

A Parametric Rosetta Energy Function
Analysis with LK Peptides on SAM
Surfaces

By
Joseph Harrison Lubin

A thesis submitted to the Johns Hopkins University in conformity with the requirements of the degree of Master of Science in Engineering

Johns Hopkins University
Baltimore, Maryland
July, 2017

© Joseph Harrison Lubin, 2017.
All Rights Reserved

Abstract

While structures have been determined for many soluble proteins and an increasing number of membrane proteins, experimental structure determination methods are limited for complexes of proteins and solid surfaces. An economical alternative or complement to experimental structure determination is molecular simulation. Rosetta is a molecular simulation software suite that can model protein–surface interactions. While Rosetta has been thoroughly benchmarked on soluble protein modeling tasks, its ability to predict protein–surface interactions requires further work. In particular, the validity of the energy function is uncertain because it is a combination of independent parameters from energy functions developed separately for solution proteins and mineral surfaces. Therefore, I have assessed the performance of the RosettaSurface algorithm and tested the accuracy of its energy function by modeling the adsorption of leucine/lysine repeat peptides on methyl- and carboxy-terminated self-assembled monolayers. I investigated how RosettaSurface predictions for this system compared with experimental results, which showed that on both surfaces, LK- α peptides folded into helices and LK- β peptides held extended structures. Utilizing this model system, I performed a parametric analysis of Rosetta’s Talaris energy function and determined that the default energy function was less able to predict the extended LK- β structures, and that adjusting solvation parameters offered improved predictive accuracy. Simultaneously increasing lysine carbon hydrophilicity and methyl head group hydrophobicity yielded computational predictions most closely matching experimental results. The findings will improve RosettaSurface and other algorithms utilizing the Rosetta energy function.

Advisor: Jeffrey J. Gray

Acknowledgements

I want to express my gratitude to my advisor, Professor Jeffrey J. Gray, for giving a MSE student with no computational background the opportunity to join this excellent group, to learn a new approach to biology, to take over a room for my 3D printer, and to be the teaching assistant for his class. He has cultivated an excellent lab culture in which I have grown immensely and thoroughly enjoyed myself. The mixture of patience, encouragement, and leadership by example he provided have been guides both in science and out in the world. Speaking of ‘out in the world,’ thanks again for the bike, Jeff! Thank you also for helping me continue to grow and advance academically and professionally.

My gratitude also goes to Professor Rebecca Schulman, for reading and refining my thesis and for supporting my pursuit of a PhD. I very much enjoyed your class, and am proud of the project you guided my team to complete. I also think I may very well have been without a PhD acceptance this coming year, if not for your candid and constructive feedback on my application materials.

To the Gray Lab as a group: it has been a great pleasure getting to know you and work with you, and a great pride to stand among you. Though I hadn’t thought of myself as a computational guy when I arrived at Johns Hopkins, I’m so glad I wound up in this group. I advise new students who are picking out labs to join that they should consider not only what they’ll be doing, but who they’ll be with; instead of just doing a job, I’m part of a collaboration with a group of friends who help and motivate me in my work. I can’t imagine any alternative way my MSE could have gone that would have been preferable to this one.

Much of my growth in the lab can be attributed to Dr. Michael Pacella, my graduate student mentor. While in the busy final stretch of earning that title, he patiently guided me through the

steep learning curve of getting started with Rosetta and taught me what I'd need to get up to speed on the projects presented herein. Most of this thesis is built on groundwork he laid, and he has generously allowed me the use of excerpts from his own thesis in several sections as noted. I look forward to the publication of my LK/SAM work. Live long and prosper!

I have been fortunate to meet many great mentors and friends in the lab, including Shourya Sonkar Roy Burman and Dr. Jason Labonte, my officemates, and Jeliazko Jeliazkov, who was TA when I took Jeff's class, and Rebecca Alford. Each helped me in learning Rosetta, Python, and various other scientific topics, and helped me to be a better TA and a more prepared PhD student. Shourya further helped me get back to the bench with the crystallization experiments, and headed up the CAPRI team. Kayvon Tabrizi worked on single amino acid surface docking, which provided some insight for my project. I also want to thank Rebecca for reviewing my PhD application materials, and helping nudge me to write, write, write. Thanks again to her, Shourya, Jeliazko, and Mike for taking the time to review this thesis and help me in improving it. I am grateful to Naireeta Biswas and Elizabeth Lagesse, who helped convince me to join the group in the first place, and to Xiaotong Zuo who blazed the trail for MSE students in the group. A shout out to Matt Mulqueen and Sergey Lyskov, who do so much behind the scenes for everyone. Cheers also to Dr. Nicholas Marze, Xiyao Long, Dr. Brian Weitzner, Morgan Nance, Kathy Wang, Mayukh Chakrabarti, Nikhil Shah, Dr. Julia Koehler Leman, and Paige Stanley.

Thanks also go to the many friends I have made outside of the lab, who helped me make the transition back to being a student again after four years, and who have celebrated the good times and lamented the bad ones with me. Among them, David Ryoo, Anjishnu Sarkar, Preetika Karnal, Swetha Kumar, Gayatri Dhara, Sean Ponce, who I met my first time visiting campus, and who showed me to the financial aid office so that I could afford to be here, Anjali Nelliath, Debonil

Maity, Tanvi Shroff, Jimmy Kirsch, Ilyssa Haar, Lanlan Ji, and Jen Dailey, with whom I 3D printed in chocolate, and many others. A special thanks to Linna Wang, my landlady and surrogate mother while I have been in Baltimore, who beyond a great place to live at a great price, provided me with many excellent home-cooked meals, rides to various places, and great conversations about life. Cheers also to my housemates, William Hua, Tu Nguyen, Eric Xiaoxiao Liu, Sam Tiansheng Wang, and Da Chen.

A special acknowledgement goes to the D&D crew, Mike, Shourya, David, Nick, Jared Labonte, and especially to our DM, Jason. Never have I seen a DM take such care to make the game thorough and immersive and fun. It's been a blast, and I look forward to continuing the campaign remotely. Though this thesis has a title reflective of the work I did, I might humorously entitle this chapter of my life, "Thesis and the Minotaur".

All results shown in this thesis were produced from simulations run on the Maryland Advanced Research Computing Center (MARCC). Thanks go to Jaime Combariza and the other good folks who keep the processors churning. Thanks also to the administrators that keep the ChemBE department running, particularly Kailey Dille, Jenny Seat, Porscha Reid, Lucy Raybon, and Deborah Norris. I also wish to thank the members of the Rosetta Commons, especially the members I was fortunate to meet at Winter RosettaCon, especially my roommate, Yuval Sadan, who was the first person outside the Gray Lab to look over my data.

Four years out in 'the real world' feels like a long time, and it wasn't easy to make the transition and come back to school, but it has been so worth it. I want to extend my thanks to Chris Hughes, my manager at EN Engineering, to Professor Thatcher Root, and to Professor Brian Pflieger, in whose lab I found the seeds of the passion that motivated my return to science, for their support in making my return to academia. Chris has continued to support my academic

advancement with recommendations to PhD programs. At Johns Hopkins, I was also helped in becoming a student again by Christine Kavanaugh.

The greatest thanks go to my family, for giving me the foundation on which all of this has been built. My parents, Patti and Barry Lubin, who have given me golden examples of love, wisdom, patience, and diligence to which I will ever aspire. I hope always to make them proud. Sarah, my younger sister, has my undying admiration for her dogged strength and loyalty, and my thanks for keeping me company as I grew up. And Becca, my little sister, has taught me so much about what it means to be a friend, a teacher, and a good person. I love you all.

As I continue on towards earning a PhD at Rutgers, I reflect that this experience has helped shape the course of my future. To everyone who has played a part in that, whether named here or not, thank you.

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
Table of Contents.....	vi
List of Figures.....	viii
Chapter 1: Introduction.....	1
1A: Molecular modeling at solution-surface interfaces.....	1
1B: LK Peptides and SAM Surfaces.....	4
Chapter 2: Methods.....	11
2A: Generation of Starting Structures.....	11
2B: Sampling Protocol.....	12
2C: Structure Analysis.....	12
Chapter 3: Baseline Results, Unmodified Rosetta Score Function.....	14
3A: Baseline Peptides Adsorbed on Surfaces, Talaris2013.....	14
3B: Solution Models, Talaris2013.....	22
3C: REF2015.....	25
Chapter 4: Results, Alternate Peptides.....	27
4A: Alternate Peptide Termini.....	27
4B: Alternate Peptide Sizes.....	31
Chapter 5: Results, Score Weight Variations.....	34

5A: Single Physical Parameter Score Weight Variations	34
5B: Single Knowledge-Based Parameter Score Weight Variations.....	40
5C: Paired Physical Parameter Score Weight Variation	42
Chapter 6: Results, Atom Parameter Variations	44
6A: Lysine Layout and LK_DGFFREE Cross Plot	44
6B: Best Discrimination Case	49
6C: Other Atom Property Variations	53
Chapter 7: Conclusions	57
Appendix 1: Table of Altered Parameters	62
Appendix 2: Code Development.....	64
Submission Script Generator.....	64
Top Decoy Selection Script.....	65
Analysis and Plotting Script.....	66
Appendix 2A: Example MARCC Submission Script.....	68
Appendix 2B: Example Flags File.....	69
Appendix 2C: Submission Script Generator.....	70
Appendix 2D: Decoy Collection Script	83
Appendix 2E: Analysis and Plotting Script	91
References.....	120
CV.....	129

List of Figures

Figure 1. LK peptides partition hydrophobic and hydrophilic residues by adopting specific secondary structures. Top: LK- α partitions hydrophobic and hydrophilic residues by adopting an α -helical structure. Bottom: LK- β partitions hydrophobic and hydrophilic residues by adopting a β -strand structure.	4
Figure 2. Self-assembled monolayer (SAM) surfaces. Left: Hydrophilic carboxylic acid-terminated SAM. Right: Hydrophobic methyl-terminated SAM.	5
Figure 3. SFGS results obtained by Castner and coworkers for the Amide I region ⁵¹ . Peaks at 1655 cm-1 indicate the presence of α -helical structures. The top two spectra are for LK- α , while the bottom two are for LK- β . The top spectrum in each pair is for the hydrophilic SAM surface, while the bottom in each pair is for the hydrophobic SAM surface.	7
Figure 4. SFGS results obtained by Castner and coworkers for the CH region ⁵¹ . The differences in spectra indicate that the leucines are oriented away from the hydrophilic surface for both peptides, while they are oriented toward the hydrophobic surface for both peptides. On the left are spectra obtained for the hydrophilic surface, in order from top to bottom: the hydrophilic SAM surface without peptides, the surface with LK- α , and the surface with LK- β . On the right are spectra obtained for the hydrophobic surface, with LK- α on the top and LK- β on the bottom.	8
Figure 5. Low energy structures of LK peptides adsorbed on hydrophobic and hydrophilic SAMs. Top left: LK- α on hydrophilic SAM. Top right: LK- β on hydrophilic SAM. Bottom left: LK- α on hydrophobic SAM. Bottom right: LK- β on hydrophobic SAM. All images were obtained from simulations with the standard Talaris2013 score function.	15
Figure 6. Secondary structure distribution of top 100 low-scoring decoys from simulations using RosettaSurface defaults. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue.	16
Figure 7. Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys from simulations using RosettaSurface defaults.	17
Figure 8. Score vs. helicity distribution of the top 100 decoys from simulations using RosettaSurface defaults.	18
Figure 9. Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys from simulations using RosettaSurface defaults.	20
Figure 10. Helicity, position, and score results for the solution-state structures obtained using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys.	23
Figure 11. Helicity, position, and score results for REF2015 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.	25
Figure 12. Peptide termini with side chains not shown. Asterisks (*) are placed above the α -carbon of the terminal residue. Capped peptides used by Latour and coworkers ³⁹ were acetylated at the	

N-terminus and amidated at the C-terminus, while uncapped peptides had unmodified termini. The peptides used by Castner *et al.*⁵¹, and which I used in the publication, were N-acetylated, but unmodified at the C-terminus. 27

Figure 13. Helicity, position, and score results for *uncapped* 14-mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys. 28

Figure 14. Helicity, position, and score results for *N- and C-capped* 14-mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys. 29

Figure 15. Helicity, position, and score results for uncapped 7mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys. 31

Figure 16. Helicity, position, and score results for N- and C-capped 7mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys. 32

Figure 17. LK peptide helicity versus the implicit solvation energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.75) of the FA_SOL term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. 34

Figure 18. LK peptide helicity versus the attractive Lennard-Jones energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.8) of the FA_ATR term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. 36

Figure 19. LK peptide helicity versus the repulsive Lennard-Jones energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.44) of the FA_REP term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. 37

Figure 20. LK peptide helicity versus the electrostatic energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.7) of the FA_ELEC term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.	38
Figure 21. LK peptide helicity versus the short-range backbone-backbone hydrogen bonding energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (1.17) of the HBOND_SR_BB term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.	39
Figure 22. LK peptide helicity versus the probability of amino acid based on backbone dihedrals score term weighting (P_{AA_PP}). Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.32) of the P_{AA_PP} term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.	40
Figure 23. LK peptide helicity versus the side-chain dihedral probability score term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.56) of the FA_DUN term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.	41
Figure 24. LK peptide helicity heat maps with implicit solvation and Lennard-Jones attraction energy score weights. The default Talaris2013 value is (0.75, 0.8). Top left: The helicity of LK- α . Top right: The helicity of LK- β . Bottom: The point-by-point difference between the two upper plots. Each black point represents the average helicity of the 100 lowest energy structures out of 10,000 generated structures. This simulation used uncapped 14-mer peptides on the hydrophobic SAM surface.	43
Figure 25. A low energy structure of LK- β on a hydrophobic SAM with multiple lysine side chains lying flat against the surface.	44
Figure 26. LK peptide helicity heat maps with varied surface methyl head group hydrophobicity and lysine C δ and C ϵ hydrophilicity. Top left: The helicity of LK- α . Top right: The helicity of LK- β . Bottom: The point-by-point difference between the two upper plots. Each black point represents the average helicity of the 100 lowest energy structures out of 10,000 generated structures. The white point shows the default LK_DGFREE values.	47
Figure 30. Secondary structure distribution of top 100 low-scoring decoys from the maximum secondary structure discrimination case. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue.	50
Figure 31. Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys from the maximum secondary structure discrimination case.	50
Figure 32. Score vs. helicity distribution of the top 100 decoys from the maximum secondary structure discrimination case.	51
Figure 33. Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys from the maximum secondary structure discrimination case.	51
Figure 27. LK peptide helicity versus the hydrophobic surface methyl Lennard-Jones well depth atom property value. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (0.1811) of the LJ_WDEPTH property of the surface CH3 atom types in Rosetta. Each point represents the	

average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. 53

Figure 28. LK peptide helicity versus the hydrophobic surface methyl Lazaridis-Karplus solvation volume atom property value. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (30.0) of the LJ_WDEPTH property of the surface CH3 atom types in Rosetta. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. .. 55

Figure 29. LK peptide helicity versus the two-body interaction cutoff range used for calculating Rosetta energies. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (5) of the detection range in Rosetta. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures. 56

Chapter 1: Introduction

I performed my research in tight collaboration with my graduate student mentor, Dr. Michael S. Pacella, and we both made significant intellectual contributions to the work. Elements of our draft research manuscript were integrated into both his doctoral dissertation and my thesis here. This chapter has elements in common with chapters 1 and 4 of his dissertation, “Modeling and Design of Peptides to Control Biomineral Nucleation and Growth,” Johns Hopkins University, Baltimore, Maryland, Copyright 2017 Michael Steven Pacella. Adapted portions are reproduced with permission.

1A: Molecular modeling at solution-surface interfaces

Biological processes, including the formation of bones and shells, rely on the interaction of proteins with solid inorganic surfaces. While ordered structures are commonplace in nature, human ability to exert control at a molecular level is a recent development. Improved understanding of natural surface-active protein systems has enabled the development of protein-based therapies to treat hypophosphatasia¹, kidney stones²⁻⁴, and amelogenesis imperfecta⁵. Discovering nature’s design principles for surface-active proteins has additional applications to molecular manufacturing⁶, including the development of novel approaches to nanoengineering and biomimetic synthesis of custom nano-structured materials. Such bioinspired methods have been employed to grow designed structures including SiO₂, BaCO₃⁷, silica, and titania⁸.

Despite these advancements, relatively little is understood about the structural differences between how proteins act in solution and how they act at solution-solid interfaces. Knowledge is limited because current experimental methods provide only sparse or low-resolution data on atomic structure at solid interfaces. There are two methods considered to be the best available. The first is solid state NMR⁹ (ssNMR), which accurately determines distances between specific atoms, but which has the limitation of being very cost intensive, and difficult to perform on substrates lacking phosphorus or carbon. The second is single molecule force spectroscopy¹⁰ (SMFS), including techniques such as atomic force microscopy^{5,10-17} (AFM), which can determine the ΔG

energy of protein adsorption, yielding thermodynamic information about protein-surface binding, but does not provide sufficient imaging resolution to determine molecular structures¹⁸.

Molecular simulation can connect limited experimental information to the structure of protein molecules adsorbed on solid surfaces, matching structural constraints from ssNMR¹⁹ or comparing equilibrium binding energies from AFM to predicted structures. Three approaches have been applied to modeling protein-surface interactions. Quantum Mechanics (QM) explicitly models electronic degrees of freedom (DOFs), offering the greatest precision for small molecule systems that can guide development of force fields²⁰⁻²³, but it does not scale well to large molecules. Molecular Dynamics (MD) allows for calculation of thermodynamic and kinetic constants^{24,25} and handles DOFs related to solvent, but like QM is limited to smaller systems and time scales, or to coarse-grained models, due to its computational requirements. Monte Carlo-plus-minimization (MCM) is less able to capture thermodynamic ensembles, instead focusing on finding the global minimum potential²⁶. It is therefore quicker and less computationally demanding, allowing for simulation of larger scale systems, and sampling of alternative peptide sequences to produce optimal designs^{27,28}.

Rosetta is an MCM software suite that functions by making random perturbations to the system, such as altering dihedral angles in a peptide's backbone or side chains, then accepting or rejecting each move based on scores that it calculates for the system before and after perturbation. An important difference between Rosetta and MD methods is that while MD tends to explicitly model water and salt molecules in the system, Rosetta models interactions with the solvent using a Gaussian exclusion implicit solvation model²⁹ with the effects of the solution abstracted into terms in the energy function. The scores are calculated using an energy function with separate parameters representing physical properties, such as attractive and repulsive Van der Waals forces

between nearby atoms in the system, and the energy of solvation, and also statistical, or knowledge-based parameters evaluating consistency with structural trends observed in the Protein Data Bank. Each parameter has a weight assigned to it, scaling its contribution to the total score of the system, and weights have been optimized for soluble proteins³⁰⁻³². The final score for the system is a weighted sum of all terms within the score function, and represents the energy of the equilibrium state of the system.

Computational force field models can now predict many protein structures in solution, in some cases to sub-angstrom accuracy³³⁻³⁸. Though energy functions have been refined considerably for solution structures, no such optimized energy functions exist for surface-bound proteins. An additional challenge arises for Rosetta: while the implicit water model greatly expedites Rosetta simulations by simplifying the system model and reducing the amount of interactions for which scores must be calculated, its assumption of a system environment of bulk water at pH 7 may not be valid at solid interfaces. One way to produce an energy function for this system is to linearly combine independent parameters from protein and mineral energy functions, using standard mixing rules; however, without benchmarking simulation results against experimental measurements of peptide-surface interactions, the validity of the resulting force field remains uncertain³⁹.

Recent work on benchmarking of computational methods to study biomineralization processes has focused on the development of energy functions to reproduce experimental results on bulk and interfacial properties of minerals in aqueous ionic solutions^{21,40-45}. Work has also been done to benchmark host-guest peptides interacting with hydrophobic and hydrophilic self-assembled monolayers⁴⁵.

1B: LK Peptides and SAM Surfaces

Previous work has explored structure prediction of proteins and peptides on flat mineral surfaces using Rosetta^{5,19,46}, and has refined a surface docking protocol, RosettaSurface⁴⁷. A recent benchmark study⁴⁸ provides insight into the accuracy of the RosettaSurface energy function for reproducing experimental results at length and time scales relevant to biomineral-peptide interactions. However, the benchmark set is not ideal for developing and refining an energy function. First, generating and analyzing results for the entire benchmark set is computationally intensive. Second, the comparison between computational results and experimental data is a binary success or failure, which is too coarse-grained for this purpose. To fine-tune the RosettaSurface

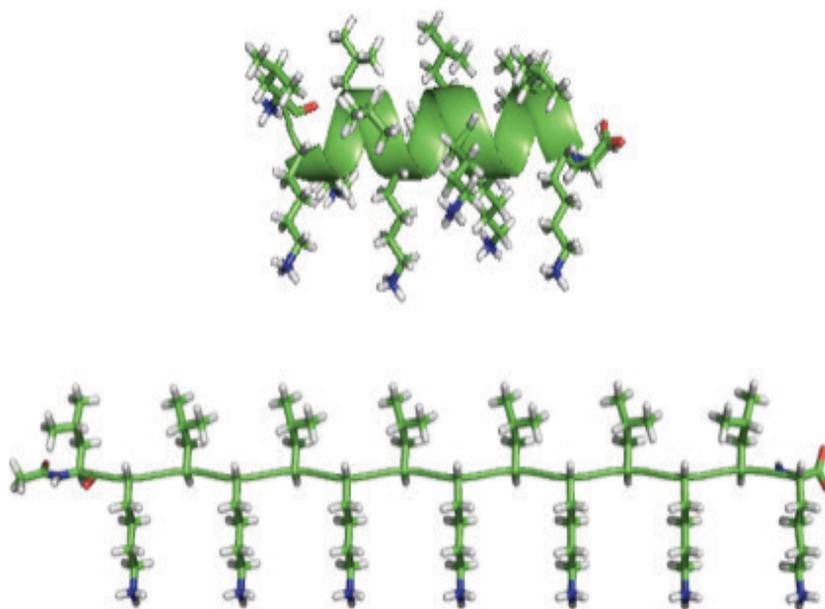


Figure 1. LK peptides partition hydrophobic and hydrophilic residues by adopting specific secondary structures. Top: LK- α partitions hydrophobic and hydrophilic residues by adopting an α -helical structure. Bottom: LK- β partitions hydrophobic and hydrophilic residues by adopting a β -strand structure.

energy function, I sought a more quantitative measure of accuracy of the RosettaSurface energy function. Thus I used the adsorption of two amphiphilic leucine/lysine repeat peptides (LK peptides), referred to as LK- α and LK- β , on hydrophobic and hydrophilic alkane-thiol self-assembled monolayers (SAM) as the model system.

LK- α and LK- β have hydrophobic periodicities of 3.5 and 2 residues, respectively. As a result, LK- α can partition its hydrophobic leucine residues and hydrophilic lysine residues by adopting an α -helical secondary structure, and LK- β can partition hydrophobic and hydrophilic residues by adopting an extended secondary structure (Figure 1). DeGrado and Lear invented these peptides in 1985 and investigated the structure at the air/water interface using circular dichroism (CD). They discovered that each LK peptide adopted the anticipated secondary structure, presumably in order to present leucines towards the air at the air/water interface⁴⁹. Moving from gas/liquid to solid/liquid interfaces, Somorjai and coworkers investigated LK peptide adsorption onto hydrophobic polystyrene and hydrophilic silica surfaces using atomic force microscopy (AFM), quartz crystal microbalance (QCM), and sum frequency generation spectroscopy

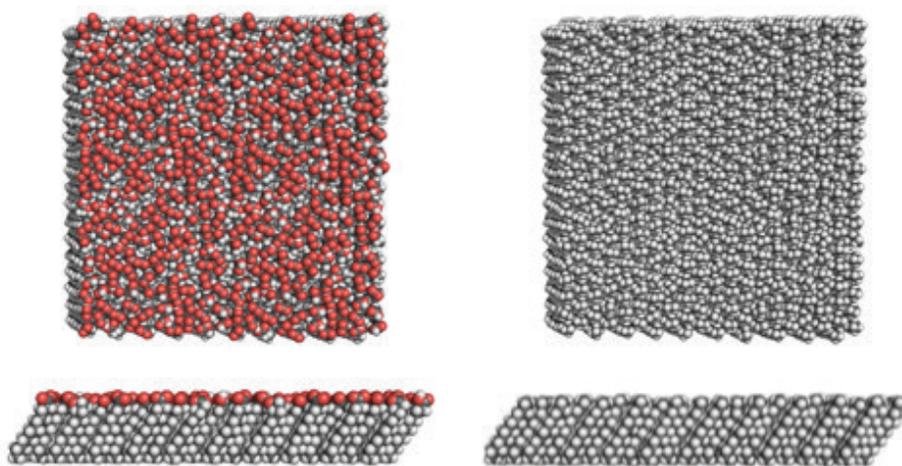


Figure 2. Self-assembled monolayer (SAM) surfaces. Left: Hydrophilic carboxylic acid-terminated SAM. Right: Hydrophobic methyl-terminated SAM.

(SFGS)⁵⁰. More recently, Castner and coworkers studied the structure of LK peptides adsorbed on a variety of material surfaces, including hydrophobic methyl-terminated and hydrophilic carboxylic acid-terminated alkane-thiol self-assembled monolayers (SAMs) (Figure 2)⁵¹⁻⁵⁴. Their experiments, using SFGS and near-edge X-ray absorption fine structure spectroscopy (NEXAFS), provide appropriate structural information to benchmark a structure prediction algorithm like RosettaSurface, including the predominant secondary structure of the peptides, and their orientation on the surface.

The experimental results presented by Castner and coworkers provide information on both LK peptide secondary structure and the orientation of the amino acid side chains relative to the SAM surface. In the SFG spectra of LK- α on both the methyl-terminated and carboxy-terminated SAMs, an amide I signal near 1655 cm^{-1} , characteristic of intact α -helical structures, suggests that LK- α adopts an α -helical structure on both the hydrophobic and hydrophilic SAMs (Figure 3, top). In the SFG spectra of LK- β on both the methyl-terminated and carboxy-terminated SAMs, no

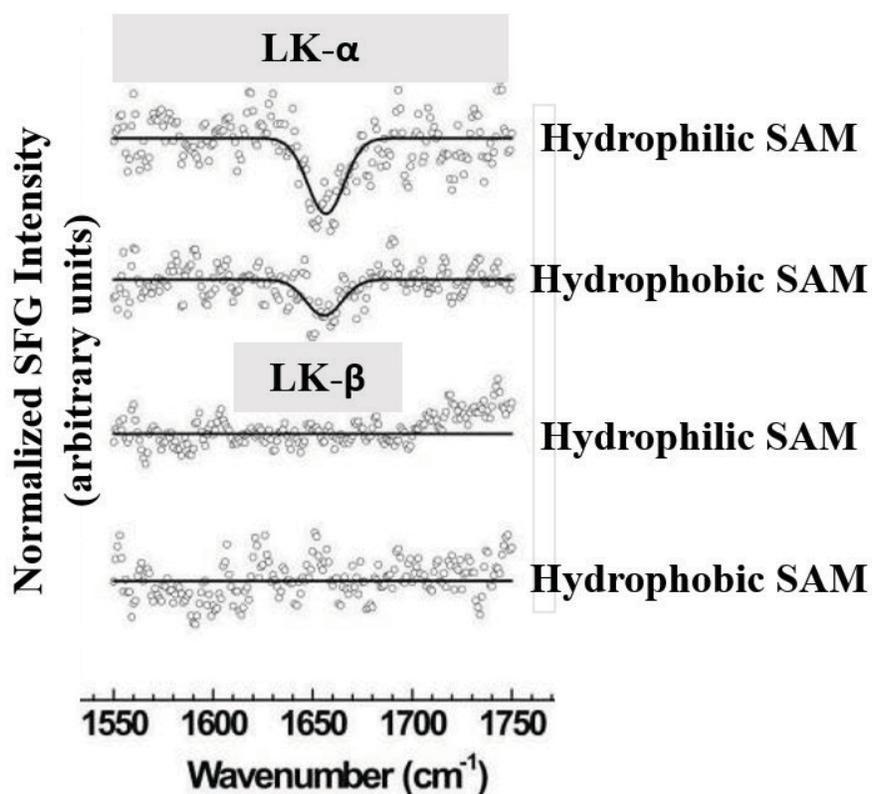


Figure 3. SFGS results obtained by Castner and coworkers for the Amide I region⁵¹. Peaks at 1655 cm^{-1} indicate the presence of α -helical structures. The top two spectra are for LK- α , while the bottom two are for LK- β . The top spectrum in each pair is for the hydrophilic SAM surface, while the bottom in each pair is for the hydrophobic SAM surface.

amide I signal is observed, indicating canceling C=O group orientations characteristic of extended β -strand structures (Figure 3, bottom). On the hydrophilic SAM for both LK- α and LK- β , the phase of the SFG resonances indicates that the leucine side chains are oriented away from the surface (Figure 4, left). On the hydrophobic SAM for both LK- α and LK- β , the phase of the SFG resonances indicates that the leucine side chains are oriented towards the surface (Figure 4, right).

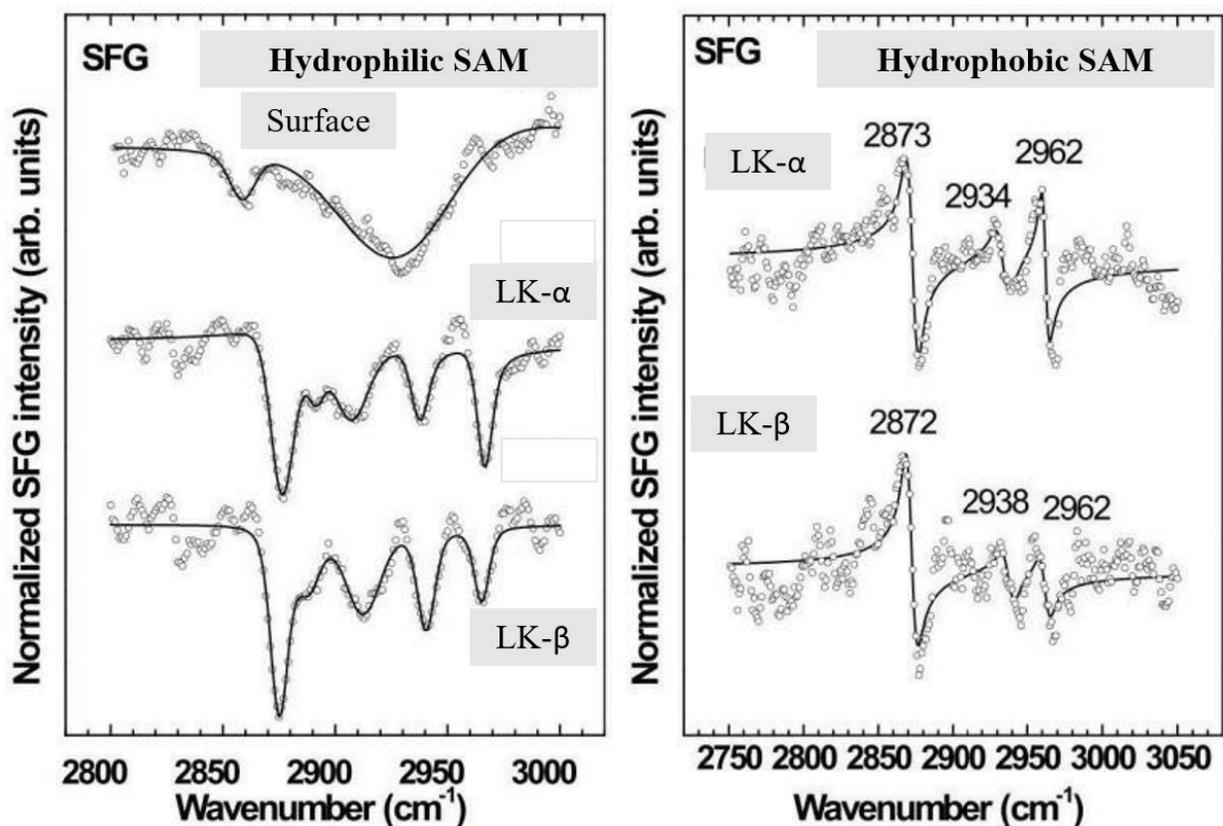


Figure 4. SFGS results obtained by Castner and coworkers for the CH region⁵¹. The differences in spectra indicate that the leucines are oriented away from the hydrophilic surface for both peptides, while they are oriented toward the hydrophobic surface for both peptides. On the left are spectra obtained for the hydrophilic surface, in order from top to bottom: the hydrophilic SAM surface without peptides, the surface with LK- α , and the surface with LK- β . On the right are spectra obtained for the hydrophobic surface, with LK- α on the top and LK- β on the bottom.

These experimental observations are ideal for comparing with structure prediction calculations. On both hydrophobic and hydrophilic SAMs, an accurate calculation would show high α -helical content for LK- α and low α -helical content for LK- β . Thus, the difference in predicted helicity between LK- α and LK- β , assessed on both hydrophobic and hydrophilic SAMs, is a quantitative measure of the calculation's ability to capture structural differences between these peptides. Proper orientation of the leucine side chains towards the hydrophobic surface and away from the hydrophilic surface is a secondary measure of structural accuracy.

Several previous computational studies compared to Castner and coworker's data on the LK peptide/SAM system. Using replica-exchange molecular dynamics simulations, Latour and coworkers assessed the ability of three commonly used protein force fields (CHARM22, AMBER94, and OPLS-AA) to reproduce the structural results and found the CHARM22 force field to be most accurate in predicting helical content³⁹. Deighan and Pfaendtner used molecular dynamics with two enhanced sampling strategies, metadynamics and umbrella sampling, to exhaustively sample LK peptide adsorption using the AMBER99SB force field⁵⁵. Additionally, Deighan and Pfaendtner compared the performance of three force fields (CHARM22, AMBER99SB, and OPLS-AA) specifically on the LK- α /methyl-terminated SAM system. In their study, metadynamics outperformed umbrella sampling in exploring a large number of conformational states. In their force field comparison, CHARMM22 and AMBER99SB reproduced the experimental results more accurately than OPLS-AA.

Here, I assess the ability of the RosettaSurface algorithm to reproduce the structural data reported by Castner and coworkers on the LK peptide/methyl-terminated SAM and LK peptide/carboxy-terminated SAM systems. The results of this assessment will provide a quantitative measure of the accuracy of the RosettaSurface energy function. After evaluating the

accuracy of the default energy function, I perform a parametric analysis to improve the energy function. Finally, I compare the results of RosettaSurface and the two molecular dynamics studies mentioned previously. This comparison tests the validity of the simplifying assumptions utilized in the RosettaSurface algorithm (implicit solvation, static surface) and helps better understand sampling and scoring strategies for simulation of peptide adsorption.

Chapter 2: Methods

This chapter has elements in common with chapter 4 of Michael S. Pacella's doctoral dissertation, "Modeling and Design of Peptides to Control Biomineral Nucleation and Growth," Johns Hopkins University, Baltimore, Maryland, Copyright 2017 Michael Steven Pacella. Adapted portions are reproduced with permission.

2A: Generation of Starting Structures

I constructed extended peptides from the published sequences of LK- α (Ac-LKLLKLLKLLKLL) and LK- β (Ac-LKLKLLKLLKLLKLL)⁵¹. I retained the acetylated N-termini for both peptides, but shortened the LK- β sequence to 14 amino acids by removing the C-terminal leucine, to match the length of LK- α to ensure consistent reference energies in Rosetta. When constructing the extended peptides, I used ideal bond lengths and angles⁵⁶. All main chain ϕ/ψ torsion angles as well as side chain χ angles were flexible in the simulations.

Structures of the methyl-terminated and carboxy-terminated alkane-thiol self-assembled monolayers were taken from Collier *et al.*³⁹. To remove edge effects, the slab dimensions were expanded to 85 x 85 Å by duplication. The slab thickness was 13 Å. The ratio of protonated to unprotonated carboxylic acid head groups on the carboxy-terminated SAM was adjusted to reflect the experimental pH of 7.4. The atom parameters for the surfaces were taken from similar groups (CH₃, CH₂, COOH) in canonical amino acids. To account for the SAM alkane having a larger hydrophobic region and lacking nitrogen and oxygen, the solvation energy ($\Delta G_{\text{solvation}}$, or LK_DGFREE in Rosetta) of the methyl termini was set to double that of the methyl R-group of alanine, 3.0 Rosetta Energy Units (REU, roughly equivalent to kcal/mol), making it twice as hydrophobic.

2B: Sampling Protocol

I simulated the LK peptide/SAM systems using the Rosetta surface docking algorithm described by Pacella and coworkers⁴⁷. Briefly, the peptide is centered over the surface, and a random number of up to five solution-state refinements are performed to increase the variety of initial conformations and address the possibility of proteins denaturing on a surface. Then the peptide is brought into contact with the surface and refinement continues with simultaneous optimization of the peptide's backbone, rotamers, and position on the surface. I did not apply harmonic constraints to keep the peptide near the surface, as was done by Latour and coworkers³⁹, to avoid bias in the simulation that might misrepresent the behavior resulting from the Rosetta energy function.

I generated 10^4 candidate structures for baseline studies, using the default Rosetta score function. I generated 10^3 structures for single-parameter weight variation trials to save on computation requirements. For two-variable simulations, where two score parameters and/or atom properties were varied, I returned to generating 10^4 structures because of the difficulty to increase precision, compared to the single-parameter studies. In both cases, analysis was performed on the 1% of structures with lowest energy. All reported simulations used Rosetta's Talaris2013 score function or single-parameter variants thereof. Simulations were run on the Maryland Advanced Research Computing Cluster (MARCC), requiring 120–180 cpu-hours per 10^3 candidate low energy structures (decoys).

2C: Structure Analysis

Secondary structure was determined at each residue by DSSP⁵⁷. Overall peptide helicity was calculated as $H = \frac{n_{\text{helical}}}{N-2}$, where n_{helical} represents the number of residues with helical

secondary structure and N represents the total length of the peptide. The length is reduced by two because the terminal residue secondary structure cannot be assigned.

Distances from the surface were measured from the average z coordinate of the top atoms in the surface: the oxygen atoms of the hydrophilic SAM surface, and the methyl carbon atoms of the hydrophobic SAM surface. Following Latour and coworkers³⁹, distance for amino acid residues was measured to the z coordinate of end of the R-group: the average of the C_δ atoms of leucine, and N_ζ for lysine.

Chapter 3: Baseline Results, Unmodified Rosetta Score Function

This chapter has elements in common with chapter 4 of Michael S. Pacella's doctoral dissertation, "Modeling and Design of Peptides to Control Biomineral Nucleation and Growth," Johns Hopkins University, Baltimore, Maryland, Copyright 2017 Michael Steven Pacella. Adapted portions are reproduced with permission.

3A: Baseline Peptides Adsorbed on Surfaces, Talaris2013

To assess the ability of the RosettaSurface algorithm to reproduce the structural data reported by Castner and coworkers, I inspected the low energy structures generated by RosettaSurface for each of the four systems produced by combining either LK- α or LK- β peptide with either the hydrophilic or the hydrophobic SAM surface (Figure 5). The lowest final score was calculated with the Talaris2013 score function⁵⁸⁻⁶¹. The low-energy structures approximate the favored equilibrium state of the system. A set of the 100 lowest energy structures for each system were included in my assessments, corresponding to 1% of the total structures generated.

Figure 6-Figure 9 show the aggregated secondary structure, scoring, and orientation results of these simulations. In each quadrant, the left pair of plots corresponds to LK- α and the right pair to LK- β , while the top pair correspond to the hydrophilic SAM surface and the bottom to the hydrophobic SAM surface. Each plot summarizes data from the 100 lowest energy models of a 10,000-model simulation, thus summarizing the diversity of low-scoring structures. This sample size was selected to allow both sufficient sampling for predominant trends to emerge, and to restrict analysis to models that scored favorably with RosettaSurface's energy function.

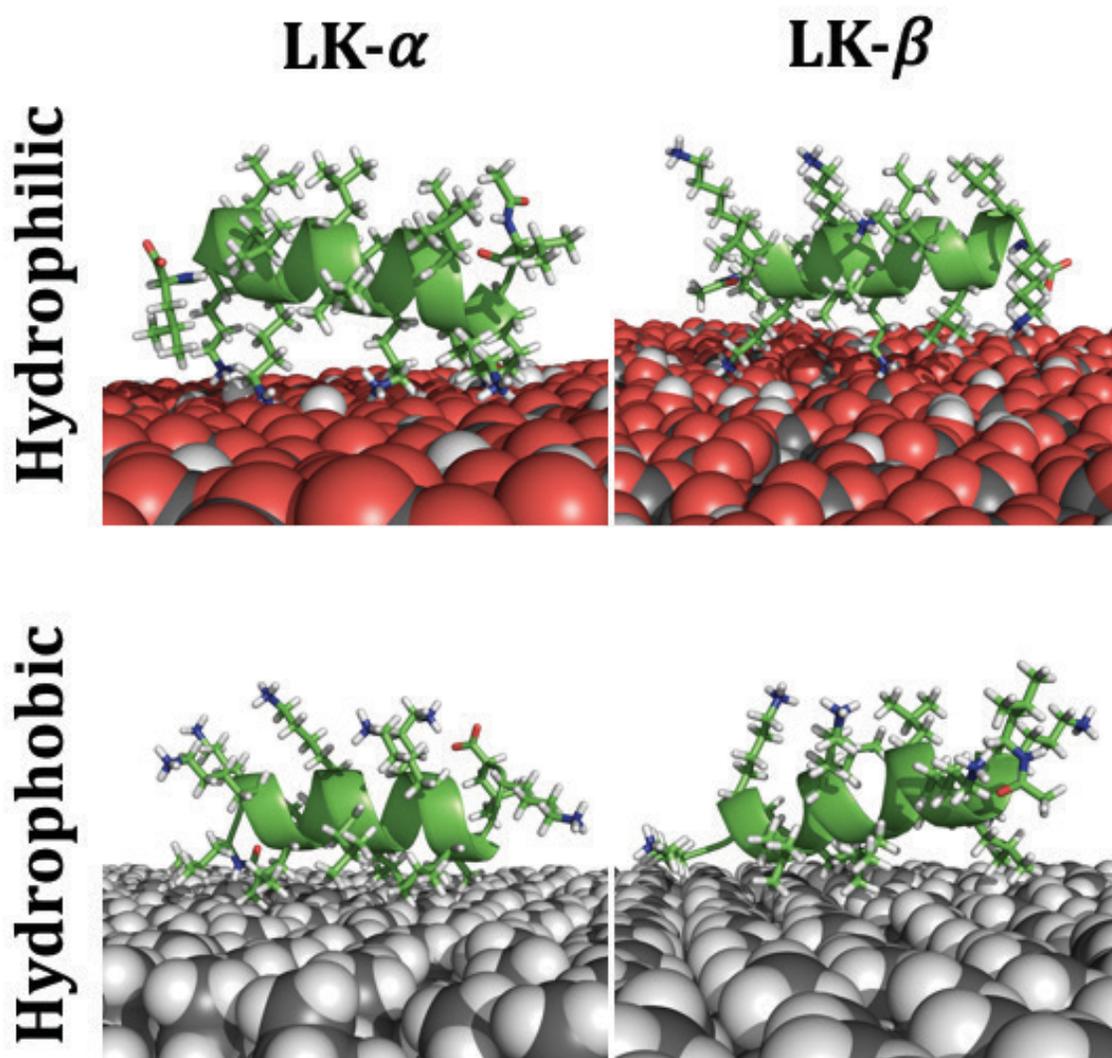


Figure 5. Low energy structures of LK peptides adsorbed on hydrophobic and hydrophilic SAMs. Top left: LK- α on hydrophilic SAM. Top right: LK- β on hydrophilic SAM. Bottom left: LK- α on hydrophobic SAM. Bottom right: LK- β on hydrophobic SAM. All images were obtained from simulations with the standard Talaris2013 score function.

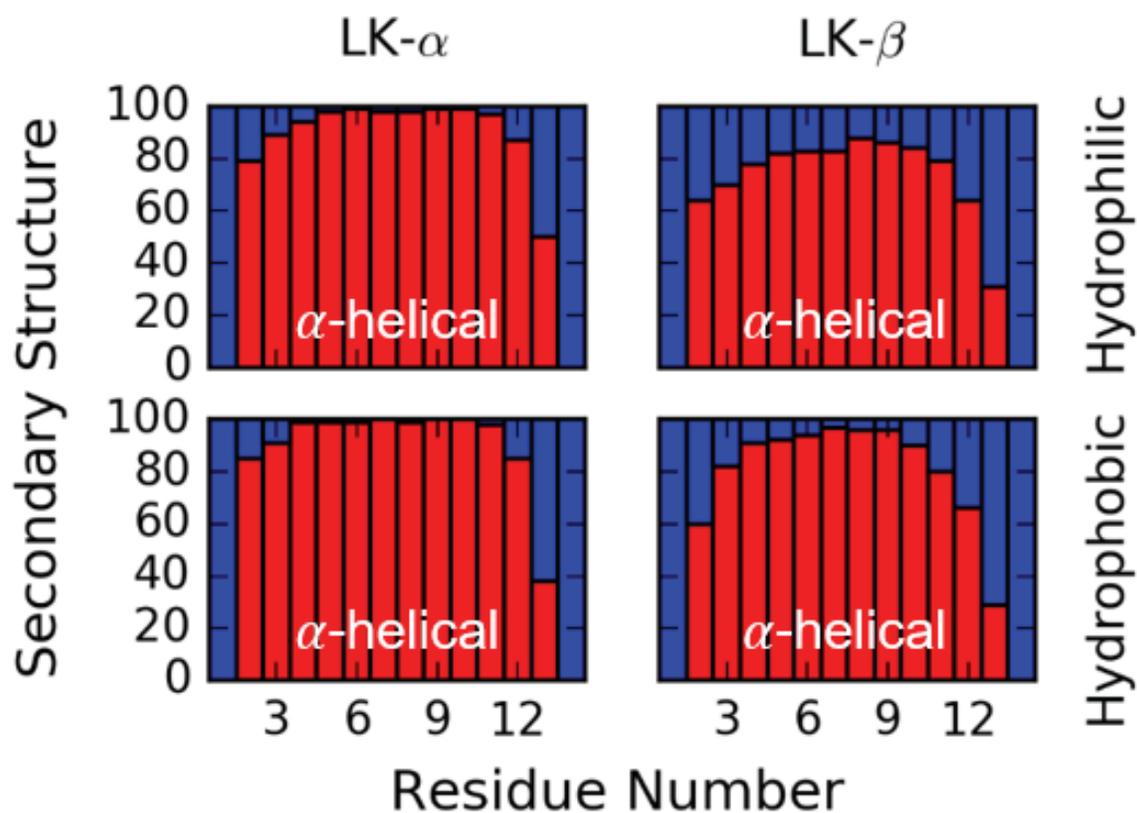


Figure 6. Secondary structure distribution of top 100 low-scoring decoys from simulations using RosettaSurface defaults. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue.

To assess the consensus of the predicted secondary structures in these simulations, I generated secondary structure distribution plots (Figure 6) that show the fraction of helical models at each residue position for each system, as well as Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the set (Figure 7). These distributions show strong consensus towards α -helical secondary structure for both LK- α and LK- β peptides on both the hydrophobic and hydrophilic surfaces. The predicted secondary structure for LK- α , with an average helicity of 91% on both surfaces, agrees with the experimental results. For LK- β , the average helicity across the 100 low-scoring structures was 74% on the hydrophilic surface and 81% on the hydrophobic surface, in contrast with the extended secondary structure observed in

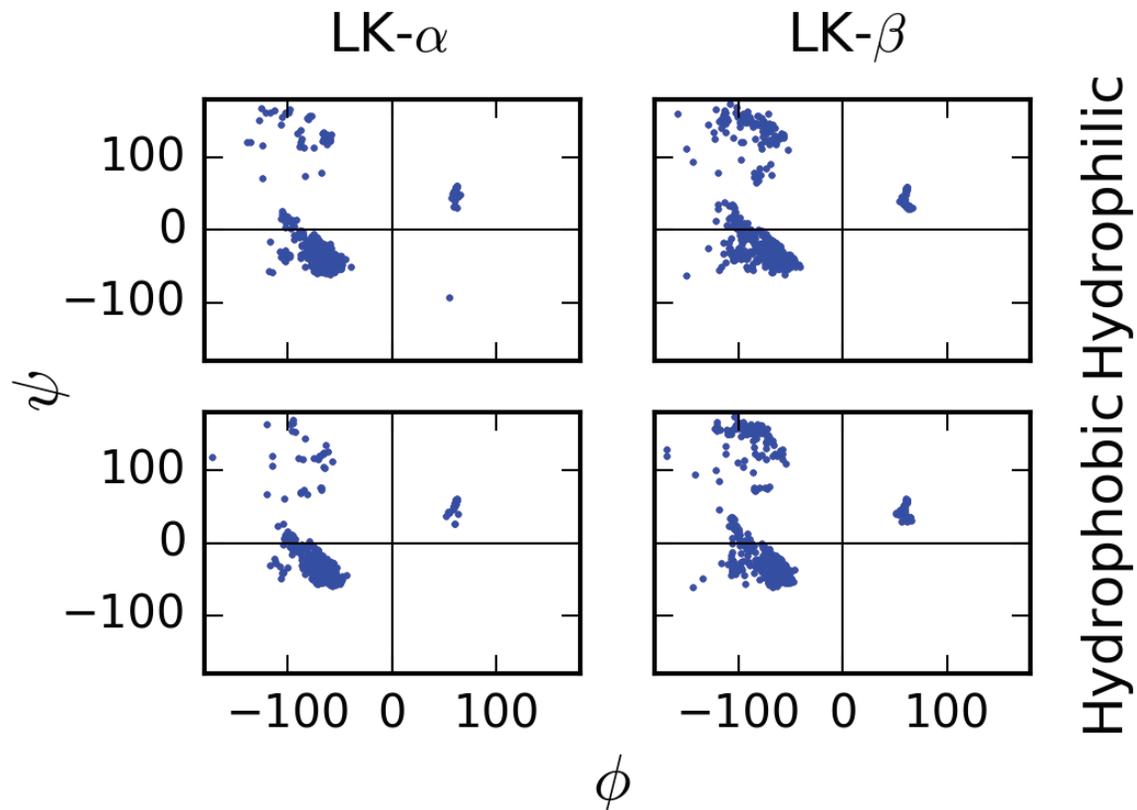


Figure 7. Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys from simulations using RosettaSurface defaults.

experiments. The conformity of points in all plots in Figure 7 to the usual regions observed in a Ramachandran plot indicate that even though LK- β is not assuming the expected structure, the structures assumed are not unusual among documented proteins.

This relationship between score and helicity is presented in Figure 8, a scatterplot showing the decoy score vs the overall peptide helicity for each of the top 100 decoys. A tendency for the lowest scoring decoys to converge on a certain structure with a steep decrease in score indicates the favorability and/or a likelihood that the structure is the native structure. Additionally, these plots would reveal that near-correct structures are not sampled (if there are few points at appropriate helicity), or if the score function does not adequately favor correct structures (if the expected structures are not the lowest energy). LK- α data reveals high helicity with favorable

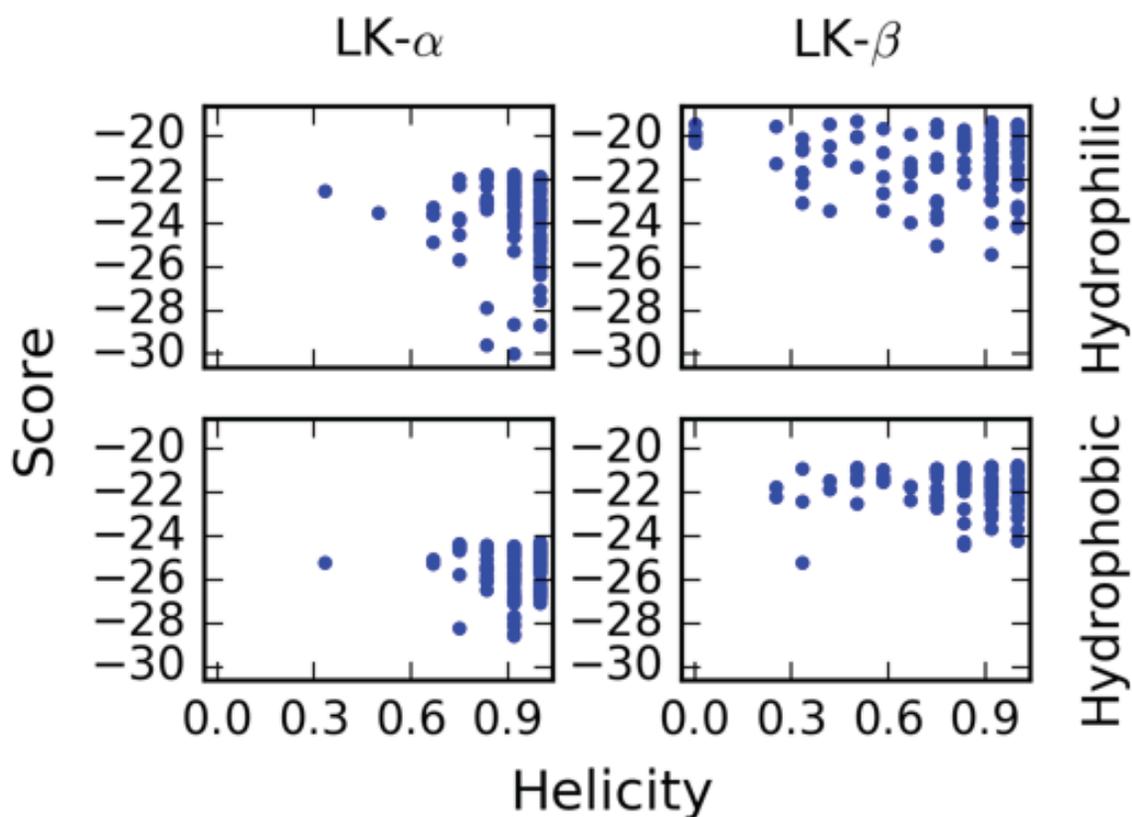


Figure 8. Score vs. helicity distribution of the top 100 decoys from simulations using RosettaSurface defaults.

energy on both surfaces, though with a larger favorable energy change with increased helicity on the hydrophilic surface. This matches the stronger helical peak that Castner and coworkers observed for the hydrophilic surface than for the hydrophobic (Figure 3). The LK- β data show that the energies are similar across different values of helicity, with helical structures having the lowest energy, in contrast with experimental results.

The presence of completely extended LK- β structures indicates that RosettaSurface samples adequately to explore some non-helical structures. The inferior scores of nonhelical LK- β structures on the hydrophilic SAM surface and their absence in the 100 lowest-scoring models on the hydrophobic SAM surface indicate a shortcoming of the score function for recognizing extended structures on the SAM surfaces. Also, the average total scores for LK- β structures on

both surfaces are 2 and 4 REU higher than the average total scores for LK- α on the hydrophilic and hydrophobic surfaces respectively. Thus, the data suggest more confidence in the accuracy of predictions for LK- α than for LK- β . The score change, which represents the energy of an interaction, is approximately 30 REU for a 14-mer forming a helix. So either the rewards for forming an extended structure would need to be increased, or the reward for forming a helix would need to be decreased, in order to reproduce the experimental observations of extended structures for the LK- β peptide.

Figure 9 shows distributions of surface separation distances for the side chains of leucine and lysine residues and for the backbone α -carbons ($C\alpha$) on each SAM surface, (see Chapter 2: Methods). I generated these plots to assess the degree to which my predicted peptide orientations relative to the surface agreed with the experimental results of leucines being oriented away from the hydrophilic SAM surface and towards the hydrophobic SAM surface for both LK peptides. If the simulations match experimental results, I would see three distinct peaks in each case, with the $C\alpha$ in the middle, and leucine and lysine peaks on opposite sides depending on the surface. Peaks would be sharper for ideal extended structures than for helical ones, as the helix's shape produces some variation in residue position.

In the simulations for LK- α , the expected pattern emerges. The lysine peak is sharp at ~ 4 Å as these residues make consistent contacts with the SAM hydroxyl groups. The $C\alpha$ peak ranges from 7-11 Å. The leucine peak for LK- α on the hydrophilic surface is broad (6-17 Å), since the leucines project from the entire upper half of helix, and are not long enough to extend away from the surface as the lysines do in the hydrophobic surface. For LK- β on the hydrophilic SAM surface, leucines are found farther from the surface than lysines, in agreement with experiment,

but because of the peptide's periodicity, a helical structure orients some leucines towards the surface. For LK- β on the hydrophobic SAM, leucines and lysines are both present near and away from the surface. In this system, lysines are distributed fairly evenly between small (~ 4 Å) and large ($\sim 10-14$ Å) separations, and there is also a significant fraction of leucines oriented away from the hydrophobic SAM. Both of these results contrast with the experimental observation of leucines oriented towards the surface and lysines oriented away. There is a significant occurrence of LK- β positioning C_α near the hydrophobic SAM surface, indicating backbone-surface interactions which were not observed in the experiments.

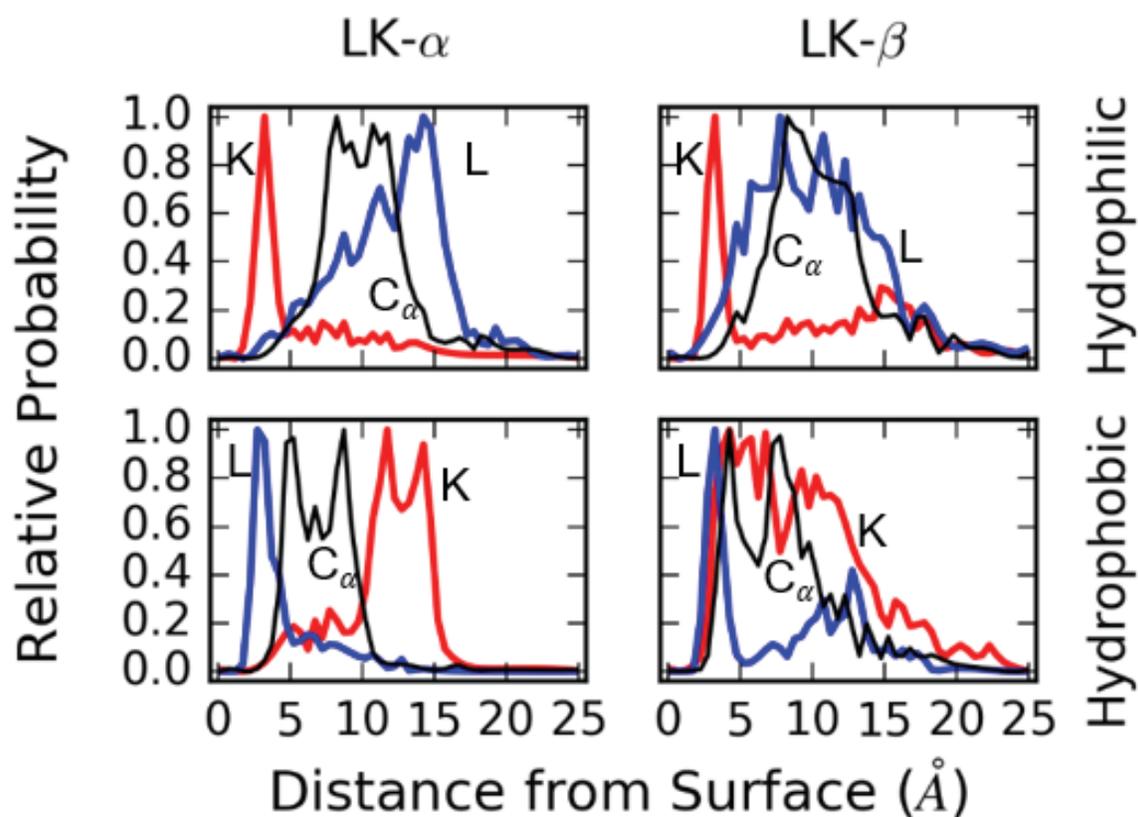


Figure 9. Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys from simulations using RosettaSurface defaults.

The plots in Figure 9 also provide an indication of the peptide's adhesion to the surface. As noted in the Experimental section, I did not employ a constraint to keep the peptide in proximity to the surface. Since the peptides adsorbed readily in experiments, I expected favorable interactions to suffice to promote surface binding. The $C\alpha$ curves show whether the simulations met this expectation. Distribution into the region past 15 Å is only possible if the peptide has residues farther from the surface than is possible for a helix lying on the surface, indicating that the peptide has tipped up such that some residues are out of contact with the surface. Since all systems except LK- α on the hydrophobic SAM surface show a small amount of high-distance positioning, I infer that surface interactions might not be adequately rewarded by the score function relative to other forces acting on the peptides. However, it should be noted that experimental results did not determine that such separations were absent in adsorbed peptides, so the Rosetta result may be reasonable.

Overall, I conclude that RosettaSurface is able to correctly calculate LK- α structures on both SAM surfaces, but has less success with LK- β . I verify that the docking algorithm is adequate to sample both completely helical and completely non-helical structures, though non-helical LK- β structures do not score better than helical ones.

3B: Solution Models, Talaris2013

Castner's experiments focused on the surface-adsorbed peptides⁵¹. However, solution data were collected in the original LK-peptides publication by DeGrado and Lear⁴⁹. They determined that LK- α 's structure depended on peptide and salt concentrations, ranging from 0%-70% helicity with the remainder in random coil. LK- β was observed to precipitate at high concentrations, but at low concentrations assumed approximately 50% β -sheets. These data provide another point of comparison to validate the Rosetta approaches. However, Rosetta's implicit water model does not account for salt and peptide concentrations, so Rosetta would not be able to capture these effects

I used a modified RosettaSurface protocol, which output a solution state model for each simulation, just before the peptide was moved into contact with the surface. These models were scored separately, since the lowest scoring surface-docked model did not necessarily have the lowest-scoring solution-state precursor. The plots comparable to Figure 6-Figure 8 are for these models are combined into Figure 10 (an equivalent of Figure 9 is not included as all atoms in the peptide are more than 25Å from the surface).

Both LK- α and LK- β tended to assume helical structures in their lowest-energy states. In the case of LK- β , this differed from experimental results. I did not include these results in the publication, confining my published data to only Castner's benchmark, because Rosetta does not model or capture differences in concentration of peptide or salt, and these aspects more

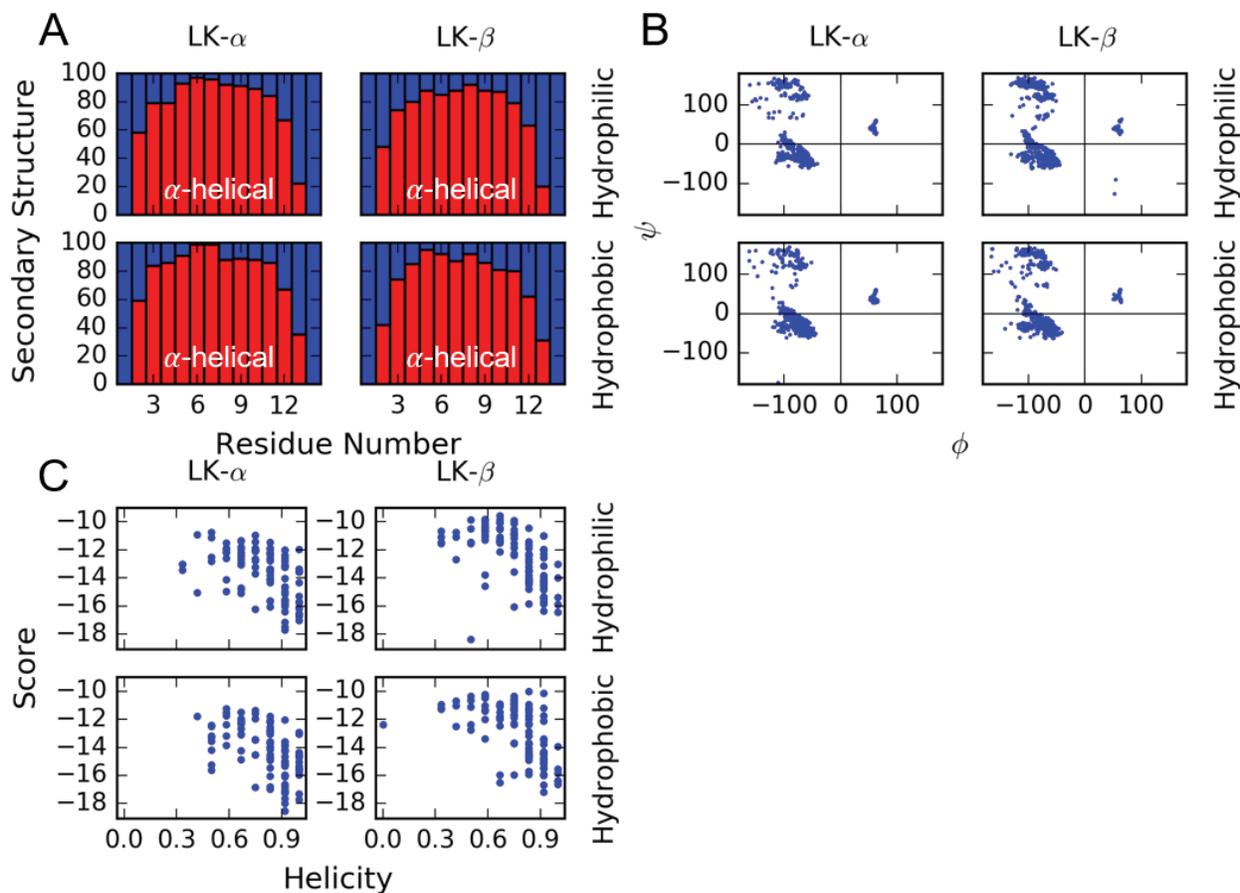


Figure 10. Helicity, position, and score results for the solution-state structures obtained using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys.

significantly affected peptide conformation in solution than when adsorbed on the SAM surfaces. This analysis serves as a check that the simulation protocol and analysis scripts were performing as desired. Each peptide's solution behavior should be independent of the SAM surface with which

it is later brought into contact. The near-symmetry of each peptide's pair of plots in Figure 10C suggests that Rosetta sees these peptides as behaving similarly in solution.

Recently, researchers have released an updated Rosetta energy function, called REF2015^{30,32}. The energy function differs from Talaris2013 in several ways. While Talaris was optimized for macromolecules in the Protein Data Bank, REF was optimized also with small

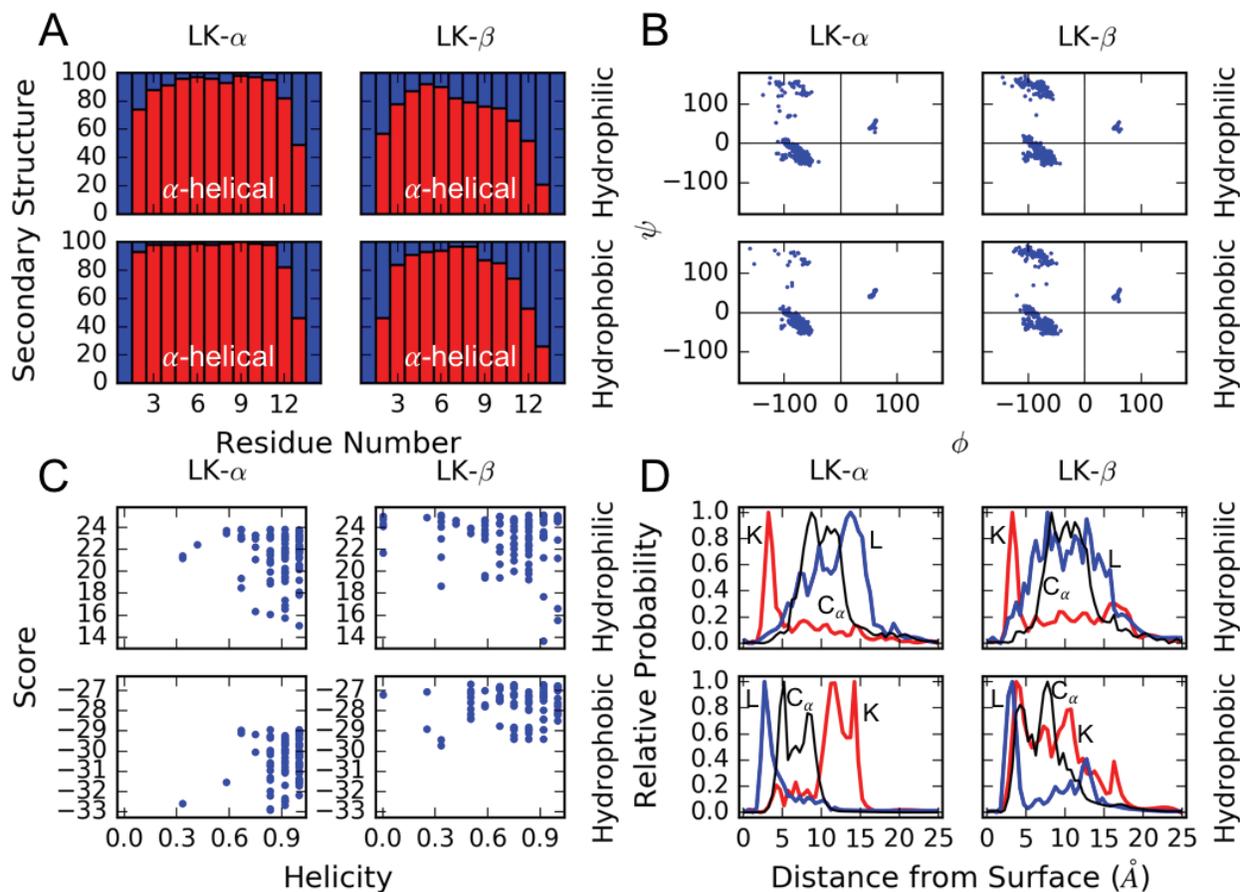


Figure 11. Helicity, position, and score results for REF2015 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.

molecules, and the score weights were tuned and optimized such that an energy unit in Rosetta would be consistent with 1 kcal/mol. Additionally, the solvation model was modified to include anisotropy, and the electrostatics and Lennard-Jones models were updated. I sought to test how the changes affected the predictions for LK peptides on SAM surfaces.

The plots comparable to Figure 6-Figure 9 are combined into Figure 11. The helicities of LK- α were 88% on the hydrophilic SAM and 92% on the hydrophobic SAM, while for LK- β they were 71% and 77%. Thus the REF2015 score function discriminated only slightly better than the Talaris2013 on the hydrophobic SAM surface, and similarly on the hydrophilic. A difference from the Talaris2013 results is in the scoring of the models on the hydrophilic surface. In this case, scores were much higher (less favorable), due to a large unfavorable contribution from the `fa_intra_sol` scoring term, representing the implicit solvation energy between atoms on the same residue, which is not a factor in the Talaris2013 score function. Surface distances and orientations were similar to those obtained with the Talaris2013 score function.

Chapter 4: Results, Alternate Peptides

4A: Alternate Peptide Termini

The experiments done by Castner *et al.*⁵¹ used 14-mer peptides that were N-acetylated, and these are the constructs used in the calculations described in the previous section. However, I also generated starting (undocked) structures for unmodified peptides, and for peptides that were both

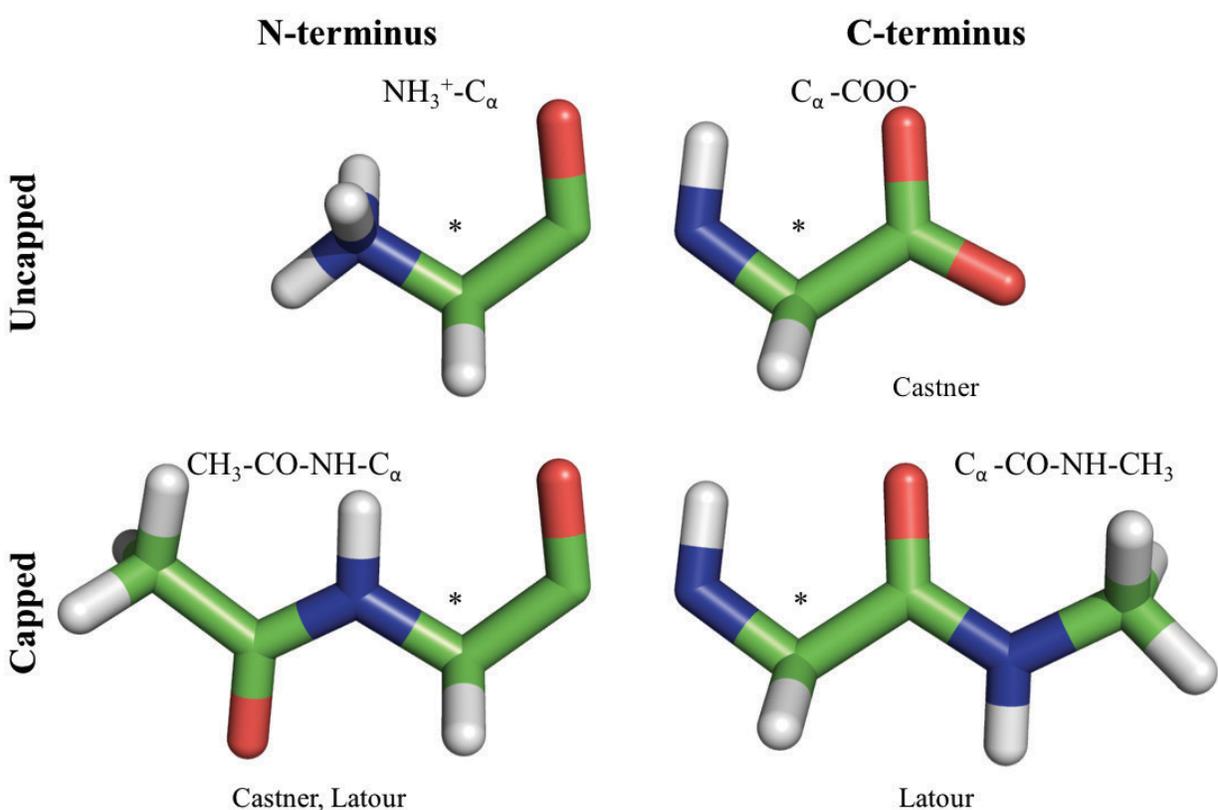


Figure 12. Peptide termini with side chains not shown. Asterisks (*) are placed above the α -carbon of the terminal residue. Capped peptides used by Latour and coworkers³⁹ were acetylated at the N-terminus and amidated at the C-terminus, while uncapped peptides had unmodified termini. The peptides used by Castner *et al.*⁵¹, and which I used in the publication, were N-acetylated, but unmodified at the C-terminus.

N-acetylated and C-amidated, matching the constructs used in the MD simulations of Latour and coworkers³⁹ (Figure 12).

Compared to the N-capped peptides, uncapped peptides (Figure 13) had a lower amount of N-terminal helicity for both LK- α and LK- β . In order of LK- α on the hydrophilic surface, LK- α

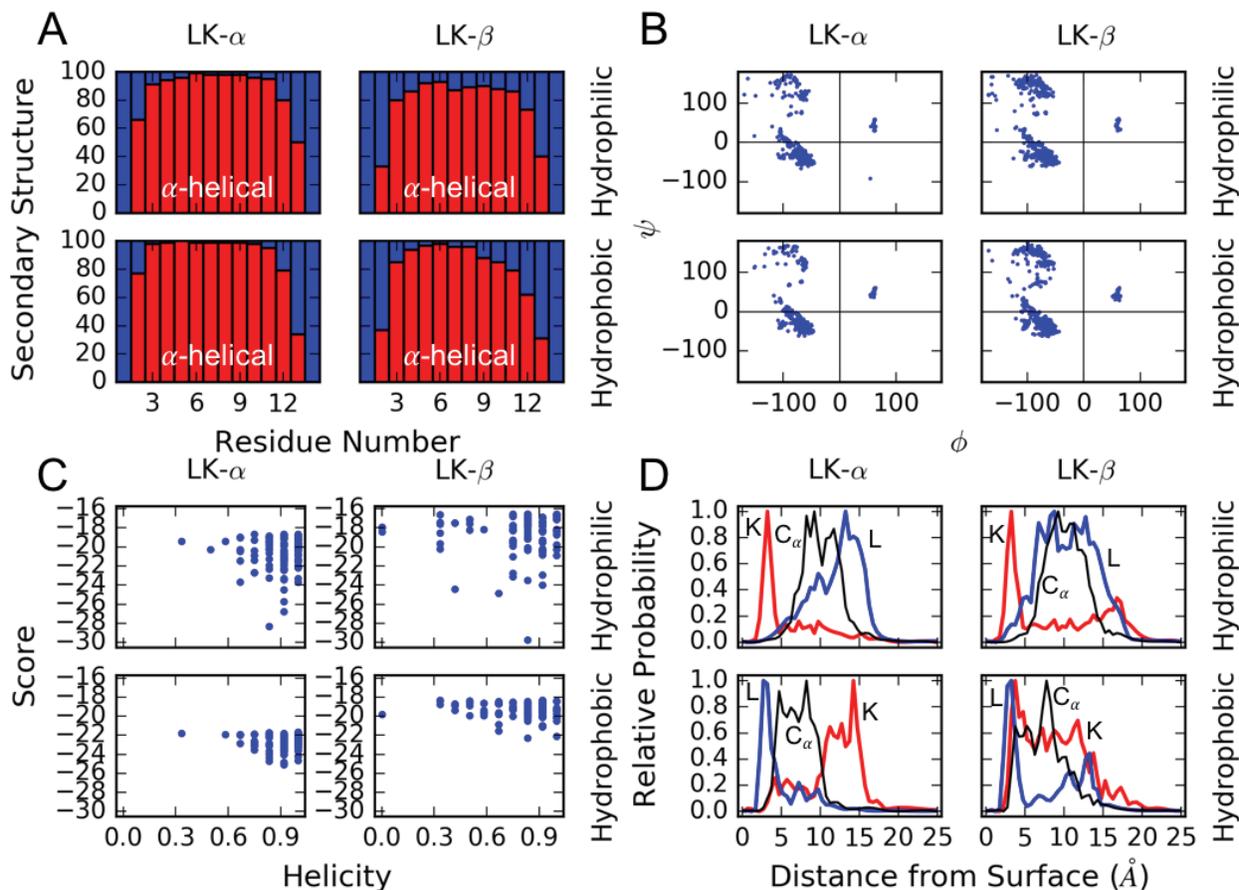


Figure 13. Helicity, position, and score results for *uncapped* 14-mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.

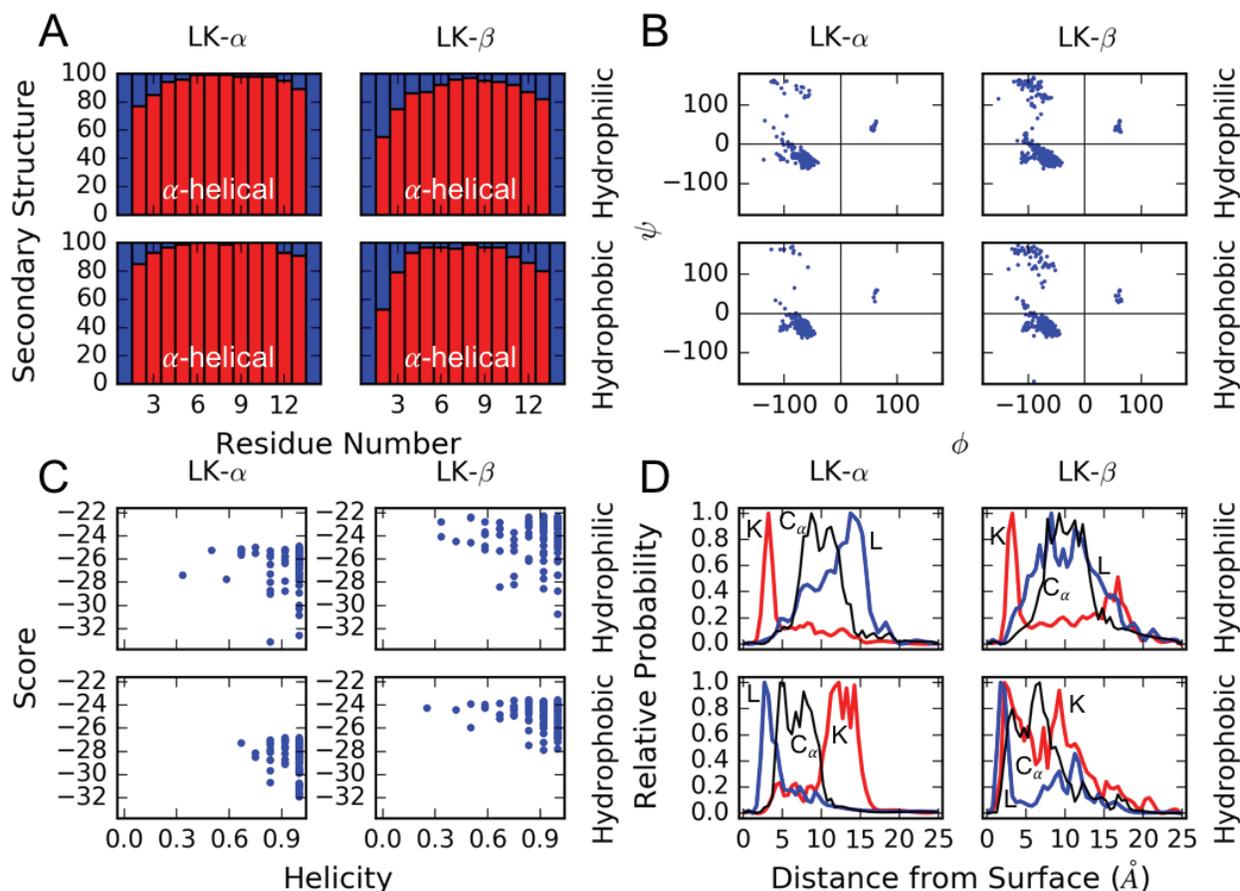


Figure 14. Helicity, position, and score results for *N*- and *C*-capped 14-mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.

on the hydrophobic surface, LK- β on the hydrophilic surface, and LK- β on the hydrophobic surface, the average helicities for the 100 lowest-scoring structures were 88%, 78%, 90%, and 79% for uncapped peptides, and 94%, 87%, 96%, and 87% for capped peptides. Meanwhile, the

peptides capped at both ends had higher C-terminal helicity than N-capped peptides (Figure 14). This suggests that in both cases, the charge-neutralizing cap favors an α -helix, whereas charged termini disfavor coiling.

In terms of scoring, in all cases, the scores were better (lower) for the capped peptides than for the uncapped peptides, with the published N-capped peptide scores bracketed in between. Typically, larger molecules get larger scores in Rosetta. In all cases, more helical models tended to score better than less helical ones. As was the case with the N-capped peptides, score-helicity distributions were tighter for LK- α than for LK- β , and scores were better overall.

The surface distance plots of uncapped, N-capped, and N- and C-capped peptides do not show any significant differences. These data suggest that terminal charge did not play a large role in the orientation of peptides on the surface.

4B: Alternate Peptide Sizes

The results in other sections refer to 14-mer peptides, LK- α and LK- β , which could be described more specifically as LK- α 14 and LK- β 14. Other work with LK peptides also used 7-mers. Specifically, the MD simulations of Latour and coworkers³⁹ used LK- α 14 and LK- β 7, and

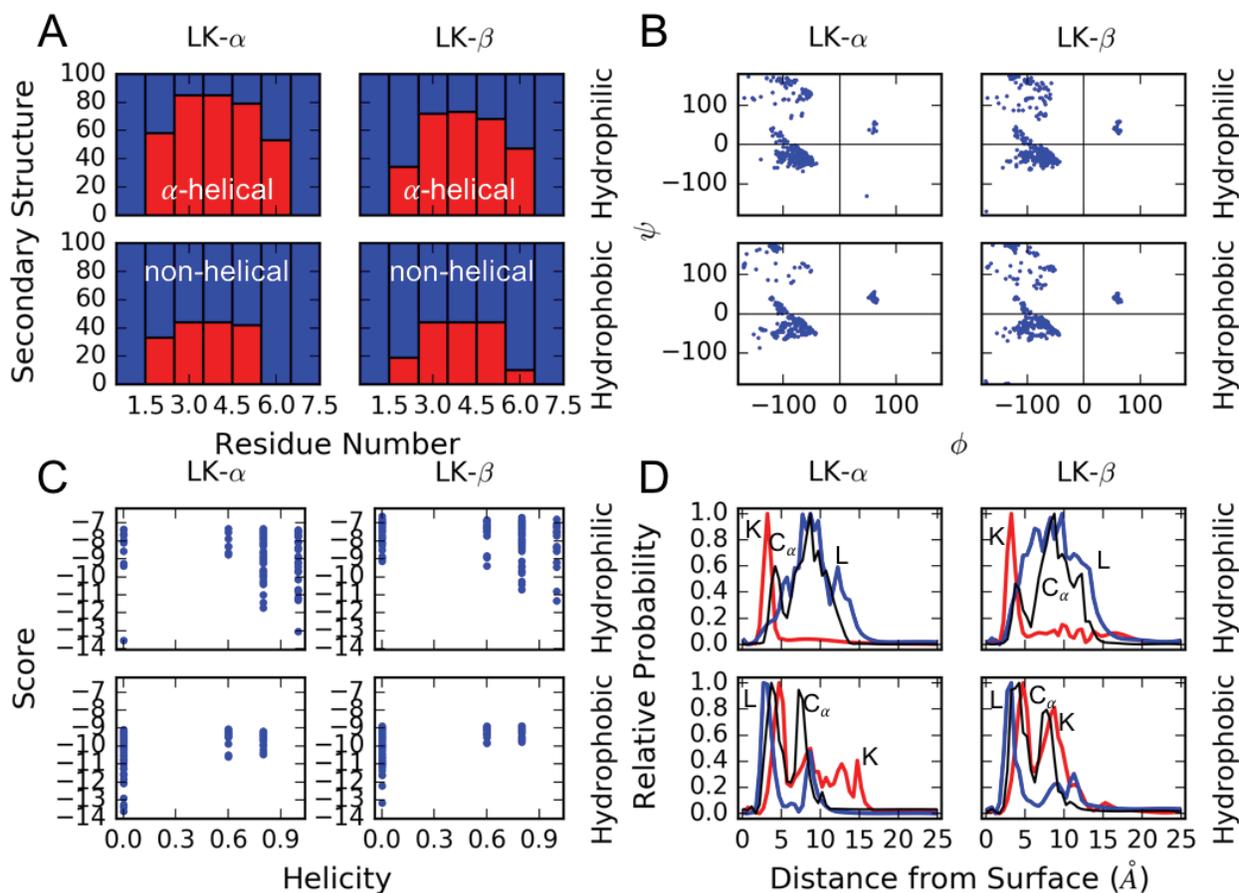


Figure 15. Helicity, position, and score results for uncapped 7mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.

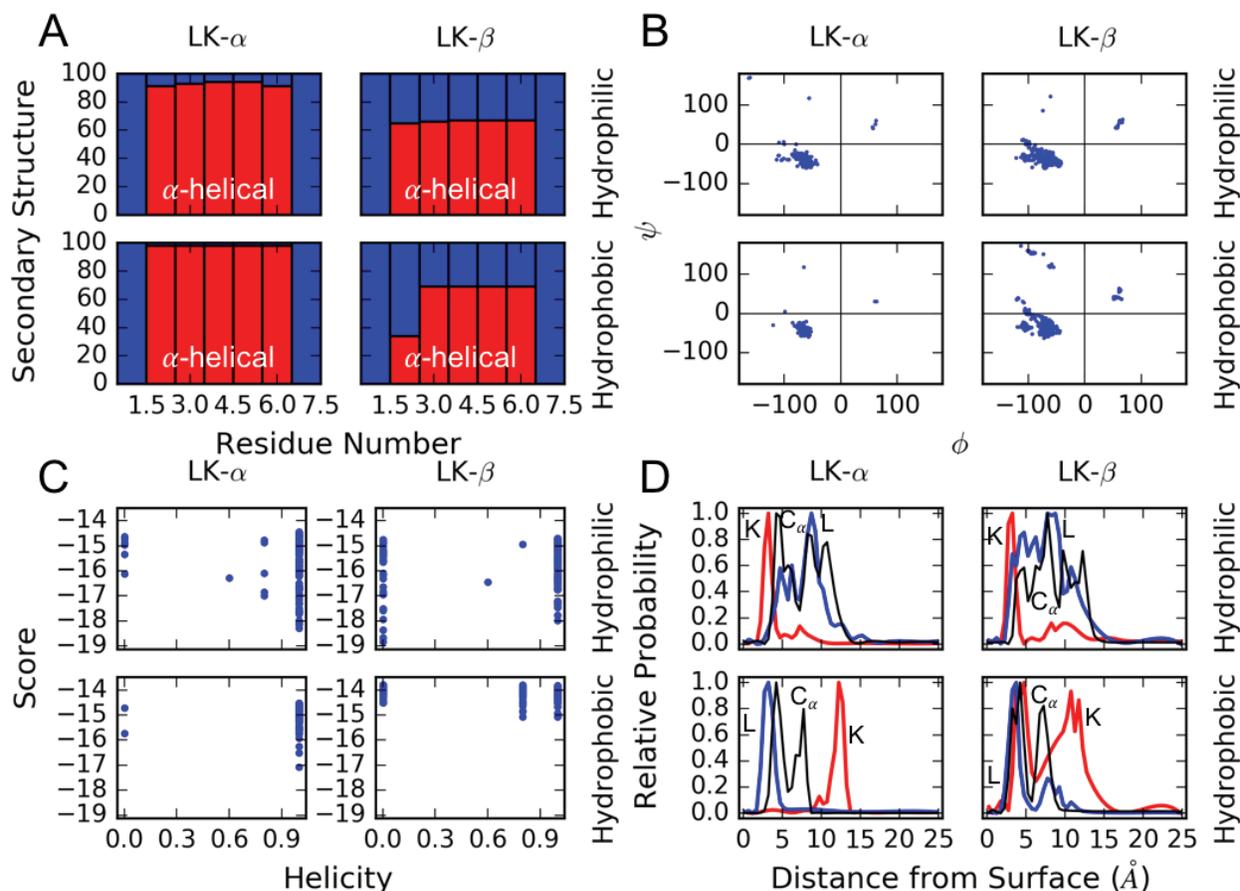


Figure 16. Helicity, position, and score results for N- and C-capped 7mer peptides using the Talaris2013 score function. A: Secondary structure distribution of 100 low-scoring decoys. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue. B: Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys. C: Score vs. helicity distribution of the top 100 decoys. D: Distributions of leucine (blue), lysine (red), and the backbone α -carbon (black) distances from the surface in the 100 low-scoring decoys.

the original publication by DeGrado and Lear⁴⁹ used LK- α 7, LK- α 14, and LK- β 7. Therefore, I generated starting structures for 7-mer peptides as well as 14-mers, and ran a number of simulations using these other starting structures.

The shorter 7-mer LK-peptides (Figure 15, Figure 16) were more susceptible to the effect of capping, since the formation of a helix requires at least four residues to coil. N- and C-capped 7-mer peptides had much higher helicity than uncapped ones for both peptides and both surfaces. Capped LK- α 7 was almost entirely helical on both surfaces. Uncapped LK- α 7's helicity was more dependent on the surface, with higher helicity on the hydrophilic surface than on the hydrophobic surface, since there was a greater tendency of the backbone contacting the hydrophobic surface to reduce unfavorable solvent-accessible surface area. Since the neutralized caps would be more attracted to the hydrophobic surface than the uncapped, charged termini, this further illustrates that the effect of the caps is to stabilize the helix, rather than to anchor the peptide to the surface.

The same observation that was true of the 14-mers, that caps made the score more negative, was true in 7mers as well. The score-helicity plots show why the 7-mers tended to be either all helical or all non-helical: the range of helicities between 0 and 0.6, not inclusive, are always empty. This is because a peptide cannot have one, two, or three helical residues.

The surface distance plots of 7mer peptides largely follow the patterns of the 14mer peptides, in reflecting the correctness or incorrectness of their secondary structure. The notable exception is the uncapped LK- β 7, which despite low helicity, still includes a significant amount of lysine near the surface. The simultaneous proximity of C_{α} to the surface indicates that the peptide is laying flat on the surface.

Chapter 5: Results, Score Weight Variations

5A: Single Physical Parameter Score Weight Variations

One discrepancy between RosettaSurface structural predictions and the experimental observations of Castner and coworkers is that the LK- β peptide favors helical structures over extended structures. This behavior should be controlled by the balance between helix breaking and helix favoring forces. Hydrogen bonding and van der Waals forces favor the formation of α -helices. Meanwhile, the unfavorable solvation of hydrophobic leucine residues disfavors any α -helical conformation that orients leucine residues away from the hydrophobic surface. With this balance of forces in mind, I hypothesized that increasing the weight on the implicit solvation energy term in the RosettaSurface energy function might favor more contact between leucines and the hydrophobic surface and extend the LK- β peptides on hydrophobic SAMs.

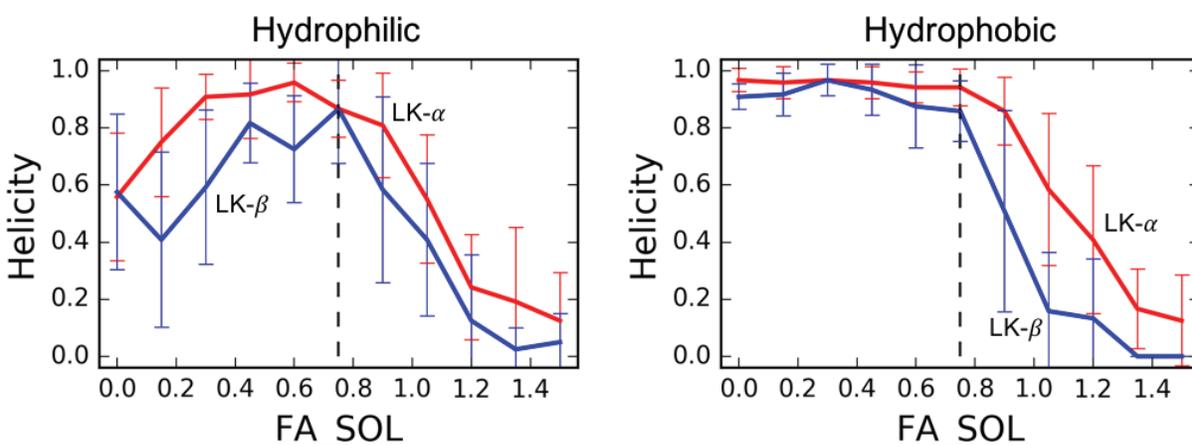


Figure 17. LK peptide helicity versus the implicit solvation energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.75) of the FA_SOL term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

To test this hypothesis, I ran a series of simulations where the weight of the implicit solvation energy term (FA_SOL) in the linear combination of energy terms was varied across a range of values from 0% to 200% of the default weight for the energy term. The weights of all other terms in the energy function were held constant at their default values. As seen in Figure 17, the α -helical content is indeed decreased at higher implicit solvation energy weights as burial of the hydrophobic surface and nonpolar leucines. At very high weights, favorable solvation of the polar backbone of the peptide outweighs the favorable hydrogen bonding and van der Waals interactions maintaining an α -helical conformation, regardless of peptide sequence.

In order for RosettaSurface structural predictions to match experiment, I would expect to see an intermediate weight range where LK- α maintains high helicity but LK- β does not. However, although discrimination between LK- α and LK- β can be improved slightly on the hydrophobic SAM surface by choosing a higher implicit solvation weight, there is no weight that captures the experimental result of LK- α adopting a helical conformation and LK- β adopting an extended conformation on both surfaces. At very low solvation score weight, attractive Lennard-Jones force becomes the dominant score contribution, accompanied in the case of the hydrophilic SAM surface by a rise in electrostatic contributions as well. These changes correspond to a drop in helicity on the hydrophilic SAM surface and an increase on the hydrophobic SAM, supporting the expectation that van der Waals forces promote helix formation, and suggesting that electrostatic interactions on the hydrophilic SAM favor pulling the helix apart and exposing more of the backbone to the surface.

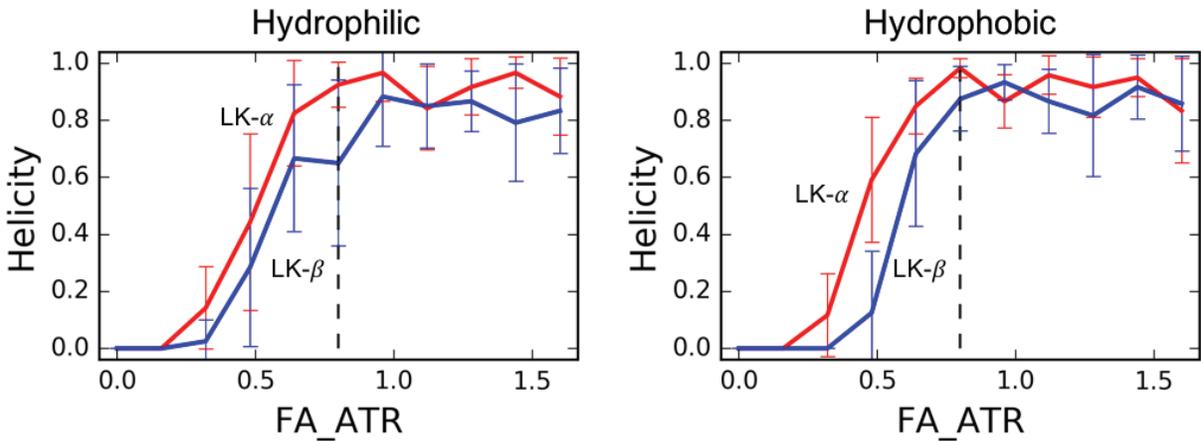


Figure 18. LK peptide helicity versus the attractive Lennard-Jones energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.8) of the FA_ATR term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

To explore the effect of other energy function terms on the helicity of LK- α and LK- β , I ran similar series of simulations, varying the energy term weights for each of the major contributing terms in the RosettaSurface energy function.

Unsurprisingly, the attractive component of the Lennard-Jones energy favors compact α -helices at high weights and extended structures at low weights (Figure 18). The repulsive component of the Lennard-Jones energy has little impact on overall helicity at most weights (Figure 19). At a weight of zero, the results are non-physical, as the atoms are allowed to superimpose.

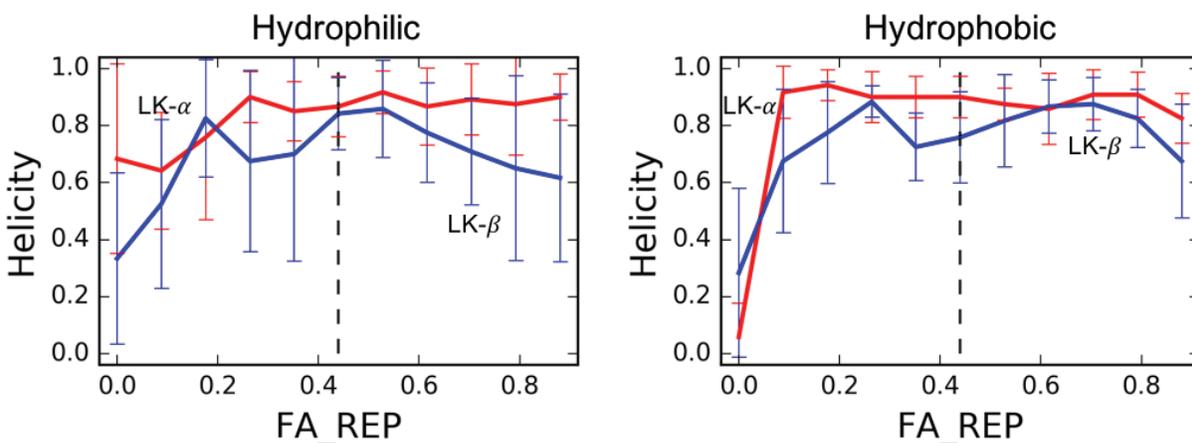


Figure 19. LK peptide helicity versus the repulsive Lennard-Jones energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.44) of the FA_REP term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

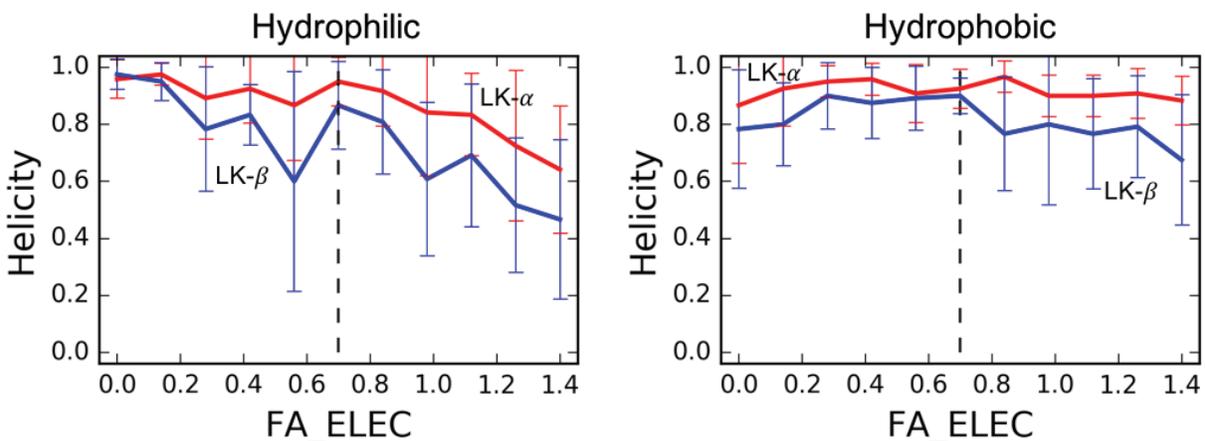


Figure 20. LK peptide helicity versus the electrostatic energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.7) of the FA_ELEC term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

The electrostatic energy, calculated using a truncated Coulomb model with a distance dependent dielectric, has little impact on predicted helicity on the hydrophobic SAM surface, but as expected, decreases helicity on the hydrophilic SAM surface when weighted higher (Figure 20). This indicates that the attractive electrostatic interaction between polar backbone atoms in an α -helical conformation is offset by the repulsive interaction between positively charged lysine residues that are placed in close proximity to one another by the α -helical conformation.

Increasing the weight on the hydrogen bonding term favors α -helical conformations (Figure 21), likely because it strengthens the backbone-backbone hydrogen bonds that define an α -helical structure, though the effect is not as dramatic as it was for solvation and attraction weights.

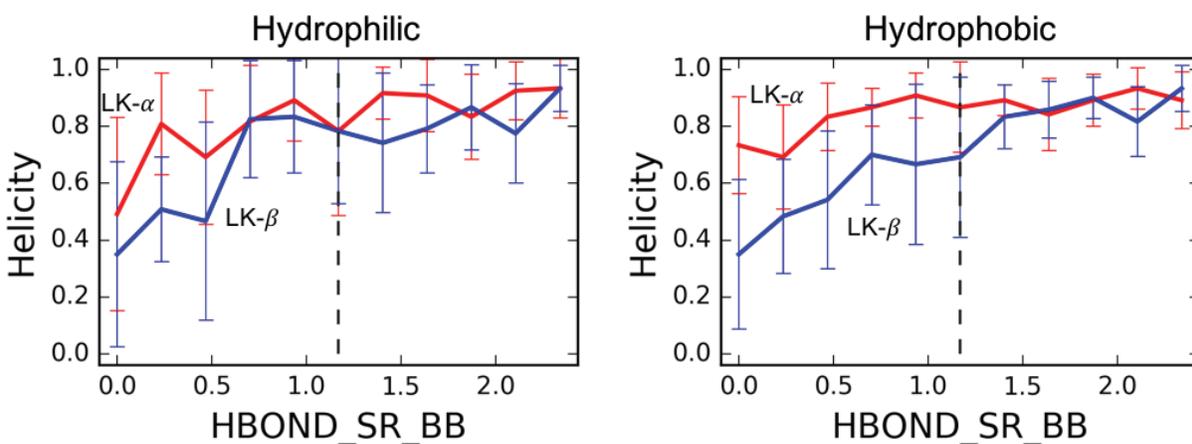


Figure 21. LK peptide helicity versus the short-range backbone-backbone hydrogen bonding energy term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (1.17) of the HBOND_SR_BB term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

5B: Single Knowledge-Based Parameter Score Weight Variations

Besides the variations of the physics-based energy terms described above, I also examined the effect of the weights of the knowledge-based score terms. Trends in the Protein Data Bank used in generating the knowledge based terms may be less applicable to surface-adsorbed proteins, since such structures are absent in the PDB. Therefore, I examined the effect of varying several knowledge-based score weights to see whether they would affect LK peptide helicity, as a significant impact would suggest an increased likelihood that they are not applicable.

I ran the variations in this section using capped peptides earlier in my research, and did not re-create them with N-capped peptides since, as observed in 4A: Alternate Peptide, capping did not significantly alter the helicity or orientation trends, and single parameter variations with capped or uncapped peptides duplicating those in the previous section were similar (not shown).

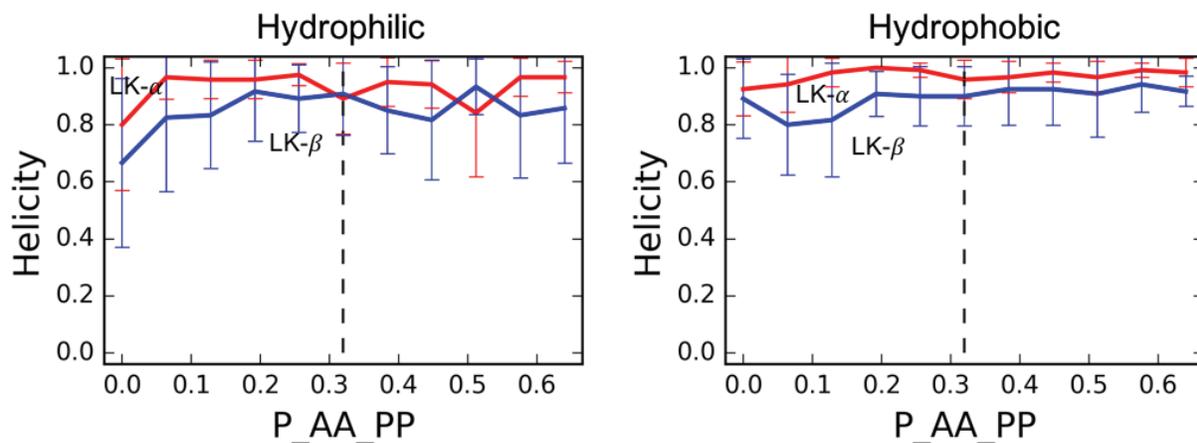


Figure 22. LK peptide helicity versus the probability of amino acid based on backbone dihedrals score term weighting (P_{AA_PP}). Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.32) of the P_{AA_PP} term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

$P(\text{aa}|\phi, \psi)$, (written as $P_{\text{AA_PP}}$ in Rosetta) is the statistical probability (P) that a given amino acid (AA) will be the type of residue that has the observed backbone ϕ/ψ (PP) conformation, compared to other amino acids. This term is related to the Ramachandran potential through Bayes' rule⁶². I thought there was a chance that the weight of this term in the score function might affect helicity because of a difference in helical tendency between leucines and lysines. However, no significant effect increasing helicity was observed (Figure 22).

Like $P_{\text{AA_PP}}$, the FA_DUN term is knowledge-based, capturing the side chain conformational statistics. This term scores the consistency of the side chain dihedral angles (χ) with the rotamers observed in the Dunbrack rotamer library⁵⁹. I tested whether the conformation of the side chains on the surface made a significant difference to peptide helicity. It does not (Figure 23).

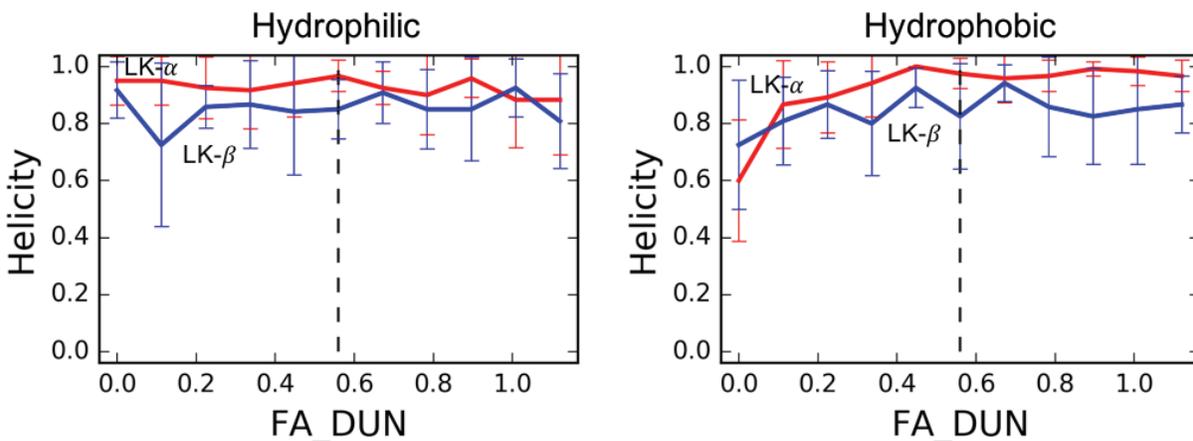


Figure 23. LK peptide helicity versus the side-chain dihedral probability score term weighting. Red: LK- α . Blue: LK- β . The dashed line indicates the default weight (0.56) of the FA_DUN term in the RosettaSurface energy function. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

5C: Paired Physical Parameter Score Weight Variation

None of the single score weight parameter variations produced a condition with a large discrimination between the helicities of LK- α and LK- β . However, scaling both implicit solvation energy and attractive Lennard-Jones potential was able to produce the full range from completely helical to completely non-helical. Thus, I sought to determine whether simultaneous variation of the two could produce some condition of improved discrimination synergistically.

I confined the scanned weight range to between 0.5 and 1.0 for both parameters, corresponding to roughly a 30% decrease and increase, respectively, because in the case of both score parameters, that was the range where much of the helicity transition occurred. Outside these ranges, the peptides tended toward complete helicity or non-helicity, suggesting that larger or smaller weights are less physically reasonable. The simulation used only the hydrophobic surface and uncapped peptides, and for the reasons mentioned above, I did not attempt to re-create the simulation with capped peptides or on the hydrophilic surface. This simulation set used the top 100 of 10,000 structures (rather than 10 of 1,000) to reduce the impact of random variability.

The combined parameter variation (Figure 24) did not sufficiently improve helicity discrimination to capture experimental results. For comparison, the baseline Talaris2013 score weights yielded a discrimination between the helicities of LK- α and LK- β of approximately 10%. Decreasing the weight of solvation energy and increasing the weight of attractive potential (top left corner) reduced discrimination to 4%. Increasing both parameter weights yielded a discrimination of 11%, while decreasing both yielded a discrimination of 17%. Increasing solvation weight and decreasing attractive weight yielded a discrimination of 13%. The highest discrimination observed in this two-parameter scan was 29%, corresponding with only an increase in solvation energy weight. This result matches the one shown in Figure 17, that increasing

solvation energy weight produces a small increase in helicity discrimination between LK- α and LK- β .

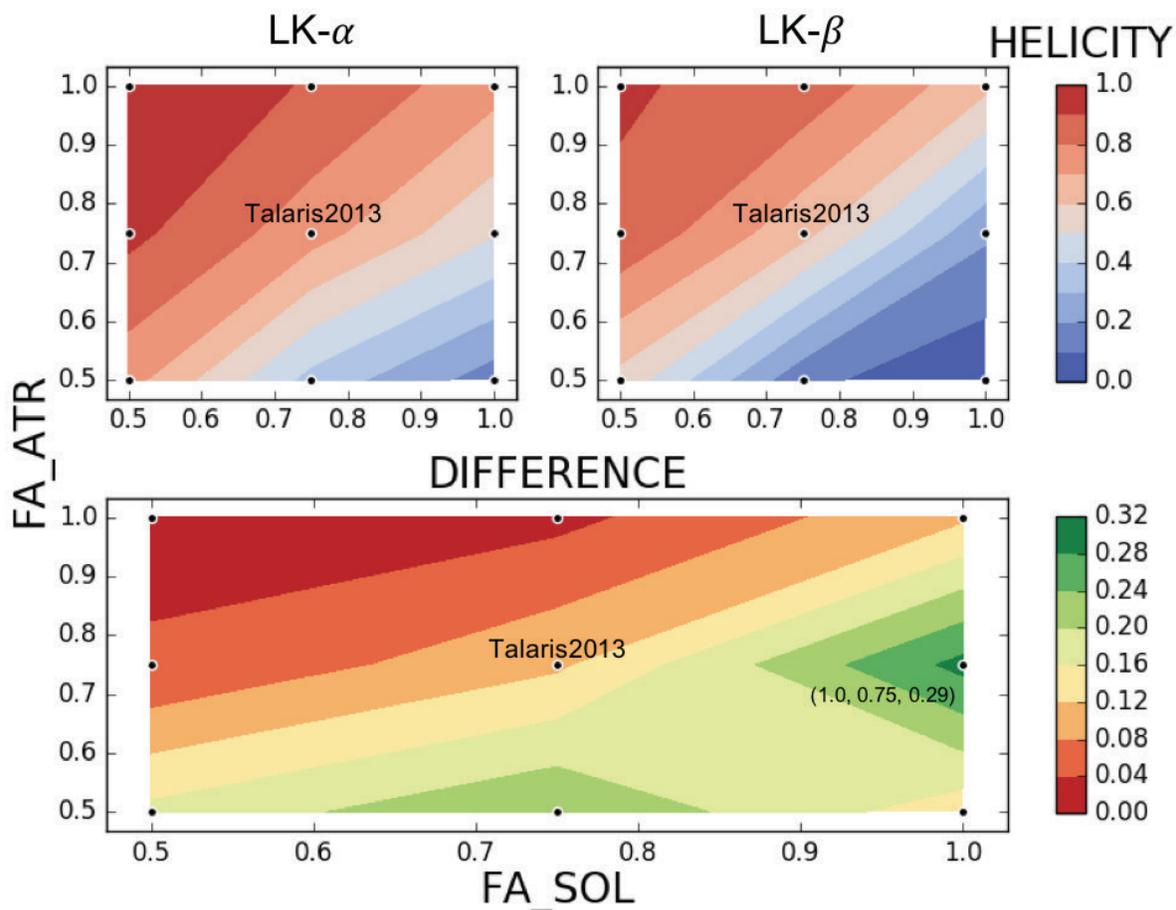


Figure 24. LK peptide helicity heat maps with implicit solvation and Lennard-Jones attraction energy score weights. The default Talaris2013 value is (0.75, 0.8). Top left: The helicity of LK- α . Top right: The helicity of LK- β . Bottom: The point-by-point difference between the two upper plots. Each black point represents the average helicity of the 100 lowest energy structures out of 10,000 generated structures. This simulation used uncapped 14-mer peptides on the hydrophobic SAM surface.

Chapter 6: Results, Atom Parameter Variations

6A: Lysine Layout and LK_DGFREE Cross Plot

In a number of the plots, there are relatively large error bars that result from averaging structures that are fully helical with structures that are fully non-helical. That is, RosettaSurface finds both helical and non-helical structures with similar low energies. In order to investigate the lack of discrimination between LK- α and LK- β on the hydrophobic surface, I visually inspected

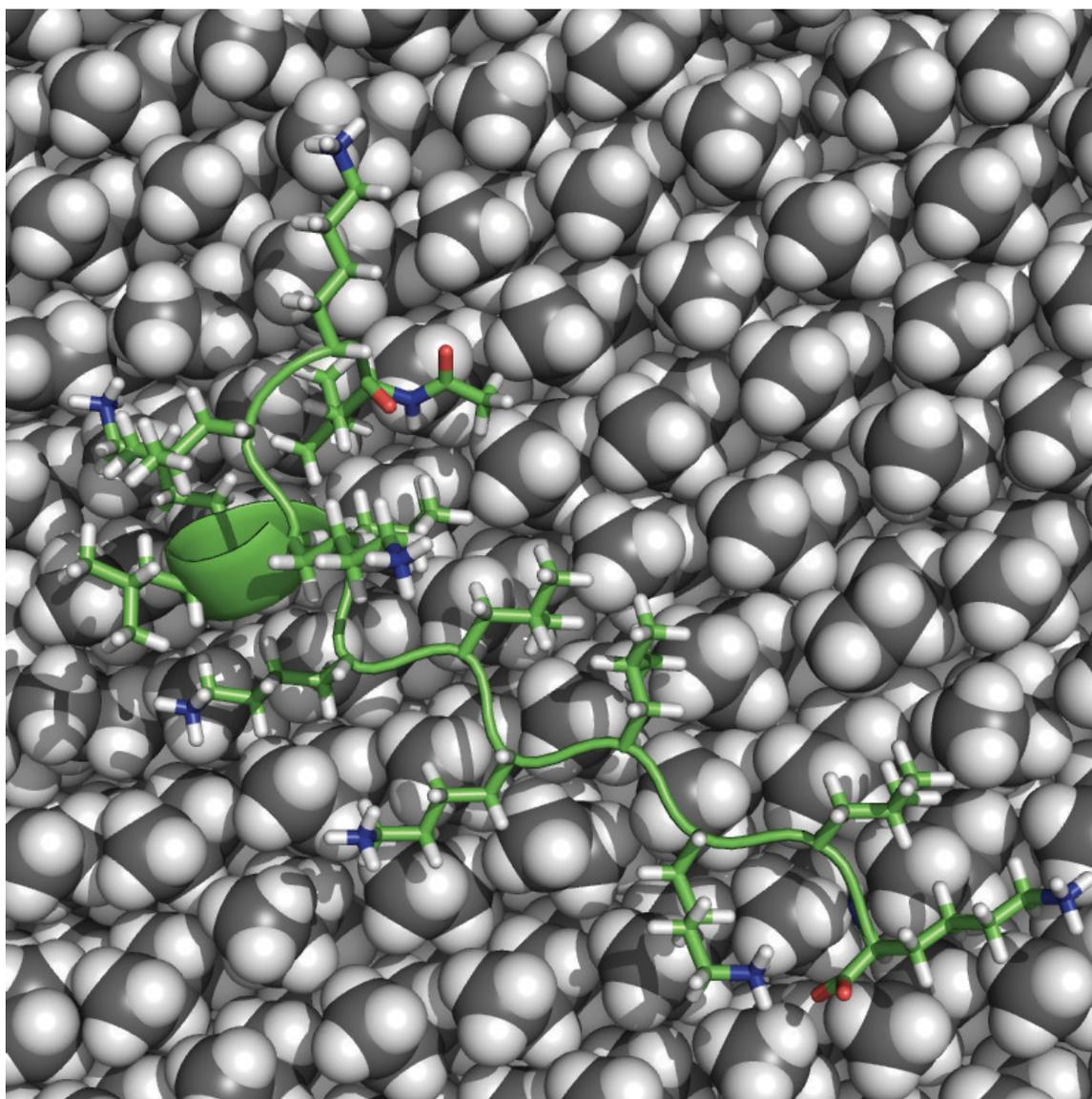


Figure 25. A low energy structure of LK- β on a hydrophobic SAM with multiple lysine side chains lying flat against the surface.

low-energy structures for LK- β and observed that in many structures lysine residues lie flat against the hydrophobic surface. Lysine is a hydrophilic residue with a charged primary amine and should prefer to be solvated; however, in these structures lysine appeared to be making higher quality contacts with the hydrophobic surface than leucine. This observation was corroborated by single amino acid docking studies (not shown) that predicted lysine to have a higher affinity for the hydrophobic surface than leucine did, by approximately 1 kcal/mol. Analysis of atom-pair energies between the lysine and the hydrophobic surface revealed that by angling the polar nitrogen at the end of the lysine side chain away from the hydrophobic surface it was considered fully solvated and not penalized for being in proximity to the surface. An example of this type of lysine structure is shown in Figure 25.

The prevalence of this adsorbed lysine conformation suggested a shortcoming in the parameters used by the implicit solvation model in RosettaSurface. Rosetta's solvation model, EEF1²⁹, is a Gaussian solvent exclusion model that approximates solvation free energy by evaluating the degree that solvent is excluded from surrounding a particular atom. The ΔG^{free} (LK_DGFREE in Rosetta) atom property controls the magnitude of the energetic penalty/reward applied when an atom is excluded from solvent.

I considered two possibilities to correct the shortcoming. The first consideration was an effort to correctly penalize lysine's nitrogen for interacting with the hydrophobic SAM with increased hydrophobicity of the hydrophobic SAM surface by altering the $\Delta G_{\text{Surface CH}_3}^{\text{free}}$ property of the carbon atoms composing the alkane-thiol methyl head groups. My second approach was to reduce the affinity of the CH₂ groups for the hydrophobic surface. Inspection of the ΔG^{free} parameters for the atoms composing lysine provided a potential explanation for the persistence of the behavior I observed: while the terminal polar nitrogen is considered hydrophilic ($\Delta G^{\text{free}} =$

-20), all of the side chain carbons that make up the lysine tail are uniformly hydrophobic ($\Delta G^{\text{free}} = 0.52$). With new $\Delta G_{\text{Lysine } C_{\delta}}^{\text{free}}$ and $\Delta G_{\text{Lysine } C_{\epsilon}}^{\text{free}}$ atom properties, I could make those residues hydrophilic, reflecting their proximity to the nitrogen and favoring solvation over contact with the hydrophobic surface. These changes used the `-chemical:set_atom_properties` flag, and did not require any alterations to the Talaris2013 score weights.

I tested a hypothesis that increasing the hydrophilicity of the lysine side chain carbons while simultaneously increasing the hydrophobicity of the surface would force lysine away from the hydrophobic surface and correct the anomalous behavior. I chose to vary C_{δ} only 10% as much as C_{ϵ} , reflecting its greater distance from the nitrogen. Consequently, when the $\Delta G_{\text{Lysine } C_{\epsilon}}^{\text{free}}$ was -10 REU, $\Delta G_{\text{Lysine } C_{\delta}}^{\text{free}}$ was -1 REU. The results of the two-parameter variation simulations, assessing the impact of both increasing the hydrophobic SAM surface's hydrophobicity and making the C_{δ} and C_{ϵ} of lysine hydrophilic, are shown in Figure 26.

Increasing only the surface hydrophobicity without altering the parameters of lysine produced peak discrimination at $\Delta G_{\text{Surface } CH_3}^{\text{free}} = 8.5$ REU, yielding average helicities of 68% and 40% for LK- α and LK- β , respectively. This increased secondary structure discrimination to 28%, compared to the default values which discriminated at 10%. However, as surface hydrophobicity increases, unfolding of both peptides on the surface becomes favored in order to maximize solvent exclusion from the SAM surface. Thus, very large surface hydrophobicity produced unfolded peptides regardless of alterations to lysine, and the peptide backbone flattened to the surface.

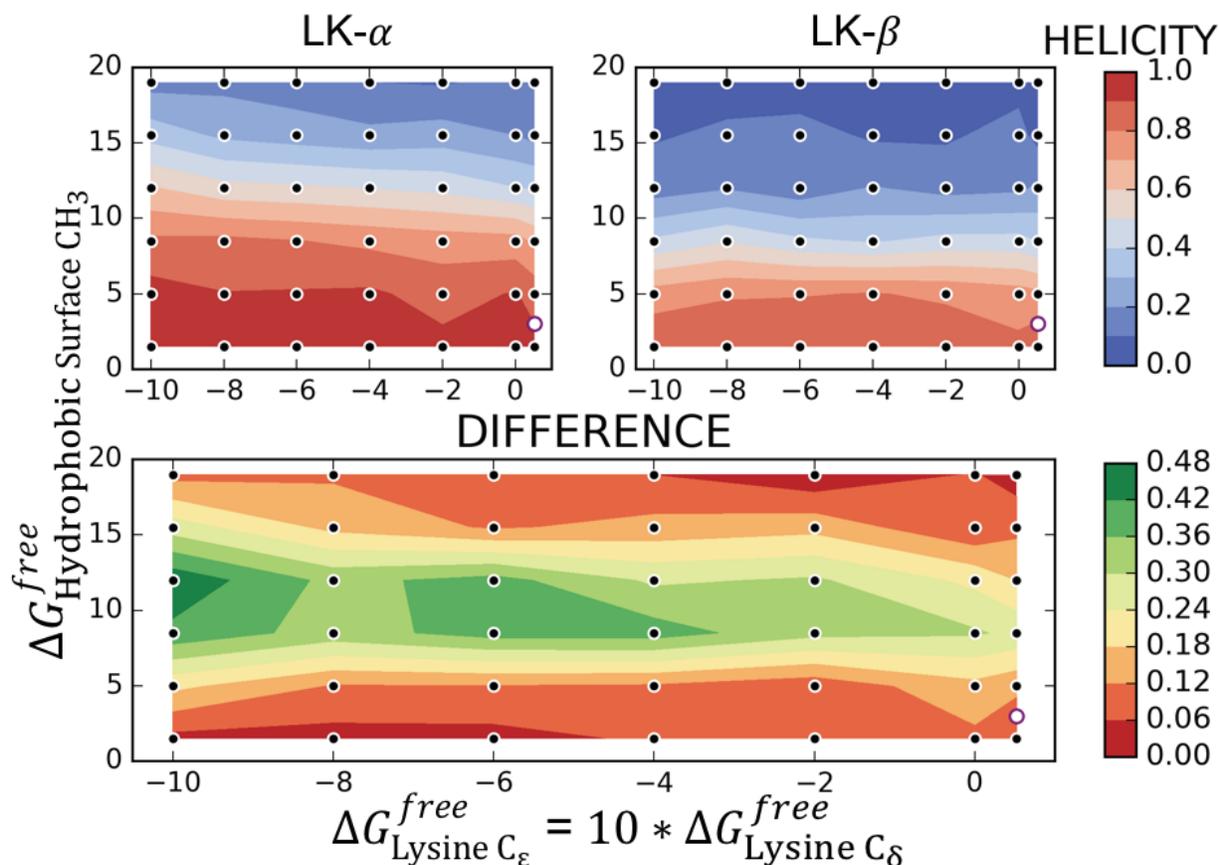


Figure 26. LK peptide helicity heat maps with varied surface methyl head group hydrophobicity and lysine C δ and C ϵ hydrophilicity. Top left: The helicity of LK- α . Top right: The helicity of LK- β . Bottom: The point-by-point difference between the two upper plots. Each black point represents the average helicity of the 100 lowest energy structures out of 10,000 generated structures. The white point shows the default LK_DGFREE values.

My results show that secondary structure discrimination above 30% was not achievable while C δ and C ϵ were hydrophobic. Increasing C δ and C ϵ hydrophilicity within reasonable range did not make as large an impact on discrimination as the SAM surface hydrophobicity. LK- α was less susceptible to uncoiling at elevated surface hydrophobicity than LK- β when the lysine side chain became hydrophobic. LK- α had a greater contribution of the attractive Lennard-Jones force, while solvation contributed more significantly to the LK- β scores, indicating that the altered

surface and lysine parameters put the forces favoring and disfavoring helicity in closer balance.

This combination produced the improved discrimination at $\Delta G_{\text{Surface CH}_3}^{\text{free}} = 12.0 \text{ REU}$, $\Delta G_{\text{Lysine C}_\epsilon}^{\text{free}} = -10 \text{ REU}$, $\Delta G_{\text{Lysine C}_\delta}^{\text{free}} = -1 \text{ REU}$.

6B: Best Discrimination Case

Table 1. Comparison of default and best discrimination simulation conditions. Average helicities are calculated from the 100 best scoring models out of 10,000.

	Default	Best discrimination
LK- α average helicity	91%	61%
LK- β average helicity	81%	15%
$\Delta G_{\text{Surface CH}_3}^{\text{free}}$	3.0 REU	12.0 REU
$\Delta G_{\text{Lysine C}_\beta, \text{C}_\gamma}^{\text{free}}$	0.52 REU	0.52 REU
$\Delta G_{\text{Lysine C}_\delta}^{\text{free}}$	0.52 REU	-1.00 REU
$\Delta G_{\text{Lysine C}_\epsilon}^{\text{free}}$	0.52 REU	-10.00 REU
$\Delta G_{\text{Lysine N}_\zeta}^{\text{free}}$	-20.00 REU	-20.00 REU

The highest discrimination between LK- α and LK- β helicity was achieved with the LK_DGFFREE parameters for the δ and ϵ carbons of lysine shown in Table 1. These values correspond to the leftmost point, third from the top in Figure 26. With these parameters, LK- α is predicted to be 46% more helical than LK- β . This result is the closest any set of parameters I tried has come to agreeing with experimental results.

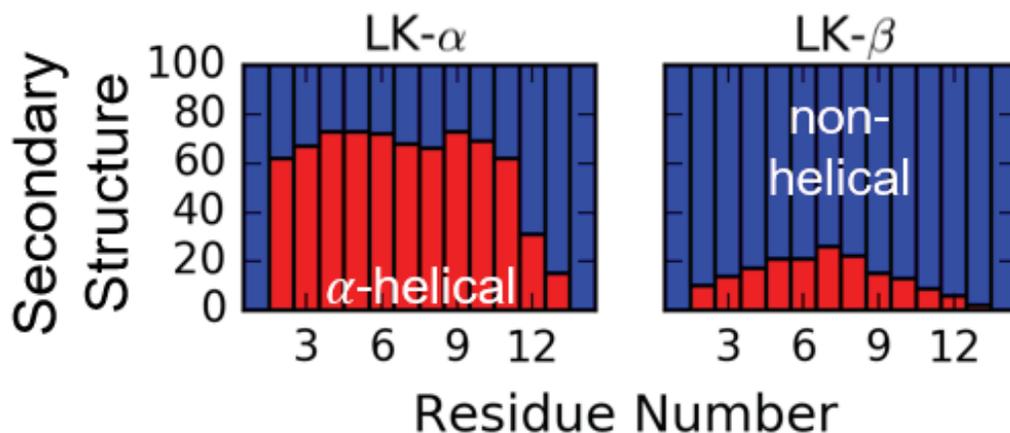


Figure 27. Secondary structure distribution of top 100 low-scoring decoys from the maximum secondary structure discrimination case. The count of helical residues at each position in the peptide is shown in red, while extended residues are shown in blue.

The full data from this best-parameters simulation are shown in Figure 27-Figure 30, which are comparable to the lower pair of plots in the corresponding Figure 6-Figure 9. Secondary structure distributions (Figure 27) show significant reduction of helicity in LK- β , compared to the baseline case (Figure 6). However, there was also a smaller reduction in the helicity of LK- α . The

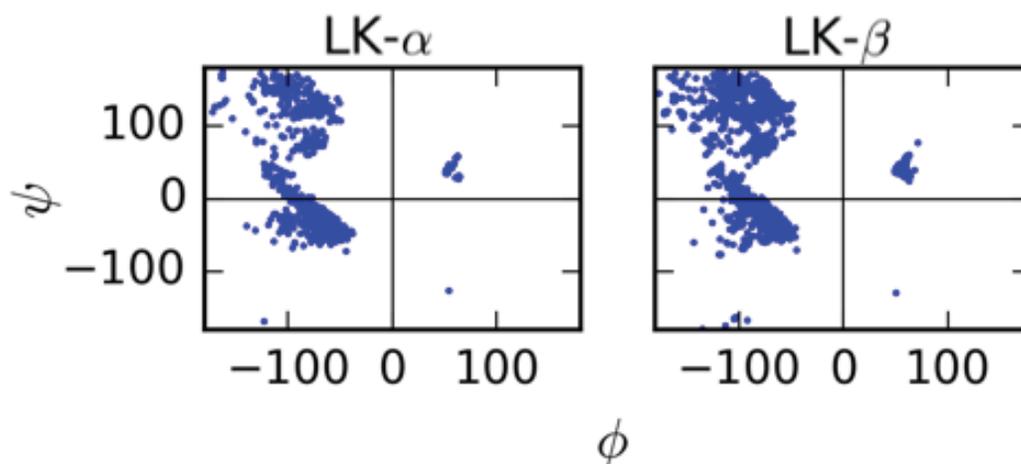


Figure 28 . Ramachandran plots showing the ϕ and ψ backbone dihedral angles for all non-terminal residues in the top 100 decoys from the maximum secondary structure discrimination case.

Ramachandran plot (Figure 28) shows that even with the altered atom parameters, the backbone

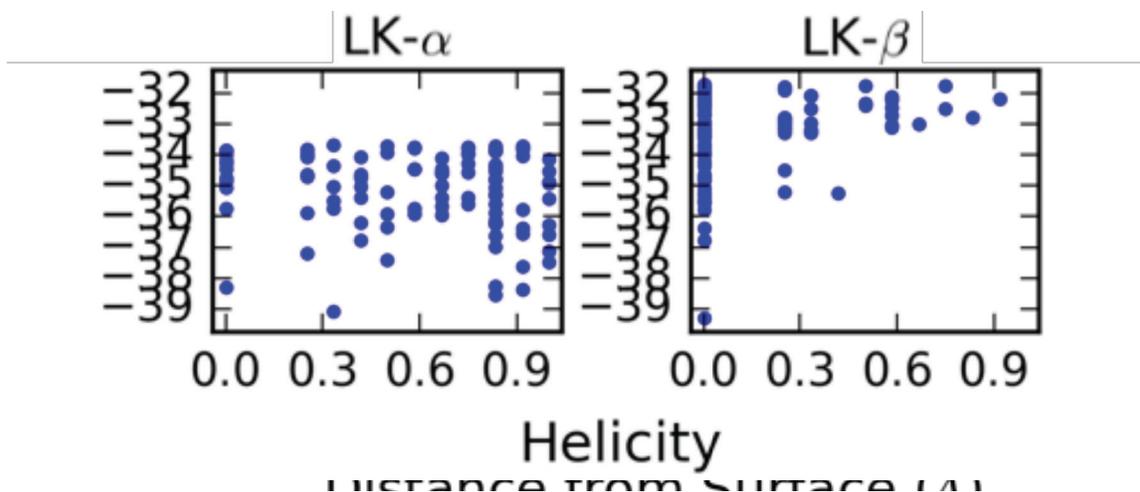


Figure 29. Score vs. helicity distribution of the top 100 decoys from the maximum secondary structure discrimination case.

structure discrimination case.

dihedrals conform to protein-like configurations, though with increased presence in the β -sheet region, compared to the baseline case (Figure 7).

The score plots (Figure 29) show distinct favorability of non-helical LK- β structures, results much more consistent with the experimental results than those obtained with unmodified atom properties. However, in this case, no clear pattern arises indicating that LK- α has found its native structure, meaning that with these parameters, RosettaSurface's score function does not have a deep energy well for the expected helical structures, hence the reduction in helicity compared to the default case.

The distance distribution plots (Figure 30) reveal that the population of leucines oriented away from the hydrophobic surface has been reduced to nearly none and all peptides are fully in contact with the surface, but lysine distribution continues to include some atoms with close proximity to the surface, and because of the high reward for covering the surface, some lysine

layout persists. Both peptides show significant C_{α} contact with the surface, which explains the reduced helicity in LK- α .

6C: Other Atom Property Variations

In addition to the two-variable LK_DGFREE cross, I conducted single parameter variations for other atom properties of the hydrophobic surface methyl termini, as alternate approaches to resolve the lysine layout issue mentioned in section 6A: Lysine Layout and LK_DGFREE Cross Plot.

The Lennard-Jones well depth represents how favorable the low-energy state is, when particles are at optimal proximity. Increasing its value (which is an atom property value, not a scoring weight) makes residues near the surface adhere more strongly (Figure 31). Significantly increasing the value of this property caused a reduction in helicity of both peptides, as the backbones tended to lay flat on the surface. There appears to be some increase in discrimination in the middle range, rising to 32% at an $\epsilon_{\text{Surface CH}_3}$ (LJ_WDEPTH in Rosetta) value of 0.4. This surface property is another promising target for future optimization. I did not test this in a two-

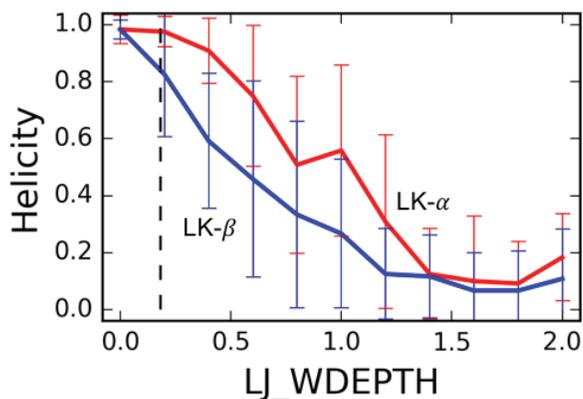


Figure 31. LK peptide helicity versus the hydrophobic surface methyl Lennard-Jones well depth atom property value. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (0.1811) of the LJ_WDEPTH property of the surface CH₃ atom types in Rosetta. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

variable trial because I expected that the synergy with other parameters would be less significant than in the case of the two LK_DGFREE properties examined in the previous section.

Since the attractive Lennard-Jones potential is dependent on the well depth, it seems counterintuitive that increasing the atom property value would have the opposite effect on helicity to that of increasing the attractive potential's score weight. The reason for this difference is that in the case of the atom property, only the surface methyl groups are affected, whereas all atoms are affected by adjusting the score weight, thus all atoms are subject to Van der Waals attractions to all other atoms except the SAM surface. At a zero LJ_WDEPTH value, the remaining score contribution from FA_ATR can be compared with default FA_ATR scores (not shown) to determine the significance of just the Van der Waals attraction to the surface. The difference is <10%, indicating that the most significant portion of attractive interactions, which were shown in Figure 18 to strongly influence helicity, is intramolecular, and independent of the surface.

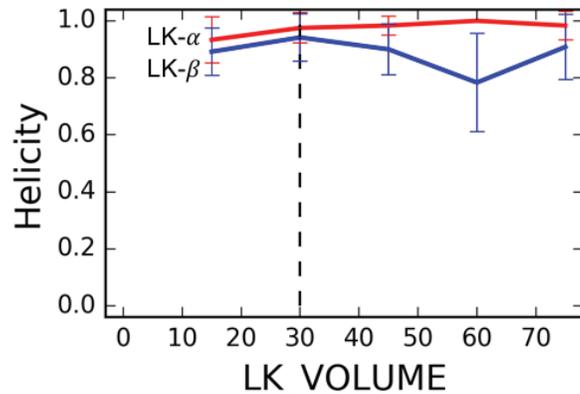


Figure 32. LK peptide helicity versus the hydrophobic surface methyl Lazaridis-Karplus solvation volume atom property value. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (30.0) of the LJ_WDEPTH property of the surface CH₃ atom types in Rosetta. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

In many cases of lysine layout, the lysine C _{ϵ} was on the surface, and the nitrogen of lysine was tipped up, just over 5Å from the surface. I varied the Lazaridis-Karplus volume ($V_{\text{Surface CH}_3}$, LK_VOLUME in Rosetta), which reflects the zone of influence for solvation of an atom, hypothesizing that increasing the volume might push the nitrogen farther from the surface and prevent side-chain adsorption. However, the helicities of LK- α and LK- β were largely independent (Figure 32).

Another approach to pushing the nitrogen off the surface was to increase the maximum range at which Rosetta calculates two-body interactions, since the default is 5Å, and the nitrogens were between 5Å and 6Å from the surface. I tested this hypothesis using the `-score::fa_max_dis` flag. Again, the effect on helicity was not appreciable (Figure 33).

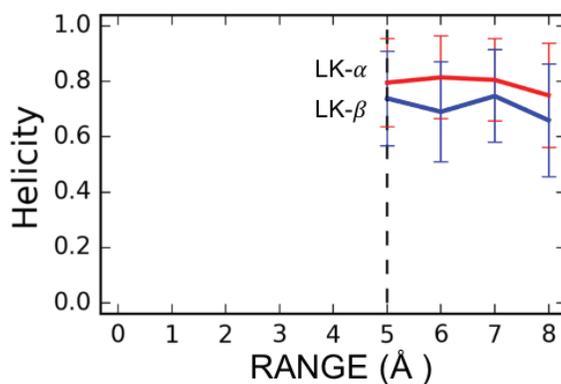


Figure 33. LK peptide helicity versus the two-body interaction cutoff range used for calculating Rosetta energies. Red: LK- α . Blue: LK- β . The dashed line indicates the default value (5) of the detection range in Rosetta. Each point represents the average helicity of the 10 lowest energy structures out of 1000 generated structures. Error bars represent one standard deviation of the helicity in the 10 lowest-energy structures.

Chapter 7: Conclusions

This chapter has elements in common with chapter 4 of Michael S. Pacella's doctoral dissertation, "Modeling and Design of Peptides to Control Biomineral Nucleation and Growth," Johns Hopkins University, Baltimore, Maryland, Copyright 2017 Michael Steven Pacella. Adapted portions are reproduced with permission.

The LK peptide/SAM system is an excellent test case for tuning the parameters of the RosettaSurface energy function. Computational results can be generated quickly and compared quantitatively with experimental results on peptide structure from SFGS and NEXAFS. As my simulations were capable of producing both completely extended and completely helical structures, I conclude that for this small system, scoring, not sampling, is the factor that limits the ability of the RosettaSurface algorithm to successfully reproduce experimental results, which simplifies the task of refining energy function parameters. My simulations reveal the sensitivity of various terms in the RosettaSurface energy function. Additionally, these simulations provide information on which energy terms control the transition from a compact folded state, such as an α -helix, to an extended unfolded state, such as a single β -strand. As a result, these findings will not only help to improve RosettaSurface, but potentially other algorithms utilizing the Rosetta energy function.

Using the default energy function, RosettaSurface successfully captures peptide secondary structure and side chain orientation for LK- α on both SAM surfaces, and produces some correct structures for LK- β on the hydrophilic SAM surface. LK- β adsorbs on the hydrophobic SAM with an α -helical secondary structure and a significant number of lysine residues oriented towards the hydrophobic surface. The cause is, in part, the higher favorability of lysine-surface interactions than leucine-surface interactions, which results in a tendency of lysine to lay flat and pull the peptide backbone into contact with the surface. In an effort to alleviate this discrepancy with experiment, I performed a parametric analysis of the RosettaSurface energy function. I calculated

average helicities for both peptides on both SAM surfaces across a range of weights for each of the major terms in the energy function. I observed a strong dependence of helicity on both implicit solvation energy weight and attractive Lennard-Jones energy weight.

After performing the parametric analysis, an inconsistency became obvious. Adsorption of lysine on the hydrophobic SAM is favored over leucine due to favorable interactions between the hydrophobic carbons on the lysine tail and the hydrophobic methyl head groups on the methyl-terminated SAM. (This problem is expected to be more prominent in the new REF_2015 energy function, due to a more significant underestimation of the lysine solvation penalty³⁰. Further work is being conducted by Park and coworkers, investigating improvements to scoring by using different atom types for lysine, as I did here with C_δ and C_ε.) To correct this inconsistency, I varied LK_DGFREE, a parameter that controls the hydrophobicity/hydrophilicity of an atom type for both the methyl head groups and the delta and epsilon carbons on lysine. I discovered a combination of increasing methyl head group hydrophobicity and lysine carbon hydrophilicity yielded computational predictions of my exception case that were in closer agreement with experimental results on the hydrophobic SAM surface.

Although tuning the RosettaSurface energy function improved accuracy considerably, the final results are not completely consistent with experiment. The final results on the hydrophobic surface show that LK- α is predicted to be 46% more helical than LK- β , with the majority (but not all) leucines of both peptides oriented towards the hydrophobic SAM surface. While it is uncertain how sensitive the experimental measurements of Castner and coworkers are to structural heterogeneity of adsorbed peptides, the predicted 40% non-helical content for LK- α would likely produce an observable differences in NEXAFS spectra. This not being the case suggests that a discrepancy remains between RosettaSurface and experimental measurements.

Multiple factors may limit the ability of the RosettaSurface energy function to correctly reproduce experimental results. The EEF1 implicit solvation model may be insufficient to capture the energetics of water at the LK peptide/hydrophobic SAM interface. Polar solvation of residues such as lysine is known to have a directional dependence that is lost in the Gaussian solvent exclusion model of EEF1⁶³. The observation of low-scoring models that partially separated from the surface indicates that the balance between scoring rewards for adsorption and those favoring solvation is not optimal.

Additionally, lateral interactions between peptides within an adsorbed monolayer may be important. The data from Castner and coworkers suggests that both LK- α and LK- β formed a densely-packed monolayer on both methyl-terminated and carboxy-terminated SAMs, which was sufficient to prevent water from permeating the surface, and MD simulations by Latour and coworkers (though not those by Deighan and Pfaendtner⁵⁵) included a second LK- β peptide. Hydrogen bonding between adjacently adsorbed LK- β peptides would stabilize an extended β -strand structure. Meanwhile, lateral van der Waals interactions between adjacently adsorbed α -helices would not significantly reduce the stability of the α -helical conformation. Thus, the importance of lateral interactions between LK- α and LK- β is one factor that could account for the deviation from experiment in the case of LK- β on the hydrophobic SAM. RosettaSurface does not presently have the capability to simultaneously dock multiple peptides on a surface, and this is a potential direction for future work.

The ability of the RosettaSurface algorithm to reproduce experimental results on three of the four LK peptide/SAM systems investigated by Castner and coworkers is comparable to two other recent computational studies. In the replica-exchange molecular dynamics study conducted by Latour and coworkers³⁹ the most accurate force field tested (CHARM22) successfully

reproduced experimental secondary structures and leucine side chain orientations in three of the four LK peptide/SAM surface combinations. Unlike my work, in their study, the CHARMM22 force field failed to produce extended β -sheet structures on the hydrophilic carboxy-terminated surface. It should be noted that MD force fields have undergone refinements since these studies were done in 2012 (the current CHARMM version is 36⁶⁴). In the metadynamics/umbrella sampling molecular dynamics study performed by Deighan and Pfaendtner⁵⁵, the AMBER99SB force field, when utilized with metadynamics, accurately reproduced secondary structures and leucine side chain orientations from Castner and coworkers for all four peptide-surface combinations, utilizing varied temperatures and solvent strengths. Although both molecular dynamics studies discussed here utilized enhanced sampling strategies (replica exchange in the case of Latour and coworkers, metadynamics in the case of Deighan and Pfaendtner) the computational resources needed to generate their results, 100,000 to 150,000 CPU hours, far exceeded the resources required by RosettaSurface. Indeed, my parametric analysis, which involved running 11 simulations per peptide-surface combination per parameter scan would not have been possible using a more computationally intensive all-atom molecular dynamics strategy. The comparable accuracy observed in all three studies suggests that the simplifications in RosettaSurface (implicit solvent, static surface) may be useful in modeling peptide adsorption to save computational resources. Future development of a surface-optimized Rosetta score function, and development and usage of a multi-body RosettaSurface docking algorithm, may help further. Such a development will require a larger benchmarking set, since testing and training on only the LK-SAM system may result in overfitting of data.

This study suggests that RosettaSurface can complement experimentation for study of proteins on surfaces. The LK peptide/SAM system is not a mineral system or directly related to

natural processes, but was used experimentally for its compatibility with NEXAFS and SFGS. The relatively large particle size of most biominerals causes light scattering that renders structural determination through many spectroscopic techniques such as SFGS impossible. The long-term goal for RosettaSurface is to develop a general tool to supplement experiments, allowing for structural determination and design of proteins involved in biomineralization and surface ordering.

Appendix 1: Table of Altered Parameters

Variable	Rosetta Parameter	Rosetta Location	Default Value	Alteration Range
Gaussian exclusion implicit solvation energy weight	FA_SOL	database/scoring/ weights/talaris2013.wts	0.75	0 - 1.5
Lennard-Jones attractive energy weight	FA_ATR	database/scoring/ weights/talaris2013.wts	0.8	0 - 1.6
Lennard-Jones repulsive energy weight	FA_REP	database/scoring/ weights/talaris2013.wts	0.44	0 - 0.88
Coulomb electrostatic interaction energy weight	FA_ELEC	database/scoring/ weights/talaris2013.wts	0.7	0 - 1.4
Short range backbone hydrogen bonding energy weight	HBOND_SR_BB	database/scoring/ weights/talaris2013.wts	1.17	0 - 2.34
Probability of amino acid identity, given backbone ϕ , ψ angles score weight	P_AA_PP	database/scoring/ weights/talaris2013.wts	0.32	0 - 0.64
Probability of rotamer, given backbone ϕ , ψ angles score weight	FA_DUN	database/scoring/ weights/talaris2013.wts	0.56	0 - 1.12
$\Delta G_{\text{Surface CH}_3}^{\text{free}}$	LK_DGFREE CH3S	database/chemical/ atom_type_sets/fa_standard/ atom_properties.txt New atom type	1.5	1.5 - 19

Variable	Rosetta Parameter	Rosetta Location	Default Value	Alteration Range
$\Delta G_{\text{Lysine } C_{\delta}}^{\text{free}}$	LK_DGFREE KCD	database/chemical/ atom_type_sets/fa_standard/ atom_properties.txt New atom type	0.52	-1 - 0.52
$\Delta G_{\text{Lysine } C_{\epsilon}}^{\text{free}}$	LK_DGFREE KCE	database/chemical/ atom_type_sets/fa_standard/ atom_properties.txt New atom type	0.52	-10 - 0.52
$\epsilon_{\text{Surface } CH_3}$	LJ_WDEPTH CH3S	database/chemical/ atom_type_sets/fa_standard/ atom_properties.txt New atom type	0.1811	0 - 2
$V_{\text{Surface } CH_3}$	LK_VOLUME CH3S	database/chemical/ atom_type_sets/fa_standard/ atom_properties.txt New atom type	30	15 - 75
Detection Range	FA_MAX_DIS	source/src/basic/options/ options_rosetta.py	5	5 - 8

Appendix 2: Code Development

To enable the generation and analysis of the data presented in this thesis, I developed the following scripts, which enabled the generation and analysis of the mass of data included in the publication. While I think there are ways the scripts could be further improved and refactored, they are probably my most essential accomplishments in enabling the study that forms this thesis. With these scripts, I am able to produce and analyze nearly any data set I could think to want for this LK-SAM study, with the limiting factor being computation time.

Submission Script Generator

Both MARCC and Rosetta require configuration files to run simulations. MARCC requires a slurm submission script to configure and run simulations, while Rosetta requires a “flags” configuration file. For my simulations, it was necessary to submit many simulations, with submission scripts and flags files varying key parameters. Manually writing these files would be both inefficient and prone to errors, even if copying and pasting from templates. To expedite and ensure accuracy, I wrote a submission script generator in Python.

The generator accepts command line input and generates submission scripts and corresponding flags file for each set of simulation conditions required. Examples of these files are in Appendix 2A: Example MARCC Submission Script and Appendix 2B: Example Flags File, respectively, and the Python script is in Appendix 2C: Submission Script Generator. The script takes the following arguments:

- The option to exclude peptide and surface combinations (by default, the generator produced simulations for all eight combinations of LK- α or LK- β , 7mer or 14mer, and hydrophilic or hydrophobic SAM surface)
- The option to use capped, uncapped, or N-capped peptides
- Zero (baseline), one, or two parameters to be varied, which could be score weights such as `fa_sol`, or atom properties of the methyl carbons of the hydrophobic SAM, such as `LK_DGFFREE`, lysine C_δ and C_ϵ as a range (with C_δ scaling at 10%

of the rate of C_ϵ), or the two-body interaction calculation range Rosetta used when calculating scores

- The start and end values for the parameter variation range(s)
- The option to adjust the LK_DGFREE atom properties of lysine C_δ and C_ϵ
- The number of desired increments within the range
- The number of models each simulation should produce
- The option to use the REF2015 score function instead of Talaris2013
- An optional folder, into which all the simulation set folders would be placed
- An optional command to submit all generated scripts to MARCC

The script would then generate a range of parameter values and peptide-surface combinations, and create a subfolder for each, containing an appropriately generated submission script and flags file and folders to receive structures outputted by the simulation and any output from the cluster. These subfolders would be moved into a specified folder (which the script could create), if one was given, and issue the SLURM sbatch command for each submission script if desired. Cluster time requested for the simulations was scaled by the number of structures to output.

I would invoke the script with a command line such as the following, which would generate the fa_sol set from 0 to 1.5 with 10 increments with 10^3 structures at each point, using N-capped 14mers for both LK- α and LK- β on both SAM surfaces, and put them into a folder for fa_sol:

```
./submit_script_gen_13.py -f fa_sol -n -d 1000 -e 7 -x s fa_sol 0 1.5 10
```

Top Decoy Selection Script

Once the simulations were complete on MARCC, it was necessary to isolate the portion of the output structures that would be used for analysis. I wrote the script in Appendix 2D: Decoy

Collection Script, to read the score list and select the best structures. It allows control over the number of structures to collect, and copies a score file sorted not by order of simulation, but by total score, as well as the flags file for the simulations so that it was not necessary to have detailed folder names to know what the simulation conditions were. The script can also select best solution state models. The score files output from each simulation do not include the score of the solution-state models, and favorable adsorbed score and favorable solution score did not necessarily correlate, thus the script recalculates the scores of all solution models.

Analysis and Plotting Script

After running each simulation and selecting the fraction of best scoring models, the next step was the analysis of patterns in peptide secondary structure and orientation on the surface. This script has three main parts. Firstly the required information is extracted from each structure in each subfolder, and combined as a list of class objects that the script outputs, and which can be read by the latter sections. During this step, a summary is generated for each simulation set, listing the score and total helicity of each structure. Sorting of the data utilized naming conventions from the Submission Script Generator. The subsequent sections are two levels of plotting. Local plots are those shown in 3A: , which are generated for each value in a parameter variation (though not included in this thesis, due to the large number of such sets), and for all baselines. Parameter variation plots are those shown in Chapter 5: Results, Score Weight Variations and Chapter 6: Results, Atom Parameter Variations, which are not generated for baseline simulations, and require reorganization of the data. This script is in Appendix 2E: Analysis and Plotting Script, and can be run directly on a folder created by the Submission Script Generator, reading the parameters and values directly from the folder names the Generator creates.

Appendix 2A: Example MARCC Submission Script

```
#!/bin/bash -l

#SBATCH --job-name=04A0BAS0
#SBATCH --partition=parallel
#SBATCH --time=15:30:0
#SBATCH --nodes=5
#SBATCH --ntasks-per-node=24
#SBATCH --mem=120GB
#SBATCH --output
/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14mer
_alpha_baseline/hydrophobic_14mer_alpha_baseline_err/report.%j.out
#SBATCH --error
/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14mer
_alpha_baseline/hydrophobic_14mer_alpha_baseline_err/report.%j.err
#SBATCH --mail-user=jlubin3@jhu.edu
#SBATCH --mail-type=ALL

module unload openmpi
module load intel-mpi

ROSETTABIN=/home-
2/jlubin3@jhu.edu/work/jgray21/jhlubin/Rosetta/main/source/bin
ROSETTAEXE=surface_docking
COMPILER=mpi.linuxgccrelease
EXE=$ROSETTABIN/$ROSETTAEXE.$COMPILER
echo Starting MPI job running $EXE

date
time mpirun $EXE
@/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14me
r_alpha_baseline/hydrophobic_14mer_alpha_baseline.flags
date
```

Appendix 2B: Example Flags File

```
-database /home-2/jlubin3@jhu.edu/work/jgray21/jhlubin/Rosetta/main/database
-include_surfaces
-s /home-
2/jlubin3@jhu.edu/scratch/samonolayer_parameter_refine/seed_files/hydrophobic
_14mer_alpha_n_capped.pdb
-in:file:surface_vectors /home-
2/jlubin3@jhu.edu/scratch/samonolayer_parameter_refine/seed_files/SAM.surf
-mute core
-mute protocols.moves.RigidBodyMover
-nstruct 10000
-score:weights talaris2013
-out:pdb_gz
-out:path:pdb
/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14mer
_alpha_baseline/hydrophobic_14mer_alpha_baseline_output
-out:path:score
/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14mer
_alpha_baseline/hydrophobic_14mer_alpha_baseline_output
-mpi_tracer_to_file
/scratch/users/jlubin3@jhu.edu/samonolayer_parameter_refine/hydrophobic_14mer
_alpha_baseline/hydrophobic_14mer_alpha_baseline_err/tracer.out
```

Appendix 2C: Submission Script Generator

```
#!/usr/bin/python
"""
Make folders, flags, sbatch files for MARCC submission for LK peptide surface
docks. Vary weight parameters, surface atom properties,
hydrophobic/hydrophilic surface.

Created by Joseph Lubin, 2016
"""

import argparse
import os
import numpy as np
import math

# Collecting inputs
parser = argparse.ArgumentParser()
subparser = parser.add_subparsers(help = "How many parameters to vary",
    dest = 'generator_type')

# Options
parser.add_argument("-f", "--subfolder", help = \
    "What folder below the current working directory do you want to use?")
parser.add_argument("-s", "--score", help = \
    "What score function do you want to use, if not talaris2013? " + \
    "(beta_nov15)")
parser.add_argument("-c", "--capped", action = "store_true", help = \
    "Should the peptides be N-acetylated and C-amidated, " + \
    "as Collier & Latour?")
parser.add_argument("-n", "--n_cap", action = "store_true", help = \
    "Should the peptides be N-acetylated, as Weidner & Castner? " + \
    "(14mers only)")
parser.add_argument("-d", "--decoys", type = int, default = 10000, help = \
    "How many decoys do you want? (Default = 10000)")
parser.add_argument("-r", "--range", type = float, default = 5, help = \
    "Desired range cutoff for LJ & LK potentials? (Default = 5)")
parser.add_argument("-kcd", type = float, default = 0.52, help = \
    "Altered hydrophobicity LK_DGFFREE of Lysine delta carbon? <0=hydrophilic")
parser.add_argument("-kce", type = float, default = 0.52, help = \
    "Altered hydrophobicity LK_DGFFREE of Lysine epsilon carbon? " + \
    "<0=hydrophilic")
parser.add_argument("-e", "--exclusions", type = str,
    action = 'append', choices = "abio74", help = \
    "What test conditions do you want to exclude? " + \
```

```

    "Can be used multiple times to exclude multiple categories. " + \
    " Sequence: a--alpha, b--beta // Surface: i--" + "hydrophilic, " + \
    "o--hydrophobic // "Size: 7--7mer, 4--14mer")
parser.add_argument("-x", "--execute", action = "store_true", help = \
    "Do you wish to submit all generated scripts?")

#Baseline case
base = subparser.add_parser('b', help = "baseline")

#Single parameter case
single = subparser.add_parser('s', help = "Vary one parameter")
single.add_argument("parameter_1", type = str, help = \
    "What parameter do you mean to vary?")
single.add_argument("begin_1", type = float, help = \
    "Minimum of desired parameter range for testing")
single.add_argument("end_1", type = float, help = \
    "Maximum of desired parameter range for testing")
single.add_argument("step_1", type = float, help = \
    "Desired increments in range")

#Double parameter case
double = subparser.add_parser('d', help = "Vary two parameters")
p1 = double.add_argument_group("first parameter")
p1.add_argument("parameter_1", type = str, help = \
    "What is the first parameter you mean to vary?")
p1.add_argument("begin_1", type = float, help = \
    "Minimum of first parameter range for testing")
p1.add_argument("end_1", type = float, help = \
    "Maximum of first parameter range for testing")
p1.add_argument("step_1", type = float, help = \
    "Desired increments in first parameter range")

p2 = double.add_argument_group("second parameter")
p2.add_argument("parameter_2", type = str, help = \
    "What is the second parameter you mean to vary?")
p2.add_argument("begin_2", type = float, help = \
    "Minimum of second parameter range for testing")
p2.add_argument("end_2", type = float, help = \
    "Maximum of second parameter range for testing")
p2.add_argument("step_2", type = float, help = \
    "Desired increments in second parameter range")

args = parser.parse_args()

#####

```

```

#Defining parameter lists and functions
def get_parameter_type(parameter):
    """
    Given a parameter name, this function checks the Rosetta default weight
    parameters and atom properties, and matches the type of the parameter
    """
    weight_params = ['fa_atr', 'fa_rep', 'fa_sol', 'fa_intra_rep', 'fa_elec',
                     'pro_close', 'hbond_sr_bb', 'hbond_lr_bb', 'hbond_bb_sc',
                     'hbond_sc', 'dslf_fa13', 'rama', 'omega', 'fa_dun',
                     'p_aa_pp', 'ref']
    atom_properties = ['LJ_RADIUS', 'LJ_WDEPTH', 'LK_DGFFREE', 'LK_LAMBDA',
                      'LK_VOLUME']
    parameter_types_list = ["Weight", "Atom Property", "Lysine CD & CE",
                             "ERROR"]

    if parameter.lower() in weight_params:
        par_type = "weight"
        print_index = 0

    elif parameter.upper() in atom_properties:
        par_type = "atom_property"
        print_index = 1
        print "\n\nParameter type: "

    elif parameter.lower() == 'kcde':
        par_type = "kcde"
        print_index = 2

    else:
        par_type = "ERROR"
        print_index = 3

    print parameter + ":\t" + parameter_types_list[print_index]

    return par_type

def trial_conditions_chooser(exclusions):
    """
    This function takes an input with letter or number codes indicating a data
    set to exclude from testing. Options are alpha or beta peptide,
    hydrophobic or hydrophilic surface, and 7mer or 14mer peptide.
    """
    condition_files = ["hydrophilic_14mer_alpha", "hydrophobic_14mer_alpha",
                       "hydrophilic_14mer_beta", "hydrophobic_14mer_beta",
                       "hydrophilic_7mer_alpha", "hydrophobic_7mer_alpha",

```

```

        "hydrophilic_7mer_beta", "hydrophobic_7mer_beta"]

test_cond = []

seq_exclude = [0,1]
surf_exclude = [0,1]
size_exclude = [0,1]

if "14" in exclusions:
    del size_exclude[0]
elif "7" in exclusions or args.n_cap:
    del size_exclude[1]

if "a" in exclusions:
    del seq_exclude[0]
elif "b" in exclusions:
    del seq_exclude[1]

if "i" in exclusions:
    del surf_exclude[0]
elif "o" in exclusions:
    del surf_exclude[1]

for i in size_exclude:
    for j in seq_exclude:
        for k in surf_exclude:
            file_index = 4*(i)+2*(j)+(k)
            test_cond.append(condition_files[file_index])

test_cond_display = str(test_cond)
text_replacements = {"[":"", "]":"", "'": "", "_": " "}
for i, j in text_replacements.iteritems():
    test_cond_display = test_cond_display.replace(i, j)

print "Test conditions:"
print test_cond_display
return test_cond

def freerange(start, stop, step):
    """
    Generates a range that doesn't need to count by 1
    """
    i = start
    while round(i, 3) <= stop:
        yield round(i, 3)

```

```

        i += step

def range_gen(start, stop, steps):
    """
    Uses the freerange function to generate a list of parameter variant values
    in both numerical and text form
    """
    param_set = []
    variable_step = (stop - start)/steps

    for i in freerange(start, stop, variable_step):
        param_set.append(i)

    return param_set

def determine_score_file(parameter_value_1, parameter_1, parameter_type_1,
                        parameter_value_2, parameter_2, parameter_type_2):
    """ Determines appropriate score weights file to flag """
    param_list = {'fa_atr': 0.8, 'fa_rep': 0.44, 'fa_sol': 0.75,
                  'fa_intra_rep': 0.004, 'fa_elec': 0.7, 'pro_close': 1,
                  'hbond_sr_bb': 1.17, 'hbond_lr_bb': 1.17,
                  'hbond_bb_sc': 1.17, 'hbond_sc': 1.1, 'dslf_fa13': 1.0,
                  'rama': 0.2, 'omega': 0.5, 'fa_dun': 0.56,
                  'p_aa_pp': 0.32, 'ref': 1}

    # Pointing to appropriate scores file
    if args.score:
        scorefile = args.score
    else:
        if [parameter_type_1, parameter_type_2].count("weight") == 0:
            scorefile = 'talaris2013'
        elif [parameter_type_1, parameter_type_2].count("weight") == 1:
            if parameter_type_1 == "weight":
                if parameter_value_1 == param_list[parameter_1.lower()]:
                    scorefile = 'talaris2013'
                else:
                    scorefile = 'talaris_' + parameter_1.lower() + '_' +
                                str(parameter_value_1)
            else:
                if parameter_value_2 == param_list[parameter_2.lower()]:
                    scorefile = 'talaris2013'
                else:
                    scorefile = 'talaris_' + parameter_2.lower() + '_' +
                                str(parameter_value_2)
        else:

```

```

        if parameter_value_1 == param_list[parameter_1.lower()] and \
            parameter_value_2 == param_list[parameter_2.lower()]:
            scorefile = 'talaris2013'
        else:
            scorefile = 'talaris_' + parameter_1.lower() + '_' +
                str(parameter_value_1) + '_' + parameter_2.lower() + '_' +
                str(parameter_value_2)

    return scorefile

def kcde_value_flags(parameter_value):
    """
    Sets KCD value based on KCE value. Changes by 14.4% of KCE, so KCD is 1
    when
    KCE is 10.
    """
    kcde_lines = []
    atom_adjust = '-chemical:set_atom_properties fa_standard:'

    if parameter_value == 'manual':
        kcd_value = args.kcd
        kce_value = args.kce
    else:
        kce_value = parameter_value
        kcd_value = 0.52 - round(0.144 * (0.52 - kce_value), 2)

    kcde_lines.append(atom_adjust + 'KCD:LK_DGFFREE:' + str(kcd_value) + '\n')
    kcde_lines.append(atom_adjust + 'KCE:LK_DGFFREE:' + str(kce_value) + '\n')

    return kcde_lines

def flags_maker(path, destination, rosetta_folder, source_folder, dataset,
                num_decoys, detection_range, parameter_value_1, parameter_1,
                parameter_type_1, parameter_value_2, parameter_2,
                parameter_type_2):
    """
    Writes flags file for LK peptide SAM surface docking simulation
    """
    loc_ident = os.path.join(path, destination, destination)
    filename = destination + ".flags"
    scorefile = determine_score_file(parameter_value_1, parameter_1,
        parameter_type_1, parameter_value_2, parameter_2, parameter_type_2)

    #addressing kcde values
    kcde_lines = []

```

```

if parameter_1 == 'kcde':
    kcde_lines = kcde_value_flags(parameter_value_1)
elif parameter_2 == 'kcde':
    kcde_lines = kcde_value_flags(parameter_value_2)
elif args.kcd != 0.52 or args.kce != 0.52:
    kcde_lines = kcde_value_flags('manual')

# Writing flags file
with open(flagname, 'w') as fl:
    fl.write('-database ' + rosetta_folder + 'database\n')
    fl.write('-include_surfaces\n')
    if args.capped:
        fl.write('-s ' + source_folder + dataset + '_capped.pdb\n')
    elif args.n_cap:
        fl.write('-s ' + source_folder + dataset + '_n_capped.pdb\n')
    else:
        fl.write('-s ' + source_folder + dataset + '.pdb\n')
    fl.write('-in:file:surface_vectors ' + source_folder + 'SAM.surf\n')
    fl.write('-mute core\n')
    fl.write('-mute protocols.moves.RigidBodyMover\n')
    fl.write('-nstruct ' + num_decoys + '\n')
    if args.range != 5:
        fl.write('-score::fa_max_dis ' + str(detection_range) + '\n')
    fl.write('-score:weights ' + scorefile + '\n')
    if parameter_type_1 == "atom_property":
        fl.write('-chemical:set_atom_properties fa_standard:CH3S:' +
            parameter_1.upper() + ':' + str(parameter_value_1) + '\n')
    if parameter_type_2 == "atom_property":
        fl.write('-chemical:set_atom_properties fa_standard:CH3S:' +
            parameter_2.upper() + ':' + str(parameter_value_2) + '\n')
    for line in kcde_lines:
        fl.write(line)
    fl.write('-out:pdb_gz\n')
    fl.write('-out:path:pdb '+ loc_ident + '_output\n')
    fl.write('-out:path:score '+ loc_ident + '_output\n')
    fl.write('-mpi_tracer_to_file ' + loc_ident + '_err/tracer.out\n')

os.rename(flagname, os.path.join(path, destination, flagname))
print "\t" + flagname
return flagname

def sbatch_maker(flags_file, path, destination, rosetta_folder,
                condition_name, wall_time, parameter_name_1,
                set_member_1, parameter_name_2, set_member_2):
    """

```

```

Writes MARCC sbatch file for LK peptide SAM surface docking simulation
"""
loc_ident = os.path.join(path, destination, destination)
subname = destination+".sbatch"

# Dictionaries to abbreviate conditions and parameters
conditions_codes = {"hydrophobic_14mer_alpha": "O4A",
                    "hydrophilic_14mer_alpha": "I4A",
                    "hydrophobic_14mer_beta": "O4B",
                    "hydrophilic_14mer_beta": "I4B",
                    "hydrophobic_7mer_alpha": "O7A",
                    "hydrophilic_7mer_alpha": "I7A",
                    "hydrophobic_7mer_beta": "O7B",
                    "hydrophilic_7mer_beta": "I7B"}

parameter_codes = {'fa_atr': 'ATR', 'fa_rep': 'REP', 'fa_sol': 'SOL',
                  'fa_intra_rep': 'IRP', 'fa_elec': 'ELC', 'pro_close': 'PCL',
                  'hbond_sr_bb': 'HSB', 'hbond_lr_bb': 'HLB',
                  'hbond_bb_sc': 'HBS', 'hbond_sc': 'HSC', 'dslf_fa13': 'DSF',
                  'rama': 'RAM', 'omega': 'OMG', 'fa_dun': 'DUN',
                  'p_aa_pp': 'PAP', 'ref': 'REF', 'LJ_RADIUS': 'RAD',
                  'LJ_WDEPTH': 'WDP', 'LK_DGFFREE': 'DGF', 'LK_LAMBDA': 'LAM',
                  'LK_VOLUME': 'VOL', 'kcde': 'KC', 'baseline': 'BAS', ':': ''}

# Getting parameter type
try:
    par_code_1 = parameter_codes[parameter_name_1]
except:
    par_code_1 = parameter_codes[parameter_name_1.upper()]

try:
    par_code_2 = parameter_codes[parameter_name_2]
except:
    par_code_2 = parameter_codes[parameter_name_2.upper()]

# Writing SBATCH files
with open(subname, 'w') as su:
    su.write('#!/bin/bash -l\n')
    su.write('\n')
    su.write('#SBATCH --job-name=' + conditions_codes[condition_name] +
            set_member_1 + par_code_1 + set_member_2 + par_code_2 + '\n')
    su.write('#SBATCH --partition=parallel\n')
    su.write('#SBATCH --time=' + wall_time + ':0' + '\n')
    su.write('#SBATCH --nodes=5\n')
    su.write('#SBATCH --ntasks-per-node=24\n')

```

```

su.write('#SBATCH --mem=120GB\n')
su.write('#SBATCH --output ' + loc_ident + '_err/report.%j.out\n')
su.write('#SBATCH --error ' + loc_ident + '_err/report.%j.err\n')
su.write('#SBATCH --mail-user=jlubin3@jhu.edu\n')
su.write('#SBATCH --mail-type=ALL\n')
su.write('\n')
su.write('module unload openmpi\n')
su.write('module load intel-mpi\n')
su.write('\n')
su.write('ROSETTABIN=' + rosetta_folder + 'source/bin\n')
su.write('ROSETTAEXE=surface_docking\n')
su.write('COMPILER=mpi.linuxgccrelease\n')
su.write('EXE=$ROSETTABIN/$ROSETTAEXE.$COMPILER\n')
su.write('echo Starting MPI job running $EXE\n')
su.write('\n')
su.write('date\n')
su.write('time mpirun $EXE @' + path + '/' + destination + '/' +
        flags_file + '\n')
su.write('date\n')
os.rename(subname, os.path.join(path, destination, subname))
print "\t" + subname
return subname, loc_ident

```

```
def par_value_stringer(value):
```

```
    """
```

```
    When given a value, this function will convert it into a string, with a
    consistent length so folders sort nicely.
```

```
    """
```

```
    string = str(value)
```

```
    if '.' not in string:
```

```
        string += '.0'
```

```
    while len(string) < 5:
```

```
        string += '0'
```

```
    return string
```

```
def marcc_file_maker(path, test_condition, decoy_count, detection_range,
                    set_item_1, set_item_2, parameter_1, parameter_value_1,
                    parameter_type_1, parameter_2, parameter_value_2,
                    parameter_type_2):
```

```
    """
```

```
    Uses flags_maker and sbatch_maker and generates flags, sbatch scripts,
    and folders for LK peptide surface docking simulations run on MARCC.
```

```
    """
```

```
    # making folder
```

```

if parameter_type_1 == "base":
    newpath = test_condition + "_baseline"
elif parameter_type_2 == "":
    newpath = test_condition + "_" + parameter_1 + "_" +
        par_value_stringer(parameter_value_1)
else:
    newpath = test_condition + "_" + parameter_1 + "_" +
        par_value_stringer(parameter_value_1) + "_" + parameter_2 + "_" +
        par_value_stringer(parameter_value_2)

os.makedirs(os.path.join(path, newpath))
print "Folder: " + newpath

ros_folder = "/home-2/jlubin3@jhu.edu/work/jgray21/jhclubin/Rosetta/main/"
work_folder = "/home-2/jlubin3@jhu.edu/scratch/" + \
    "samonolayer_parameter_refine/seed_files/"

# making flags
flagname = flags_maker(path, newpath, ros_folder, work_folder,
    test_condition, decoy_count, detection_range,
    parameter_value_1, parameter_1, parameter_type_1,
    parameter_value_2, parameter_2, parameter_type_2)

# making submission script
subname, loc_ident = sbatch_maker(flagname, path, newpath, ros_folder,
    test_condition, time, parameter_1,
    set_item_1, parameter_2, set_item_2)

subscript = (path + "/" + newpath + "/" + subname)

#making output and error folders
outname = loc_ident + "_output"
os.makedirs(outname)
print "\tFolder: " + newpath + "_output"
ername= loc_ident + "_err"
os.makedirs(ername)
print "\tFolder: " + newpath+ "_err\n"

return subscript, subname

#####
print "\n"

# Checking parameter type
if args.generator_type == "b":

```

```

par_type_1 = "base"
parameter_1 = "baseline"
par_type_2 = ""
parameter_2 = ""
print "Parameter type: "
print "Baseline"

elif args.generator_type == "s":
    print "Parameter type: "
    par_type_1 = get_parameter_type(args.parameter_1)
    parameter_1 = args.parameter_1
    par_type_2 = ""
    parameter_2 = ""

elif args.generator_type == "d":
    print "Parameter type: "
    par_type_1 = get_parameter_type(args.parameter_1)
    par_type_2 = get_parameter_type(args.parameter_2)
    parameter_1 = args.parameter_1
    parameter_2 = args.parameter_2
    if par_type_2 == "ERROR":
        par_type_1 = par_type_2

print ""

#Response to incorrect parameter
if par_type_1 == "ERROR":
    print "\nPlease check the name of the parameter you want to vary."
    print "\tWeight parameters:"
    for i in range(len(weight_params)):
        print "\t\t" + weight_params[i]
    print "\tAtom Properties:"
    for i in range(len(atom_properties)):
        print "\t\t" + atom_properties[i]
    print "\tLysine CD & CE:\n\t\ttkcde"
    exit()

# Data sets to generate
if args.exclusions:
    test_cond = trial_conditions_chooser(args.exclusions)
else:
    test_cond = trial_conditions_chooser([])

# Generating list of values from given min, max, and step count
if args.generator_type == "b":

```

```

param_set_1 = [0]
param_set_2 = [0]

elif args.generator_type == "s":
    param_set_1 = range_gen(args.begin_1, args.end_1, args.step_1)
    param_set_2 = [0]
    print "\nList of parameter values:"
    print param_set_1

elif args.generator_type == "d":
    param_set_1 = range_gen(args.begin_1, args.end_1, args.step_1)
    param_set_2 = range_gen(args.begin_2, args.end_2, args.step_2)
    print "\nList of parameter values:"
    print parameter_1 + ":\t" + str(param_set_1)
    print parameter_2 + ":\t" + str(param_set_2)

# Adjusting time for increased for decoy counts
decoys = str(args.decoys).replace(".0", "")
hours = math.ceil(args.decoys / 1000) * 1.5 + 0.5

if hours % 1 == 0:
    hours = str(hours).replace(".0", "")
    minutes = "0"
else:
    minutes = math.ceil((hours % 1) * 60)
    minutes = str(minutes).replace(".0", "")
    hours = math.floor(hours)
    hours = str(hours).replace(".0", "")
time = hours + ":" + minutes

print "\n" + decoys + " decoys"
print "Requesting time per simulation:\t" + time

trials = len(test_cond) * len(param_set_1) * len(param_set_2) * int(decoys)
print "Total decoys generated:\t" + str(trials)

# Setting destination
if args.subfolder:
    path = os.path.join(os.getcwd(), args.subfolder)
    if not os.path.isdir(path):
        os.makedirs(path)
        print "\nCreated destination folder:"
    else:
        print "Destination folder:"
    print path

```

```

else:
    path = os.getcwd()
    print "\nDestination folder:"
    print path

# Creating files
subscripts = []
subnames = []
print "\n\nMaking the following:"
for i in range(len(test_cond)):
    for j in range(len(param_set_1)):
        for k in range(len(param_set_2)):
            subscript, subname = marcc_file_maker(path, test_cond[i], decoys,
                                                  args.range, str(j), str(k),
                                                  parameter_1, param_set_1[j], par_type_1,
                                                  parameter_2, param_set_2[k], par_type_2)

            subscripts.append(subscript)
            subnames.append(subname)

# Submitting the sbatch files
# Should only be done after the weight files are generated, if varying a
# weight
if args.execute:
    for i in range(len(subscripts)):
        bashline = "sbatch " + str(subscripts[i])
        os.system(bashline)
    print "\nAll jobs submitted"

# Printing list of submission scripts
# The conversion below makes it easier to execute the list of subscripts with
# a loop bash command (for i in [block]; do sbatch $i; done) if the execute
# option is not used
else:
    sub_string = ""
    for i in subscripts:
        sub_string = sub_string + i
        if i != len(subscripts):
            sub_string = sub_string + " "
    print "\nList of sbatch files:"
    print sub_string
    print "\n"

```

Appendix 2D: Decoy Collection Script

```
#!/usr/bin/python
"""
Selects top decoys from folders made by submit_script_gen.py on MARCC for
LK peptide SAM surface docks. If solution state decoys are being analyzed, a
new score list will be generated.

Created by Joseph Lubin
"""

import argparse
import os
from shutil import copyfile, move
# Further import of PyRosetta is only necessary to analyze solution models

def parse_args():
    """ Collecting inputs """
    parser = argparse.ArgumentParser()
    parser.add_argument("folder", help="What folder below the current \
        working directory do you want to use?")
    parser.add_argument("-sol", "--solution_state", action="store_true",
        help="Do ou want the solution state decoys, rather \
        than the surface docked ones?")
    parser.add_argument("-d", "--decoy_count", type=int, default=100,
        help="Collect the top [how many] decoys?")
    parser.add_argument("-l", "--location", type=str, choices=['m', 'l'],
        default='m', help="Are PDB's on MARCC (m, default) \
        or downloaded to local (l)?")
    parser.add_argument("-s", "--silence", action="store_true",
        help="Silence output from calculations")
    args = parser.parse_args()

    return args

def get_folders(direc, f_name, args):
    """
    This function returns an input folder with PDB files and an output folder
    to which top decoys, a score file, and the flags file will be copied
    """
    # Getting path for folder containing decoys
```

```

if args.location == 'm':    # Running from MARCC-formatted folders
    input_folder = os.path.join(direc, f_name + '_output')
else:    # Running from folders already created by this script
    input_folder = direc

# Making top decoys folder
td_folder_name = f_name + '.top' + str(args.decoy_count)
top_decoys_folder = os.path.join(input_folder, td_folder_name)
if not os.path.isdir(top_decoys_folder):
    os.makedirs(top_decoys_folder)

return input_folder, [td_folder_name, top_decoys_folder]

def display_time(start, elapsed, run_count, total_count):
    """ Displays the time for scoring PDBs """
    # Giving an initial estimate of computation time
    if run_count == 1:
        total_time = elapsed / 2 * total_count / 3600
        out = "Rough estimated time to score this folder: {} hours"
        print out.format(round(total_time,2))
        return

    # Giving a total at completion
    if (run_count + 1) == total_count:
        out = "Scoring completed for this folder. Completion time: {} hours"
        print out.format(round(elapsed,2))
        return

    # Updating estimate every 100 decoys
    if (run_count + 1) % 100 == 0:
        est = elapsed * (float(total_count / (run_count + 1)) - 1) / 3600
        out = "\t{} PDBs scored. Estimated time remaining: {} hours"
        print out.format(run_count + 1, round(est, 2))
        return

def rescore_sol(folder, args):
    """
    This function takes an input folder and scores all the solution state
    decoys in the folder. It returns a sorted list of names and scores.
    This requires the importation of PyRosetta. There is a limitation that
    this function will only use the default Rosetta score function. In this
    case, it is using talaris2014.
    """

```

```

import rosetta
import rosetta.core.scoring.solid_surface
opts = '-include_surfaces -mute basic -mute core -mute protocols'
rosetta.init(extra_options = opts)
if not args.silence:
    from time import time

# Getting list of all files in the folder
f_name = os.path.basename(folder).replace('_output', '')
if not args.silence:
    print '\n\nFolder:\t{}'.format(f_name)

folder_list = os.listdir(folder)    # Full folder

# Narrowing list to only solution models
sol_pdbs = []
for i in folder_list:
    if 'Sol' in i:
        sol_pdbs.append(i)
sol_pdbs.sort()
count = len(sol_pdbs)

if not args.silence:
    print "Scoring {} PDBs".format(count)

# Writing unsorted scores file
scoresc = os.path.join(folder, 'sol_score.sc')
header = ('\t' * 6).join(['Description', 'Score'])
with open(scoresc, 'w') as s:
    s.write(header)

# Scoring solution PDBs and listing scores
score_errors = {}
sf = rosetta.get_fa_scorefxn()
sol_scores = []
start = time()
for i in range(len(sol_pdbs)):
    try:
        pdb = sol_pdbs[i]
        p = rosetta.pose_from_pdb(os.path.join(folder, pdb))
        score = sf(p)
        sol_scores.append(score)

        # Adding score to unsorted list
        with open(scoresc, 'a') as s:

```

```

        s.write('\n{}\t{}'.format(pdb, score))

    if not args.silence:
        elapsed = time() - start
        display_time(start, elapsed, i, count)

    except RuntimeError:
        print "Unable to read PDB: {}".format(pdb)
        if score_errors.has_key(f_name):
            score_errors[f_name].append(pdb)
        else:
            score_errors.update({f_name: [pdb]})

# Combining files names and scores, sorting by scores
s_name_scores = sorted(zip(sol_pdbs, sol_scores), key=lambda x:x[1])

return s_name_scores, header, score_errors

def read_scorefile(scorefile):
    """
    This function uses the output score.sc file from the MARCC run to find the
    top docked decoys based on Rosetta score. It requires a score file input.
    Outputs a sorted scores list
    """
    # Reading score file
    score_text = open(scorefile).readlines()
    score_header = score_text[1]
    raw_scores = score_text[2:]

    # Getting total scores and filenames
    total_scores = []
    for line in raw_scores:
        total_scores.append(line.split())

    # Converting string numbers to floats and sorting
    for line in range(len(total_scores)):
        # Removing "SCORE:" from each line
        total_scores[line] = total_scores[line][1:]
        for i in range(len(total_scores[line])-1):
            # Last item is file name
            float_val = float(total_scores[line][i])
            total_scores[line][i] = float_val

    total_scores.sort()

```

```

# Reorganizing data
row_size = len(total_scores[0])
# Cleaning header
headlist = score_header.split()[-row_size:] # Strips all before "score"
header = str(headlist[-1]) # Convert to str with file name before
scores
for column in range(len(headlist)-1):
    header += '\t' + str(headlist[column]).replace(':', '')

# Re-ordering scores to match header
for line in range(len(total_scores)):
    total_scores[line] = [total_scores[line][-1]] + \
        total_scores[line][1:-1]

return total_scores, header

def pdb_copy(pdb_name, source, destination, args):
    """ Copy a PDB file from one folder to another """
    if args.solution_state:
        dec = pdb_name
    else:
        dec = pdb_name + '.pdb.gz'

    decoy_source = os.path.join(source, dec)
    decoy_destination = os.path.join(destination, dec)
    copyfile(decoy_source, decoy_destination)

    if not args.silence:
        print '\t' + dec

def score_and_flags(name, dir_in, dir_out, score_list, header, args):
    """ Output a sorted scores file and copy over the simulation flags """
    if not args.silence:
        print 'Writing sorted scores file and copying flags\n\n'

    # Making output scores file
    out_scores = os.path.join(dir_out, 'score.sc')
    with open(out_scores, 'w') as sc:
        sc.write(header + '\n')
        for line in score_list:
            line_text = '\t'.join([str(i) for i in line])
            sc.write(line_text + '\n')

```

```

#Copying flags file
flag_name = name + '.flags'
flag_source = os.path.join(dir_in, flag_name)
flag_destination = os.path.join(dir_out, flag_name)
copyfile(flag_source, flag_destination)

def take_top_decoys(folder, name, args):
    """
    Given a folder containing PDB files, this function will get the
    appropriate directory or subdirectory containing PDB files. Then it will
    either rescore solution state PDBs or read and sort the score.sc file to
    get the lowest scoring decoys. The specified number of best scorers will
    be copied to a top decoys folder, along with the flags file that generated
    them, and a sorted scores file.
    """
    folder_full = os.path.join(folder, name)
    input_folder, out_folder = get_folders(folder_full, name, args)

    # Getting scores list
    if args.solution_state:      # For solution models, need to re-score
        scores, header, errors = rescore_sol(input_folder, args)

    else:      # Otherwise, can read scores directly from score.sc
        if not args.silence:
            print 'Folder:\t' + name

        scorefile = os.path.join(input_folder, 'score.sc')
        scores, header = read_scorefile(scorefile)
        errors = {}

    # Getting top models
    top_models = []
    for i in range(args.decoy_count):
        top_models.append(scores[i][0])

    # Copying top decoys
    print '\nCopying top {} decoys'.format(args.decoy_count)
    for decoy in top_models:
        pdb_copy(decoy, input_folder, out_folder[1], args)

    # Making output scores file and copying over flags for reference
    score_and_flags(name, folder_full, out_folder[1],
                    scores, header, args)

```

```

    return out_folder, errors

def main():
    args = parse_args()

    # Setting path
    main_directory = os.getcwd() + "/" + args.folder

    # Getting list of subfolders
    subdirectories = next(os.walk(main_directory))[1]
    subdirectories.sort()

    # Picking out top decoys in each subfolder
    out_folders = {}
    pdb_exclusions = {}
    folder_exclusions = []

    for folder in subdirectories:
        #verify that folder has PDBs (exclude results folders, etc.)
        run = False
        for file in os.listdir(os.path.join(main_directory, folder)):
            if any(x in file for x in ['.pdb.gz', '_output']):
                run = True
                break

        if run:
            out_folder, errors = take_top_decoys(main_directory, folder, args)
            out_folders.update({out_folder[0]: out_folder[1]})
            pdb_exclusions.update(errors)

        else:
            print 'Excluding:\t\t' + folder
            folder_exclusions.append(folder)

    # Making results directory
    results_folder = os.path.join(main_directory, 'results')
    if not os.path.isdir(results_folder):
        os.makedirs(results_folder)
    for folder in out_folders:
        move(out_folders[folder], os.path.join(results_folder, folder))

    # Displaying results directory to copy-paste for scp/rsync
    print '\n\n' + results_folder + '/*'

```

```
# Displaying excluded folders and PDBs
if len(folder_exclusions) > 0:
    print '\nFOLDERS EXCLUDED:'
    for exclusion in folder_exclusions:
        print exclusion

if len(pdb_exclusions) > 0:
    print '\nPDBS EXCLUDED:'
    for folder in pdb_exclusions:
        print folder
        for pdb in pdb_exclusions[folder]:
            print '\t' + pdb

if __name__ == '__main__':
    main()
```

Appendix 2E: Analysis and Plotting Script

```
#!/usr/bin/python
"""
This script is intended to take data from self-assembled monolayer docking
simulations, and generate appropriate plots. The script reads a folder with
subfolders containing the top decoys from a set of trials generated by
another
script, 'submit_script_gen'.

A note about octets: now they're quadrants. An older version of the script
handled 7mers and 14mers together, whereas for the publication, I needed only
14mers, so I changed it around but never got around to beautifying.

Script for extracting data and making plots originally written by M. Pacella
Modified and expanded by J. H. Lubin
"""
import argparse
import operator
from os import getcwd, listdir, makedirs, walk
from os.path import basename, dirname, isdir, join, normpath
import pickle
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.patches as mpat
from matplotlib import cm
import matplotlib.gridspec as gridspec
import numpy as np
from pylab import *
from scipy.interpolate import UnivariateSpline
# Following functions only used for calculating helicity
from rosetta import *
from rosetta.core.scoring.dssp import Dssp
import rosetta.core.scoring.solid_surface
from glob import glob
import gzip

# parameter info, with default values
param_list = { 'fa_atr': 0.8, 'fa_rep': 0.44, 'fa_sol': 0.75,
               'fa_intra_rep': 0.004, 'fa_elec': 0.7, 'pro_close': 1,
               'hbond_sr_bb': 1.17, 'hbond_lr_bb': 1.17, 'hbond_bb_sc': 1.17,
               'hbond_sc': 1.1, 'dslf_fa13': 1.0, 'rama': 0.2, 'omega': 0.5,
               'fa_dun': 0.56, 'p_aa_pp': 0.32, 'ref': 1, 'lj_radius': 2.000,
               'lj_wdepth': 0.1811, 'lk_dgfree': 1.5000, 'lk_lambda': 3.5000,
```

```
'lk_volume': 30.0000, 'range': 5, 'kcde': 0.52}
```

```
def parse_args():
    """
    Collect input arguments for helicity plotter. By default, the script will
    take a data file it has generated on a previous run and produce plots.
    -c is an option to calculate such a data file for a new dataset
    Parameters -p1 and -p2 can be entered manually, though by default, the
    script will read the folder name for parameter names, and finding none,
    will assume baseline.
    -bw will generate the plots in black and white, rather than default colors
    -s will silence the default output of PDB's being processed and plots made
    """
    info = 'Make helicity plots for decoys generated by submit_script_gen'
    parser = argparse.ArgumentParser(description=info)
    parser.add_argument("folder", type=str,
                        help="What folder below the current working \
                              directory do you want to use?")
    parser.add_argument("-c", "--calculate", action="store_true",
                        help="Use this option the first time handling a \
                              dataset")
    parser.add_argument("-p1", "--parameter_1", type=str,
                        help="Manually enter a varied parameter for \
                              calculation")
    parser.add_argument("-p2", "--parameter_2", type=str,
                        help="Manually enter second varied parameter for \
                              calculation.")
    parser.add_argument("-bw", "--black_white", action="store_true",
                        help="Use this option to print the plots in black \
                              and white")
    parser.add_argument("-sl", "--skip_local", action="store_true",
                        help="Make only parameter varition plots")
    args = parser.parse_args()

    return args

#####
"""Data management"""

class pose_helicity_data:
    """
    This class is designed to hold helicity data from a single PDB.
```

Pose name will be the PDB name string. Peptide size, PDB score, and helicity will be float values, with helicity being the count of helical residues divided by two less than the peptide length. Leucine and lysine distances and phis and psis will be lists with length matching the number of each residue in the peptide. Helix and loop hist will be arrays of length equal to the peptide size, counting whether each residue is helical or not.

"""

```
def __init__(self, pdb_file):
```

```
    self.pose_name = ''
```

```
    self.peptide_size = 0
```

```
    self.score = 0
```

```
    self.helicity = 0
```

```
    self.leucine_distances = []
```

```
    self.lysine_distances = []
```

```
    self.c_alpha_distances = []
```

```
    self.helix_hist = np.array([])
```

```
    self.loop_hist = np.array([])
```

```
    self.phis = []
```

```
    self.psis = []
```

```
    self.pose_helicity_analysis(pdb_file)
```

```
def pose_helicity_analysis(self, pdb_file):
```

"""

Function takes input PDB file and extracts multiple pieces of data, including the name of the PDB file, the peptide length, the PDB score (which includes any weight parameter changes), a count of helical residues, lists of the distances between leucine and lysine residues and the surface, arrays of length equal to the number of residues with 1's or 0's indicating whether a residue is helical, and the backbone dihedrals.

"""

```
# Getting just the PDB file name from directory
```

```
self.pose_name = basename(pdb_file)
```

```
print self.pose_name
```

```
# Reading PDB file for secondary structure
```

```
pose = pose_from_pdb(pdb_file) # load a pose from the file
```

```
first_protein_residue = pose.num_jump()+1
```

```

# Getting peptide size
self.peptide_size = pose.total_residue() + 1 - first_protein_residue

# Getting the score of the current pdb
self.score = self.read_score(pdb_file)

# Initializing histogram lists (size is required)
self.helix_hist = np.array([0] * self.peptide_size)
self.loop_hist = np.array([0] * self.peptide_size)

# Getting secondary structure, coordinates, dihedrals for each residue
sec_struct = Dssp(pose)
sec_struct.insert_ss_into_pose(pose)
H_count = float(0)

for resnum in range(first_protein_residue, pose.total_residue()+1):
    residue = resnum - first_protein_residue
    self.get_secstruct(pose, resnum, residue)

def read_score(self, pdb_file):
    """
    Reads the score from the text of a PDB file. This yields the score
    reflecting any altered weight parameters, rather than rescoring the
    PDB with default weights. Searches PDB text from the bottom, since the
    relevant line is near the end. The total score is the last item of
    that line.
    """
    # Read PDB file
    with gzip.open(pdb_file, 'r') as pdbgz:
        line_search = pdbgz.readlines()

    # Find line with pose scores
    for line in line_search[::-1]:
        line = line.rstrip()
        if line[0:4] == "pose":
            score_line = line
            break

    # Convert last portion of total score line to a float value
    total_score = float(score_line.split()[-1])

    return total_score

def get_secstruct(self, pose, pdb_number, peptide_number):
    """

```

Inputs of a PDB file and a residue number, and returns whether the residue is helical, the amino acid type, the distance from the surface, and the backbone dihedrals. Distance is determined based on average of CD's for leucine and NZ for lysine.

```
"""
# Getting residue secondary structure
sec_struct_string = pose.secstruct()
res_sec_struct = sec_struct_string[peptide_number-1]

# Incorporating residue's secondary structure into histogram
if res_sec_struct == 'H':
    self.helix_hist[peptide_number] += 1
    self.helicity += float(1)/(self.peptide_size -2)
    # Peptide size -2 because termini cannot be helical
else:
    self.loop_hist[peptide_number] += 1

# Getting residue name
res = pose.residue(peptide_number)
res_name = res.name1()

# Getting and adjusting the z coordinates
if res_name == 'L': # average of carbon deltas for leucine
    z1 = res.atom('CD1').xyz().z
    z2 = res.atom('CD2').xyz().z
    z_val = self.adjust_surface_z(float(np.mean([z1, z2])), pose)
    self.leucine_distances.append(z_val)

if res_name == 'K': # z coordinate of nitrogen for lysine
    z_val = self.adjust_surface_z(res.atom('NZ').xyz().z, pose)
    self.lysine_distances.append(z_val)

c_a_z = self.adjust_surface_z(res.atom('CA').xyz().z, pose) # CA
self.c_alpha_distances.append(c_a_z)

# Getting backbone dihedrals, ignoring terminal residues
if peptide_number not in [0, self.peptide_size-1]:
    self.phis.append(pose.phi(peptide_number))
    self.psis.append(pose.psi(peptide_number))

def adjust_surface_z(self, z_value, pose):
    """
    Takes a z coordinate of an LK peptide residue and corrects position.
    The z coordinate is adjusted because the surfaces are not at precisely
    z = 0A. The average z coordinate for the top of the hydrophilic
```

surface is -17.868Å. There was an error in early simulations where the peptide may be above or below the hydrophobic surface (which is symmetrical). The average z for the top of the hydrophobic surface is -19.064Å, and for the bottom is -30.964. The center of the surface was therefore considered -25.

```
"""
if pose.residue(1).name() in ['COO', 'COH']:    # hydrophilic surface
# Surface is always before peptide in PDB file
    coordinate = z_value + 17.868
elif z_value > -25:        # above hydrophobic surface
    coordinate = z_value + 19.064
else:                    # below hydrophobic surface
    coordinate = -z_value - 30.964

return coordinate
```

```
class folder_helicity_data:
```

```
    """
```

This class is designed to take in and organize data from individual decoys

Test condition and both parameters will be strings; the parameter values and set helicity mean standard deviation will be float values with the parameter defaults at 0 signifying baseline; helix count and loop count will be lists with length matching the peptide length, indicating degree of helicity at each residue; file names, scores, helicities will be lists of the respective data for each PDB in the folder; L_ and K_ distances will be lists of length equal times the product of peptide length and the number of PDB's.

```
    """
```

```
def __init__(self, place, parameter_1, parameter_2):
    self.location = place
    self.folder_name = basename(place)

    self.parameter_1 = parameter_1
    self.parameter_1_value = self.name_to_num(parameter_1)

    self.parameter_2 = parameter_2
    self.parameter_2_value = self.name_to_num(parameter_2)

    self.surface_type = ''
    self.peptide_size = ''
    self.peptide_length = None
    self.peptide_sequence = ''
    self.data_typer()
```

```

self.test_condition = None
self.parameter_group = None
self.variation_group = None
self.parameters_varied = 0
self.report_strings()

self.file_names = []
self.scores = []
self.helicities = []
self.leucine_distances = []
self.lysine_distances = []
self.c_alpha_distances = []
self.phis = []
self.psis = []

self.helix_count = np.array([0] * self.peptide_length)
self.loop_count = np.array([0] * self.peptide_length)

self.helicity_mean = None
self.helicity_sd = None

self.folder_helicity_analysis()
self.local_readable_summary()
self.post_process()

def name_to_num(self, parameter):
    """
    Method will take a folder name generated by submit_script_gen and
    isolates the numerical value of the varied parameter, regardless of
    1 or 2 variables
    """
    if parameter in ["baseline", None]:
        return None

    elif 'baseline' in self.folder_name:
        return param_list[parameter]

    else:
        fix_text = self.folder_name.lower()

        # strip out the decoy count, ex: '.top100'
        fix_text = fix_text.split(".top")[0]

        # isolating parameter value (hard-coded to fit submit_script_gen)

```

```

# the parameter value follows the parameter in the folder name
text_split = fix_text.split(parameter + '_')[1]
target_value = text_split.split('_')[0] # value will be first item

# convert to float
return float(target_value)

def data_typer(self):
    """
    Method takes folder name output from submit_script_gen and determines
    which of the eight possible combinations of peptide sequence and
    length, and surface type that folder contains, so that the data can be
    added to the appropriate bin for analysis.
    """
    for i in ['hydrophilic', 'hydrophobic']:
        if i in self.folder_name:
            self.surface_type = i

    peptide_length = '14mer' # early trials did not include the 7mer
                            # option

    for j in ['7mer', '14mer']:
        if j in str(self.folder_name):
            self.peptide_size = j
            self.peptide_length = int(j.replace('mer', ''))

    for k in ['alpha', 'beta']:
        if k in self.folder_name:
            self.peptide_sequence = k

def report_strings(self):
    """ Method concatenates various values for file naming """
    # Test condition
    self.test_condition = '_'.join([self.surface_type,
                                    self.peptide_size,
                                    self.peptide_sequence])

    # Adjusting parameter group to consistent float so plot names align
    par_1_string = str(self.parameter_1_value)
    par_2_string = str(self.parameter_2_value)
    for string in [par_1_string, par_2_string]:
        while len(string) < 5:
            if '.' in string:
                string += '0'
            else:
                string += '.'

```

```

# Parameter group and parameters varied
if self.parameter_1 is None and self.parameter_2 is None:
    self.parameter_group = 'baseline'
    self.parameters_varied = 0

elif self.parameter_2 is None:
    self.parameter_group = '_'.join([self.parameter_1, par_1_string])
    self.parameters_varied = 1

else:
    self.parameter_group = '_'.join([self.parameter_1, par_1_string,
                                     self.parameter_2, par_2_string])
    self.parameters_varied = 2

# Variation group
self.variation_group = self.surface_type + '_' + self.peptide_size

def add_pose(self, pose_data):
    """
    This function adds a pose_helicity_data class object to the group.
    """
    self.file_names.append(pose_data.pose_name)
    self.scores.append(pose_data.score)
    self.helicities.append(pose_data.helicity)
    self.leucine_distances += pose_data.leucine_distances
    self.lysine_distances += pose_data.lysine_distances
    self.c_alpha_distances += pose_data.c_alpha_distances
    self.phis += pose_data.phis
    self.psis += pose_data.psis
    self.helix_count += pose_data.helix_hist
    self.loop_count += pose_data.loop_hist

def folder_helicity_analysis(self):
    """
    This method enables the class to collect all pose helicity analysis
    data on each PDB in the input folder, and stores the combined data from
    the entire folder, including the test condition, both parameters and
    their values, the mean and standard deviation of helicity for all PDBs,
    histogram data of residues are helical or non-helical in secondary
    structure, and lists of files, scores, helicities, leucine and lysine z
    coordinates, and backbone dihedral angles.
    """
    # displaying folder name
    print "\n", self.folder_name, ":"

```

```

# getting list of PDBs
pdbs = join(self.location, "*.pdb.gz")
pdb_list = glob(pdbs)
pdb_list.sort()

# Analyzing pose secondary structures and adding them to list
for file_name in pdb_list:
    try:
        p = pose_from_pdb(file_name)

    except Exception:
        print 'Unable to read ', file_name
        continue

    self.add_pose(pose_helicity_data(file_name))

def post_process(self):
    """
    This function calculates helicity mean and standard deviation. It also
    converts the numpy arrays (helix and loop count) to lists for later
    ease of use.
    """
    # Getting average and sd of helicity
    self.helicity_mean = np.mean(self.helicities)
    self.helicity_sd = np.std(self.helicities)

    # Converting np arrays to lists
    self.helix_count = list(self.helix_count)
    self.loop_count = list(self.loop_count)

def local_readable_summary(self):
    """
    This method makes a reference text file listing the PDB file names,
    helicities, and scores for all PDB's in the folder, sorted by score.
    """
    # making a combined list
    name_heli_score = zip(self.file_names, self.helicities, self.scores)

    # sorting by score
    sorted_helicities = sorted(name_heli_score, key=lambda x: x[2])

    # saving file
    template = '{:50s} {:20s} {:10s} \n'
    local_out = join(self.location, 'scores_helicities.txt')

```

```

with open(local_out, 'w') as l_o:
    l_o.write(template.format("File", "Helicity", "Score")) # Header
    for line in sorted_helicities:
        lineout = [str(x) for x in line]
        l_o.write(template.format(*lineout))

#####
"""Analysis functions"""

def identify_parameters(place, parameter_1, parameter_2=None):
    """
    This function takes the parameter arguments (or lack thereof) and
    determines whether the dataset is baseline, or one or two varied
    parameters, and which parameters are varied if applicable. Default is
    baseline. If parameter arguments are given, this function will verify
    that they are in the list of possible parameters. If not, before
    accepting default, the function will read the folder name and identify
    parameters there.
    """
    # default: baseline
    par_name = None

    # given input
    if parameter_1:
        par_name = parameter_1.lower()
        assert par_name in param_list

    # checking folder name
    else:
        for parameter in param_list:
            if parameter == parameter_2: # for second parameter case
                continue
            elif parameter in basename(place):
                par_name = parameter

    return par_name

def display_parameters(directory_name, parameter_1, parameter_2, calc=True):
    """
    This function uses identify_parameters function to determine one or two
    varied parameters. It will display the filepath and parameters as well.
    """
    # calculate parameters

```

```

name_1 = identify_parameters(directory_name, parameter_1)
name_2 = identify_parameters(directory_name, parameter_2, name_1)

# display parameters
print "\n\nPATH:\t" + directory_name

if name_1 is None:
    print "\nBaseline\n"
else:
    print "\nVaried Parameter:\t" + name_1 + '\n'

if name_2 is not None:
    print "Varied Parameter:\t" + name_2 + '\n'

# returning parameters
if calc:
    return name_1, name_2

def master_data_extraction(path, parameter_1, parameter_2):
    """
    This is the overall function for reading helicity data from a folder of
    folders generated by submit_script_gen. The data from each PDB in a
    subfolder is collected in a class object, and all class objects are stored
    in a list. Each object will include the folder's test condition, and the
    varried parameter(s) and parameter values if applicable. It will also
    include the helicity mean and standard deviation for the folder, a helix
    count and loop count for each residue in the peptide, lists of file names,
    PDB scores, and helicities for each PDB in the folder, and lists of the
    leucine and lysine distances from the surface acrtoss all peptides and
    residues in the folder.
    """
    # identifying parameter(s) being analyzed
    par_name_1, par_name_2 = display_parameters(path,
                                                parameter_1,
                                                parameter_2)

    # getting list of subfolders
    subdirectories = next(walk(path))[1]
    subdirectories.sort()
    print "\nFolder list:"
    for subdirectory in subdirectories:
        print '\t' + subdirectory

    # Performing local helicity analysis

```

```

global_datasets, excluded_folders = [[] for i in range(2)]
print "\nReading PDB's"

for folder in range(len(subdirectories)):
    place = join(path, subdirectories[folder])
    # verify that folder has PDBs (exclude plots folders, etc.)
    run = False
    for file in listdir(place):
        if '.pdb.gz' in file:
            run = True
            break

    # analyzing folder PDB's
    if run:
        fhd = folder_helicity_data(
            place, par_name_1, par_name_2)
        global_datasets.append(fhd)

    else:
        print "Excluding folder:", subdirectories[folder]
        excluded_folders.append(subdirectories[folder])

# Making data dump text file
folder_name = basename(path)
master_out = join(path, folder_name + "_helicity_data.txt")
with open(master_out, "wb") as f:
    pickle.dump(global_datasets, f, pickle.HIGHEST_PROTOCOL)

return global_datasets

#####
"""Plotting functions"""

def make_octets(dataset_list):
    """
    Local plots (the residue helicity histogram, the L/K surface distance
    plot, Ramachandran, and the score-helicity scatterplot) show all surface/
    peptide combinations for a single test condition, e.g. baseline or a
    single varied parameter value. This function takes the list of folder
    data and sorts it into sets to pass to those plotting functions.
    """
    # getting list of groups
    octet_groups = []
    for datum in dataset_list:

```

```

    if datum.parameter_group in octet_groups:
        continue
    else:
        octet_groups.append(datum.parameter_group)
octet_groups.sort()

# populating groups
octets = []
for i in range(len(octet_groups)):
    octets.append([])
    for datum in dataset_list:
        if datum.parameter_group == octet_groups[i]:
            octets[i].append(datum)

return octets

def add_textline(text, x, y, ha='left', va='center', r=90, fs=11):
    """ This function adds text labels to a figure """
    plt.figtext(x, y, text, horizontalalignment=ha, verticalalignment=va,
                rotation=r, fontsize=fs)

class local_helicity_plots:
    """
    This class will generate the three local plot types, the residue helicity
    histogram, the L/K surface distance plot, and the score-helicity
    scatterplot. The reason for calling it an octet, even if there aren't
    eight items, is because the figure has eight plot spaces, representing
    each combination of surface type (hydrophilic or hydrophobic), peptide
    size (7mer or 14mer), and peptide sequence (alpha or beta).
    """
    def __init__(self, condition_octet, filepath, out_folder, black_white):
        self.octet = condition_octet
        self.path = filepath
        self.folder_name = out_folder
        self.condition = self.octet[0].parameter_group

        # Plot settings
        self.plot_types = {'heli_hist': self.helicity_plotter,
                          'surf_dist': self.distance_plotter,
                          'funnel': self.scatter_plotter,
                          'rama': self.rama_plotter}
        self.color = {False: 0, True: 1}[black_white]

```

```

# Getting min and max scores for funnel plots
self.min_score = []
self.max_score = []
self.scatter_range()

for plot_type in self.plot_types:
    print 'Making Plot:\t' + self.condition + '\t\t' + plot_type
    self.local_plot_master(plot_type)

def scatter_range(self):
    """ Determining plot range for funnel-type scatter plots """
    # Finding absolute maxima and minima
    abs_min_score = min([min(data.scores) for data in self.octet])
    abs_max_score = max([max(data.scores) for data in self.octet])

    # Checking if max score is positive. Necessary for beta_november sets
    # because hydrophilic surface scores highly positive
    if abs_max_score < 0:
        self.min_score = [abs_min_score, abs_min_score]
        self.max_score = [abs_max_score, abs_max_score]

    # Splitting score ranges
    else:
        # Separating hydrophobic and hydrophilic
        hydrophilic_sets = []
        hydrophobic_sets = []
        for i in self.octet:
            if i.surface_type == 'hydrophilic':
                hydrophilic_sets.append(i)
            if i.surface_type == 'hydrophobic':
                hydrophobic_sets.append(i)

        # Finding set maxima and minima
        hydrophilic_min_score = \
            min([min(data.scores) for data in hydrophilic_sets])
        hydrophilic_max_score = \
            max([max(data.scores) for data in hydrophilic_sets])

        hydrophobic_min_score = \
            min([min(data.scores) for data in hydrophobic_sets])
        hydrophobic_max_score = \
            max([max(data.scores) for data in hydrophobic_sets])

        self.min_score = [hydrophilic_min_score, hydrophobic_min_score]
        self.max_score = [hydrophilic_max_score, hydrophobic_max_score]

```

```

def local_plot_master(self, p_type):
    """
    Structure for creating 4x2 plots of the three different local types.
    Each subplot is populated by the appropriate type function.
    """
    # creating a figure with eight plots arranged 4x2
    fig, axes = plt.subplots(nrows=2, ncols=2)

    # formatting and labeling plot
    axis_labels = {'heli_hist': ['Secondary Structure', 'Residue Number'],
                   'surf_dist': ['Relative Probability',
                                 r'Distance from Surface ($\AA$)'],
                   'funnel': ['Score', 'Helicity'],
                   'rama': [r'$\psi$', r'$\phi$']}
    self.local_plot_formatter(fig, axis_labels[p_type])

    # mapping each dataset to the appropriate plot
    set_2_plot = { 'hydrophilic_14mer_alpha': axes[0][0],
                   'hydrophilic_14mer_beta': axes[0][1],
                   'hydrophobic_14mer_alpha': axes[1][0],
                   'hydrophobic_14mer_beta': axes[1][1],
                   'hydrophilic_7mer_alpha': axes[0][0],
                   'hydrophilic_7mer_beta': axes[0][1],
                   'hydrophobic_7mer_alpha': axes[1][0],
                   'hydrophobic_7mer_beta': axes[1][1]}

    # populating each plot
    for data in self.octet:
        plot_data = {
            'heli_hist': [data.helix_count, data.loop_count],
            'surf_dist': [data.leucine_distances,
                          data.lysine_distances,
                          data.c_alpha_distances],
            'funnel': [data.scores, data.helicities, data.surface_type],
            'rama': [data.psis, data.phis]}

        self.plot_types[p_type](set_2_plot[data.test_condition],
                                plot_data[p_type])

    # Removing superfluous ticks
    if p_type != 'funnel': # Scores not necessarily the same for all 4
        for a in [a[1] for a in axes]: #Plots on the right
            a.set_yticklabels(())
    else:
        # Leave space for scores

```

```

plt.subplots_adjust(wspace=0.37)

for a in axes[0]:    # Plots on the top
    a.set_xticklabels(())

# Setting tick fonts
for a in [axes[0][0], axes[0][1], axes[1][0], axes[1][1]]:
    for t in a.xaxis.get_major_ticks() + a.yaxis.get_major_ticks():
        t.label.set_fontsize(10)

# saving figure
title = {'heli_hist': '_sec_struct_distribution_',
        'surf_dist': '_distance_distribution_',
        'funnel': '_helicity_vs_score_',
        'rama': '_rama_'}
p_name = join(self.path, self.folder_name + title[p_type] +
             self.condition + '.png')
plt.savefig(p_name, dpi=300)
plt.close()

def local_plot_formatter(self, fig, y_x_labels):
    """
    Applies standard formatting and labels to 4x2 local data plots
    """
    # formatting
    fig.set_size_inches(3.5, 2.5)
    #plt.subplots_adjust(hspace = 0.3)
    #plt.subplots_adjust(wspace = 0.25)
    fig.subplots_adjust(top=0.88, bottom=0.18, left=0.18, right=0.9)

    # labels
    add_textline(y_x_labels[0], 0.01, 0.525, fs=12)
    add_textline(y_x_labels[1], 0.535, 0.01,
                ha='center', va='bottom', r=0, fs=12)
    add_textline(r'LK- $\alpha$ ', 0.34, 0.99, ha='center', va='top', r=0)
    add_textline(r'LK- $\beta$ ', 0.74, 0.99, ha='center', va='top', r=0)
    add_textline('Hydrophobic', 0.94, 0.32)
    add_textline('Hydrophilic', 0.94, 0.73)

def helicity_plotter(self, ax, helix_loop_hist_data):
    """
    Generates a bar plot with bins for each of the residues in the peptide,
    with red indicating the number of decoys in the set in which that
    residue is helical, and blue indicating the number of decoys in which
    it is not helical

```

```

"""
width = 1

assert len(helix_loop_hist_data[0]) == len(helix_loop_hist_data[1])
bin_count = len(helix_loop_hist_data[0])
bins = np.arange(0, bin_count)+0.5

ax.bar( bins, helix_loop_hist_data[0], width,
        color=['red', 'gray'][self.color])
ax.bar( bins, helix_loop_hist_data[1], width,
        color=['blue', 'white'][self.color],
        bottom=helix_loop_hist_data[0])

ax.axis([width / 2.0, bin_count + width / 2.0, 0,
        max(helix_loop_hist_data[0] + helix_loop_hist_data[1])])
ax.xaxis.set_major_locator(plt.MaxNLocator(5))

def distance_plotter(self, ax, L_K_z_coords):
    """
    Generates a double lineplot showing the distance distribution
    frequency of leucine any lysine residues for all peptides in the
    folder, with lysine in red and leucine in blue.
    """
    # Generating bins based on range of distances
    L_count, x_L = np.histogram(L_K_z_coords[0], bins=50, range=(0, 25))
    K_count, x_K = np.histogram(L_K_z_coords[1], bins=50, range=(0, 25))
    C_count, x_C = np.histogram(L_K_z_coords[2], bins=50, range=(0, 25))
    x_L = x_L[:-1] + (x_L[1] - x_L[0]) / 2
    x_K = x_K[:-1] + (x_K[1] - x_K[0]) / 2
    x_C = x_C[:-1] + (x_C[1] - x_C[0]) / 2
    f_L = UnivariateSpline(x_L, L_count, s=50)
    f_K = UnivariateSpline(x_K, K_count, s=50)
    f_C = UnivariateSpline(x_C, C_count, s=50)

    # Normalizing L/K distances
    max_L, min_L = max(f_L(x_L)), min(f_L(x_L))
    max_K, min_K = max(f_K(x_K)), min(f_K(x_K))
    max_C, min_C = max(f_C(x_C)), min(f_C(x_C))
    l_norm = [(float(i) - min_L)/(max_L-min_L) for i in f_L(x_L)]
    k_norm = [(float(i) - min_K)/(max_K-min_K) for i in f_K(x_K)]
    c_norm = [(float(i) - min_C)/(max_C-min_C) for i in f_C(x_C)]

    # plotting
    c = self.color
    ax.plot(x_K, k_norm, label="Lysine", lw=1.5,

```

```

        color=['red', 'gray'][c], linestyle=['-', '--'][c])
ax.plot(x_L, l_norm, label="Leucine", lw=1.5,
        color=['blue', 'gray'][c], linestyle=['-', ':'][c])
ax.plot(x_C, c_norm, label="CA", lw=1,
        color='black', linestyle=['-', '-:'][c])

ax.axis([-0.5, 25.5, -0.04, 1.04])

def scatter_plotter(self, ax, scores_helicities):
    """
    Generates a scatter of score vs helicity. This plot approximates a
    funnel plot, with helicity taking the place of RMSD.
    """
    ax.scatter(scores_helicities[1], scores_helicities[0],
               s=4, color=['blue', 'gray'][self.color])

    # setting axis limits so points aren't right on axis lines
    which_scores = ['hydrophilic', 'hydrophobic'].index(
        scores_helicities[2])
    w = which_scores
    margin = 0.06 * (self.max_score[w] - self.min_score[w])
    m = margin
    ax.axis([-0.04, 1.04, self.min_score[w] - m, self.max_score[w] + m])

    # setting a max tick count on the Y axis so the numbers don't overlap
    max_ticks = 5
    loc = plt.MaxNLocator(max_ticks)
    ax.xaxis.set_major_locator(loc)
    #ax.yaxis.set_major_locator(loc)

def rama_plotter(self, ax, psi_phis):
    """ Generates a Ramachandran scatter plot. """
    ax.scatter(psi_phis[1], psi_phis[0], s=0.25,
               color=['blue', 'gray'][self.color])

    # Setting axis limits and tick counts
    ax.axis([-180, 180, -180, 180])
    max_ticks = 5
    loc = plt.MaxNLocator(max_ticks)
    ax.yaxis.set_major_locator(loc)
    ax.xaxis.set_major_locator(loc)

    # Adding lines at 0
    ax.plot([0, 0], [-180, 180], '-', lw=0.5, color="black")
    ax.plot([-180, 180], [0, 0], '-', lw=0.5, color="black")

```

```

def surf_size_cluster(dataset_list):
    """
    This function takes the list of folder data and sorts it into sets of
    consistent surface type and peptide size (peptide sequence is mixed a/b).
    These groups are passed to the varied parameter plots.
    """
    # getting list of groups
    surf_size_groups = []
    for datum in dataset_list:
        if datum.variation_group not in surf_size_groups:
            surf_size_groups.append(datum.variation_group)

    # surf_size_groups.sort()

    # populating groups
    quadrants = []
    for i in range(len(surf_size_groups)):
        quadrants.append([])
        for datum in dataset_list:
            if datum.variation_group == surf_size_groups[i]:
                quadrants[i].append(datum)

    # splitting groups into alpha and beta
    quads_split = []
    for quadrant in quadrants:
        alpha_list, beta_list = [[] for i in range(2)]
        for datum in quadrant:
            if datum.peptide_sequence == 'alpha':
                alpha_list.append(datum)
            elif datum.peptide_sequence == 'beta':
                beta_list.append(datum)
        quads_split.append([alpha_list, beta_list])

    return quads_split

class parameter_varriation_plots:
    """
    Whereas the local helicity plots compare the eight surface/peptide
    combinations at a single experimental condition or at baseline, this class
    is intended to handle larger data sets that include the variation of one
    or two parameters. The reason for calling the data a quadrant, even if
    there aren't four of them, is due to the nature of the figure taking both

```

```

alpha and beta data from one of four possible combinations of surface
(hydrophilic or hydrophobic) and peptide size (7mer or 14mer).
"""
def __init__(self, a_list, b_list, filepath, folder_name, black_white):
    self.alpha_sets = a_list
    self.beta_sets = b_list
    self.path = filepath
    self.folder_name = folder_name
    self.variation_group = a_list[0].variation_group      # same for all
    self.parameters_varied = a_list[0].parameters_varied # same for all
    self.parameter_1 = a_list[0].parameter_1             # same for all
    self.parameter_2 = a_list[0].parameter_2             # same for all

    self.match_param_values = ['parameter_1_value', 'parameter_2_value',
                               'helicity_mean', 'helicity_sd', 'scores',
                               'helicities']

    # initializing list parameters
    for seq in ['alpha_', 'beta_']:
        for param in self.match_param_values:
            setattr(self, seq + param, [])
        for coord in ['x', 'y', 'z']:
            setattr(self, seq + '2_var_' + coord, [])
    for coord in ['x', 'y', 'z']:
        setattr(self, 'difference_2_var_' + coord, [])

    self.one_var_plots = {'1-var_plot': self.helicity_parameter_plot,
                          'score-helicity_plot': self.score_helicity_plot}
    self.color = {False: 0, True: 1}[black_white]

    # incorporating data
    for seq in ['alpha_', 'beta_']:
        for trial in getattr(self, seq + 'sets'):
            for param in self.match_param_values:
                getattr(self, seq + param).append(getattr(trial, param))

    # plotting
    if self.parameters_varied == 1:
        for p_type in self.one_var_plots:
            print '\t'.join(['Making Plot:', self.variation_group, p_type])
            self.one_var_plots[p_type]()

    elif self.parameters_varied == 2:
        self.two_var_grid()
        print '\t'.join(['\nMaking Plot:',

```

```

        self.variation_group,
        '2-var plot'])
    self.two_var_heat_plot()

def plot_namer(self, name):
    """ Makes plot name form path, parameters, given text """
    if self.parameter_2 is None:
        p2_text = ''
    else:
        p2_text = '_' + self.parameter_2

    return join(self.path, self.folder_name + name + self.parameter_1 +
                p2_text + "_" + self.variation_group + ".png")

def helicity_parameter_plot(self):
    """
    Generates a line plot, with a red line for alpha-sequence peptides and
    a blue line for beta-sequence peptides, which shows the average
    helicity of the set as a single score weight or surface atom property
    is varied. A vertical dotted black line indicates the Talaris2013
    default value for the varied parameter, and error bars at each point
    indicate the standard deviation in the dataset.
    """
    fig, ax0 = plt.subplots(nrows=1, ncols=1)

    c = self.color

    # helicity vs parameter lines
    ax0.plot(
        self.alpha_parameter_1_value, self.alpha_helicity_mean,
        color=['red', 'gray'][c], linestyle=['-', '--'][c], lw=1.75)
    ax0.plot(
        self.beta_parameter_1_value, self.beta_helicity_mean,
        color=['blue', 'gray'][c], linestyle=['-', '-.'][c], lw=1.75)

    # default line
    par_default = param_list[self.parameter_1]
    ax0.plot([par_default, par_default], [0, 1],
             '--', lw=1, color="black")

    # error bars
    ax0.errorbar(self.alpha_parameter_1_value, self.alpha_helicity_mean,
                 yerr = self.alpha_helicity_sd,
                 color=['red', 'gray'][c], lw=0.5)
    ax0.errorbar(self.beta_parameter_1_value, self.beta_helicity_mean,

```

```

        yerr=self.beta_helicity_sd,
        color=['blue', 'gray'][c], lw=0.5)

# axes
ax0.set_xlabel(self.parameter_1.upper(), fontsize=14)
ax0.set_ylabel("Helicity", fontsize=14)
max_pt = max(self.alpha_parameter_1_value +
             self.beta_parameter_1_value)
margin = 0.04 * (max_pt)
ax0.axis([-margin, max_pt + margin, -0.04, 1.04])

# Size
fig.set_size_inches(3.5, 2.25)
fig.subplots_adjust(bottom=0.21, top=0.98, left=0.18, right=0.95)
for t in ax0.xaxis.get_major_ticks() + ax0.yaxis.get_major_ticks():
    t.label.set_fontsize(10)

# save plot
plt.savefig(self.plot_namer("_parameter_helicity_plot_"), dpi=300)
plt.close()

def score_heli_axis_limits(self):
    """ Sets axis limits for score-helicity plots """
    # Determining score range
    scores = self.alpha_scores + self.beta_scores
    y_max = max([max(i) for i in scores])
    y_min = min([min(i) for i in scores])

    # Getting axis limits
    margin = 0.06 * (y_max - y_min) # prevents points being cut off
    ax_min, ax_max = y_min - margin, y_max + margin
    axlims = [-0.06, 1.06, ax_min, ax_max]

    return axlims

def colormapper(self, scorelist):
    """ Generate iterable cmap patterns for use in score_helicity_plot """
    if self.color == 0:
        colors = iter(cm.rainbow(np.linspace(0, 1, len(scorelist))))
        markers = iter(['o'] * len(scorelist))
    else:
        colors = iter(['gray'] * len(scorelist))
        markers = iter(['o', 's', 'D', '^', 'h', 'p', '*',
                       'v', 'd', '8', '>', '<', 'H'])

```

```

return colors, markers

def score_helicity_plot(self):
    """
    Generates a pair of scatterplots intended to mimic score-RMSD funnel
    plots. The first represents the alpha sequence peptides, and the second
    the beta sequence peptides. The plot coordinates for each decoy are its
    score and its helicity. This plot includes the data from all subsets.
    """
    fig, ax = plt.subplots(nrows=2, ncols=1)

    # alpha plot
    colors, markers = self.colormapper(self.alpha_scores)
    legend_keys = []
    for group in range(len(self.alpha_scores)):
        key = ax[0].scatter(self.alpha_helicities[group],
                           self.alpha_scores[group], s=10, alpha=0.6,
                           c=next(colors), marker=next(markers))
        legend_keys.append(key)

    # beta plot
    colors, markers = self.colormapper(self.beta_scores)
    for group in range(len(self.beta_scores)):
        ax[1].scatter(self.beta_helicities[group],
                     self.beta_scores[group], s=10, alpha=0.6,
                     c=next(colors), marker=next(markers))
    ax[1].set_xlabel("Helicity", fontsize=14)

    # Setting axis limits and ticks
    axis_limits = self.score_heli_axis_limits()
    for a in ax:
        a.axis(axis_limits)
        a.set_xticks([0, 0.5, 1])
        cur_y_ticks = a.get_yticks()
        a.set_yticks(cur_y_ticks[1:-1:2])

    # Y labels
    plt.figtext(0.02, 0.525, 'Score', horizontalalignment='left',
               verticalalignment='center', rotation=90, fontsize=14)
    plt.figtext(0.71, 0.76, r'LK- $\alpha$ ', horizontalalignment='left',
               verticalalignment='center', rotation=90, fontsize=12)
    plt.figtext(0.71, 0.34, r'LK- $\beta$ ', horizontalalignment='left',
               verticalalignment='center', rotation=90, fontsize=12)

```

```

# legend
#title
title_switches = { 'hbond_sr_bb': 'hbond',
                  'lj_wdepth': 'wdepth',
                  'lk_volume': 'volume'}
if self.parameter_1 in title_switches:
    title = title_switches[self.parameter_1]
else:
    title = self.parameter_1

assert self.alpha_parameter_1_value == self.beta_parameter_1_value
fig.legend(legend_keys, self.alpha_parameter_1_value, ncol=1,
          bbox_to_anchor=[0,0.05,1,0.9], loc='center right',
          scatterpoints=1, fontsize=8,
          title=title.upper())

# Size
fig.subplots_adjust(bottom=0.2, top=0.9, left=0.2, right=0.69)
fig.set_size_inches(3.5, 2.25)
plt.subplots_adjust(hspace = 0.5)
plt.subplots_adjust(wspace = 0.25)

# save plot
plt.savefig(self.plot_namer("_score_plot_"), dpi=300)
plt.close()

def two_var_grid(self):
    """
    The two-variable heat map is essentially 3-D, and therefore requires
    the parameter values and helicities to be arrayed. This function
    does that.
    """
    for seq in ['alpha_', 'beta_']:
        # making x and y coordinate arrays
        vals_1 = sorted(list(set(getattr(self, seq + 'parameter_2_value'))))
        vals_2 = sorted(list(set(getattr(self, seq + 'parameter_1_value'))))
        x, y = np.meshgrid(vals_1, vals_2)
        setattr(self, seq + '2_var_x', x)
        setattr(self, seq + '2_var_y', y)

        # making z coordinate array
        x_length = len(vals_1)
        y_length = len(vals_2)
        z = np.zeros((y_length, x_length))

```

```

    for datum in getattr(self, seq + 'sets'):
        x_index = vals_1.index(datum.parameter_2_value)
        y_index = vals_2.index(datum.parameter_1_value)
        z[y_index][x_index] = datum.helicity_mean

    setattr(self, seq + '_2_var_z', z)

# making alpha - beta difference array
    assert all(self.alpha_2_var_x) == all(self.beta_2_var_x)
    setattr(self, 'difference_2_var_x', self.alpha_2_var_x)
    assert all(self.alpha_2_var_y) == all(self.beta_2_var_y)
    setattr(self, 'difference_2_var_y', self.alpha_2_var_y)

    delta = self.alpha_2_var_z - self.beta_2_var_z
    setattr(self, 'difference_2_var_z', delta)

def heat_plotter(self, which_plot):
    """
    This function makes heat plots for the subsections of the
    two_var_heat_plot function. which_plot refers to alpha, beta,
    or delta.
    """
    subplot = {'alpha': 221, 'beta': 222, 'difference': 212}

    ax = plt.subplot(subplot[which_plot])
    x = getattr(self, which_plot + '_2_var_x')
    y = getattr(self, which_plot + '_2_var_y')
    z = getattr(self, which_plot + '_2_var_z')
    if which_plot in ['alpha', 'beta']:
        cs = ax.contourf(x, y, z, vmin=0, vmax=1,
                        cmap=[cm.coolwarm, cm.gray][self.color],
                        levels=np.arange(0, 1.1, 0.1))
    else:
        cs = ax.contourf(x, y, z, cmap=[cm.RdYlGn, cm.gray][self.color])

    # Points
    ax.scatter(x, y, color="black", marker='o', edgecolor='white')
    title = which_plot.upper()
    ax.set_title(title, fontsize=18)
    for t in ax.xaxis.get_major_ticks() + ax.yaxis.get_major_ticks():
        t.label.set_fontsize(11)

    # Default
    ax.scatter([0.52], [3], color='white', edgecolor='purple',

```

```

        marker='o', s=30)

    # Setting axis limits and tick counts
    ax.axis([-10.5, 1, 0, 20])
    #plt.xticks(list(plt.xticks()[0]) + [0.52])
    #plt.yticks(list(plt.xticks()[0]) + [0])

    return cs

def two_var_heat_plot(self):
    """
    Generates heat plots displaying the average helicity of the dataset at
    each point in the two parameter grid. The top plots represent the
    alpha sequence and beta sequence peptides, while the bottom plot is the
    difference between the two.
    """
    axes = []
    fig = plt.figure()

    for subplot in ['alpha', 'beta', 'difference']:
        ax = self.heat_plotter(subplot)
        axes.append(ax)

    # formatting
    plt.subplots_adjust(hspace=0.3)
    plt.subplots_adjust(wspace=0.2)
    fig.subplots_adjust(top=0.9, bottom=0.12, left=0.11, right=0.85)
    plt.figtext(0.02, 0.525, 'Hydrophobic Surface LK_DGFFREE', fontsize=16,
                rotation=90, va='center')
    plt.figtext(0.475, 0.02, r'Lysine C$\epsilon$ LK_DGFFREE', fontsize=16,
                ha='center')
    plt.figtext(0.91, 0.915, 'HELICITY', fontsize=16, ha='center')

    # colorbars
    color_ax_1 = fig.add_axes([0.89, 0.565, 0.025, 0.33])
    fig.colorbar(axes[1], cax=color_ax_1)
    color_ax_2 = fig.add_axes([0.89, 0.125, 0.025, 0.33])
    fig.colorbar(axes[2], cax=color_ax_2)

    # Size
    fig.set_size_inches(7, 5)

    # save plot
    plt.savefig(self.plot_namer("_helicity_heat_plot_"), dpi=300)
    plt.close()

```

```
#####

def main():
    args = parse_args()

    # setting path, eliminating extra '/', lowering case
    path = normpath(join(getcwd(), args.folder))

    # isolating folder name
    directory_name = basename(path)

    # getting data, either by folder analysis, or by reading the dump file
    # if the analysis has previously been run
    if args.calculate:
        opts = '--include_surfaces -mute core'
        rosetta.init(extra_options=opts)

        datasets = master_data_extraction( path,
                                          args.parameter_1,
                                          args.parameter_2)

    else:
        # display path and varied parameter(s)
        display_parameters(path, args.parameter_1, args.parameter_2,
                          calc=False)

        # importing the saved data from previous calculation
        master_out = join(path, directory_name + "_helicity_data.txt")
        with open(master_out, "rb") as f:
            datasets = pickle.load(f)

    # sort datasets by parameter 1 and parameter 2 values
    datasets.sort(key=operator.attrgetter(
        'test_condition', 'parameter_1_value', 'parameter_2_value'))

    # making plots folder
    plots_folder = join(path, directory_name + '_plots')
    if not isdir(plots_folder):
        makedirs(plots_folder)

    # plotting
    print "\nPlotting"
```

```

# Generating local plots
octet_groups = make_octets(datasets)
if not args.skip_local:
    for group in octet_groups:
        local_helicity_plots( group,
                               plots_folder,
                               directory_name,
                               args.black_white)

# no aggregate plotting possible for baselines
if len(octet_groups) == 1:
    exit()

# parameter variation plots
var_groups = surf_size_cluster(datasets)
for group in var_groups:
    parameter_varriation_plots( group[0],
                                 group[1],
                                 plots_folder,
                                 directory_name,
                                 args.black_white)

print '\n'

if __name__ == '__main__':
    main()

```

References

1. McKee, M. D., Nakano, Y., Masica, D. L., Gray, J. J., Lemire, I., Heft, R., Whyte, M. P., Crine, P. & Millan, J. L. Enzyme Replacement Therapy Prevents Dental Defects in a Model of Hypophosphatasia. *J. Dent. Res.* **90**, 470–476 (2011).
2. Qiu, S. R., Wierzbicki, A., Salter, E. A., Zepeda, S., Orme, C. A., Hoyer, J. R., Nancollas, G. H., Cody, A. M. & De Yoreo, J. J. Modulation of Calcium Oxalate Monohydrate Crystallization by Citrate through Selective Binding to Atomic Steps. (2005).
3. De Yoreo, J. J., Qiu, S. R. & Hoyer, J. R. Molecular modulation of calcium oxalate crystallization. *Am. J. Physiol. - Ren. Physiol.* **291**, (2006).
4. Wang, L., Zhang, W., Qiu, S. R., Zachowicz, W. J., Guan, X., Tang, R., Hoyer, J. R., Yoreo, J. J. De & Nancollas, G. H. Inhibition of calcium oxalate monohydrate crystallization by the combination of citrate and osteopontin. *J. Cryst. Growth* **291**, 160–165 (2006).
5. Masica, D. L., Gray, J. J. & Shaw, W. J. Partial high-resolution structure of phosphorylated and non-phosphorylated leucine-rich amelogenin protein adsorbed to hydroxyapatite. *J. Phys. Chem. C. Nanomater. Interfaces* **115**, 13775–13785 (2011).
6. KE, D. *Engines of creation*. (Anchor Press/Doubleday, 1986).
7. Noorduyn, W. L., Grinthal, A., Mahadevan, L. & Aizenberg, J. Rationally Designed Complex, Hierarchical Microarchitectures. *Science (80-.)*. **340**, (2013).
8. Kröger, N. & Sandhage, K. H. From Diatom Biomolecules to Bioinspired Syntheses of Silica-and Titania-Based Materials Silica Formation In Vitro Using Biomolecules from Diatoms.
9. Roehrich, A., Ash, J., Zane, A., Masica, D. L., Gray, J. J., Goobes, G. & Drobny, G. Solid-State NMR Studies of Biomineralization Peptides and Proteins. in 77–96 (2012).

10. Das, P. & Reches, M. Review insights into the interactions of amino acids and peptides with inorganic materials using single molecule force spectroscopy. *Biopolymers* **104**, 480–494 (2015).
11. De Yoreo, J. J., Chung, S. & Friddle, R. W. In Situ Atomic Force Microscopy as a Tool for Investigating Interactions and Assembly Dynamics in Biomolecular and Biomineral Systems. *Adv. Funct. Mater.* **23**, 2525–2538 (2013).
12. Friddle, R. W., Battle, K., Trubetskoy, V., Tao, J., Salter, E. A., Moradian-Oldak, J., De Yoreo, J. J. & Wierzbicki, A. Single-Molecule Determination of the Face-Specific Adsorption of Amelogenin's C-Terminus on Hydroxyapatite. *Angew. Chemie Int. Ed.* **50**, 7541–7545 (2011).
13. Razvag, Y., Gutkin, V. & Reches, M. Probing the Interaction of Individual Amino Acids with Inorganic Surfaces Using Atomic Force Spectroscopy. *Langmuir* **29**, 10102–10109 (2013).
14. Jarzynski, C. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Phys. Rev. E* **56**, 5018–5035 (1997).
15. Wu, S., Zhai, H., Zhang, W. & Wang, L. Monomeric Amelogenin's C-Terminus Modulates Biomineralization Dynamics of Calcium Phosphate. *Cryst. Growth Des.* **15**, 4490–4497 (2015).
16. Li, S., Wu, S., Nan, D., Zhang, W. & Wang, L. Inhibition of Pathological Mineralization of Calcium Phosphate by Phosphorylated Osteopontin Peptides through Step-Specific Interactions. *Chem. Mater.* **26**, 5605–5612 (2014).
17. De Yoreo, J. J., Zepeda-Ruiz, L. A., Friddle, R. W., Qiu, S. R., Wasylenki, L. E., Chernov, A. A., Gilmer, G. H. & Dove, P. M. Rethinking Classical Crystal Growth Models through

- Molecular Scale Insights: Consequences of Kink-Limited Kinetics. *Cryst. Growth Des.* **9**, 5135–5144 (2005).
18. Neuman, K. C. & Nagy, A. Single-molecule force spectroscopy: optical tweezers, magnetic tweezers and atomic force microscopy. *Nat. Methods* **5**, 491–505 (2008).
 19. Masica, D. L., Ash, J. T., Ndao, M., Drobny, G. P. & Gray, J. J. Toward a Structure Determination Method for Biomineral-Associated Protein Using Combined Solid-State NMR and Computational Structure Prediction. *Structure* **18**, 1678–1687 (2010).
 20. Rimola, A., Aschi, M., Orlando, R. & Ugliengo, P. Does Adsorption at Hydroxyapatite Surfaces Induce Peptide Folding? Insights from Large-Scale B3LYP Calculations. *J Am Chem Soc* **134**, 10899–10910 (2012).
 21. Iori, F., Di Felice, R., Molinari, E. & Corni, S. GolP: An atomistic force-field to describe the interaction of proteins with Au(111) surfaces in water. *J. Comput. Chem.* **30**, 1465–1476 (2009).
 22. Latour, R. A. Molecular simulation of protein-surface interactions: Benefits, problems, solutions, and future directions (Review). *Biointerphases* **3**, FC2-FC12 (2008).
 23. Corno, M., Rimola, A., Bolis, V. & Ugliengo, P. Hydroxyapatite as a key biomaterial: quantum-mechanical simulation of its surfaces in interaction with biomolecules. *Phys. Chem. Chem. Phys.* **12**, 6309 (2010).
 24. Wang, F., Stuart, S. J. & Latour, R. A. Calculation of adsorption free energy for solute-surface interactions using biased replica-exchange molecular dynamics. *Biointerphases* **3**, 9–18 (2008).
 25. Freeman, C. L., Harding, J. H., Quigley, D. & Rodger, P. M. Simulations of Ovocleidin-17 Binding to Calcite Surfaces and Its Implications for Eggshell Formation. *J. Phys. Chem. C*

- 115**, 8175–8183 (2011).
26. Li, Z. & Scheraga, H. A. Monte Carlo-minimization approach to the multiple-minima problem in protein folding (Metropolis criterion/Markov process/absorbing state/global optimization). *Chemistry (Easton)*. **84**, 6611–6615 (1987).
 27. Kuhlman, B. & Baker, D. Native protein sequences are close to optimal for their structures. *Proc. Natl. Acad. Sci. United States Am.* **97**, 10383–10388 (2000).
 28. Kuhlman, B., Dantas, G., Ireton, G. C., Varani, G., Stoddard, B. L. & Baker, D. Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science (80-.)*. **302**, (2003).
 29. Lazaridis, T. & Karplus, M. Effective energy function for proteins in solution. *Proteins Struct. Funct. Genet.* **35**, 133–152 (1999).
 30. Park, H., Bradley, P., Greisen, P., Liu, Y., Mulligan, V. K., Kim, D. E., Baker, D. & DiMaio, F. Simultaneous Optimization of Biomolecular Energy Functions on Features from Small Molecules and Macromolecules. *J. Chem. Theory Comput.* **12**, 6201–6212 (2016).
 31. Leaver-Fay, A., O’Meara, M. J., Tyka, M., Jacak, R., Song, Y., Kellogg, E. H., Thompson, J., Davis, I. W., Pache, R. A., Lyskov, S., Gray, J. J., Kortemme, T., Richardson, J. S., Havranek, J. J., Snoeyink, J., Baker, D. & Kuhlman, B. Scientific benchmarks for guiding macromolecular energy function improvement. *Methods Enzymol.* **523**, 109–43 (2013).
 32. Alford, R. F., Leaver-Fay, A., Jeliaskov, J. R., O’Meara, M. J., DiMaio, F. P., Park, H., Shapovalov, M. V, Renfrew, P. D., Mulligan, V. K., Kappel, K., Labonte, J. W., Pacella, M. S., Bonneau, R., Bradley, P., Dunbrack, R. L., Das, R., Baker, D., Kuhlman, B., Kortemme, T. & Gray, J. J. The Rosetta all-atom energy function for macromolecular modeling and design. *J. Chem. Theory Comput.* (2017).
 33. Leaver-Fay, A., Tyka, M., Lewis, S. M., Lange, O. F., Thompson, J., Jacak, R., Kaufman,

- K., Renfrew, P. D., Smith, C. A., Sheffler, W., Davis, I. W., Cooper, S., Treuille, A., Mandell, D. J., Richter, F., Ban, Y.-E. A., Fleishman, S. J., Corn, J. E., Kim, D. E., Lyskov, S., Berrondo, M., Mentzer, S., Popović, Z., Havranek, J. J., Karanicolas, J., Das, R., Meiler, J., Kortemme, T., Gray, J. J., Kuhlman, B., Baker, D. & Bradley, P. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Methods Enzymol.* **487**, 545–74 (2011).
34. King, N. P., Sheffler, W., Sawaya, M. R., Vollmar, B. S., Sumida, J. P., Andre, I., Gonen, T., Yeates, T. O. & Baker, D. Computational Design of Self-Assembling Protein Nanomaterials with Atomic Level Accuracy. *Science (80-.)*. **336**, 1171–1174 (2012).
35. Brooks, B. R., Brooks, C. L., Mackerell, A. D., Nilsson, L., Petrella, R. J., Roux, B., Won, Y., Archontis, G., Bartels, C., Boresch, S., Caffisch, A., Caves, L., Cui, Q., Dinner, A. R., Feig, M., Fischer, S., Gao, J., Hodoscek, M., Im, W., Kuczera, K., Lazaridis, T., Ma, J., Ovchinnikov, V., Paci, E., Pastor, R. W., Post, C. B., Pu, J. Z., Schaefer, M., Tidor, B., Venable, R. M., Woodcock, H. L., Wu, X., Yang, W., York, D. M. & Karplus, M. CHARMM: The biomolecular simulation program. *J. Comput. Chem.* **30**, 1545–1614 (2009).
36. Hornak, V., Abel, R., Okur, A., Strockbine, B., Roitberg, A. & Simmerling, C. Comparison of multiple Amber force fields and development of improved protein backbone parameters. *Proteins Struct. Funct. Bioinforma.* **65**, 712–725 (2006).
37. Jorgensen, W., Maxwell, D. & Tirado-Rives, J. Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids. *J. Am. Chem. Soc.* **118**, 11225–11236 (1996).
38. Dror, R. O., Jensen, M. Ø., Borhani, D. W. & Shaw, D. E. Exploring atomic resolution

- physiology on a femtosecond to millisecond timescale using molecular dynamics simulations. *J. Gen. Physiol.* **135**, (2010).
39. Collier, G., Vellore, N. A., Yancey, J. A., Stuart, S. J. & Latour, R. A. Comparison Between Empirical Protein Force Fields for the Simulation of the Adsorption Behavior of Structured LK Peptides on Functionalized Surfaces. *Biointerphases* **7**, 1–19 (2012).
 40. Wallace, A. F., Hedges, L. O., Fernandez-Martinez, A., Raiteri, P., Gale, J. D., Waychunas, G. A., Whitlam, S., Banfield, J. F. & De Yoreo, J. J. Microscopic Evidence for Liquid-Liquid Separation in Supersaturated CaCO₃ Solutions. *Science* (80-.). **341**, (2013).
 41. Raiteri, P., Gale, J. D., Quigley, D. & Rodger, P. M. Derivation of an Accurate Force-Field for Simulating the Growth of Calcium Carbonate from Aqueous Solution: A New Model for the Calcite–Water Interface. *J. Phys. Chem. C* **114**, 5997–6010 (2010).
 42. Heinz, H., Lin, T., Mishra, R. & Emami, F. Thermodynamically consistent force fields for the assembly of inorganic, organic, and biological nanostructures: the INTERFACE force field. *Langmuir* **29**, 1754–1765 (2013).
 43. Kokh, D. B., Corni, S., Winn, P. J., Hoefling, M., Gottschalk, K. E. & Wade, R. C. ProMetCS: An Atomistic Force Field for Modeling Protein-Metal Surface Interactions in a Continuum Aqueous Solvent. *J Chem Theory Comput.* **6**, 1753–1768 (2010).
 44. Freeman, C. L., Harding, J. H., Cooke, D. J., Elliott, J. A., Lardge, J. S. & Duffy, D. M. New forcefields for modeling biomineralization processes. *J Phys Chem C* **111**, 11943–11951 (2007).
 45. Wei, Y. & Latour, R. A. Benchmark experimental data set and assessment of adsorption free energy for peptide-surface interactions. *Langmuir* **25**, 5637–46 (2009).
 46. Masica, D. L. & Gray, J. J. Solution- and Adsorbed-State Structural Ensembles Predicted

- for the Statherin-Hydroxyapatite System. *Biophys. J.* **96**, 3082–3091 (2009).
47. Pacella, M. S., Koo, D. C. E., Thottungal, R. A. & Gray, J. J. Using the RosettaSurface algorithm to predict protein structure at mineral surfaces. *Methods Enzymol.* **532**, 343–66 (2013).
 48. Pacella, M. S. & Gray, J. J. A Benchmarking Study of Peptide-Biomineral Interactions. *J. Cryst. Growth Des.* (2017).
 49. DeGrado, W. F. & Lear, J. D. Induction of peptide conformation at apolar water interfaces. 1. A study with model peptides of defined hydrophobic periodicity. *J. Am. Chem. Soc.* **107**, 7684–7689 (1985).
 50. Mermut, O., Phillips, D. C., York, R. L., McCrea, K. R., Ward, R. S. & Somorjai, G. A. In situ adsorption studies of a 14-amino acid leucine-lysine peptide onto hydrophobic polystyrene and hydrophilic silica surfaces using quartz crystal microbalance, atomic force microscopy, and sum frequency generation vibrational spectroscopy. *J. Am. Chem. Soc.* **128**, 3598–3607 (2006).
 51. Weidner, T., Apte, J. S., Gamble, L. J. & Castner, D. G. Probing the orientation and conformation of α -helix and β -strand model peptides on self-assembled monolayers using sum frequency generation and NEXAFS spectroscopy. *Langmuir* **26**, 3433–3440 (2010).
 52. Apte, J. S., Collier, G., Latour, R. A., Gamble, L. J. & Castner, D. G. XPS and ToF-SIMS Investigation of α -Helical and β -Strand Peptide Adsorption onto SAMs. *Langmuir* **26**, 3423–3432 (2010).
 53. Weidner, T., Breen, N. F., Li, K., Drobny, G. P. & Castner, D. G. Sum frequency generation and solid-state NMR study of the structure, orientation, and dynamics of polystyrene-adsorbed peptides. *Proc. Natl. Acad. Sci. U. S. A.* **107**, 13288–93 (2010).

54. Weidner, T., Samuel, N. T., McCrea, K., Gamble, L. J., Ward, R. S. & Castner, D. G. Assembly and structure of α -helical peptide films on hydrophobic fluorocarbon surfaces. *Biointerphases* **5**, 9–16 (2010).
55. Deighan, M. & Pfendtner, J. Exhaustively Sampling Peptide Adsorption with Metadynamics. *Langmuir* **29**, 7999–8009 (2013).
56. Engh, R. A., Huber, R. & IUCr. Accurate bond and angle parameters for X-ray protein structure refinement. *Acta Crystallogr. Sect. A Found. Crystallogr.* **47**, 392–400 (1991).
57. Kabsch, W. & Sander, C. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* **22**, 2577–2637 (1983).
58. O’Meara, M. J., Leaver-Fay, A., Tyka, M. D., Stein, A., Houlihan, K., DiMaio, F., Bradley, P., Kortemme, T., Baker, D., Snoeyink, J. & Kuhlman, B. Combined Covalent-Electrostatic Model of Hydrogen Bonding Improves Structure Prediction with Rosetta. *J. Chem. Theory Comput.* **11**, 609–622 (2015).
59. Shapovalov, M. V, Dunbrack, R. L., Zhou, Y., Richardson, D. C., Song, Y., Richardson, D. C., Richardson, J. S., Baker, D., Smith, C. A., Sheffler, W. & al., et. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure* **19**, 844–58 (2011).
60. Song, Y., Tyka, M., Leaver-Fay, A., Thompson, J. & Baker, D. Structure-guided forcefield optimization. *Proteins Struct. Funct. Bioinforma.* **79**, 1898–1909 (2011).
61. Leaver-Fay, A., O’Meara, M. J., Tyka, M., Jacak, R., Song, Y., Kellogg, E. H., Thompson, J., Davis, I. W., Pache, R. A., Lyskov, S., Gray, J. J., Kortemme, T., Richardson, J. S., Havranek, J. J., Snoeyink, J., Baker, D. & Kuhlman, B. Scientific benchmarks for guiding macromolecular energy function improvement. *Methods Enzymol.* **523**, 109–43 (2013).

62. Shortle, D. Propensities, probabilities, and the Boltzmann hypothesis. *Protein Sci.* **12**, 1298–302 (2003).
63. Bazzoli, A., Kelow, S. P., Karanicolas, J., Rarey, M., Stahl, M. & Blundell, T. Enhancements to the Rosetta Energy Function Enable Improved Identification of Small Molecules that Inhibit Protein-Protein Interactions. *PLoS One* **10**, e0140359 (2015).
64. Huang, J., Rauscher, S., Nawrocki, G., Ran, T., Feig, M., de Groot, B. L., Grubmüller, H. & MacKerell, A. D. CHARMM36m: an improved force field for folded and intrinsically disordered proteins. *Nat. Methods* **14**, 71–73 (2016).

Pfleger Laboratory, University of Wisconsin-Madison, Madison WI
Undergraduate Summer Researcher, 2009

- Conducted laboratory research including DNA extraction from bacteria, primer design, PCR, and transformation by electroporation
- Launched and maintained a lab Wiki
- Read, summarized articles on methods of converting algal biomass to usable fuels
- Maintained cultures of cyanobacteria, prepared solutions, electrocompetent cells, and petri dishes, autoclaved dishware, disposed of chemical and biological wastes

Bioinspired Materials Laboratory, University of Wisconsin-Madison
Undergraduate Student Researcher, 2007

- Studied the swelling behavior of polyethylene-glycol (PEG) polymer hydrogels
- Attempted to synthesize variant compound by attaching another small organic molecule to the PEG monomer
- Analyzed mass spectra to determine whether the reaction was successful

Pathfinder Day Camp, Park District of Highland Park, Highland Park IL
Counselor, 2008

- Demonstrated and instructed various outdoor skills, including camping, cooking, boating, and ecology, while maintaining a safe environment for campers, ages 9-10

Ma-Ka-Ja-Wan Scout Reservation, Pearson WI.

Kitchen Staff; Trading Post Manager/Quartermaster; Handicraft, summers 2005-2007

- Prepared lesson plans and taught classes merit badges
- Managed sales and inventory of store and quartermaster equipment. Sold ~\$30,000 in food and merchandise.
- Maintained adherence to health standards in kitchen serving 150-400 guests
- Provided assistance when necessary (e.g., minor first aid) for campers, ages 11-17

Elms for Highland Park, Highland Park, IL

President, Project Coordinator, 2004-05

- Raised \$3,000 in private donations and led 35 volunteers to plant 28 American Liberty Elm trees
- Presented project on local television and published articles in newspaper and local magazine

Skills, Proficiencies Microsoft Office and SharePoint, especially Excel functions; Python; Adobe Acrobat; BASH; PyRosetta; PyMOL; 3D printing with Slic3r and 3D Builder; Meta-analysis of scientific literature; Familiar with speaking and reading Spanish

Activities, Volunteering Eagle Scout, Boy Scouts of America; STEM Achievement in Baltimore Elementary Schools; EN Engineering Community Involvement Committee; Engineers Without Borders; Creating Hall Events in Friedrick; University of Wisconsin Marching Band

Other Literature Researched Biofuels and sustainable energies; biomass processing; possible host organisms for bio-butanol production; organic photovoltaics; feasibility of engineering gut bacteria to mitigate lacose and gluten intolerance; tau protein aggregation in Alzheimer's disease

Anticipated Publications *A Parametric Rosetta Energy Function Analysis with LK Peptides on SAM Surfaces*
(in preparation)
CAPRI 2017 (work in progress)