nsamples
    temps = temps + "Because the autocorrelation time is %.5e" % lags[acorr_i]
    temps = temps + " (%d frames)\n" % int(lags[acorr_i])
    #print(temps)
    return decorrelation_time, decorrelation_stride

def get_mean_and_std_from_time_step(job, time_steps, prop_values,start_t):
    start_i = np.where(time_steps >= start_t)[0]
    if len(start_i) >0:
        start_i=start_i[0]
    else:
        start_i = 0

    if start_i < len(time_steps):
        independent_vals_i = np.arange(start_i, len(prop_values)-1, 1)
        independent_vals = prop_values[independent_vals_i]
        #print(independent_vals)
        mean=np.mean(independent_vals)
        std=np.std(independent_vals)
    else:
        print('the {} values given have not reached equilibrium.'.format(prop))
        mean = None
        std = None
    return mean, std

def get_mean_and_std(job, time_steps, prop_values,pe,mean_from_second_half=False):
    if mean_from_second_half:
        start_i = int(len(time_steps)*0.75)
        start_t = time_steps[start_i]
    else:
        start_i, start_t = autocorr.find_equilibrated_window(time_steps, pe)

    if start_i < len(time_steps):
        decorrelation_time, decorrelation_stride = _get_decorrelation_time(prop_values[start_i:],
                                                                           time_steps[start_i:])
        #print('decorrelation time:',decorrelation_time)
        independent_vals_i = np.arange(start_i, len(prop_values)-1, decorrelation_stride)
        independent_vals = prop_values[independent_vals_i]
        #print(independent_vals)
        mean=np.mean(independent_vals)
        std=np.std(independent_vals)
    else:
        print('the {} values for {} given have not reached equilibrium.'.format(job,prop))
```

```
        mean = None
        std = None
    return mean, std



def get_mean_and_std_from_log(job, prop):
    if job.isfile('out.log'):
        log_path = job.fn('out.log')
        data = np.genfromtxt(log_path, names=True)
        prop_values = data[prop]
        time_steps = data['timestep']
        start_i, start_t = autocorr.find_equilibrated_window(time_steps, prop_values)
        if start_i < len(time_steps):
            decorrelation_time, decorrelation_stride = _get_decorrelation_time(prop_values[start_i:],
                                                                               time_steps[start_i:])
            independent_vals_i = np.arange(start_i, len(prop_values)-1, decorrelation_stride)
            independent_vals = prop_values[independent_vals_i]
            #print(independent_vals)
            mean=np.mean(independent_vals)
            std=np.std(independent_vals)
        else:
            print('the_{}_values_given_have_not_reached_equilibrium.'.format(prop))
            mean = None
            std = None
    else:
        print('could_not_find_log_file_for_{}'.format(job))
        mean=None
        std=None
    #print(mean)
    return mean, std


def plot_equilibriation(df_filtered,
                        project,
                        prop_name,
                        draw_decorrelated_samples=False,
                        draw_equilibrium_window=True,
                        mean_from_second_half=False):
    df_sorted = df_filtered.sort_values(by=['quench_T'])
    df_grouped = df_sorted.groupby('quench_T')


    quenchTs=[]
    mean_vols=[]
    vol_stds=[]
    colors = plt.cm.plasma(np.linspace(0,1,len(df_grouped)))
    i=0
    for name,group in df_grouped:
        time_steps_temp = []
        mean_vals_temp = []
```

```
    val_stds_temp = []
    for job_id in group.index:
#for i,job_id in enumerate(df_sorted.index):
        job = project.open_job(id=job_id)
        #print(job)
        if job.isfile('out.log'):
            log_path = job.fn('out.log')
            data = np.genfromtxt(log_path, names=True)
            PROP_NAME =prop_name
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            time_steps = data['timestep']
            if mean_from_second_half:
                start_i = int(len(time_steps)*.75)
                #print(job,'start_i',start_i,len(time_steps),time_steps)
                start_t = time_steps[start_i]
            else:
                start_i, start_t = autocorr.find_equilibrated_window(time_steps,
                                                                     data['potential_energy'])
            decorrelation_time, decorrelation_stride = _get_decorrelation_time(data\
                    ['potential_energy'][start_i:],
                                                                       time_steps[start_i:])
            #print('decorrelation_time:',decorrelation_time)
            independent_vals_i = np.arange(start_i, len(prop_values)-1, decorrelation_stride)
            independent_vals = time_steps[independent_vals_i]
            #starttime_steps.index(start_t)

            if 'quench_T' in job.sp:
                label = 'q_T:{},cure:{}'.format(job.sp.quench_T,job.sp.stop_after_percent)
                #label = 'tau:{}, tauP:{}'.format(job.sp.tau,job.sp.tauP)
            else:
                label = 'kT:{},cure:{}'.format(job.sp.kT,job.sp.stop_after_percent)
            time_steps_temp.append(time_steps)
            mean_vals_temp.append(prop_values)
        else:
            print('did_not_find_out.log_for',job)
    mean_time_steps = np.mean(time_steps_temp,axis=0)
    mean_prop_values = np.mean(mean_vals_temp,axis=0)
    plt.plot(mean_time_steps,mean_prop_values,label=label,color=colors[i],linewidth=1.0)
    i+=1
    if draw_decorrelated_samples:
        for xval in independent_vals:
            plt.axvline(x=xval,linestyle='--',linewidth=0.2)
    if draw_equilibrium_window:
        plt.plot(mean_time_steps[start_i],
                 mean_prop_values[start_i],
                 marker='*',
                 color='r',
                 markersize=10)
```

```
def get_values_for_quenchTs(df_filtered,project, prop,mean_from_second_half=False):
    df_sorted = df_filtered.sort_values(by=['quench_T'])
    df_grouped = df_sorted.groupby('quench_T')
    quenchTs=[]
    mean_vals=[]
    val_stds=[]
    for name,group in df_grouped:
        quench_Ts_temp = []
        mean_vals_temp = []
        val_stds_temp = []

        for job_id in group.index:
            #job_id = group.signac_id
            #print(name,job_id)
            job = project.open_job(id=job_id)
            #print(job)
            if job.isfile('out.log'):
                log_path = job.fn('out.log')
                data = np.genfromtxt(log_path, names=True)
                prop_value = data[prop]
                time_steps = data['timestep']
                pe = data['potential_energy']
                #print(job)
                mean,std = get_mean_and_std(job,time_steps,prop_value,pe,mean_from_second_half)
                if mean is not None:
                    quench_Ts_temp.append(job.sp.quench_T)
                    mean_vals_temp.append(mean)
                    val_stds_temp.append(std)

        quenchTs.append(np.mean(quench_Ts_temp))
        mean_vals.append(np.mean(mean_vals_temp))
        val_stds.append(np.mean(val_stds_temp))
    return quenchTs,mean_vals,val_stds


def line_intersect(m1, b1, m2, b2):
    if m1 == m2:
        print ("These lines are parallel!!!")
        return None
    x = (b2 - b1) / (m1 - m2)
    y = m1 * x + b1
    return x,y

from scipy.optimize import curve_fit
from scipy.interpolate import InterpolatedUnivariateSpline
from piecewise.regressor import piecewise #https://www.datadoghq.com/blog/engineering/piecewise-regression/
from piecewise.plotter import plot_data_with_regression

def DiBenedetto(alphas,T1,T0,inter_param):
    Tgs = []
    for alpha in alphas:
```

```
        Tg = inter_param*alpha*(T1-T0)/(1-(alpha*(1-inter_param))) +T0
        Tgs.append(Tg)
    return Tgs

def fit_Tg_to_DiBenedetto(alphas,Tgs,T1,T0=None):
    import warnings
    np.seterr(all='raise')
    plot_fit_fails=True
    inter_parm=0.5
    try:
        if T1==None and T0==None:
            smallestTg=Tgs[0]
            largestTg=Tgs[-1]
            popt, pcov = curve_fit(lambda Xs,T1,T0: DiBenedetto(Xs,T1,T0,inter_parm),
                                   alphas,Tgs,
                                   #p0=[0,0],
                                   p0=[largestTg,smallestTg],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   bounds=([0,0],[largestTg*1.5,smallestTg*1.2]))#,maxfev=200000)
        elif T1==None and T0!=None:
            popt, pcov = curve_fit(lambda Xs,T1: DiBenedetto(Xs,T1,T0,inter_parm),
                                   alphas,Tgs,
                                   #p0=[0,0],
                                   p0=[1],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   bounds=([0],[np.infty]))#,maxfev=200000)
        else:
            popt, pcov = curve_fit(lambda Xs,T0: DiBenedetto(Xs,T1,T0,inter_parm),
                                       alphas,Tgs,
                                       #p0=[0,0],
                                       p0=[0],
                                       #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                       bounds=([-np.infty],[np.infty]))#,maxfev=200000)
        #print('found fit')
    except FloatingPointError:
        print('Curve_fitting_failed(FloatingPointError)')
    except RuntimeError:
        print('Curve_fitting_failed(RuntimeError)')
    except TypeError:
        print('Curve_fitting_failed(TypeError)')
    except ValueError:
        print('Curve_fitting_failed(ValueError)')

    ydata = np.asarray(Tgs)
    if T1==None and T0==None:
        fit_ydata = DiBenedetto(alphas,*popt,inter_parm)
    elif T1==None and T0!=None:
        fit_ydata = DiBenedetto(alphas,*popt,T0,inter_parm)
    else:
        fit_ydata = DiBenedetto(alphas,T1,*popt,inter_parm)
    residuals = ydata - fit_ydata
```

```
    ss_res = np.sum(residuals**2)
    ss_tot = np.sum((ydata-np.mean(ydata))**2)
    #print('ss_res',ss_res,'ss_tot',ss_tot)
    if ss_tot == 0:
        #print('found ss_tot: 0')
        r_squared = 0
    else:
        r_squared = 1 - (ss_res / ss_tot)
    if T1==None and T0==None:
        return r_squared,fit_ydata,popt[0],inter_parm,popt[1]
    else:
        return r_squared,fit_ydata,popt[0],inter_parm#,popt[1]

def find_Tg(quenchTs, mean_vals,sap):
    print(sap)
    if True:#sap<=50.:
        use_first_deviation = False
        if use_first_deviation:
            model = piecewise(quenchTs, mean_vals)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dxdT = f.derivative(n=1)
            dx_dTs = dxdT(quenchTs)
            dev_index = np.where(np.abs(dx_dTs)>m1)[0][0]
            x=quenchTs[dev_index]
            y=mean_vals[dev_index]
        else:
            print('using derivatives')
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dxdT = f.derivative(n=1)
            d2xdT = f.derivative(n=2)
            dx_dTs = dxdT(quenchTs)
            d2x_dT2s = d2xdT(quenchTs)
            max_dx2 = np.max(d2x_dT2s)
            min_dx2 = np.min(d2x_dT2s)
            max_i = np.where(d2x_dT2s==max_dx2)[0][0]
            min_i = np.where(d2x_dT2s==min_dx2)[0][0]
            x = (quenchTs[min_i]+quenchTs[max_i])/2
            y = (mean_vals[min_i]+mean_vals[max_i])/2
    else:
        print('using line iftting')
        #plot_data_with_regression(quenchTs, mean_vals)
        model = piecewise(quenchTs, mean_vals)
        #print(model)
```

```
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)

        else:
            print('WARNING: found more or less than 2 line segments in regression!')
    return x,y

def plot_this(job,
              time_steps,
              prop_values,
              pe,
              color,
              label=None,
              normalize_by_mean=False,
              mean_from_second_half=True):
    if mean_from_second_half:
        start_i = int(len(time_steps)*.75)
        start_t = time_steps[start_i]
    else:
        start_i, start_t = autocorr.find_equilibrated_window(time_steps, pe)
    decorrelation_time, decorrelation_stride = _get_decorrelation_time(prop_values[start_i:],
                                                                       time_steps[start_i:])
    independent_vals_i = np.arange(start_i, len(prop_values)-1, decorrelation_stride)
    independent_vals = time_steps[independent_vals_i]
    indices = list(range(0,len(prop_values)))
    if len(indices) != len(prop_values):
        print('Check the length of arrays')
    if normalize_by_mean:
        mean,std = get_mean_and_std(job,time_steps,prop_values,pe)
        prop_values = prop_values/mean
        plt.axhline(y=1.0,linewidth=1.0,linestyle='--')
    plt.plot(indices,prop_values,label=label,linewidth=1,color=color)
    plt.plot(start_i,prop_values[start_i],marker='*',color='r', markersize=10)

def get_split_quench_job_property_mean_std(job,prop_name):
    means = []
    stds = []
    times = []
    temps = []
    if job.isfile('out.log'):
        log_path = job.fn('out.log')
        data = np.genfromtxt(log_path, names=True)
        PROP_NAME =prop_name
        prop_values = data[PROP_NAME]#'pair_lj_energy']
```

```
        time_steps = data['timestep']
        pe = data['potential_energy']
        print(job)
        len_prof = len(job.sp.quench_temp_prof)
        for i in range(0,len_prof,2):
            current_point = job.sp.quench_temp_prof[i]
            next_point = job.sp.quench_temp_prof[i+1]
            start_time = current_point[0]
            end_time = next_point[0]
            if current_point[1]!=next_point[1]:
                print('WARNING! Detected a non-isothermal step')
            target_T = current_point[1]
            #print(time_steps)
            #print(start_time,end_time)
            indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
            start_index = indices[0][0]
            end_index = indices[0][-1]
            sliced_ts = time_steps[start_index:end_index+1]
            sliced_prop_vals = prop_values[start_index:end_index+1]
            sliced_pe = pe[start_index:end_index+1]
            mean,std = get_mean_and_std(job,sliced_ts,sliced_prop_vals,sliced_pe)
            means.append(mean)
            stds.append(std)
            times.append((start_time,end_time))
            temps.append(target_T)
    return means,stds,times,temps

def split_log(df_filtered,
              project,
              prop_name,
              filter_temp,
              rtol=0.1,
              show_all=True,
              normalize_by_mean=False):
    df_sorted = df_filtered.sort_values(by=['quench_T'])


    for job_id in df_sorted.index:
        job = project.open_job(id=job_id)
        #print(job)
        if job.isfile('out.log'):
            log_path = job.fn('out.log')
            data = np.genfromtxt(log_path, names=True)
            PROP_NAME =prop_name
            prop_values = data[PROP_NAME]#'pair_lj_energy']
            time_steps = data['timestep']
            pe = data['potential_energy']
            print(job)
            len_prof = len(job.sp.quench_temp_prof)
            colors = plt.cm.plasma(np.linspace(1,0,len_prof/2))
            for i in range(0,len_prof,2):
```

```
                current_point = job.sp.quench_temp_prof[i]
                next_point = job.sp.quench_temp_prof[i+1]
                start_time = current_point[0]
                end_time = next_point[0]
                if current_point[1]!=next_point[1]:
                    print('WARNING! Detected a non isothermal step')
                target_T = current_point[1]
                #print(start_time,end_time)
                #print(time_steps)
                if np.isclose(target_T,filter_temp,rtol=rtol) or show_all:
                    #print(time_steps)
                    #print(start_time,end_time)
                    indices = np.where((time_steps>=start_time)&(time_steps<=end_time))
                    #print(indices)
                    start_index = indices[0][0]
                    end_index = indices[0][-1]
                    #print('start_index',start_index,'end_index',end_index)
                    #print('start_index',start_index,'end_index',end_index)
                    sliced_ts = time_steps[start_index:end_index+1]
                    sliced_prop_vals = prop_values[start_index:end_index+1]
                    sliced_pe = pe[start_index:end_index+1]
                    #print(sliced_ts)
                    #print(sliced_prop_vals)
                    label = 'T:{}'.format(target_T)
                    #print(i/2)
                    plot_this(job,
                              sliced_ts,
                              sliced_prop_vals,
                              sliced_pe,
                              colors[int(i/2)],
                              label,
                              normalize_by_mean=normalize_by_mean)

def find_Tg(quenchTs, mean_vals):
    if False:#sap<=50.:
        use_first_deviation = True
        if use_first_deviation:
            model = piecewise(quenchTs, mean_vals)
            if len(model.segments) == 2:
                lines = []
                l1 = model.segments[0]
                m1 = l1.coeffs[1]
                b1 = l1.coeffs[0]
                l2 = model.segments[1]
                m2 = l2.coeffs[1]
                b2 = l2.coeffs[0]
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dxdT = f.derivative(n=1)
            dx_dTs = dxdT(quenchTs)
            dev_index = np.where(np.abs(dx_dTs)>m1)[0][0]
            x=quenchTs[dev_index]
```

```
            y=mean_vals[dev_index]
        else:
            print('using derivatives')
            f = InterpolatedUnivariateSpline(quenchTs, mean_vals, k=2)
            dxdT = f.derivative(n=1)
            d2xdT = f.derivative(n=2)
            dx_dTs = dxdT(quenchTs)
            d2x_dT2s = d2xdT(quenchTs)
            max_dx2 = np.max(d2x_dT2s)
            min_dx2 = np.min(d2x_dT2s)
            max_i = np.where(d2x_dT2s==max_dx2)[0][0]
            min_i = np.where(d2x_dT2s==min_dx2)[0][0]
            x = (quenchTs[min_i]+quenchTs[max_i])/2
            y = (mean_vals[min_i]+mean_vals[max_i])/2
    else:
        print('using line iftting')
        #plot_data_with_regression(quenchTs, mean_vals)
        model = piecewise(quenchTs, mean_vals)
        #print(model)
        if len(model.segments) == 2:
            lines = []
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)

        else:
            print('WARNING: found {} line segments in regression! Expecting 2'.format(len(model.segments)))
    return x,y

def Fit_Diffusivity1(Ts,
                     Ds,
                     method='use_viscous_region',
                     min_D=1e-8,
                     ver=1,
                     viscous_line_index=1,
                     l1_T_bounds=[0,1],
                     l2_T_bounds=[0,1]):
    indices = np.where(Ds>min_D)#0.00000095)
    start_index = indices[0][0]
    D_As=Ds[start_index:]
    quenchTs=Ts[start_index:]
    #print('quenchTs',quenchTs)
    model = piecewise(quenchTs, D_As)
    #print(ver)
    if ver==4:
        line_vals = []
        l1Ts=Ts[(Ts>=l1_T_bounds[0])&(Ts<=l1_T_bounds[1])]
```

```
    l1Ds=Ds[(Ts>=l1_T_bounds[0])&(Ts<=l1_T_bounds[1])]
    l2Ts=Ts[(Ts>=l2_T_bounds[0])&(Ts<=l2_T_bounds[1])]
    l2Ds=Ds[(Ts>=l2_T_bounds[0])&(Ts<=l2_T_bounds[1])]
    par = np.polyfit(l1Ts, l1Ds, 1, full=True)
    m1 = par[0][0]#0-slope, 1-intercept
    b1 = par[0][1]
    xs = np.linspace(l1Ts[0],l1Ts[-1])
    ys = m1*xs+b1
    line_vals.append((xs,ys))
    par = np.polyfit(l2Ts, l2Ds, 1, full=True)
    m2 = par[0][0]#0-slope, 1-intercept
    b2 = par[0][1]
    xs = np.linspace(l2Ts[0],l2Ts[-1])
    ys = m2*xs+b2
    line_vals.append((xs,ys))
    x,y = line_intersect(m1,b1,m2,b2)
    Tg=x
    Tg_prop = y
    return Tg,Tg_prop,line_vals
elif ver==3:
    line_vals=[]
    Ts_low_i = np.where(Ts>=l1_T_bounds[0])[0]
    if len(Ts_low_i)==0:
        raise ValueError('lower_bound_for_T_fitting_of_line_1_too_low._Use_a_higher_T')
    l1_low_i = Ts_low_i[0]
    Ts_low_i = np.where(Ts>=l2_T_bounds[0])[0]
    if len(Ts_low_i)==0:
        raise ValueError('lower_bound_for_T_fitting_of_line_2_too_low._Use_a_higher_T')
    l2_low_i = Ts_low_i[0]

    Ts_high_i = np.where(Ts<=l1_T_bounds[1])[0]
    if len(Ts_high_i)==0:
        raise ValueError('upper_bound_for_T_fitting_of_line_1_too_high._Use_a_lower_T')
    l1_high_i = Ts_high_i[-1]
    Ts_high_i = np.where(Ts<=l2_T_bounds[1])[0]
    if len(Ts_high_i)==0:
        raise ValueError('upper_bound_for_T_fitting_of_line_2_too_high._Use_a_lower_T')
    print('Ts_high_i',Ts_high_i)
    l2_high_i = Ts_high_i[-1]

    l1Ts=Ts[l1_low_i:l1_high_i]
    l1Ds=Ds[l1_low_i:l1_high_i]
    print(l1_low_i,l1_high_i,l1Ts)
    l2Ts=Ts[l2_low_i:l2_high_i]
    l2Ds=Ds[l2_low_i:l2_high_i]
    print(l2_low_i,l2_high_i,l2Ts)
    par = np.polyfit(l1Ts, l1Ds, 1, full=True)
    m1 = par[0][0]#0-slope, 1-intercept
    b1 = par[0][1]
    xs = np.linspace(l1Ts[0],l1Ts[-1])
    ys = m1*xs+b1
```

```
        line_vals.append((xs,ys))

        par = np.polyfit(l2Ts, l2Ds, 1, full=True)
        m2 = par[0][0]#0-slope, 1-intercept
        b2 = par[0][1]
        xs = np.linspace(l2Ts[0],l2Ts[-1])
        ys = m2*xs+b2
        line_vals.append((xs,ys))
        if viscous_line_index==0:
            Tg = -b1/m1
            Tg_prop = 0.
        elif viscous_line_index==1:
            Tg = -b2/m2
            Tg_prop = 0.
        else:
            x,y = line_intersect(m1,b1,m2,b2)
            Tg=x
            Tg_prop = y

        return Tg,Tg_prop,line_vals
    elif ver==2:
        n_lines=len(model.segments)
        if n_lines == 0:
            raise ValueError('Found zero lines in piecewise fitting')
        lines=[]
        line_vals=[]
        for i in range(n_lines):
            line = model.segments[i]
            lines.append(line)
            xs = np.linspace(line.start_t,line.end_t)
            ys = line.coeffs[1]*xs+line.coeffs[0]
            line_vals.append((xs,ys))

        if method=='use_viscous_region':
            if n_lines>1:
                l2=lines[viscous_line_index]
            else:
                l2=lines[0]
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            Tg = -b2/m2
            Tg_prop = 0.
        else:
            Tg,Tg_prop=find_Tg(mean_vals=Ds,quenchTs=Ts)
        return Tg,Tg_prop,line_vals
    elif ver==1:
        if len(model.segments) == 2:
            l1 = model.segments[0]
            m1 = l1.coeffs[1]
            b1 = l1.coeffs[0]
            l2 = model.segments[1]
```

```
            m2 = l2.coeffs[1]
            b2 = l2.coeffs[0]
            x,y = line_intersect(m1,b1,m2,b2)
            xs1 = np.linspace(l1.start_t,l1.end_t)#np.linspace(l1.start_t,(x+(l1.end_t-l1.start_t)*0.2))
            ys1 = l1.coeffs[1]*xs1+l1.coeffs[0]
            xs2 = np.linspace(l2.start_t,l2.end_t)#np.linspace((x-(l2.end_t-l2.start_t)*0.2),l2.end_t)
            ys2 = l2.coeffs[1]*xs2+l2.coeffs[0]

            if method=='use_viscous_region':
                Tg = -b2/m2
                Tg_prop = 0.
            elif method == 'intersection':
                Tg=x
                Tg_prop=y
        else:
            print('WARNING: found {} line segments in regression!'.format(len(model.segments)))

        return Tg,Tg_prop,xs1,ys1,xs2,ys2


def Calc_Diffusivity(eq_time,
                     eq_msd,
                     fit_method='curve_fit'):
    #fit_method='curve_fit'#'power_law','poly_fit'
    if fit_method=='curve_fit':
        norm_eq_time = (eq_time-eq_time[0])
        #print(norm_eq_time,eq_msd)
        popt, pcov = curve_fit(lambda t,m,b: m*t+b ,
                                    eq_time,
                                    eq_msd,
                                    p0=[1.,0.0],
                                    bounds=([0.0,0.0],[np.infty,np.infty]))
        drdt_A = popt[0]
        m=popt[0]
        b=popt[1]

        ydata = np.asarray(eq_msd)
        fit_ydata = m*eq_time+b
        residuals = ydata - fit_ydata
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((ydata-np.mean(ydata))**2)
        #print('ss_res',ss_res,'ss_tot',ss_tot)
        if ss_tot == 0:
            #print('found ss_tot: 0')
            r_squared = 0
        else:
            r_squared = 1 - (ss_res / ss_tot)
        #print('R2:',r_squared)
    elif fit_method=='poly_fit':
        par = np.polyfit(time, msd, 1, full=True)
        drdt_A = par[0][0]#0-slope, 1-intercept
```

```
            m=par[0][0]
            b=par[0][1]
        elif fit_method=='power_law':
            popt, pcov = curve_fit(lambda t,w,x1: (w*t)**x1 ,
                                   time,
                                   msd,
                                   p0=[0.2,1.0],
                                   #p0=[1.0],
                                   #bounds=([-np.infty,-np.infty],[np.infty,np.infty])
                                   #bounds=([0],[4.0]))
                                   maxfev=2000000,
                                   bounds=([0.0,0.0],[1.0,4.0]))
            raise NotImplementedError('Diffusivity not determined')

        #calculate the diffusion coefficient
        dimensions=3
        D = drdt_A/(2*dimensions)
        return D,m,b,r_squared

    def getDiffusivities(project,
                         df_curing,
                         sortby='quench_T',
                         name='bparticles',
                         quench_time=1e7,
                        equilibriated_ts_percentage = 0.0):
        """
        returns diffusivity in units of D^2/tau where D and tau are distance and time units.
        Note that time is not in time steps.
        """
        Ts=[]
        Ds=[]
        for key,df_grp in df_curing.groupby('cooling_method'):
            if key=='quench' and quench_time is not None:
                df_filt = df_grp[(df_grp.quench_time==quench_time)]
            else:
                df_filt = df_grp
            df_sorted=df_filt.sort_values(sortby)
            for q_T,q_T_grp in df_sorted.groupby('quench_T'):
                for job_id in q_T_grp.index:
                    job = project.open_job(id=job_id)
                    if job.isfile('msd.log'):
                        log_path = job.fn('msd.log')
                        data = np.genfromtxt(log_path, names=True)
                        prop_values = data[name]#'pair_lj_energy']
                        if key=='anneal':
                            times,msds,qTs = get_split_quench_job_msd(job,name)
                            #print('split qTs from anneal',qTs)
                            for j,msd in enumerate(msds):
                                start_index = int(len(times[j])*equilibriated_ts_percentage)
                                time=times[j]*job.sp.md_dt
                                quench_T = qTs[j]
```

```
                        eq_msd = msd[start_index:]
                        eq_time = time[start_index:]
                        D_A,m,b,r_2 = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                        Ts.append(quench_T)
                        Ds.append(D_A)
                else:
                    all_time_steps = data['timestep']
                    start_index = int(len(all_time_steps)*equilibriated_ts_percentage)
                    time=all_time_steps*job.sp.md_dt
                    quench_T = job.sp.quench_T
                    eq_msd = prop_values[start_index:]
                    eq_time = time[start_index:]
                    #print(job)
                    D_A,m,b,r_2 = Calc_Diffusivity(eq_time,eq_msd,'curve_fit')
                    #if r_2 <0.9:
                    #    print(r_2,job)
                    Ts.append(quench_T)
                    Ds.append(D_A)
                break#just using the first data point in this quench_T instead of mean.
            else:
                print('msd_file_not_present_in',job)

    Ts=np.asarray(Ts)
    Ds=np.asarray(Ds)
    #Ts_sem=np.asarray(Ts_sem)
    return Ts,Ds

def savefig(plt,nbname,figname,transparent=True):
    import os
    if not os.path.exists(nbname):
        os.makedirs(nbname)
    plt.savefig(os.path.join(nbname,figname),transparent=transparent,bbox_inches='tight')
```