

Structure, Flexibility, And Overall Motion Of Transmembrane  
Peptides Studied By NMR Spectroscopy And Molecular  
Dynamics Simulations

by

Tyler Reddy

Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy

at

Dalhousie University  
Halifax, Nova Scotia  
July 2011

© Copyright by Tyler Reddy, 2011

DALHOUSIE UNIVERSITY

DEPARTMENT OF BIOCHEMISTRY & MOLECULAR BIOLOGY

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “Structure, Flexibility, And Overall Motion Of Transmembrane Peptides Studied By NMR Spectroscopy And Molecular Dynamics Simulations” by Tyler Reddy in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dated: July 14, 2011

Supervisor:

Readers:

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Departmental Representative: \_\_\_\_\_

DALHOUSIE UNIVERSITY

DATE: July 14, 2011

AUTHOR: Tyler Reddy

TITLE: Structure, Flexibility, And Overall Motion Of Transmembrane Peptides  
Studied By NMR Spectroscopy And Molecular Dynamics Simulations

DEPARTMENT OR SCHOOL: Department of Biochemistry & Molecular Biology

DEGREE: PhD CONVOCATION: October YEAR: 2011

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than the brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

---

Signature of Author

# Dedication

To my parents (*John & Sandra*),

my sister (*Rose-Marie*),

and amazing friends from the Dalhousie

& Halifax running communities.

# Table Of Contents

List Of Tables	x
List Of Figures	xii
Abstract	xxii
List Of Abbreviations Used	xxiii
Acknowledgements	xxvi
Chapter 1: Introduction	1
Chapter 2: NMR Structure Of NHE1 TM IX (Based On Published Manuscript (1))	3
2.1 Introduction . . . . .	3
2.2 Materials And Methods . . . . .	4
2.2.1 Materials . . . . .	4
2.2.2 Peptide Synthesis And Purification . . . . .	4
2.2.3 NMR Spectroscopy And Structure Calculations . . . . .	5
2.2.4 CD Spectroscopy And Analysis . . . . .	7
2.3 Results . . . . .	7
2.3.1 Peptide Design And Conditions For NMR Spectroscopy . . . . .	7
2.3.2 Resonance Assignment And Structure Calculation . . . . .	8
2.3.3 Structural Analysis Of TM IX Peptide . . . . .	8

2.3.4	Structural Superposition Of TM IX . . . . .	10
2.4	Discussion . . . . .	10
2.4.1	Structural Analysis Of TM IX . . . . .	10
2.4.2	Structure-Function Correlation . . . . .	12
<b>Chapter 3: NMR Spin Relaxation Studies And The Dynamics Of</b>		
<b>NHE1 TM VII (Based On The Published Manuscripts (2) And</b>		
<b>(3))</b> . . . . . <b>27</b>		
3.1	Introduction . . . . .	27
3.1.1	Introduction To NMR Spin Relaxation . . . . .	28
3.1.1.1	Assessing The Trend Of $T_1$ , $T_2$ , And NOE With Mag- netic Field Strength . . . . .	33
3.1.1.2	The Overall Rotational Correlation Time, $\tau_c$ . . . . .	34
3.1.1.3	The Generalized Order Parameter, $S^2$ . . . . .	35
3.1.1.4	The Effective Internal Correlation Time, $\tau_e$ . . . . .	38
3.2	NMR Spin Relaxation Studies Of NHE1 TM VII . . . . .	39
3.2.1	NHE1 TM VII Background . . . . .	39
3.2.2	Materials And Methods . . . . .	40
3.2.2.1	Materials . . . . .	40
3.2.2.2	Peptide Synthesis And Purification . . . . .	40
3.2.2.3	NMR Spectroscopy . . . . .	41
3.2.2.4	$^{15}\text{N}$ Relaxation Parameters . . . . .	41
3.2.2.5	Model-Free Calculations . . . . .	42
3.2.2.6	Reduced Spectral Density Mapping . . . . .	43
3.2.2.7	Theoretical Calculations . . . . .	44
3.2.3	Results . . . . .	45
3.2.3.1	Relaxation Parameters: $T_1$ , $T_2$ , and NOE . . . . .	45
3.2.3.2	Reduced Spectral Density Mapping . . . . .	45

3.2.3.3	Model-Free Analysis . . . . .	46
3.2.4	Discussion . . . . .	47
3.2.4.1	Relaxation Parameters: Comparing Theory And Experiment . . . . .	47
3.2.4.2	Reduced Spectral Density Mapping . . . . .	48
3.2.4.3	Correlating Structure, Dynamics, And Function . . . . .	49
<b>Chapter 4: Technicalities Of Spitz Preparation</b>		<b>58</b>
4.1	Introduction . . . . .	58
4.2	Solid-Phase Peptide Synthesis . . . . .	60
4.3	Production And Purification Of A 34-Residue Spitz TMD Construct (tr-08-1) . . . . .	61
4.4	Production And Purification Of A 34-Residue C→S Spitz TMD Construct (tr-08-2) . . . . .	62
4.5	Production And Purification Of A 21-Residue Spitz TMD Construct (tr-09-1) . . . . .	64
4.6	Production And Purification Of A 16-Residue TatA TMD Construct (tr-10-2) . . . . .	68
4.7	Production Of A 37-Residue Gurken TMD Construct (tr-10-1) . . . . .	69
4.8	Production And Purification Of A Spitz Construct By Expression In <i>E. coli</i> . . . . .	70
4.9	Conclusions . . . . .	71
<b>Chapter 5: FGFR3 Simulation</b>		<b>108</b>
5.1	Fibroblast Growth Factor Receptors . . . . .	108
5.2	Achondroplasia: Specific Relevance Of FGFR3 . . . . .	108
5.3	Coarse-Grained Simulations And Glycophorin A Control . . . . .	110
5.4	Analysis Of FGFR3 And GpA <i>Dimer</i> Trajectories . . . . .	111

5.4.1	Tracking The Distance Between Helices In A Trajectory . . .	111
5.4.2	Relative Helical Motion . . . . .	113
5.4.3	Helix Crossing Angle . . . . .	114
5.4.4	Correlated Helical Motion . . . . .	115
5.4.5	Identification Of Predominant Interhelical Contacts . . . . .	116
5.4.6	Identification Of Dimer Interfaces . . . . .	118
5.4.7	Dimer Interface Transitions . . . . .	120
5.4.8	Identification Of Representative Dimer Interface Structures And Contacts . . . . .	121
5.4.9	Population-Based Dimer Interface Classification . . . . .	123
5.5	Analysis Of Lipid Bilayer In GpA And FGFR3 Simulations . . . . .	126
5.5.1	Testing For Phospholipid ‘Flip-Flop’ . . . . .	127
5.5.2	Protein-Local And -Distal Bilayer Thickness . . . . .	128
5.5.3	Protein-Local And -Distal Lipid Shell Counts . . . . .	129
5.5.4	Overall Bilayer Thickness Analysis For GpA And FGFR3 Sim- ulations . . . . .	131
5.5.5	DPPC: The Effect Of Phospholipid Type On FGFR3 Dimer Behaviour . . . . .	132
5.5.5.1	FGFR3 Dimer Stability In DPPC . . . . .	132
5.5.5.2	FGFR3 Dimerization Interface In DPPC . . . . .	133
5.6	FGFR3 Monomer Simulations . . . . .	134
5.6.1	Helix Tilt Angle . . . . .	134
5.6.2	FGFR3 SIDEKICK Monomer Simulations . . . . .	135
5.7	Summary And Conclusions . . . . .	136
<b>Chapter 6: Spitz-Rhomboid Simulation</b>		<b>191</b>
6.1	Introduction . . . . .	191
6.2	Tracking Enzyme-Substrate Separation . . . . .	192



6.3	Position Of Spitz In Fixed Rhomboid Reference Frame . . . . .	192
6.4	Analysis Of The POPE Lipid Bilayer . . . . .	193
6.5	Identification Of Predominant Interprotein Contacts . . . . .	194
6.6	Conclusions . . . . .	195
<b>Chapter 7: Conclusions</b>		<b>203</b>
<b>Appendix A: Source Code For Fractional Isotope Incorporation In Peptide Mass Calculations</b>		<b>205</b>
A.1	Introduction . . . . .	205
A.2	Source Code Proper . . . . .	205
<b>Appendix B: Additional Data From Spitz Peptide/Protein Production And Purification</b>		<b>216</b>
B.1	Mass Spectrometry Results . . . . .	216
B.1.1	TR-09-1 Construct . . . . .	216
B.1.2	TR-10-2 Construct . . . . .	249
B.1.3	TR-08-2 Construct . . . . .	249
<b>Appendix C: Source Code For Website Which Predicts NMR Spin Relaxation Parameter Trends With Magnetic Field Strength</b>		<b>253</b>
C.1	Introduction . . . . .	253
C.2	Source Code Proper . . . . .	253
<b>Appendix D: Source Code For FGFR3 MD Simulation Analysis</b>		<b>338</b>
D.1	Introduction . . . . .	338
D.2	Python Source Code For Trajectory Parsing With MDAnalysis . . . . .	338
<b>Bibliography</b>		<b>465</b>

# List of Tables

2.1	NHE1 TM IX NMR structural statistics summary . . . . .	25
2.2	Summary of secondary structure analysis of NHE1 TM IX CD spectra	26
3.1	Parameters for setup of NMR relaxation experiments . . . . .	55
3.2	The full set of $^{15}\text{N}$ relaxation parameters for NHE1 TM VII . . . . .	56
3.3	Model-free model selections and parameters for NHE1 TM VII . . . . .	57
4.1	TR-09-1 MALDI results summary (November 30/2009) . . . . .	102
4.2	TR-09-1 MALDI results summary (December 2/2009) . . . . .	103
4.3	TR-09-1 MALDI results summary (December 4/2009) . . . . .	103
4.4	TR-09-1 MALDI results summary (December 7/2009) . . . . .	104
4.5	TR-09-1 MALDI results summary for first bulk purification (December 22/2009) . . . . .	104
4.6	TR-09-1 MALDI results summary for second bulk purification (December 22/2009) . . . . .	105
4.7	TR-09-1 MALDI results summary for bulk purification starting at 40% ACN (January 5/2010) . . . . .	105
4.8	TR-09-1 MALDI results summary for bulk purification starting at 30% ACN (January 5/2010) . . . . .	106
4.9	Summary of TR-10-2 MALDI results . . . . .	107
5.1	The full set of FGFR3 and GpA <i>dimer</i> replicate simulations as they were tracked during production . . . . .	189

5.2	The full set of FGFR3 <i>monomer</i> replicate simulations as they were tracked during production . . . . .	190
-----	---	-----

# List of Figures

2.1	Comparison of NHE1 TM IX TOCSY and NOESY spectra . . . . .	17
2.2	Summary of NOE contacts in the context of NHE1 TM IX primary sequence . . . . .	18
2.3	Unique NOE restraints in the final NMR ensemble for NHE1 TM IX	18
2.4	Tracking total energy for NHE1 TM IX structure calculations . . . . .	19
2.5	$\alpha$ -proton secondary chemical shift analysis for NHE1 TM IX . . . . .	20
2.6	Circular dichroism spectropolarimetry of NHE1 TM IX in DPC micelles	21
2.7	Representative and superposed NHE1 TM IX NMR structures . . . . .	22
2.8	Ramachandran plot for NHE1 TM IX final NMR ensemble . . . . .	23
2.9	Dihedral angle order parameters for NHE1 TM IX in DPC micelles .	24
3.1	Sample one-dimensional $^1\text{H}$ - $^{15}\text{N}$ NMR spectrum for NHE1 TM VII . . .	52
3.2	$T_1$ , $T_2$ and steady-state NOE values for NHE1 TM VII at three field strengths . . . . .	53
3.3	Reduced spectral density mapping results for NHE1 TM VII at three magnetic field strengths . . . . .	54

4.1	A cartoon representation of the TM segments from the ecGlpG structure (PDB: 2IC8; (4)). The top of this representation would represent the extracellular environment, the loops between helices have been excluded, and helix colouring follows the scheme: TM 1 (dark blue), TM 2 (red), TM 3 (green), TM 4 (purple), TM 5 (orange), TM 6 (pink). Note that TM 4, which includes the catalytic serine residue, is protected from the exterior of the protein by the ring of TM helices. . . .	73
4.2	TR-08-2 C18 analytical HPLC trace (August 20/2008) . . . . .	74
4.3	TR-08-2 fraction 2 MALDI result (November 3/2008) . . . . .	75
4.4	TR-08-2 representative HPLC purification (November 13/2008) . . .	76
4.5	TR-08-2 HPLC trace (November 21/2008) . . . . .	77
4.6	TR-08-2 fraction 1 CD spectropolarimetry results . . . . .	78
4.7	TR-08-2 NOESY NMR spectrum . . . . .	79
4.8	TR-08-2 TOCSY NMR spectrum . . . . .	79
4.9	Comparing spitz TMD constructs for solid-phase peptide synthesis . .	80
4.10	Fractional isotope incorporation scheme for solid-phase peptide synthesis	81
4.11	TR-09-1 MALDI result (November 20/2009) . . . . .	82
4.12	TR-09-1 HPLC traces (November 26/2009) . . . . .	83
4.13	TR-09-1 crude HPLC trace (December 1/2009) . . . . .	84
4.14	C18 semi-preparative HPLC run for C3-column-purified TR-09-1 fraction 5 (December 2/2009) . . . . .	85
4.15	TR-09-1 C18 semi-preparative HPLC work (December 3/2009) . . . .	86
4.16	Direct purification of TR-09-1 by C18 semi-preparative HPLC (December 5/2009) . . . . .	87
4.17	TR-09-1 crude direct load to C18 semi-preparative HPLC column with proper fraction collection (December 6/2009) . . . . .	88
4.18	C18 analytical HPLC trace for two-column purified TR-09-1 fraction 5	89

4.19	TR-09-1 HPLC purification (December 8/2009)	90
4.20	TR-09-1 HPLC purification (December 9/2009)	91
4.21	TR-09-1 bulk purification by C18 semi-preparative HPLC	92
4.22	TR-09-1 bulk HPLC purification starting at 40% ACN	93
4.23	TR-09-1 bulk HPLC purification starting at 30% ACN	94
4.24	Comparison of selected TR-09-1 C18 semi-preparative HPLC purifications	95
4.25	TR-10-2 crude MALDI result	96
4.26	TR-10-2 crude MALDI result (zoom-in)	97
4.27	TR-10-2 C18 analytical HPLC (210 nm)	98
4.28	TR-10-2 C18 analytical HPLC (280 nm)	99
4.29	The expressed spitz construct (June 30/2009)	99
4.30	Unlabelled expressed spitz C3 semi-preparative HPLC	100
4.31	MALDI result for 50% <sup>15</sup> N-enriched expressed spitz	101
4.32	MALDI result demonstrating the presence of a monomer/dimer equilibrium for expressed spitz	102
5.1	Tracking closest C <sub>α</sub> helix-helix approach for the first GpA coarse-grained simulation.	138
5.2	Tracking closest C <sub>α</sub> helix-helix approach for the third WT FGFR3 coarse-grained simulation.	139
5.3	Tracking closest C <sub>α</sub> helix-helix approach for the ninth FGFR3 heterodimer coarse-grained simulation.	139
5.4	Tracking closest C <sub>α</sub> helix-helix approach for the second FGFR3 mutant homodimer coarse-grained simulation.	140
5.5	Summary of closest interhelical approach results for GpA and FGFR3	141
5.6	Tracking relative helical motion for the GpA dimer	142
5.7	Tracking relative helical motion for the FGFR3 WT construct	142

5.8	Tracking relative helical motion for the FGFR3 heterodimer construct	143
5.9	Tracking relative helical motion for the FGFR3 mutant homodimer construct . . . . .	143
5.10	Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the GpA WT homodimer construct . . . . .	144
5.11	Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the FGFR3 WT homodimer construct . . . . .	145
5.12	Helix crossing angle ( $\Omega$ ) distribution for the ninth replicate of the FGFR3 heterodimer construct . . . . .	146
5.13	Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the FGFR3 mutant homodimer construct . . . . .	147
5.14	Helix crossing angle ( $\Omega$ ) distribution merged over all replicates of GpA and FGFR3 constructs . . . . .	148
5.15	Correlating the closest $C_\alpha$ interhelical approach with helix crossing angle ( $\Omega$ ) for GpA and FGFR3 . . . . .	149
5.16	Tracking Z coordinate (along bilayer normal) for each helix $C_\alpha$ geometric center relative to center of bilayer during the first GpA replicate simulation . . . . .	150
5.17	The absolute correlation coefficients ( $ R $ ) between the Z coordinates of the geometric centers of the GpA or FGFR3 helices in each replicate simulation before and after dimerization . . . . .	151
5.18	Overall average and standard deviation for GpA and FGFR3 coordinated helical motion . . . . .	152
5.19	Closest contact probabilities for the first replicate GpA simulation . .	152
5.20	Closest contact probabilities taken from all FGFR3 simulation replicates for each of the three conditions . . . . .	153

5.21	A contour plot of the positional probability of GpA helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) in the first GpA replicate simulation . . . . .	154
5.22	A contour plot of the positional probability of helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) calculated over all ten replicate trajectories for each of the FGFR3 dimer conditions . . .	155
5.23	GpA dimer replicate 4 frame-abstracted position of helix 2 in rmsd-fixed frame of helix 1 . . . . .	156
5.24	A contour plot of the positional probability of helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) in the fourth mutant homodimer FGFR3 simulation . . . . .	157
5.25	FGFR3 mutant homodimer replicate 4 frame-abstracted position plot for helix 2 in reference frame of rmsd-fixed helix 1 . . . . .	158
5.26	Polar angle of helix 2 ( $\theta$ ) in the rmsd-fixed frame of helix 1 and helix crossing angle ( $\Omega$ ) are tracked during the first GpA replicate simulation	159
5.27	Polar angle of helix 2 ( $\theta$ ) in the rmsd-fixed frame of helix 1 and helix crossing angle ( $\Omega$ ) are tracked during the fourth FGFR3 mutant homodimer replicate simulation . . . . .	160
5.28	Direct correlation of polar angle ( $\theta$ ) and helix crossing angle ( $\Omega$ ) for the frames of representative FGFR3 simulations . . . . .	160
5.29	Representative coarse-grain $C_\alpha$ structures (N-terminus top) for the FGFR3 primary and secondary dimer interfaces . . . . .	161
5.30	FGFR3 reciprocal closest contact distances for each residue in each helix of a representative dimer interface structure . . . . .	162
5.31	Comparing coarse-grained MD simulation trajectory structural clustering by the single linkage method . . . . .	163



5.32	Comparing coarse-grained MD simulation trajectory structural clustering by the gromos algorithm . . . . .	164
5.33	Overall clustering comparison (gromos vs. single linkage) for GRO-MACS gcluster results for all GpA/FGFR3 replicate trajectories . . .	165
5.34	Adjusted (weighted moving average and spike-filtered) polar $\theta$ data for FGFR3 mutant homodimer replicate 4 . . . . .	165
5.35	Closest contacts for FGFR3 filtered by $\theta$ (dimer interface classification)	166
5.36	Network propagation issue for leaflet selection in MDAnalysis . . . .	167
5.37	Lipid flip-flop tracking results for the fourth replicate GpA simulation	167
5.38	Lipid flip-flop tracking results for the first replicate FGFR3 WT simulation . . . . .	168
5.39	Lipid flip-flop tracking results for the tenth replicate FGFR3 heterodimer simulation . . . . .	169
5.40	Lipid flip-flop tracking results for the tenth replicate FGFR3 mutant homodimer simulation . . . . .	170
5.41	Protein-local and protein-distal interphosphate bilayer thickness tracking results for GpA replicate 4 . . . . .	171
5.42	Protein-local and protein-distal interphosphate bilayer thickness tracking results for FGFR3 WT replicate 10 . . . . .	172
5.43	Protein-local and protein-distal interphosphate bilayer thickness tracking results for FGFR3 heterodimer replicate 10 . . . . .	173
5.44	Protein-local and protein-distal interphosphate bilayer thickness tracking results for FGFR3 mutant homodimer replicate 9 . . . . .	173
5.45	Lipid phosphate protein-local and -distal shell counts for the third replicate GpA simulation . . . . .	174
5.46	Lipid phosphate protein-local and -distal shell counts for the third replicate GpA simulation (zoom-in) . . . . .	174

5.47	Lipid phosphate protein-local and -distal shell counts for the seventh replicate FGFR3 WT simulation . . . . .	175
5.48	Lipid phosphate protein-local and -distal shell counts for the seventh replicate FGFR3 WT simulation (zoom-in) . . . . .	175
5.49	Lipid phosphate protein-local and -distal shell counts for the ninth replicate FGFR3 heterodimer simulation . . . . .	176
5.50	Lipid phosphate protein-local and -distal shell counts for the ninth replicate FGFR3 heterodimer simulation (zoom-in) . . . . .	176
5.51	Lipid phosphate protein-local and -distal shell counts for the seventh replicate FGFR3 mutant homodimer simulation . . . . .	177
5.52	Lipid phosphate protein-local and -distal shell counts for the seventh replicate FGFR3 mutant homodimer simulation (zoom-in) . . . . .	177
5.53	Average and standard deviation values for interphosphate bilayer thickness in GpA and FGFR3 simulation conditions . . . . .	178
5.54	Tracking the closest $C_{\alpha}$ interhelical distance for representative SIDEKICK-based CG simulations in DPPC bilayers . . . . .	179
5.55	Probability map for the position of helix 2 in the rmsd-fixed reference frame of helix 1 compared for FGFR3 CG simulations in POPC and DPPC . . . . .	180
5.56	Representative plot tracking the helix tilt angle of the FGFR3 WT monomer (replicate 1) relative to the bilayer normal . . . . .	181
5.57	Representative plot tracking the helix tilt angle of the FGFR3 mutant monomer (replicate 1) relative to the bilayer normal . . . . .	182
5.58	FGFR3 WT membrane burial depth from SIDEKICK monomer simulations in POPC . . . . .	183
5.59	FGFR3 G380R construct membrane burial depth from SIDEKICK monomer simulations in POPC . . . . .	184

5.60	FGFR3 WT helix tilt angle distribution from SIDEKICK monomer simulations in POPC . . . . .	185
5.61	FGFR3 G380R helix tilt angle distribution from SIDEKICK monomer simulations in POPC . . . . .	186
5.62	FGFR3 WT helix rotation angle distribution from SIDEKICK monomer simulations in POPC . . . . .	187
5.63	FGFR3 G380R helix rotation angle distribution from SIDEKICK monomer simulations in POPC . . . . .	188
6.1	Association between spitz and rhomboid TM5 monitored as the closest $C_\alpha$ approach in fourth replicate simulation . . . . .	196
6.2	$C_\alpha$ separation monitored between spitz and ecGlpG during the first replicate coarse-grained simulation . . . . .	197
6.3	Positional probability of the geometric center of the spitz TMD construct monitored in an rmsd-fixed reference frame . . . . .	198
6.4	Average protein-local and -distal bilayer thickness before and after spitz-rhomboid association (spitz starts near ecGlpG TMDs 1 and 3) . . . . .	199
6.5	Probability for each spitz (TMD construct) residue to reside in the closest contacts with ecGlpG (spitz starting near TMs 1 and 3) . . . . .	200
6.6	Predominant contacts between ecGlpG <i>helix 1</i> and the spitz TMD construct sorted by POPE bilayer burial depth . . . . .	201
6.7	Structure of ecGlpG with spitz-contact probability indicated for each residue . . . . .	202
B.1	MALDI-MS result for TR-09-1 fraction 4 with DTT treatment (November 30/2009) . . . . .	217
B.2	MALDI-MS result for TR-09-1 fraction 4 with no DTT treatment (November 30/2009) . . . . .	218

B.3	MALDI-MS result for TR-09-1 fraction 5 with DTT treatment . . . .	219
B.4	MALDI-MS result for TR-09-1 fraction 5 with no DTT treatment . .	220
B.5	TR-09-1 MALDI-MS result for fraction 5 (December 2/2009) . . . . .	221
B.6	TR-09-1 MALDI-MS result for fraction 6 (December 2/2009) . . . . .	222
B.7	TR-09-1 MALDI-MS result for fraction 7 (December 2/2009) . . . . .	223
B.8	TR-09-1 MALDI-MS result for fraction 8 (December 2/2009) . . . . .	224
B.9	TR-09-1 MALDI-MS result for fraction 9 (December 2/2009) . . . . .	225
B.10	TR-09-1 MALDI-MS result for fraction 12 (December 2/2009) . . . . .	226
B.11	MALDI-MS result for TR-09-1 (2-column purification) fraction 4 . . .	227
B.12	MALDI-MS result for TR-09-1 (2-column purification) fraction 5 . . .	228
B.13	MALDI-MS result for TR-09-1 (2-column purification) fraction 5 (zoom- in) . . . . .	229
B.14	MALDI-MS result for TR-09-1 (2-column purification) fraction 6 . . .	230
B.15	MALDI-MS result for TR-09-1 (direct C18 purification) fraction P1 (December 7/2009) . . . . .	231
B.16	MALDI-MS result for TR-09-1 (direct C18 purification) fraction P2 (December 7/2009) . . . . .	232
B.17	MALDI-MS result for TR-09-1 (direct C18 purification) fraction P3 (December 7/2009) . . . . .	233
B.18	MALDI-MS for a control solution of 50% H <sub>2</sub> O/ACN (0.1% TFA) (De- cember 7/2009) . . . . .	234
B.19	Representative MALDI-MS result for TR-09-1 C18 bulk purification (December 22/2009) . . . . .	235
B.20	The TR-09-1 ‘recovery fraction’ MALDI-MS result (December 22/2009)	236
B.21	MALDI-MS result for TR-09-1 fraction 1 (run #2) (December 22/2009)	237
B.22	MALDI-MS result for TR-09-1 fraction 10 (run #2) (December 22/2009)	238
B.23	MALDI-MS result for TR-09-1 fraction 14 (run #2) (December 22/2009)	239

B.24 TR-09-1 MALDI-MS result for fraction 1.1 (January 5/2010) . . . . .	240
B.25 TR-09-1 MALDI-MS result for fraction 1.13 (January 5/2010) . . . . .	241
B.26 TR-09-1 MALDI-MS result for fraction 2.9 (January 5/2010) . . . . .	242
B.27 TR-09-1 MALDI-MS result for fraction 2.23 (January 5/2010) . . . . .	243
B.28 TR-09-1 MALDI-MS result for sample TR-1 (January 14/2010) . . . . .	244
B.29 TR-09-1 MALDI-MS result for sample TR-2 (January 14/2010) . . . . .	245
B.30 TR-09-1 MALDI-MS result for sample TR-3 (January 14/2010) . . . . .	246
B.31 MALDI-MS result for control sample (January 14/2010) . . . . .	247
B.32 MALDI-MS result for TR-09-1 bulk purification recollected eluent (January 18/2010) . . . . .	248
B.33 TR-10-2 MALDI result for fraction 5 (April 23/2010) . . . . .	249
B.34 TR-10-2 MALDI result for fraction 6 (April 23/2010) . . . . .	250
B.35 MALDI-MS result for TR-08-2 fraction 1 (November 25/2008) . . . . .	251
B.36 MALDI-MS result for TR-08-2 fraction 2 (November 25/2008) . . . . .	252

## Abstract

Nuclear magnetic resonance (NMR) spectroscopy was used to determine the structure of transmembrane (TM) segment IX of the  $\text{Na}^+/\text{H}^+$  exchanger isoform 1 (NHE1) in dodecylphosphocholine micelles. Studying isolated TM segments in this fashion constitutes a well-established “divide and conquer” approach to the study of membrane proteins, which are often extremely difficult to produce, purify, and reconstitute in full-length polytopic form. A similar approach was combined with NMR spin relaxation experiments to determine the peptide backbone flexibility of NHE1 TM VII. The combined NMR structural and dynamics studies are consistent with an important role for TM segment flexibility in the function of NHE1, a protein involved in apoptosis and myocardial disease. The study of the rhomboid protease system is also described from two perspectives: 1) I attempted to produce several TM constructs of the substrate spitz or a related construct and the production and purification are described in detail; and 2) I present coarse-grained molecular dynamics simulation results for the *E. coli* rhomboid ecGlpG and a spitz TM construct. Spitz appears to preferentially associate with rhomboid near TMs 1 and 3 rather than the proposed substrate gate at TM 5. The two proteins primarily interact at the termini of helices rather than within the hydrocarbon core of the bilayer. Finally, I present a detailed analysis of coarse-grained molecular dynamics simulations of the fibroblast growth factor receptor 3 TM domain dimerization. Specifically, algorithms are described for analyzing critical features of wild-type and G380R mutant constructs. The G380R mutation is the cause of achondroplasia, the most common form of human dwarfism. The results suggest that the proximity of a residue to the dimer interface may impact the severity of the mutant phenotype. Strikingly, heterodimer and mutant homodimer constructs exhibit a secondary dimer interface which may explain the increased signaling activity previously reported for the G380R mutation—the helices may rotate with the introduction of G380R. The unifying theme of this work is the ‘study of membrane proteins’ using complementary techniques from structural biology and computational biochemistry.

# List Of Abbreviations Used

ACN	acetonitrile
AIC	Akaike's information criteria
Boc	tert-butoxycarbonyl
CD	circular dichroism
CG	coarse-grained
CG-MD	coarse-grained molecular dynamics
cNHE1	cysteineless NHE1
CSI	chemical shift index
DI-H <sub>2</sub> O	deionized water
DIEA	N,N-diisopropylethylamine
DMSO	dimethyl sulfoxide
DPC	dodecylphosphocholine
DPPC	dipalmitoylphosphatidylcholine
DSS	( <i>d</i> <sub>6</sub> )-2,2-dimethyl-2-silapentane-5-sulfonic acid
DTT	dithiothreitol

EGFR	epidermal growth factor receptor
ER	endoplasmic reticulum
ESI	electrospray ionization
FGF	fibroblast growth factor
FGFR	fibroblast growth factor receptor
FGFR3	fibroblast growth factor receptor 3
Fmoc	9-fluorenylmethoxycarbonyl
FRET	fluorescence resonance energy transfer
GAF	Gaussian axial fluctuation
GOBP	general odorant-binding protein
GpA	glycophorin A
HATU	2-(1H-7-azabenzotriazol-1-yl)-1,1,3,3-tetramethyl uronium hexafluorophosphate methanaminium
HBTU	O-benzotriazole-N,N,N',N'-tetramethyl-uronium-hexafluoro-phosphate
HPLC	high performance liquid chromatography
HSQC	heteronuclear single quantum correlation
IDP	intrinsically disordered protein
Ig	immunoglobulin
iRED	isotropic reorientational eigenmode dynamics
MALDI	matrix-assisted laser desorption/ionization



MD	molecular dynamics
MTSES	(2-sulfonatoethyl) methanethiosulfonate
MTSET	(2-(trimethylammonium) ethyl)methanethiosulfonate
NHE1	Na <sup>+</sup> /H <sup>+</sup> exchanger isoform 1
NMR	nuclear magnetic resonance
NOESY	nuclear Overhauser effect spectroscopy
PARL	presenilin-associated rhomboid-like
Pbf	2,2,4,6,7-pentamethyl-dihydrobenzofuran
POPC	1-palmitoyl-2-oleoyl-phosphatidylcholine
PPS	propeptide subtilisin
RTK	receptor tyrosine kinase
SNase	staphylococcal nuclease
SPPS	solid-phase peptide synthesis
tBu	tert-butyl
TFA	trifluoroacetic acid
TM	transmembrane
TMD	transmembrane domain
TOCSY	total correlation spectroscopy
Trt	triphenylmethane
WT	wild-type

# Acknowledgements

My PhD project supervisor, Dr. Jan K. Rainey, provided exceptional guidance for the duration of my thesis work in his laboratory. Jan's open-door policy, career advice, and willingness to send trainees to training workshops and conferences is commendable. Jan also encouraged my application for a Natural Sciences and Engineering Research Council of Canada (NSERC) Michael Smith Foreign Study Supplement. This scholarship allowed me to pursue an interest in learning to perform and analyze molecular dynamics simulations at one of the world-leading laboratories in this field at the University of Oxford. It strikes me as unlikely that most graduate supervisors would encourage their trainees to explore working for another group for six months, and this was one of the most important experiences of my PhD work. I am very thankful to Jan for supporting (or at least tolerating!) my transition from experimental to computational biochemistry.

Substantial collaborations include work with Drs. Brian D. Sykes, Larry Fliegel and Joanne Lemieux at the University of Alberta, who have all been very kind and helpful. A number of current and former members of the Rainey laboratory at Dalhousie University provided assistance for parts of my project: Bruce Stewart, Caitlin Reid, David N. Langelaan, Marie-Laurence Tremblay, Aaron Banks, Kyungsoo Shin, Lesley Seto, and Jonathan Melong. Drs. Stephen Bearne (also my honours project supervisor), David Waisman, Barbara Karten, Carmichael Wallace, and Andrew Roger kindly provided access to crucial experimental instruments. I thank Dr. Edward

d'Auvergne for extensive discussion of NMR spin relaxation calculations and Dr. Leo Spyropoulos for providing Mathematica notebooks and suggestions for relaxation analysis. Jason Moses and Marc Genest were involved in peptide synthesis and purification for an important part of my project. Dr. Mostafa Hatam (AAPPTec) and Wayne Kottkamp (JASCO) provided excellent technical advice.

I've also had the luxury of working in the laboratory of Dr. Mark S. P. Sansom at the University of Oxford. Mark was very kind for the duration of my stay and encouraged my interest in returning for post-doctoral studies. I'd like to thank a number of current and former members of the Structural Bioinformatics & Computational Biochemistry Unit (SBCB) for helpful discussions and suggestions: Dr. Philip W. Fowler, Dr. Oliver Beckstein, Dr. Phillip J. Stansfeld, Antreas Kalli, Khairul Adb Halim, and Dave Marshall.

I would also like to thank Dr. Boris Kablar, who supervised my early years of undergraduate summer research and encouraged a career in research. Much of my practical scientific training came from work in his group at the Anatomy & Neurobiology Department at Dalhousie University.

I have been supported by an NSERC Canada Graduate Scholarship (CGS) D, a Killam Honourary Predoctoral Fellowship, an NSERC CGS M, a Nova Scotia Health Research Foundation (NSHRF) Student Research Capacity Award, and Dalhousie University. I also acknowledge support from the Department of Biochemistry & Molecular Biology at Dalhousie University, and the SBCB and Department of Biochemistry at the University of Oxford. Parts of the research were supported by grants from Canadian Institutes of Health Research (CIHR), NSHRF, the E. Gordon Young Endowment Fund, NSERC, and the Protein Engineering Network of Centres of Excellence.

A 500 MHz spectrometer at the Nuclear Magnetic Resonance Research Resource (NMR-3; Halifax, NS) was used for several experiments with expert assistance from

Drs. Michael Lumsden and Kathy Robertson, and more recently a 700 MHz spectrometer at the Biomolecular Magnetic Resonance Facility (BMRF—National Research Council Canada; Halifax, NS) has been employed. Many NMR experiments were performed at the Canadian National High Field NMR Centre (NANUC; Edmonton, AB) on an 800 MHz spectrometer with assistance from Dr. Ryan McKay. Operation of NANUC is funded by CIHR, NSERC, and the University of Alberta. Molecular dynamics simulations were performed using resources provided at the SBCB (Oxford, U.K.) and by ACEnet, the regional high performance computing consortium for universities in Atlantic Canada. ACEnet is funded by the Canada Foundation for Innovation (CFI), the Atlantic Canada Opportunities Agency (ACOA), and the provinces of Newfoundland & Labrador, Nova Scotia, and New Brunswick.

# Chapter 1

## Introduction

If there is a single unifying theme to this body of work it would have to be the ‘study of membrane proteins.’ Chapter 2 (page 3) concerns the nuclear magnetic resonance (NMR) spectroscopic based elucidation of the structure of the ninth transmembrane (TM) segment of the  $\text{Na}^+/\text{H}^+$  exchanger isoform 1 (NHE1) in a membrane-mimetic medium. Studying an isolated TM segment is a well-established strategy to improve tractability by overcoming the difficulty of producing, purifying, and reconstituting polytopic membrane proteins. This “divide and conquer” approach (5) was also used to study the seventh TM segment of NHE1 as detailed in Chapter 3 (page 27), but in this case I examined the ps-ns and  $\mu\text{s}$ -ms timescale dynamics of peptide backbone flexibility in a membrane-mimetic environment using NMR spin relaxation experiments. I employed the common Lipari-Szabo model-free analysis (6, 7) as well as reduced spectral density mapping (8, 9) to gain insight into the dynamics of the system. The following two chapters are based on three of my published manuscripts (1, 2, 3).

In Chapter 4 (page 58), I switch to the study of the rhomboid protease system. However, in keeping with the above theme, rhomboids are polytopic *membrane proteins* which cleave single-pass TM proteins within the phospholipid bilayer in prokaryotes and eukaryotes (10). My specific objective was the production, purification, and NMR-based structural study of a construct of the TM portion of the *Drosophila*

spitz peptide—part of the natural substrate of Rhomboid-1. This is a labwork-heavy chapter as it was extremely challenging to produce and purify a number of different spitz-based or related constructs.

Although NMR-based structural and dynamics studies of membrane proteins can provide a lot of valuable information, one missing piece of information is how the peptide or protein structures move (on a larger scale) within phospholipid bilayers and interact with other proteins. I have employed coarse-grained molecular dynamics (CG-MD) simulations to study protein-lipid and protein-protein interactions for two membrane protein systems. In chapter 5 (page 108), the dimerization of the fibroblast growth factor receptor 3 (FGFR3) TM domain (of particular interest because of its role in human achondroplasia (*i.e.*, reference (11))) is studied in detail using CG-MD. This chapter integrates a detailed discussion of analytical algorithms used to parse crucial information from wild-type homodimer, heterodimer, and mutant homodimer FGFR3 simulation trajectories. Chapter 6 (page 191) outlines preliminary CG-MD simulation results for a system consisting of a rhomboid protease construct and a spitz TMD construct in a phospholipid bilayer.

This work therefore employs complementary techniques from structural biology and computational biochemistry to probe a number of membrane protein systems. The self-contained introductions in subsequent chapters will introduce the specific systems and techniques in detail, and the appendices contain documented source code and additional results.

# Chapter 2

## NMR Structure Of NHE1 TM IX (Based On Published Manuscript (1))

### 2.1 Introduction

The human  $\text{Na}^+/\text{H}^+$  exchanger isoform 1 (NHE1) is involved in a number of important biological processes. These include regulation of intracellular pH, cell growth and differentiation, cell migration, and regulation of sodium fluxes (reviewed in (12)). There is also substantial interest in NHE1 because of its established role in the myocardial damage that occurs during ischemia, reperfusion and a possible role in mediating cardiac hypertrophy. Despite the motivations for studying NHE1, there are no high-resolution atomic structures of the protein available in the literature.

NHE1 is a ubiquitously expressed plasma membrane protein, and based on functional and phylogenetic analysis it is proposed to include 12 N-terminal TM segments and a C-terminal regulatory domain (13, 14). Our objective was to use an NMR spectroscopy-based approach to obtain high-resolution atomic structural information on NHE1. In light of the poor tractability of the polytopic TM protein, we opted to use the “divide and conquer” approach to membrane protein structure determination (reviewed in (5)). This method consists of producing isolated TM segments, solving their structures in membrane-mimetic environments, and then recombining the

individual TM segment structures to get an overall picture of the polytopic configuration. The NMR structures of a number of NHE1 TM peptide segments have been solved as part of a “divide and conquer” strategy, and I have described the details of these structures (along with several successful examples of the “divide and conquer” approach) in the published manuscript (1).

We specifically chose to study NHE1 TM segment IX (residues 339-363 in the Wakabayashi topology (13)) for three reasons: 1) TM IX is important in mediating sensitivity to NHE1 antagonists (15); 2) Alteration of H349 in TM IX reduced exchanger sensitivity to amiloride (inhibitor) compounds (16); 3) Site-directed mutagenesis is consistent with additional importance for TM IX in NHE function and drug sensitivity (17, 18).

My work on a 31-residue synthetic peptide construct of NHE1 TM IX employed circular dichroism (CD) spectropolarimetry and NMR spectroscopy for structural characterization. In parallel, our collaborators in the laboratory of Dr. Fliegel (University of Alberta) conducted a cysteine-scanning mutagenesis study to determine the importance of each residue in TM IX for full-length cysteineless NHE1 (cNHE1) activity. The latter functional methods and results are detailed in the manuscript (1), while the structural results are described below with integration of crucial functional results where appropriate.

## **2.2 Materials And Methods**

### **2.2.1 Materials**

Deuterated dodecylphosphocholine (DPC) was purchased from C/D/N isotopes (Pointe-Claire, Quebec, Canada).

### **2.2.2 Peptide Synthesis And Purification**

A (> 95 %) purified 31-residue peptide construct of NHE1 TM IX containing



cationic caps at the N- and C- termini (KSYMAYLSAELFHLSGIMALIASGVVMPKK; acetyl-capped N-terminus, amide-capped C-terminus) was purchased from GL Biochem (Shanghai, China). Purity was assessed by high performance liquid chromatography (HPLC) and identity was confirmed by matrix-assisted laser desorption ionization (MALDI) mass spectrometry and sequential assignment of NMR spectra.

### 2.2.3 NMR Spectroscopy And Structure Calculations

An NMR sample was prepared by dissolving  $0.9 \pm 0.1$  mM peptide (concentration estimated from  $^1\text{H}$  one-dimensional NMR spectrum integrations relative to the internal standard using assigned resonances) in 95%  $\text{H}_2\text{O}$ , 5%  $\text{D}_2\text{O}$  solution containing  $\sim 75$  mM deuterated DPC and 0.25 mM ( $d_6$ )-2,2-dimethyl-2-silapentane-5-sulfonic acid (DSS) for chemical shift referencing. The experiments were conducted at 30 °C on the sample with pH 5.05 (without accounting for deuterium isotope effects). One-dimensional  $^1\text{H}$ , natural abundance gradient-enhanced  $^1\text{H}$ - $^{13}\text{C}$  HSQC (heteronuclear single quantum correlation), two-dimensional  $^1\text{H}$ - $^1\text{H}$  TOCSY (total correlation spectroscopy) (60-ms mix; decoupling in the presence of scalar interactions spin lock), and NOESY (nuclear Overhauser effect spectroscopy) (225-ms mix) experiments were acquired on the Canadian National High Field NMR Centre Varian INOVA 800-MHz spectrometer. All experiments were used as configured within the Varian BioPack software package. Spectra were processed using NMRPipe (19) and analyzed using Sparky 3 (20). All spectral assignments were carried out by manual peak picking in Sparky, and resonance assignments have been deposited in the BioMagResBank (code 15747).

Structure calculations were performed in the Python scripting interface of XPLOR-NIH version 2.18 (21) using NOE restraints from the assigned spectrum. Peak volumes were independently calculated using Gaussian and Lorentzian fit Sparky algorithms, with no allowed peak center motion. Those peaks that were not assigned a volume or were assigned a negative volume by the algorithm ( $\sim 27.5\%$ ) were either manu-

ally assigned a summed signal intensity as defined by a user-specified region or fit by Sparky algorithm after contour adjustment. Peak volumes were empirically calibrated to distance ranges of 0-5.0, 1.8-5.0, and 1.8-6.0 Å; the Lorentzian fit volumes were more conservative and used for subsequent analysis. Restraints were divided into bin ranges corresponding to strong (1.8-2.8 Å), medium (1.8-4.0 Å), weak (1.8-5.0 Å), and very weak (1.8-6.0 Å) contacts. Ambiguous assignments and NOE potential scaling were handled as described previously (22). Following 21 rounds of structural refinement by simulated annealing, an additional 6 rounds were performed with dihedral angle potential scaling factors of 5, 50, 100, 100, 50, and 25. Simulated annealing parameters were similar to those described previously (22), but with 15000 cooling steps.

A single extended polypeptide was generated and subjected to simulated annealing for each round. To handle the families of 100 structures generated per round an in-house tcl/tk script (freely available upon request from J. Rainey) was used; the script allowed for assessment of violations and iterative refinement of NOE restraints. Refinement gradually increased in stringency, initially violations  $> 0.5$  Å in  $> 50$  % of structures were lengthened, but subsequently violations  $> 0.1$  Å in  $> 25$  % of structures were modified. After 21 cycles of simulated annealing and NOE refinement, calculated XPLOR-NIH structure energies contained minimal contributions from NOE violations, and magnitudes of all observed violations were minimal. All NOE restraints were satisfied without pruning. Six further cycles of simulated annealing were carried out with incorporation of dihedral angle restraints as described above. The lowest 40 energy structures in the final ensemble of 100 were retained for further analysis. The final sets of restraints have been deposited in the Protein Data Bank with this ensemble of 40 structures (Research Collaboratory for Structural Bioinformatics Protein Data Bank code 2k3c).

## 2.2.4 CD Spectroscopy And Analysis

Samples for CD spectroscopy were diluted from the previously described NMR sample to obtain samples of  $\sim 10 \mu\text{M}$  peptide,  $\sim 3 \text{ mM}$  deuterated DPC, and the unbuffered pH was adjusted to  $\sim 4.8$  in a chloride-free solution. Nine replicate measurements were collected over 3 separate days on a Jasco J-810 spectropolarimeter (Easton, MD) at  $30 \text{ }^\circ\text{C}$  in a 0.1-cm path length (Hellma, Müllheim, Germany) water-jacketed cell (20 nm/min scan rate). Data were collected in a wavelength range of 260-180 nm but the signal to noise ratio was considered reasonable only at a detector (photomultiplier tube) voltage under 700 V (based on previous instrumental observation). Averaged data were analyzed by several algorithms (23) using the DICHROWEB interface (24, 25).

## 2.3 Results

### 2.3.1 Peptide Design And Conditions For NMR Spectroscopy

The synthetic peptide contains three lysine residues, one at the N terminus and two at the C terminus, which are not present in the endogenous sequence of the NHE1 protein. Cationic residues at the termini of peptides have been shown to facilitate both peptide purification and maintenance of transbilayer orientation (26). For convenience, the lysine residues are numbered relative to the endogenous sequence. All structural studies were conducted on the 31-residue peptide produced by chemical synthesis with no isotope labels.

TM peptide solubility is often problematic, and while NHE1 TM IV was soluble in organic solvents (27), TM VII was substantially more soluble in DPC micelles (22). We opted to use DPC micelles for solubilization of TM IX because this system is proposed to be a more appropriate membrane-mimetic than organic solvents (28). Sample components were  $0.9 \pm 0.1 \text{ mM}$  peptide,  $\sim 75 \text{ mM}$  deuterated DPC, and  $0.25 \text{ mM}$  DSS in 95%  $\text{H}_2\text{O}$ , 5%  $\text{D}_2\text{O}$  adjusted to pH  $\sim 5.05$  and studied at  $30 \text{ }^\circ\text{C}$ . This

temperature provided good NMR spectral characteristics, prevented precipitation for extended periods of spectroscopic study, and allows for use of the cryogenically cooled triple-resonance probe on the 800-MHz Canadian National High Field NMR Centre spectrometer. This combination of factors allowed determination of the structure of the TM IX segment in DPC micelles.

### 2.3.2 Resonance Assignment And Structure Calculation

Sequential chemical shift assignments were carried out using two-dimensional TOCSY, natural abundance  $^1\text{H}$ - $^{13}\text{C}$  HSQC, and NOESY experiments (22, 29) (Figure 2.1 on page 17). Acquisition of a natural abundance  $^1\text{H}$ - $^{15}\text{N}$  HSQC data set was not feasible because of low signal-to-noise arising from the tumbling rate of the peptide-containing micelles. Resonance assignments for  $^1\text{H}$  were complete except for Met- $\text{H}^\epsilon$  and Phe- $\text{H}^\delta$ . In some cases there were missing or ambiguous  $^{13}\text{C}$  assignments for Tyr- $\text{C}^\alpha$ , Met- $\text{C}^\alpha/\text{C}^\epsilon$ , Leu- $\text{C}^\beta/\text{C}^\gamma/\text{C}^{\delta 1}$ , Ser- $\text{C}^\alpha/\text{C}^\beta$ , Glu- $\text{C}^\alpha$ , Phe- $\text{C}^\beta$ , His- $\text{C}^\alpha/\text{C}^{\delta 2}$ , and Lys- $\text{C}^\gamma$ . Assessment of unambiguous assignments was made as described previously (22). The latter approach involves the use of ambiguous assignments in structure calculations where unambiguous assignments were not possible. A total of 1231 unique NOE restraints (Table 2.1 on page 25) were used for calculation of the TM IX structure. These are summarized graphically in terms of the standard connectivities examined for secondary structure characterization (Figure 2.2 on page 18) and in terms of the number of unique restraints per residue (Figure 2.3 on page 18). A progressive energy minimization of calculated NMR structural ensembles with each round of refinement is detailed in Figure 2.4 on page 19.

### 2.3.3 Structural Analysis Of TM IX Peptide

An ensemble of the 40 lowest energy structures from 100 calculated peptide structures was obtained that satisfy the 1231 observed unique NOE restraints with minimal violations (Table 2.1 on page 25). Dihedral angle restraints ( $\phi = -60 \pm 30$ ;

$\psi = -40 \pm 40$ ) consistent with  $\alpha$ -helical structure were imposed over residues M340-S344 and I353-S359, based on the secondary chemical shift analysis (Figure 2.5 on page 20). The dihedral restraints resulted in a significant but reasonable ( $\sim 17$  %) inflation in total energy relative to an ensemble restrained only by NOEs (Table 2.1 and Figure 2.4).

A 9-replicate averaged CD spectrum was analyzed using the DICHROWEB (24, 25) interface by applying several secondary structure deconvolution algorithms (results summarized in Table 2.2 on page 26). A 190 nm low-wavelength cutoff was used for meaningful signals as detailed in section 2.2.4. The normalized standard deviation parameter was used to filter those algorithms that did not produce a good fit to the data (normalized rmsd  $> 0.2$ ) (30). The calculated average secondary structure contributions are  $26 \pm 5$  % helix,  $31 \pm 6$  % sheet,  $16 \pm 4$  % turn, and  $29 \pm 10$  % random (Table 2.2). In light of the relatively high average deviations across algorithms, a qualitative inspection of the CD spectra is relevant (Figure 2.6 on page 21). To reflect the 12/31 ( $\sim 39$  %) helical residues predicted based on NMR dihedral restraints, a (theoretical) 39 % helix CD spectrum from 200 to 240 nm was generated by K2D (23). The experimental curves are qualitatively consistent with a strong helical contribution.

Superposition of all members of the ensemble over the full length of the peptide was not possible. However, based on rmsd analysis as detailed in (29), K337-L350 and G352-V362 were respective N- and C-terminal segments consistent with a relatively invariant structural fold (Figure 2.7 on page 22). The intervening S351 serves as a pivot point between the N- and C-terminal portions of the peptide, although its dihedral angles across the ensemble of structures are well clustered (Figure 2.8 on page 23), and its dihedral angle order parameters are close to unity, but in close proximity to a flexible region (Figure 2.9 on page 24). Residues assigned to the two helical segments were also well clustered by dihedral angle (Figure 2.8). While the

penultimate C-terminal K residue was surprisingly well-clustered by dihedral angle, M363 (also near the C-terminus) had a large dispersion of backbone dihedral angles over the ensemble (Figure 2.8).

### **2.3.4 Structural Superposition Of TM IX**

Peptide segments between 4 and 19 residues long were iteratively superimposed to provide a minimum rmsd relative to the backbone of the lowest energy structure (Figure 2.7 on page 22). Those superpositions producing the largest contiguous segments of residues with rmsd values  $< 1.0$  were additionally filtered based on average rmsd across the entire segment. The following permutations were assessed: N-terminal segments between residues 337-342 and 345-354 and C-terminal segments between residues 349-354 and 357-367 were superimposed by the method of Kabsch (31) as implemented in the LSQKAB software of the CCP4 suite (32). K337-L350 and G352-V362 were the respective N- and C-terminal regions with rmsd values most consistent with a fixed region of structure (Figure 2.7 on page 22). The dihedral angle order parameter (Figure 2.9) was consistent with a region of structural flexibility (*i.e.*, a pivot point at S351) between the determined segments.

## **2.4 Discussion**

### **2.4.1 Structural Analysis Of TM IX**

Where the full-length structure of proteins is available, studies have shown that isolated TM segments both reflect the structure of intact proteins and often retain their functional characteristics. Specific examples include bacteriorhodopsin (33, 34), rhodopsin (35), the cystic fibrosis TM conductance regulator (36, 37), and the fungal G-protein-coupled receptor Ste2p (38). Stabilization of the physiological structure of a TM segment may be solvent-dependent as observed previously with bacteriorhodopsin (33, 34), and TM IV (27) and TM VII (22) of NHE1. For this reason, we solubilized

the 31-residue synthetic TM IX peptide in membrane-mimetic DPC micelles, a system that has been well established (39, 40). We found that the structure of the TM IX peptide in DPC micelles is an interrupted helix with a sharp, although potentially flexible, bend immediately N-terminal to S351 (Figure 2.7 on page 22). This bend results in a kinked, “L”-shaped structure retained across amino acids 338-365. The relative position for the N- and C-terminal superposition segments around the S351 pivot is slightly variable across the ensemble. We have previously reported a similar observation for NHE1 TM VII in DPC micelles, which was also determined by NMR spectroscopy to be an interrupted helix, although with more variability in the angle of the bend between converged segments (22). The residues involved in the kink of TM VII were observed to be functionally critical residues. Similarly, it has been reported that helix IV of NHE1 is kinked at functionally important amino acids (27). The actual degree of flexibility in the context of the full-length protein would depend on constraining interactions arising from other helices, from the surrounding lipid bilayer, and through homodimer contacts (41). Kinked helices are thought to play an important role in transport function for the prokaryotic  $\text{Na}^+/\text{H}^+$  exchanger NhaA (42), and the trend of studies on TM segments of NHE1 shows residues appearing at kinked locations that are functionally critical.

There are two different membrane-spanning topologies for the NHE1 protein proposed in the literature. The first, by Wakabayashi and co-workers (13), is based on cysteine-scanning accessibility analysis and led to the present widely used model with TM IX including amino acids 338-360. More recently a model was developed based on fold recognition using the *E. coli* NhaA crystal structure as a template for phylogenetic analysis and homology modeling (14). In this model, amino acids 338-360 of TM IX correspond to part of TM VII, a subsequent extracellular loop, and all of TM VIII. Landau *et al.* (14) suggest that the previously described TM IX is an artifact of the window size used for hydropathy analysis. However, the TM VIII arising from

their model has a membrane-spanning length of  $\sim 19$  Å, which is short for a eukaryotic membrane (43).

## 2.4.2 Structure-Function Correlation

In addition to the structural studies described above, a complementary set of functional experiments on full-length cNHE1 in living cells were conducted by researchers in the laboratory of our collaborator (Dr. Fliegel, University of Alberta), and the results are detailed in the published manuscript (1). In all cases, they were able to express an intact  $\text{Na}^+/\text{H}^+$  exchanger protein. However, five of the mutants had decreased NHE1 protein activity. For L343 and S351 this was due, at least in part, to decreased expression and improper targeting. For Y339, I353, A355, and V361, significant defects in protein function were introduced by mutation. The E346, I353, and V361 side chains face the opposite direction of Y339, S351 and A355 side chains in our structure (Figure 2.7). This would seem to conflict with their importance in function; however, Landau *et al.* (14) suggested that S351 faces the lipid bilayer despite its functional importance and that it can rotate by  $180^\circ$  to face the pore for ion coordination. If true, the segment may sometimes face the pore and sometimes the lipid bilayer.

Only two mutants, E346C and S351C, were strongly inhibited by reaction with MTSET ((2-(trimethylammonium) ethyl)methanethiosulfonate) (a positively charged sulfhydryl-reactive reagent), and no mutants were inhibited by MTSES ((2-sulfonatoethyl) methanethiosulfonate) (a negatively charged sulfhydryl-reactive reagent). The most likely explanation for their reactivity with MTSET is an interaction at a site that lines and blocks the ion translocation pore (44). There were minor inhibitory effects of MTSET on the adjacent amino acids A345 and L350 (1). Their mutation could cause a minor perturbation of the structure of TM IX in this region that affects E346 and S351 but is also consistent with these residues having at least partial exposure to the pore. In support of this concept, A345, E346, and L350 all cluster with side chains



on the same face of the segment throughout the ensemble of structures. Although S351 is consistently (37/40 ensemble members) on the opposite face of the peptide, this observation may be explained by the rotation mechanism suggested by Landau *et al.* (14).

Positively charged MTSET but not negatively charged MTSES inhibited NHE1 activity in the E346C and S351C mutants. This contrasts with the results for the F161C mutant of TM IV that was inhibited by both compounds (27) but mirrors results with TM VII (45). MTSET may disrupt cation translocation by direct electrostatic repulsion, whereas negative MTSES would not. Alternatively, MTSES may not be able to react with these amino acids because of repulsion on the protein surface from negatively charged amino acids important in ion coordination (46). Local protein conformation and chemistry have been shown to affect accessibility to sulfhydryl-reactive reagents in both  $K^+$  channels (47) and in the FMRF-amide-activated sodium channel (48). The E346C cNHE1 mutant exhibited a higher degree of kinetic inhibition for the same measured parameters as S351C (1) and was similarly affected by MTSET (inhibition) and MTSES (no inhibition). In both topology models (13, 14) S351 is located in the intramembrane region of the protein. Therefore, our results are consistent with S351 being pore-lining in both cases and MTSET obstructing the pore. In the first topology model (13), E346 is also in the intramembrane region and should be pore-lining, potentially on the same face of a TM segment as S351. However, our structural analysis did not place these residues on the same face, and only 3/40 ensemble members show this positioning. This may suggest an extracellular loop position for E346, as proposed by Landau *et al.* (14). MTSET modification could then obstruct cation transport by preventing entry to the pore (E346).

The observed increase in  $K_{m(Na^+)}$  for S351C mutants (in kinetic studies by our collaborators detailed in the published manuscript) is consistent with at least moderately reduced extracellular coordination efficacy, and may therefore be supportive of

a role in the extracellular cation funnel for S351 as suggested by Landau *et al.* (14). The decrease in  $V_{max(Na^+)}$  may imply a measurable contribution to inhibition by a perturbation in overall pore structure or that ion coordination by S351 is rate-limiting (*i.e.*, that we are not simply observing reduced electrostatic interaction that can be overcome by ligand saturation). Finally, the reduced  $V_{max}$  values for  $H^+$  transport and especially the reduced intracellular (*i.e.*, allosteric) activation of S351C cNHE1 by  $H^+$  are in agreement with an overall perturbation in tertiary structure.

Based on dihedral restraints and rmsd analysis, our data suggest that M340-S344 is helical, and although this is consistent with these residues lining a TM segment as they do in both models, the lack of contiguous  $\alpha$ -helical character observed in the ensemble C-terminal to this segment until the second major helical segment (I353-S359) would be consistent with the extracellular loop assignment of residues 345-352 (14). The observed pivot at S351 (Figure 2.7 on page 22) also supports the possibility of an adjacent flexible extracellular loop at the location suggested by Landau *et al.* (14). E346 and G352 have been implicated as being critical in NHE1 inhibitor binding (17, 18). Placement of these residues at or near the extracellular surface is consistent with a site of drug interaction.

A distorted and interrupted helix with a bent L-shape would result if the peptide NMR structure was treated as a single TM segment (13). The bent structure we observe is consistent with a eukaryotic membrane span, the distance between  $C^\alpha$  of S338 and S359 is  $28 \pm 2 \text{ \AA}$  in our structures. Conversely, if the Landau *et al.* (14) topology is correct, residues H349 and V362 would be at the extracellular and intracellular faces, giving a transbilayer length of  $19 \pm 1 \text{ \AA}$ , a length typically assumed to be insufficient to span a eukaryotic membrane (43). To attempt to resolve these topological issues, our collaborators performed a series of experiments with amino acids 345-348 to determine the accessibility to the extracellular surface. However, it was not possible to get consistent results that either verified or disproved their

extracellular or TM location (not shown). Even when externally accessible residues are detected, they are sometimes assigned intracellular locations using the argument that they represent pore-lining residues (13), making assignment of location somewhat arbitrary. The functional studies (by our collaborators) showing inhibition by MTSET or our structural studies on the isolated TM IX peptide provided valuable data on the importance of these residues in function, but they are unable to shed light on their precise topology.

We found that amino acid M363 displayed an elevated dispersion of backbone dihedral angles, and this may suggest functionally important flexibility at this position (Figure 2.8 on page 23). However, the functional work of our collaborators showed that the M363C mutation did not eliminate activity and that it did not react with MTSET or MTSES (1), which is somewhat contradictory to the idea that it is critical to function. Wakabayashi *et al.* (13) suggested that M363C is in an extracellular loop. Landau *et al.* (14) placed M363 in an intracellular loop. Our results strongly support M363 as a loop residue displaying a high degree of flexibility, but they cannot localize this residue to either face of the membrane nor assign a critical pore-lining role to this amino acid.

Overall, the combination of my NMR structural studies with the work of our collaborators provides a detailed structural and functional picture of the functionally critical amino acids 339-363 of the NHE1 isoform of the  $\text{Na}^+/\text{H}^+$  exchanger. Our collaborators demonstrate that 5 of 25 residues are very sensitive to mutation to cysteine, and that E346 and S351 are involved in cation translocation and likely line the cation pore (1). The ensemble of structures, representing a kinked helical peptide, is similar to that previously reported for TM VII (22), but with a larger bend angle at the pivot point, S351. Amino acids M340-S344 and I353-S359 are helical. To resolve whether amino acids 339-363 represent one TM segment or parts of two, a full-length structure of NHE1 is necessary. It is encouraging that expression and purification

of full-length NHE1 has been achieved, but currently only a low-resolution structure obtained by electron microscopy is available (41).

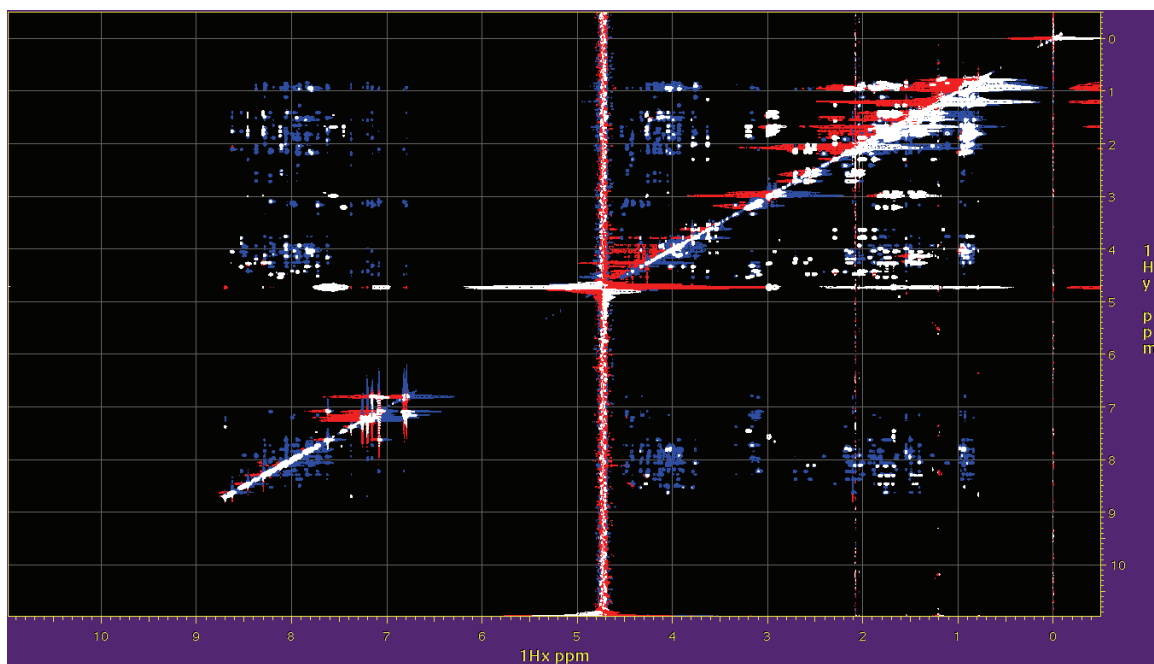


Figure 2.1: The NHE1 TM IX construct in DPC micelles was studied by TOCSY (white peaks) and NOESY (blue peaks) NMR experiments. There are substantially more peaks in the  $^1\text{H}$ - $^1\text{H}$  NOESY (225-ms mix) experiment than in the  $^1\text{H}$ - $^1\text{H}$  TOCSY (60-ms mix; decoupling in the presence of scalar interactions spin lock) on an 800 MHz spectrometer. This is consistent with the detection of through-space interactions in the NOESY and the restriction to scalar correlations in the TOCSY.



Figure 2.2: A representation of NOE contacts, which are limited to internuclear distances of  $\leq 6 \text{ \AA}$ , between residues in the context of the primary sequence of NHE1 TM IX. This representation is modified from CYANA (L.A. Systems, Inc.) output and is consistent with the contacts outlined in Figure 2.3 on page 18 (*i.e.*, consider the paucity of long-range NOE restraints).

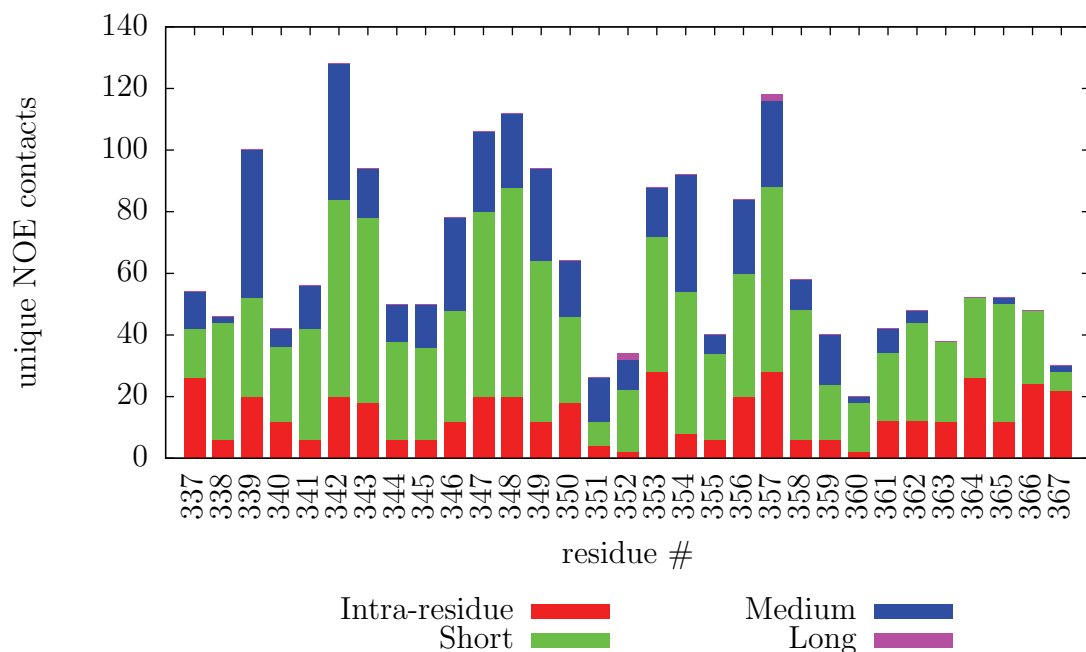


Figure 2.3: The number of per-residue unique NOE contacts in the final set of NMR restraints for NHE1 TM IX is summarized in this stacked histogram modified from CYANA (L.A. Systems, Inc.) output. Medium range restraints vary between 2 and 4 positions in the primary sequence, while long range restraints are five or more residues apart.

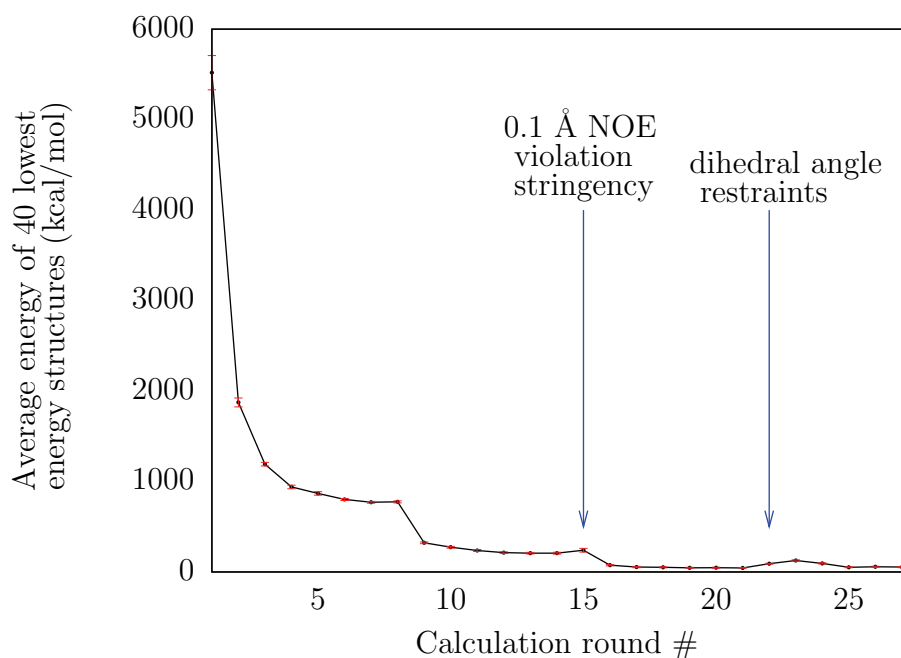


Figure 2.4: Tracking the total energy from each of the 27 rounds of XPLOR-NIH structure calculations for NHE1 TM IX. After each round, the average energy and standard deviation of the 40 lowest energy structures (out of 100 produced) were calculated. Iterative refinement of NOE restraints progressed by allowing restraints with violations  $> 0.5 \text{ \AA}$  in  $> 50\%$  of structures to be lengthened, and after round 15 restraints with violations  $> 0.1 \text{ \AA}$  in  $> 25\%$  of structures were lengthened. The latter progression in NOE restraint stringency clearly resulted in a reduction in average total energy for the calculated ensemble. After round 21, dihedral angle restraints consistent with  $\alpha$ -helicity were imposed over candidate helical residues based on secondary chemical shifts. The new restraints account for the rise in total energy in round 22, and energy returns to the optimized level after subsequent rounds.

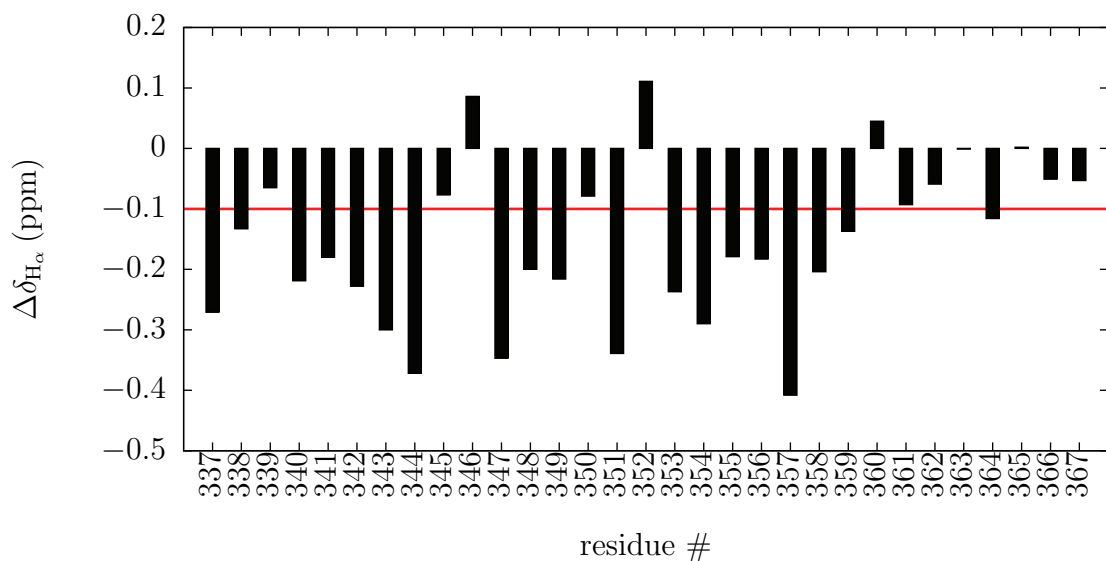


Figure 2.5: The secondary chemical shift ( $\Delta\delta$ ) for NHE1 TM IX  $H_\alpha$  nuclei is calculated as  $\delta_{H_\alpha} - \delta_{\text{random coil}}$ . The random coil values for each of the amino acids were obtained from (49), and observed shifts upfield to random coil values are consistent with helicity. Specifically, the widely-used chemical shift index (CSI) uses a secondary chemical shift requirement of  $< -0.1$  as an indicator of  $\alpha$ -helical secondary structure (50), and the red horizontal line marks this threshold.



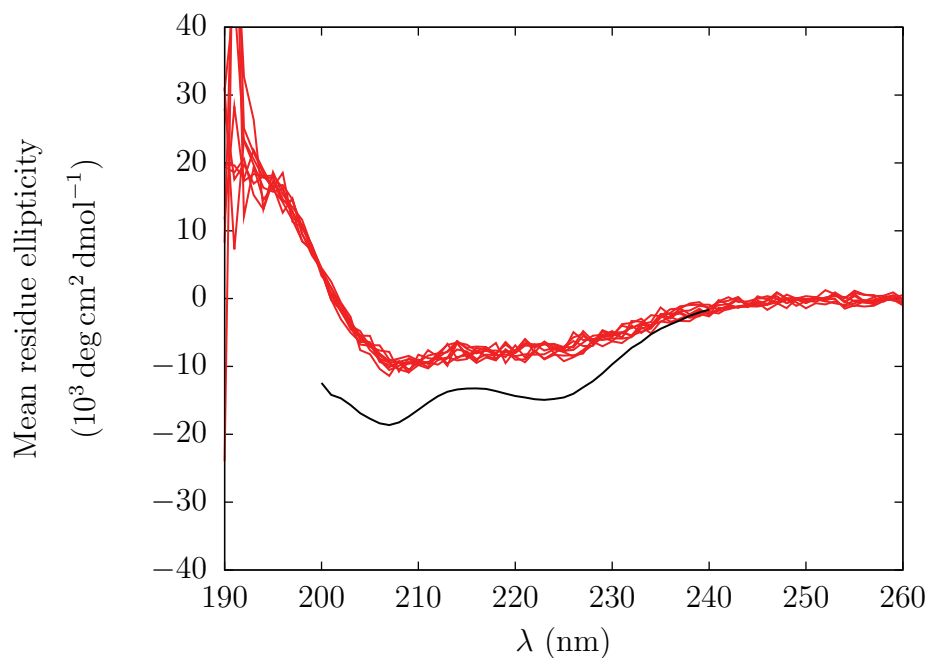


Figure 2.6: The synthetic NHE1 TM IX peptide NMR sample was diluted to produce a solution composed of  $\sim 10 \mu\text{M}$  peptide and  $\sim 3 \text{ mM}$  deuterated DPC (pH  $\sim 4.8$ ). Nine replicate CD measurements were collected over three separate days on a Jasco J-810 spectropolarimeter at  $30 \text{ }^\circ\text{C}$  in a  $0.1 \text{ cm}$  path length water-jacketed cell, and these results are plotted in red. For comparison, the predicted CD spectrum for a peptide with  $\sim 39 \%$   $\alpha$ -helical content is shown in black (prediction by the K2D algorithm (23)).

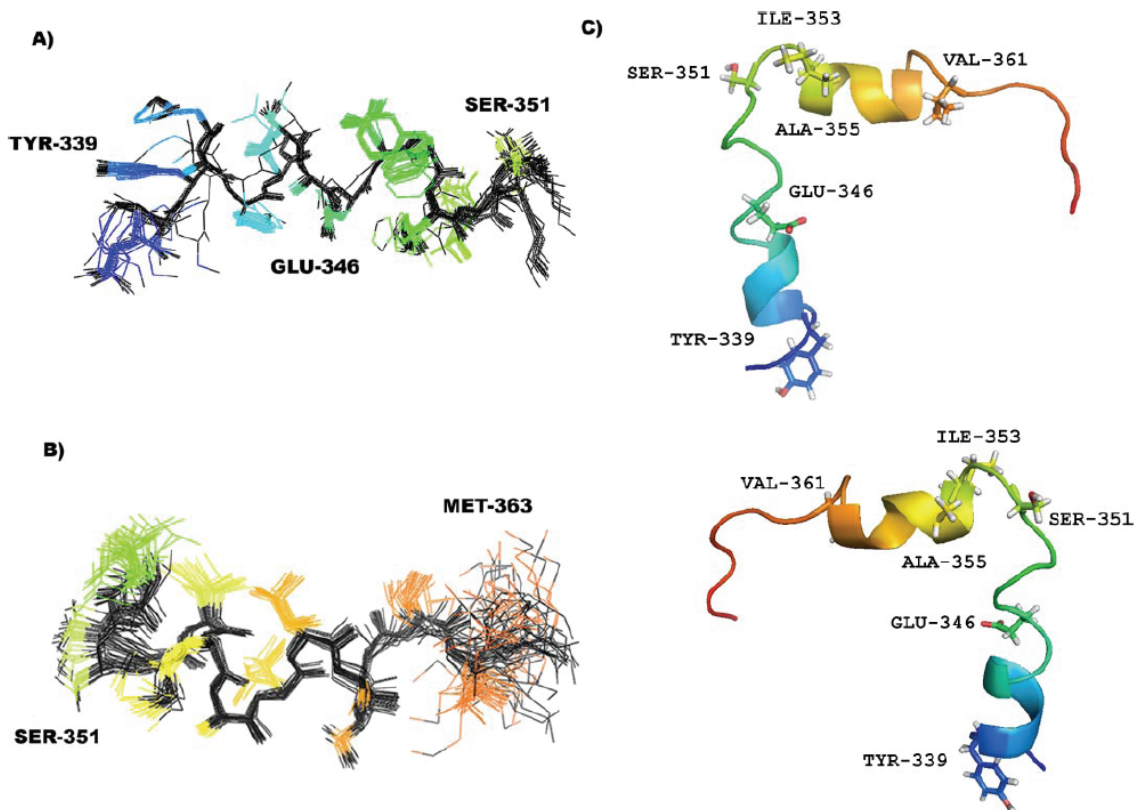


Figure 2.7: Assessment of superposable segments of the NHE1 TM IX NMR ensemble was performed using the LSQKAB program of the CCP4 suite (32), with evaluation of optimal superposition using criteria that include minimization of both local and global rmsd. The results include two superposable segments: between residues 337-350 (A) and 352-362 (B). Both of these segments have been extended in this figure to include the putative pivot point at S351, which was also found to be critical for NHE1 activity. Other functionally relevant residues (Y339, E346) and the strikingly mobile M363 (see Figure 2.8 on page 23) are also highlighted. Two alternative views of the lowest energy NHE1 TM IX NMR ensemble member are also shown (C) with colouring from N-terminus (blue) to C-terminus (red). This is a representative structure, but the angle of the kink near S351 and side chain positions are variable in the ensemble.

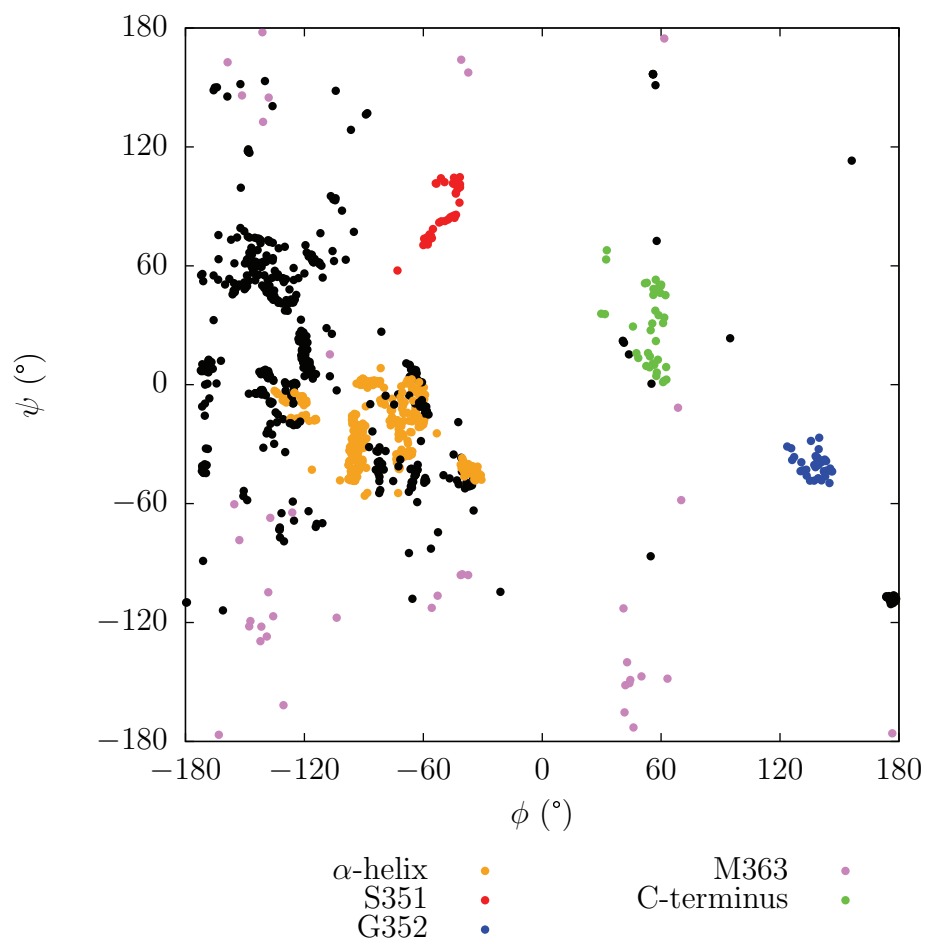


Figure 2.8: Ramachandran plot for the residues of NHE1 TM IX with  $\phi$  and  $\psi$  values from all 40 retained structures in the final NMR ensemble.  $\alpha$ -helical residues, the penultimate C-terminal residue, the pivot point at S351, the well-clustered G352 dihedral angles, and the strikingly variable M363 dihedral angles are all highlighted.

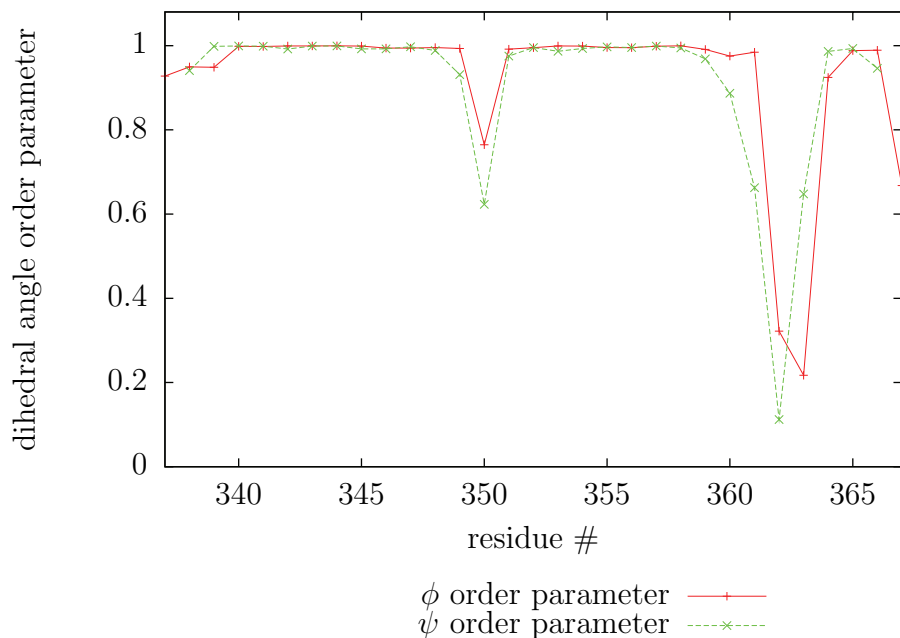


Figure 2.9: The backbone dihedral angle order parameters ( $\phi$ ,  $\psi$ ) were calculated for each residue in the NHE1 TM IX construct based on the final ensemble of retained structures, as described in (51). An order parameter of 1.0 indicates a dihedral angle that is consistent over the entire ensemble of NMR structures, a value of 0.0 is consistent with a completely random distribution of dihedral angles, and intermediate values represent varied degrees of dihedral angle fluctuation within the NMR ensemble. Angular standard deviation is not used because truly random angle distributions are not defined.

Table 2.1: Summary of statistics for the final retained ensemble of 40 NMR structures (out of 100 calculated) for NHE1 TM IX.

	dihedral restraints	no dihedral restraints
<b>Unique NOE restraints</b>		
Total	1231	
Intra-residue	432	
Sequential	536	
Medium range $ i - j  \leq 4$	238	
Long range $ i - j  > 4$	2	
Ambiguous	23	
<b>Ramachandran plot statistics</b>		
Core	54.3 %	
Allowed	34.6 %	
Generously allowed	7.0 %	
Disallowed	4.0 %	
<b>XPLOR-NIH energies (kcal/mol)<sup>a</sup></b>		
Total	54.2 ± 5.6	44.8 ± 3.1
NOE	9.95 ± 2.4	8.69 ± 2.3
<b>NOE violations</b>		
Violations > 0.5 Å	0	0
Violations of 0.3-0.5 Å	13	15
Violations of 0.2-0.3 Å	25	33

<sup>a</sup>average deviations shown

Table 2.2: Summary of NHE1 TM IX peptide secondary structure estimates based on various algorithms available via the DICHROWEB interface (24, 25). Nine replicates of CD data collected over three days between 260 and 190 nm were averaged and submitted as input. The % structural contributions are aggregate values even for algorithms which estimate the number of discrete contiguous segments for a given secondary structure type (52), and average deviations are shown with average values.

Algorithm	Reference database	Helix (%)	Sheet (%)	Turn (%)	Random (%)	Total (%)	Normalized rmsd
K2D	None <sup>a</sup>	30	14	N/A	55	99	0.167
CDSSTR	4	21	34	20	25	100	0.018
CDSSTR	7	22	38	20	19	99	0.016
CDSSTR	SP175 (53)	19	39	8	33	99	0.021
CONTIN-LL	4	31.2	29.9	15.0	23.9	100.0	0.196
CONTIN-LL	7	31.7	31.5	16.9	20.0	100.1	0.196
Average		26 ± 5	31 ± 6	16 ± 4	29 ± 10	102	

<sup>a</sup>K2D is a neural network algorithm that does not directly require a reference protein database.

## Chapter 3

# NMR Spin Relaxation Studies And The Dynamics Of NHE1 TM VII (Based On The Published Manuscripts (2) And (3))

### 3.1 Introduction

The previous chapter (chapter 2 on page 3) introduced the biological significance of the human NHE1 protein and I presented my structural NMR studies of the ninth TM segment of NHE1. The structures of a number of other NHE1 TM segments have also been studied by NMR spectroscopy in membrane-mimetic environments as part of a “divide and conquer” approach to the study of NHE1 structure. A common theme in many of these structures is the presence of a disruption in helicity—a kink in the regular secondary structure which may confer structural flexibility. However, apparent flexibility in an NMR ensemble may result from a lack of observable NOE restraints and not necessarily true motion in the peptide. NMR spin relaxation experiments are well suited to probing the ps-ns and  $\mu$ s-ms time scale dynamics of amino acids, and I employed  $^{15}\text{N}$  backbone relaxation NMR experiments to probe the dynamics at selectively labelled positions in the NHE1 TM VII peptide construct reconstituted in DPC micelles. This introduction is based on a review of NMR spin

relaxation I published (3), and followed by a section based on the NHE1 TM VII spin relaxation studies I published (2).

### 3.1.1 Introduction To NMR Spin Relaxation

NMR spectroscopy is a powerful technique for studying the dynamics of biomolecules using spin relaxation experiments. Frequently, when studying a protein or peptide, the objective of a backbone  $^{15}\text{N}$  NMR dynamics study is to identify and differentiate restricted versus mobile residues in the primary sequence and relate this information to structure and function. This type of information has led to many biologically relevant insights, with some recent examples following.  $^{15}\text{N}$  relaxation analysis of PSE-4  $\beta$ -lactamase revealed  $\mu\text{s}$ - $\text{ms}$  time scale mobility for residues in the active site, consistent with the ability of this enzyme to degrade a number of diverse  $\beta$ -lactam antibiotics (54). General odorant-binding proteins (GOBPs) also have promiscuous interaction profiles, and a recent relaxation study of honeybee GOBP ASP2 revealed increased backbone  $^{15}\text{N}$ - $^1\text{H}$  ms time scale mobility at the ligand entry site (55).  $^{15}\text{N}$  spin-lattice ( $T_1$ ), spin-spin ( $T_2$ ), and heteronuclear nuclear Overhauser effect (NOE) relaxation measurements revealed positions of active site flexibility in the essential dihydrofolate reductase enzyme from *Bacillus anthracis* (56), the highly resilient causative agent of anthrax. This structure-activity relationship is important for the design of potent inhibitors.

Binding events are also frequently detected by NMR relaxation. The acidic residues in calcium-binding domain 1 of the  $\text{Na}^+/\text{Ca}^{2+}$  exchanger are significantly restricted upon binding of calcium (57). In contrast, *Escherichia coli* 6-hydroxymethyl-7,8-dihydropterin pyrophosphokinase exhibits increased  $^{15}\text{N}$ - $^1\text{H}$  mobility when binding substrate analogues, mobilizing loops that may stabilize the transition state (58). Based on a September 2009 ISI Web of Knowledge (Thomson Reuters Inc.) search, nearly 2000 original research articles and 170 reviews cite the original Lipari and Szabo (6) model-free approach to NMR relaxation analysis. This demon-



strates not only the popularity of the Lipari-Szabo model-free approach but, more generally, the widespread degree to which biomolecular NMR relaxation methods are used. Accordingly, the field of NMR spin relaxation has been extensively reviewed (59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75).

Understandably, in the more than 60 years since the first detection of an NMR signal (76, 77), the methodologies and computer software employed for extracting dynamics information have diverged. In practice, several programs may be compared in the calculation of relaxation parameters. d’Auvergne and Gooley (78) present a detailed analysis of the limitations of various algorithms used for the interpretation of NMR relaxation data. For the NMR spectroscopist uninitiated to spin relaxation theory, however, the ready analysis of experimental data is compounded by the diversity of analytical approaches, the use of CGS versus SI units in defining constants and equations by different authors, and the lack of available sample calculations. It is therefore difficult to assess the quality of newly collected relaxation data without prior experience in the field. I will therefore review relaxation parameters and describe a web interface for graphing the theoretical trends of relaxation parameters to provide a first-check as to whether collected relaxation data matches with the predictions of theory. This is a crucial concern for the NMR spectroscopist preparing to embark on an exhaustive relaxation analysis for the first time. Thorough theoretical reviews of NMR spin relaxation analysis are available (*i.e.*, (71, 69)), but are not necessarily accessible to the spectroscopist new to relaxation. Although our focus is on solution-state peptide and protein  $^{15}\text{N}$ -backbone NMR relaxation experiments, the fundamental principles are applicable to other biomolecular systems, and a diverse set of examples are considered that may be placed in an experimental context using the new web interface I introduce.

Before diving into the specifics, it is important to understand the microscopic origins of spin relaxation in solution-state NMR. Perhaps the most intuitive way to

understand the NMR spin relaxation process is to first consider the fully relaxed thermal equilibrium state (79). At thermal equilibrium, two important conditions are met: 1) the populations of each spin state (*i.e.*,  $\alpha$  and  $\beta$ , for a spin 1/2 nucleus) are given by the Boltzmann distribution; and 2) all coherence between  $\alpha$  and  $\beta$  spin-states is lost. Spin excitation with an on-resonance radiofrequency pulse may disturb the equilibrium and break either of these conditions. For example, a  $\pi$ -pulse inverts the spin state populations (*i.e.*,  $\alpha$  and  $\beta$  populations are exchanged, for a spin 1/2 nucleus), breaking the condition of a Boltzmann distribution. Alternatively, a  $\pi/2$ -pulse produces coherent transverse magnetization, breaking the second equilibrium condition. Return to thermal equilibrium occurs by two relaxation mechanisms.

The restoration of the Boltzmann distribution, or spin-lattice relaxation (characterized by a time constant  $T_1$ ), occurs as a result of the coupling of excited spins with the surrounding environment, termed the lattice for historical reasons (80). Random reorientation of molecules in a liquid modulates the local magnetic field experienced by a given spin. In typical macromolecular cases, two orientationally dependent components come into play: the magnetic fields induced by direct dipole-dipole coupling with neighbouring spins, which depend upon relative dipole orientation; and the fields caused by molecular electron currents, which depend upon molecular orientation relative to the static magnetic field. These fluctuations drive spin-lattice relaxation by inducing transitions between spin states, specifically by field fluctuations at the Larmor frequency of the nucleus.

Spin-spin relaxation (characterized by  $T_2$ ) refers to the restoration of the second condition of thermal equilibrium, a loss of coherence between spins, as a result of individual variation in Larmor frequency for each spin due to local field fluctuations (81). Unlike spin-lattice relaxation, spin-spin relaxation of a given nuclear spin does not rely on a transition of *total* spin state; however, field fluctuations are dependent on spin state transitions by neighbouring nuclei.

Enhancement of magnetization relative to the thermal equilibrium level by the heteronuclear NOE is also frequently exploited for dynamics measurements in biomolecular NMR. The NOE is, of course, dependent upon the dipole-dipole distance and the relative gyromagnetic ratios of each spin (79). Assuming that these factors are known for a given pair of heteronuclei in sufficient proximity for the NOE to occur, the enhancement of longitudinal magnetization for a non-irradiated spin that is dipole-dipole coupled to a spin being irradiated by a weak radiofrequency field depends on cross-relaxation between the irradiated spin and non-irradiated spin. The cross-relaxation rates (zero-quantum and double-quantum transitions) depend on the molecular tumbling rate (81).

It is intuitive that spin relaxation is sensitive to dynamics, the motion of bond vectors for example, because each of these relaxation processes is driven by motion. After collection of  $T_1$ ,  $T_2$ , and NOE relaxation data, there are two main routes for mathematical analysis: 1) the Lipari and Szabo (6) model-free approach mentioned above; and 2) reduced spectral density mapping (8, 9).

The model-free approach provides intuitive motional parameters.  $S^2$ , the order parameter, is a measure of motion amplitude and ranges from 0 for an isotropic bond vector to 1 for complete restriction of motion.  $S^2$  does not provide sufficient information to obtain the path or model of motion for a bond vector—hence the model-free label for this technique (59). The model-free approach assumes separability of internal bond vector motion and overall molecular motion. The former timescale is quantified as the effective internal correlation time,  $\tau_e$ , whereas the latter is referred to as the overall rotational correlation time,  $\tau_c$ . These three model-free parameters are generally easier to interpret than the spectral density values obtained by reduced spectral density mapping.

The spectral density function,  $J(\omega)$ , is the Fourier transform of the rotational correlation function. In  $^{15}\text{N}$  biomolecular spin relaxation, the correlation function is

typically defined for the motion of a  $^{15}\text{N}$ - $^1\text{H}$  bond vector (6).  $J(\omega)$  is thus defined as the probability of observing rotational motion at a particular frequency,  $\omega$ . The case of  $J(0)$ , for example, could represent the (relative) population of a particular backbone  $^{15}\text{N}$ - $^1\text{H}$  bond vector that has zero frequency, or that is rotationally immobile, at any given time. As a whole, spectral density mapping provides the capability to demonstrate the relative degree of high- to low-frequency oscillations for a given bond vector (extensively reviewed in (64)). It has one significant advantage over the model-free approach, in that it does not require that internal bond vector motions occur independent of overall molecular tumbling.

Motional separability tends to be a particularly poor assumption for non-globular, unfolded proteins (59). As an example, Eliezer *et al.* (82) characterized the progression of myoglobin from an unfolded precursor to the fully folded globular protein by NMR relaxation experiments, demonstrating much greater mobility for residues in the unfolded versus folded state of the protein. Clearly, local reptation implied by dramatically increased mobility on a per-residue basis in an unfolded protein makes the separability of internal and overall motions unlikely. In any situation where this separability is suspect, spectral density mapping as opposed to, or at least in parallel to, model-free analysis may be essential (59).

Although I focus here on  $^{15}\text{N}$  as a probe of local dynamics,  $^2\text{H}$  and  $^{13}\text{C}$  isotopes may also be used to study the dynamics of biomolecules, as is especially popular for the study of amino acid side chain motion. Isotope enrichment is typically employed, but it should be noted that  $^{13}\text{C}$  relaxation experiments may also be performed at natural abundance (83). Uniform labelling with  $^{13}\text{C}$  and fractional labelling with  $^2\text{H}$  was used with  $^2\text{H}$ -NMR relaxation experiments to study the side chain methyl group dynamics in the C-terminal SH2 domain of phospholipase- $\text{C}_{\gamma 1}$  (84), the first solution-state NMR study of this kind. The relaxation of a deuteron is dominated by the quadrupolar interaction that can simplify interpretation of relaxation data relative

to other nuclei, which can have multiple significant contributions to relaxation. Tugarinov *et al.* (85) introduced a new set of deuterium NMR relaxation experiments, allowing the study of side chain dynamics for large (~100 kDa) proteins using malate synthase G (a 723 residue single polypeptide chain enzyme) as the test system for the experiments. Experiments using different methyl isotopomer probes ( $\text{CHD}_2$ ;  $\text{CH}_2\text{D}$ ) were consistent with each other, with previous experiments, and with molecular dynamics simulations. A comparison of the strengths and weaknesses of  $^2\text{H}$  and  $^{13}\text{C}$  spin relaxation experiments has been conducted on recombinant human ubiquitin (86).  $^2\text{H}$  relaxation methods were better for providing methyl symmetry axis information with limited data, whereas  $^{13}\text{C}$  relaxation provided more robust model-free parameters for the time scale of methyl rotation and methyl symmetry axis motion.

#### **3.1.1.1 Assessing The Trend Of $T_1$ , $T_2$ , And NOE With Magnetic Field Strength**

Any biological insights from NMR relaxation result from translating raw experimental peak height or volume data into a set of intuitive relaxation parameters relating a residue to its mobility. The NMR pulse sequences and the calculation of initial  $^{15}\text{N}$   $T_1$ ,  $T_2$ , and NOE values are well described by the seminal work of Farrow *et al.* (87). Since it is quite common to perform these experiments at multiple magnetic field strengths (8, 88), one of the first checkpoints of data integrity is the trend of  $T_1$ ,  $T_2$ , and NOE with magnetic field strength. For this reason, I have created an online plotting tool (available at [http://structbio.biochem.dal.ca/jrainey/Tyler\\_relaxation/](http://structbio.biochem.dal.ca/jrainey/Tyler_relaxation/)) using the Python code in appendix C on page 253, which accepts a number of NMR relaxation parameters that can be tailored to a specific system to produce the theoretical trends of the relaxation times  $T_1$  and  $T_2$  (as well as the related rates  $R_1$  and  $R_2$ ) and the heteronuclear NOE with magnetic field strength. Initially, the user may not have an accurate estimate of certain relaxation parameters,  $\tau_e$ ,  $S^2$ ,  $\tau_c$ , which are discussed in detail below. In these cases, the user may

input minimum and maximum values, which are plotted as a series of graded values. A set of detailed tables of parameter values for various peptide/protein systems are available in the published manuscript (3).

### 3.1.1.2 The Overall Rotational Correlation Time, $\tau_c$

The overall rotational correlation time represents the time required for the molecule to tumble through one radian in an arbitrary direction (81). It depends on the size and shape of the molecule and the solvent viscosity. Some studies are consistent with the general rule of 0.5 ns overall correlation time per 1 kDa of molecular weight in a system (89, 90, 91). In contrast, there are also several cases where this estimate is not accurate (92, 93, 94).  $\tau_c$  is normally observed to decrease with increasing temperature. Recombinant human ubiquitin  $\tau_c$  varied as follows: 8.84 ns (5 °C), 6.36 ns (15 °C), 4.71 ns (25 °C), 3.58 ns (35 °C), 2.77 ns (45 °C), and 2.17 ns (55 °C) (95). A progressive drop in  $\tau_c$  with increasing temperature was also observed for a calmodulin-peptide complex: 11.81 ns (22 °C), 8.26 ns (35 °C), 6.33 ns (47 °C), 5.00 ns (60 °C), and 4.10 ns (73 °C) (96).

In the case of proteins solvated by micelles or bicelles in solution, the “aggregate weight” versus molecular weight must be taken into account, making estimation of  $\tau_c$  more difficult. For example, Yan *et al.* (94) proposed that a concentrated protein solution can increase the aggregation number of micellar systems, as discussed previously (97). Furthermore, 100 mmol/L NaCl can increase the SDS aggregation number from approximately 60 to 91 (98). Elevated aggregation numbers in these situations may account for increases in observed correlation times.

One simple theoretical approach for the estimation of  $\tau_c$  is the Stokes-Einstein-Debye relation, which depends on isotropic motion for approximately spherical globular proteins (80). Of course, many proteins are not spherical and, even in the absence of micelles or bicelles, there is still the possibility that the protein itself aggregates in solution. More sophisticated approaches to estimation of  $\tau_c$  include a relationship

between rotational correlation time and solvent accessible surface area (99), hydrodynamic calculations (100), and NMR relaxation interference experiments (101). Independent experimental measurement of  $\tau_c$  may also be performed by time-resolved fluorescence spectroscopy or light scattering (80). However, most conveniently, NMR relaxation experiments are an excellent method for estimating  $\tau_c$  if the protein size is less than 30 kDa (101). The normal procedure is to use a ratio of transverse ( $R_2 = \frac{1}{T_2}$ ) and longitudinal ( $R_1 = \frac{1}{T_1}$ ) relaxation rates (91, 102, 103, 104). Large deviations between theoretical predictions for  $\tau_c$  and the experimentally determined value may suggest oligomerization.

### 3.1.1.3 The Generalized Order Parameter, $S^2$

The generalized order parameter ( $S^2$ ), introduced in the development of the model-free formalism by Lipari and Szabo (6, 7), is an intuitive way to characterize the amplitude of internal picosecond-nanosecond timescale motions of bond vectors studied by NMR relaxation experiments.  $S^2$  values are always between 0 (fully isotropic) and 1 (a completely restricted bond vector). In the case of the extended Lipari-Szabo formalism (105),  $S^2 = S_f^2 S_s^2$ , where  $S_f^2$  and  $S_s^2$  are order parameters representing fast (ps timescale) and slow (ns timescale) internal motions, respectively. In both formalisms, the degree of motional restriction does not specify a particular model of motion, hence the “model-free” terminology.

Intuitively,  $S^2$  is expected to decrease at higher temperature due to increased thermal motion, although only a 0.03 average decrease in  $S^2$  was observed between 12 and 37 °C for Ribonuclease H (106).  $^{15}\text{N}$ - $^1\text{H}$  NMR relaxation experiments on recombinant human ubiquitin at temperatures ranging from 5 to 55 °C also demonstrated a small drop in the order parameter as temperature was increased, with average  $\frac{dS^2}{dT} = -2.3 \pm 0.95 \times 10^{-3} \text{ K}^{-1}$  (95). Similar studies on a calmodulin-peptide complex over the range of 22-73 °C also demonstrated a small  $\frac{dS^2}{dT}$  of  $-1.5 \times 10^{-3} \text{ K}^{-1}$ , on average, with  $S^2$  actually increasing slightly from the minimum value at 60 °C to

the value obtained at 73 °C (96). The loop residues not involved with  $\text{Ca}^{2+}$  binding had order parameters lower by almost 0.10, and had the greatest temperature dependence; notably residues 115 and 116 had  $\frac{dS^2}{dT} \sim -5 \times 10^{-3} \text{ K}^{-1}$ . Furthermore,  $^{15}\text{N}$ -H order parameters showed significantly larger changes with temperature for the unfolded states of staphylococcal nuclease (SNase) and the N-terminal SH3 domain of drk (drkN SH3) versus their respective folded states (107). For SNase, between 15 and 32 °C,  $\Delta S_{avg}^2 = 0.045 \pm 0.031$ , but  $\Delta S_{avg}^2 = 0.143 \pm 0.032$  for the partially unfolded SNase mutant  $\Delta 131\Delta$ . The authors also observed this striking contrast between the folded drkN SH3 ( $\Delta S_{avg}^2 = 0.004 \pm 0.036$ ) and the protein denatured in 2 mol/L GuHCl ( $\Delta S_{avg}^2 = 0.087 \pm 0.025$ ) for a temperature change between 14 and 30 °C. These changes were related to the different contributions of motion to the overall heat capacity of folded and unfolded proteins. Clearly, the folding state of a protein is relevant to the temperature sensitivity of  $^{15}\text{N}$ -H order parameters.

An extensive database analysis of the relationship between generalized order parameter values and protein secondary structure has been conducted (108). Of the 1855 order parameters surveyed from 20 proteins,  $S_{avg}^2 = 0.839 \pm 0.106$ . The authors find that the backbone mobility of an amino acid is strongly correlated with its side chain size, and to a lesser extent, the size of the neighbouring residue side chains. The smallest residue, glycine, is also the most mobile ( $S_{avg}^2 = 0.81 \pm 0.14$ ), whereas tryptophan, which has the largest volume, is the most restricted ( $S_{avg}^2 = 0.87 \pm 0.07$ ). It is commonly reported that  $S^2$  is larger in canonical secondary structure elements versus disordered loops (109), but (108) report only a slightly more mobile average for residues in loops ( $S_{avg}^2 = 0.81 \pm 0.11$ ) versus helices ( $S_{avg}^2 = 0.88 \pm 0.07$ ) or  $\beta$ -structures ( $S_{avg}^2 = 0.85 \pm 0.07$ ). Terminal residues were mobile ( $S_{avg}^2 = 0.61 \pm 0.24$ ), as might be expected for these often less-structured regions.

It must be noted that the 20 proteins in the database used in (108) are all stable and folded. Intrinsically disordered proteins (IDPs) are increasingly recognized as



biologically important effectors with a high binding plasticity (110, 111), which may exhibit a very different set of dynamic behaviours. Human securin, a regulator of cell division, is a 202 residue protein with 24 prolines, and is considered an IDP (112). NMR relaxation analysis is consistent with transient structuring in certain segments of securin. There is a central plateau for relaxation rates in the primary sequence and higher rates at the termini, although no model-free parameters were provided. Transient helical structuring was also observed in the IDP Sml1 based on chemical shifts, and NMR relaxation results ( $R_1$ ,  $R_2$ , and steady-state NOE) consistent with restricted motion for residue segments 4-20 and 60-86 (113). In another study, the Sml1 backbone structure was incredibly flexible with all  $S^2 < 0.6$ , except for the transiently helical regions (114). A remarkably similar degree of flexibility was observed for the backbone of the N-terminal half of hepatitis C virus core protein (C82), also an IDP, with  $S_{avg}^2 = 0.59 \pm 0.04$  (115). Similarly, the natively unfolded propeptide subtilisin (PPS) was flexible with  $S_{avg}^2 = 0.57 \pm 0.06$  (116). To fit the PPS  $^{15}\text{N}$  relaxation data, the authors modified the traditional model-free approach by fitting each residue with a distribution of rotational correlation times to reflect the ensemble of states for the unfolded protein.  $S^2$  and  $\tau_e$ , the next parameter in our survey, were both separately fit for each individual residue. Clearly, different analytical strategies are being explored for the dynamics of natively unfolded proteins, and  $S^2$  values are frequently lower on average than those considered canonical for structural proteins.

It has recently been suggested that the order parameters determined from model-free analysis are merely starting values (117). A thorough assessment of N-H bond vector reorientational motion involves a variety of order parameter simulation methods that provide separate pieces of dynamic information: 1) the isotropic reorientational eigenmode dynamics method (iRED) for assessing separability of internal and overall motion as assumed by the Lipari-Szabo approach; 2) a first-order expansion in local variances and covariances accounts for contributions from local dihedral angle

fluctuations to the order parameter; 3) the three-dimensional Gaussian axial fluctuation (GAF) method describes anisotropic peptide plane motion (118); and 4) the local contact model provides direct estimation of order parameter from 3D structure. Surprisingly, the local contact model produces an excellent agreement between observed  $S^2$  values and values predicted directly from X-ray crystallography or NMR structures of lysozyme, ubiquitin, interleukin-4, calmodulin, and an HIV-1 protease-ligand complex, suggesting a strong link between structure and dynamic information (119).

#### 3.1.1.4 The Effective Internal Correlation Time, $\tau_e$

The effective internal correlation time ( $\tau_e$ ) measures the time scale for internal motions of bond vectors sweeping through an amplitude quantified by the order parameter.  $\tau_e$  is notoriously difficult to interpret quantitatively because it is a complex combination of geometric factors (6, 7, 120). Low  $\tau_e$  precision from model-free analysis is especially pronounced for restricted residues ( $S^2 \geq 0.8$ ) (121), as I have recently observed for a  $^{15}\text{N}$  NMR relaxation analysis of the seventh TM segment of the  $\text{Na}^+/\text{H}^+$  exchanger isoform 1 (2). Palmer (68) specifically indicates that  $\tau_e$  is imprecisely determined and usually not analyzed in detail. In the case of the extended Lipari-Szabo formalism, the parameter is expanded into  $\tau_f$  and  $\tau_s$ , which are internal correlation times for bond vector motions on picosecond and nanosecond timescales, respectively (105). The authors report ranges of 200-300 ps for  $\tau_f$  and 1-3 ns for  $\tau_s$  for relaxation studies on staphylococcal nuclease and interleukin-1 $\beta$ . The extended method is normally used when timescales for internal motions differ by at least one order of magnitude. Typically,  $\tau_e$  is on the picosecond timescale and overall tumbling occurs on the nanosecond timescale.

## 3.2 NMR Spin Relaxation Studies Of NHE1 TM VII

The above description of NMR spin relaxation, based on my published manuscript (3), provides the necessary foundation to understand the model-free and reduced spectral density mapping analyses of NHE1 TM VII in DPC micelles. The TM VII work is based on another published manuscript (2) and it is detailed below.

### 3.2.1 NHE1 TM VII Background

I have used  $^{15}\text{N}$  NMR relaxation methods to study the dynamics of a  $^{15}\text{N}$ -labelled peptide of TM VII of NHE1 and I correlate these studies with structural information. Previous studies on functional aspects of the full-length NHE1 protein were performed using alanine scanning and insertion mutagenesis at the TM VII segment (residues 251-273) (22). Ala is the fourth most common amino acid in protein TM segments and is the fifth most effective helix-inducer in a hydrophobic environment (122, 123). Ala substitutions at 13 of 22 TM VII residues resulted in severely reduced activity in the full-length NHE1 protein (22). Beyond perturbing intramolecular interactions or removing key chemical moieties required for ion transport, if flexibility at TM VII is important for NHE1 function (*i.e.*, ion transport) Ala substitutions may interfere by promoting the structural rigidity of an  $\alpha$ -helix in addition to replacing important charged or steric residues. The potential importance of flexibility in TM VII is spurred by the ensemble of NMR structures for the peptide in DPC micelles, which show an  $\alpha$ -helix interrupted at G261-S263 (22). E262 is critical to activity in the full-length NHE1 protein and, more specifically, an acidic residue at this position is hypothesized to be important for cation coordination (124). Since E262D retains much of the NHE1 activity (124), it is possible that the reduced helix forming propensity of acidic residues in comparison to Ala in addition to charge retention is important for conserving both a disruption in helicity and ion coordination (22). The

G261A and E262A mutants of NHE1 were 50 % and 48 % less active than the wild-type protein, respectively, even after correction for reduced expression and targeting (22). The G261-S263 region in the TM VII peptide in DPC micelles was found in two predominant conformations, one in which G261-S263 is fairly extended and the N- and C-terminal helical regions are distal to each other and the other in which the helical regions are in close proximity and G261-S263 is allowing the formation of a tight kink. Observation of a single set of NMR chemical shifts in this region implies relatively rapid interconversion between conformations, and hence a rather dynamic structure for the TM segment despite its reconstitution in DPC micelles. Given the reduction in NHE1 activity following Ala substitution mutagenesis at G261 and E262 that were observed previously (22), a new set of mutations were also explored by our collaborators (lab of Dr. Fliegel, University of Alberta) with the goal of testing the effect of further restriction of motion in the TM VII segment. For this purpose, mutation to Ile was chosen since Ile is the most common amino acid found in protein TM segments and, of the 20 common amino acids, has the highest propensity for  $\alpha$ -helix formation in a hydrophobic environment (122, 123). Herein, my NMR spin relaxation studies are presented in detail, while the details of collaborative mutagenesis studies are presented in the published manuscript (2).

## **3.2.2 Materials And Methods**

### **3.2.2.1 Materials**

Deuterium oxide (99.9% D), deuterium oxide (99.9% D) with 1% sodium 2,2-dimethyl-2-silapentane-5-sulphonate, and DPC-d<sub>38</sub> (99.1% D) were purchased from CDN Isotopes (Pointe-Claire, QC, Canada). A 535-PP NMR tube (Wilmad Glass Co., Buena, NJ, USA) was used for all NMR relaxation experiments.

### **3.2.2.2 Peptide Synthesis And Purification**

The TM VII peptide (HINELLHILVFGESLLNDAVTVVLYKK; free N-terminus,

amide-capped C-terminus; the bold red-coloured residues having backbone  $^{15}\text{N}$  labels) was synthesized using solid-phase Boc chemistry (125) and purified as previously described (22). Peptide identity was confirmed by matrix-assisted laser desorption ionization mass spectrometry and by amino acid analysis (Institute for Biomolecular Design, Edmonton, AB, Canada).

### 3.2.2.3 NMR Spectroscopy

The NMR sample was prepared by dissolving  $\sim 770 \mu\text{M}$  TM VII peptide in 90%  $\text{H}_2\text{O}$ , 10%  $\text{D}_2\text{O}$  solution containing  $\sim 75 \text{ mM}$  DPC- $\text{d}_{38}$ . Chemical shifts were referenced to 2,2-dimethyl-2-silapentane-5-sulfonic acid at 1.0 mM. Solution pH was adjusted to 4.8 (deuterium isotope effects not taken into account), and all experiments were carried out at 30 °C for consistency with structural studies (22). One-dimensional  $^1\text{H}$  observed,  $^{15}\text{N}$  NMR relaxation experiments were performed on 500, 600, and 800 MHz (800 equipped with cryogenic probe) Varian Inc. (Palo Alto, CA, USA) INOVA spectrometers, with parameters listed in Table 3.1 on page 55. The BioPack (Varian Inc.) gNhsqc pulse sequence (126) was used for measurement of  $^{15}\text{N}$  relaxation of labelled TM VII residues. The  $^{15}\text{N}$  relaxation rates were measured from 1D  $\{^1\text{H}-^{15}\text{N}\}$ -HSQC spectra. All spectra were processed and analyzed with VnmrJ 2.1B (Varian Inc.).

### 3.2.2.4 $^{15}\text{N}$ Relaxation Parameters

$^{15}\text{N}$  relaxation time constants ( $T_1$ ,  $T_2$ ) and their standard errors from the covariance matrix were calculated using a nonlinear least-squares fit to a two parameter monoexponential decay using xcrvfit version 4.0.12<sup>1</sup>. Errors for the first order rate constants ( $R_1$ ,  $R_2$ ) were propagated (127) from the time constant errors using:

$$\delta R_i = \frac{|R_i|}{|T_i|} \delta T_i \quad (3.1)$$

---

<sup>1</sup><http://www.bionmr.ualberta.ca/bds/software/xcrvfit/>

where  $R_i$  and  $T_i$  represent a pair of rate and time constants, respectively.

Steady-state  $\{^1\text{H}\}$ - $^{15}\text{N}$  NOE values were calculated using the software relax version 1.3.2 (78, 128) as the peak height ( $I$ ) ratios in proton saturated versus reference spectra:

$$\text{NOE} = \frac{I_{\text{sat}}}{I_{\text{ref}}} \quad (3.2)$$

Standard deviation ( $\sigma$ ) was propagated from the root-mean-square baseline noise as previously reported (87):

$$\sigma_{\text{NOE}} = \text{NOE} \sqrt{\left(\frac{\sigma_{I_{\text{sat}}}}{I_{\text{sat}}}\right)^2 + \left(\frac{\sigma_{I_{\text{ref}}}}{I_{\text{ref}}}\right)^2} \quad (3.3)$$

### 3.2.2.5 Model-Free Calculations

The calculated  $T_1$ ,  $T_2$ , and NOE values at three field strengths were used to determine the model-free parameters  $\tau_M$  (overall rotational correlation time),  $S^2$  (generalized order parameter), and  $\tau_e$  (effective internal correlation time) through spectral density function fitting using the following relaxation expressions (129, 8):

$$\frac{1}{T_1} = \left(\frac{d^2}{4}\right) [J(\omega_H - \omega_N) + 3J(\omega_N) + 6J(\omega_H + \omega_N)] + c^2 J(\omega_N) \quad (3.4)$$

$$\begin{aligned} \frac{1}{T_2} &= \left(\frac{d^2}{8}\right) [4J(0) + J(\omega_H - \omega_N) + 3J(\omega_N) + 6J(\omega_H) + 6J(\omega_H + \omega_N)] \\ &+ \left(\frac{c^2}{6}\right) [3J(\omega_N) + 4J(0)] \end{aligned} \quad (3.5)$$

$$\text{NOE} = 1 + \left(\frac{d^2}{4}\right) \left(\frac{\gamma_H}{\gamma_N}\right) [6J(\omega_H + \omega_N) - J(\omega_H - \omega_N)] T_1 \quad (3.6)$$

where  $d = \left[\frac{\mu_0 h \gamma_N \gamma_H}{(8\pi^2)} \left\langle \frac{1}{r_{\text{NH}}^3} \right\rangle\right]$ ,  $c = \left(\frac{\omega_N}{\sqrt{3}}\right)(\sigma_{\parallel} - \sigma_{\perp})$ ,  $\mu_0$  is the permeability of free space,  $\omega_N$  and  $\omega_H$  are the respective nuclear Larmor frequencies of  $^{15}\text{N}$  and  $^1\text{H}$ ,  $\gamma_N$  and  $\gamma_H$  are the respective gyromagnetic ratios of  $^{15}\text{N}$  and  $^1\text{H}$ ,  $h$  is Planck's constant,  $r_{\text{NH}}$  is the length of the amide bond, and  $\sigma_{\parallel}$  and  $\sigma_{\perp}$  are the parallel and perpendicular

components of the axially symmetric chemical shift tensor. A value of  $-160$  ppm was used for  $(\sigma_{\parallel} - \sigma_{\perp})$  (87, 130). The form of the spectral density function used in the Lipari-Szabo formalism is given by (6, 7):

$$J(\omega) = \frac{2}{5} \left\{ \frac{S^2 \tau_M}{[1 + (\omega^2 \tau_M^2)]} + \frac{(1 + S^2) \tau}{[1 + (\omega \tau)^2]} \right\} \quad (3.7)$$

where  $\frac{1}{\tau} = \frac{1}{\tau_M} + \frac{1}{\tau_e}$ .

Fitting of the model-free parameters  $\tau_M$ ,  $S^2$ , and  $\tau_e$  to the relaxation data was performed using a suite of Mathematica (Wolfram Research Inc., Champaign, IL, USA) notebooks previously described (131) but modified by Spyropoulos to include data collected at multiple field strengths in a single calculation. Briefly, five forms of the spectral density function are considered to account for mixtures of motion on various time scales (131, 132). An optimization procedure is performed to fit the experimental input ( $T_1$ ,  $T_2$ , steady-state NOE) to each of these five mathematical models, labelled 1-5. The appropriate model for each of the residues is selected using the statistical approach of Akaike's information criteria (AIC) (133), and 100 Monte Carlo simulations were performed to estimate parameter errors (121).

### 3.2.2.6 Reduced Spectral Density Mapping

Using the reduced spectral density mapping approach (8, 9), measurement at three field strengths for the steady-state NOE, spin-lattice ( $T_1$ ) and spin-spin ( $T_2$ ) relaxation times of  $^{15}\text{N}$  allows for sampling of seven spectral density values describing the motion of the system (88). The spectral density values, under the high frequency approximation that  $J(0.921\omega_H)$  and  $J(0.955\omega_H)$  are both equivalent to  $J(0.870\omega_H)$

(8), are obtained from the following set of equations:

$$\text{NOE} = 1 + \left(\frac{d^2}{4}\right) \left(\frac{\gamma_H}{\gamma_N}\right) [5J(0.870\omega_H)]T_1 \quad (3.8)$$

$$R_1 = \left(\frac{d^2}{4}\right) [3J(\omega_N) + 7J(0.870\omega_H)] + c^2 J(\omega_N) \quad (3.9)$$

$$R_2 = \left(\frac{d^2}{8}\right) [4J(0) + 3J(\omega_N) + 13J(0.870\omega_H)] \\ + \left(\frac{c^2}{6}\right) [3J(\omega_N) + 4J(0)] \quad (3.10)$$

Equations 3.8 to 3.10 allow solving for the three unknown spectral density functions  $J(0.870\omega_H)$ ,  $J(\omega_N)$ , and  $J(0)$ . The field-dependent  $J(0.870\omega_H)$  and  $J(\omega_N)$  alongside the field-independent  $J(0)$  give seven spectral density values for a given bond vector at three field strengths. Uncertainties in the values of spectral density functions were calculated by propagating the uncertainties of the independent variables using a sum of squares equation:

$$\delta q = \sqrt{\left(\frac{\delta q}{\delta x} \delta x\right)^2 + \dots + \left(\frac{\delta q}{\delta z} \delta z\right)^2} \quad (3.11)$$

where  $x \dots z$  represent any number of independent variables and  $\delta$  values are parameter uncertainties (127).

### 3.2.2.7 Theoretical Calculations

The software Maple 11.02 (Waterloo Maple, Inc., Waterloo, ON, Canada) was used to predict the trend for  $T_1$ ,  $T_2$  or NOE as a function of magnetic field strength according to equations 3.4-3.7 (prior to the development of the web-accessible program described in section 3.1.1.1 on page 33). Specifically, each of the three model-free parameters used to describe the motion of  $^{15}\text{N}$ -H bond vector ( $S^2, \tau_e, \tau_M$ ) was independently varied to study the robustness of the trend of  $T_1$ ,  $T_2$  or NOE with increasing magnetic field strength.  $\tau_M$  of the NHE1 TM VII peptide in DPC micelles was esti-



mated using a  $T_1/T_2$  fit strategy in the Mathematica notebooks introduced above, as previously described (87, 131).

### 3.2.3 Results

#### 3.2.3.1 Relaxation Parameters: $T_1$ , $T_2$ , and NOE

A set of  $^{15}\text{N}$  NMR relaxation data ( $T_1 = \frac{1}{R_1}$ ,  $T_2 = \frac{1}{R_2}$ , and NOE) was acquired at 500, 600, and 800 MHz for a specifically  $^{15}\text{N}$  labelled TM VII peptide in DPC micelles at 30 °C (*i.e.*, Figure 3.1 on page 52). The  $T_1$  values are similar for all six  $^{15}\text{N}$ -labelled TM VII residues, with slightly lower values for L254 at 500 MHz and 600 MHz but not at 800 MHz (Figure 3.2 on page 53). There is a trend toward increasing  $T_1$  at higher field strength, although the increase is within the bounds of experimental error. In contrast,  $T_2$  values clearly decrease at higher magnetic field strength. For each magnetic field strength, the  $T_2$  values are approximately constant over the six residues. The steady-state NOE values are also approximately constant over the six  $^{15}\text{N}$ -labelled residues at a given field strength. Although NOE values compared between different field strengths overlap within the bounds of experimental error, there is a trend toward increasing NOE at higher field. The complete set of relaxation parameter values and errors are presented in Table 3.2 on page 56.

#### 3.2.3.2 Reduced Spectral Density Mapping

$J(0.870\omega_H)$ , calculated directly from equation 3.8, is plotted for the six  $^{15}\text{N}$ -labelled TM VII residues and values are compared between field strengths in Figure 3.3 on page 54. The general trend is a decrease in  $J(0.870\omega_H)$  with increasing field strength, although there is overlap between values within the bounds of experimental error, and in the case of G261  $J(0.870\omega_H)$  is slightly greater at 800 MHz versus 600 MHz. From a sequential standpoint, all six  $^{15}\text{N}$ -labelled residues have very similar  $J(0.870\omega_H)$ . Using the calculated  $J(0.870\omega_H)$  and experimental  $T_1$  values, equation 3.9 can be solved for  $J(\omega_N)$ .  $J(\omega_N)$  values calculated from 500 MHz and

600 MHz experiments overlap within the bounds of experimental error, but values calculated from 800 MHz data are clearly smaller. In terms of primary sequence, none of the six  $^{15}\text{N}$ -labelled residues differs within the bounds of experimental error. Finally, experimental  $T_2$  values and calculated  $J(\omega_N)$  and  $J(0.870\omega_H)$  values can be used to solve equation 3.10 for  $J(0)$ . There is considerable overlap in  $J(0)$  across field strengths within the bounds of experimental error, although there is a trend toward higher values at 800 MHz. However, from a sequential standpoint, none of the six residues has a significantly different  $J(0)$  at a given field strength.

### 3.2.3.3 Model-Free Analysis

The most common analysis of  $^{15}\text{N}$  amide relaxation data for proteins and peptides is the Lipari-Szabo model-free approach where three parameters ( $S^2$ ,  $\tau_e$ , and  $\tau_M$ ) describe the motion of a  $^{15}\text{N}$ -H bond vector based on spectral density functions (*i.e.*, equation 3.7) (6, 7). The complete model-free results are summarized in Table 3.3 on page 57. For L254, L258, and L273, AIC selection favored model five, but each fit had a highly skewed  $\chi^2$  distribution so the next most likely model was chosen. This is also preferable because model five has the most fitting parameters (four), which can increase the strength of fit independent of its true reflection of the empirical relaxation data. The AIC approach selected model four for G261, L264, and A268, which includes a chemical exchange term,  $R_{\text{ex}}$ —defined as a relaxation contribution from  $\mu\text{s}$ -ms time scale motions (8). This is consistent with a strand of the peptide spanning from G261 to A268 that is subject to chemical exchange. G261 has a slightly lower order parameter ( $S^2 = 0.65 \pm 0.02$ ) than the other five  $^{15}\text{N}$ -labelled residues ( $S^2_{\text{average}} = 0.80 \pm 0.02$ ). For G261, the slightly reduced order parameter is not consistent with an elevated  $J(0)$  (Figure 3.3). Although neither  $J(0)$  or  $S^2$  is remarkably different from those of the other five residues, it is noteworthy that chemical exchange

( $R_{\text{ex}}$ ) motions in the  $\mu\text{s}$ -ms range can inflate  $J(0)$  for affected residues (88, 134):

$$J(0)_{\text{obs}} = J(0)_{\text{corr}} + \lambda R_{\text{ex}} \quad (3.12)$$

where  $\lambda$  is a positive scaling factor,  $\lambda = (\frac{3}{2})[\frac{1}{3d^2+c^2}]$ ;  $c$  and  $d$  are the chemical shift and dipolar constants defined in section 3.2.2; and  $J(0)_{\text{obs}}$  and  $J(0)_{\text{corr}}$  are the values before and after correction for the contribution from chemical exchange, respectively. Since G261 has the largest  $R_{\text{ex}}$  value estimated from model-free analysis ( $1.9 \pm 0.2$ )  $\text{s}^{-1}$  (Table 3.3), it is certainly possible that chemical exchange can account for the discrepancy of  $J(0)_{\text{obs}}$  and order parameter for G261. I avoid a more detailed consideration of chemical exchange since accurate quantification requires relaxation dispersion experiments (88, 135).

## 3.2.4 Discussion

### 3.2.4.1 Relaxation Parameters: Comparing Theory And Experiment

The six  $^{15}\text{N}$ -labelled residues of the TM VII peptide have similar values for all three relaxation parameters at each field strength (Figure 3.2 on page 53) suggesting similar flexibility on the ps-ns time scale along the length of the peptide. Both  $T_1$  and  $T_2$  follow the expected theoretical trends within experimental error for relaxation time versus field strength at the experimentally determined rotational correlation time ( $\tau_M$ ) of  $\sim 10$  ns. Analysis of the steady-state NOE is a bit more convoluted. The predicted trend is a decrease in the NOE with increasing field strength for a  $\tau_e$  in the tens of ps range as estimated from model-free analysis (Table 3.3 on page 57) coupled with the experimentally estimated  $\tau_M$ . However, we generally observe that NOE increases with field strength (Figure 3.2). It is possible that the effective internal correlation times are underestimated, in which case a  $\tau_e$  of 350 ps would be sufficient to account for the observed trend. This is not surprising given the low precision in the  $\tau_e$  estimations from model-free analysis (Table 3.3). Low  $\tau_e$  precision from model-free

analysis is especially pronounced for restricted residues ( $S^2 \geq 0.8$ ) (121).

#### 3.2.4.2 Reduced Spectral Density Mapping

No significant differences were observed for  $J(0.870\omega_H)$ ,  $J(\omega_N)$  or  $J(0)$  at a given field strength for the six  $^{15}\text{N}$ -labelled residues (Figure 3.3 on page 54), again suggesting that motions on the ps-ns time scale are similar for the tested residues. The propagated errors at the three spectral density frequencies vary considerably, and also depend on the field strength of measurement. The former result is consistent with previous NMR dynamics analyses on a series of peptides where errors varied between negligible and large on a per-residue basis at a given spectral density frequency (83). The most commonly used measure of structural flexibility in spectral density mapping is  $J(0)$ , the value of the function at zero frequency. It is normally interpreted as a measure of restricted motion, with large values suggesting increased local structure, and small values consistent with flexibility (88, 83). Chemical exchange ( $R_{\text{ex}}$ ) and  $^{15}\text{N}$  chemical shift anisotropy (CSA) variations along the primary sequence of the peptide may contribute to errors in  $J(0)$  and may explain the deviations between field strengths we observed (Figure 3.3) for this theoretically field-independent parameter (88). Specifically,  $J(0)$  values calculated from higher field strength data are more susceptible to inflation by chemical exchange contributions to  $R_2$  (88), and our  $J(0)$  values are generally greater using 800 MHz data (Figure 3.3).  $J(0)$  values are similar for all six  $^{15}\text{N}$ -labelled residues, suggesting an equal degree of motional restriction on the ps-ns time scale. The slightly reduced order parameter for G261 does not contradict its  $J(0)$  because chemical exchange can inflate the observed  $J(0)$  (88, 134). If G261 is a pivot point allowing motion of the portions of TM VII C- and N-terminal to it relative to each other, as we have previously suggested (22), the motion at this residue is most likely to be on the  $\mu\text{s}$ -ms time scale. This estimate of the motional time scale is based both on the observation of a single set of exchange-averaged chemical shifts despite extensive conformational sampling in the peptide

apparent from nuclear Overhauser effect contacts (22) and on the mathematical fit of a chemical exchange term ( $R_{\text{ex}}$ ) from our model-free analysis (Table 3.3).

### 3.2.4.3 Correlating Structure, Dynamics, And Function

The structure of TM VII in DPC micelles is an interrupted  $\alpha$ -helix (22). For a converged structural ensemble without discarding a significant portion ( $> 34\%$ ) of the NOE contacts assigned, it was necessary to employ a new dual-conformer calculation protocol (22, 29). Through parallel calculation of pairs of conformers, all NOEs were satisfied, implying extensive conformational sampling about the G261-S63 region of the TM segment. The dual-conformer protocol is equally accommodating of pairs of non-interacting conformers existing simultaneously in the ensemble vs. oligomerization. Because NOEs were almost entirely satisfied through isolated dual conformers, rather than dimer formation, TM VII was attributed to be a monomer undergoing conformational exchange (22). The TM VII peptide gave a single set of averaged chemical shifts, rather than multiple sets of independently sequentially assignable shifts implying distinct and long-lived conformations (*i.e.*, our recent structure of apelin-17 (136)). Chemical shift averaging demonstrates that interconversion between the two major conformers assumed by TM VII in micelles is rapid on the NMR time scale ( $\sim$ ms or faster).

Placing this in context of  $^{15}\text{N}$ -backbone relaxation, the highly similar  $T_1$ ,  $T_2$ , and steady-state NOE values imply highly similar dynamics at the ps-ns time scale along the length of the TM VII peptide. In light of both the chemical shift averaging and the lack of distinctive variations in ps-ns dynamics, conformational interconversion is therefore most likely to be in the  $\mu\text{s}$ -ms regime. To produce the observed dual-conformer set of TM VII structures, this exchange must be occurring in the G261-S263 region and is consistent with a chemical exchange model being selected from model-free analysis for residues G261, L264, and A268 (Table 3.3 on page 57), with the largest  $R_{\text{ex}}$  parameter for G261. From these model-free results, a hypothetical

mechanism is that the N-terminal region of TM VII is undergoing relatively little  $\mu$ s-ms level motion while the G261-S63 pivot point is allowing the C-terminal region containing L264 and A268 to exchange between two conformations relative to the N-terminus.

In order to consider the necessity of flexibility allowed by the break in helicity of the otherwise helical TM VII segment at residues G261-S263, I compare two sets of mutagenesis studies performed by our collaborators (for the details of the functional studies see (2, 22)). E262 is not considered in this structural-dynamic correlation analysis, given the likelihood of involvement in ion translocation (14). There is a perturbation to function with Ala mutation and almost complete loss of function with Ile mutation for the full-length NHE1 protein with mutation at F260, G261 and S263. Liu and Deber have tested the propensity for all 20 amino acids to form an  $\alpha$ -helix in a hydrophobic environment (122, 123). Using these results, the effect of the various Ala and Ile mutations presented both previously (22) and herein can be examined in the context of a perturbation to the formation of a non-helical region. Mutation of F260, which is in the N-terminal  $\alpha$ -helical region of TM VII, to Ala or Ile would be fairly non-perturbing in terms of secondary structure propensity and F260I reasonably conservative in terms of side-chain size; therefore, the loss of the F260 side-chain itself appears to be the critical feature of this mutation. Both G261A and G261I are relatively non-perturbing in terms of  $\alpha$ -helical propensity in a membrane. For this position, therefore, the lack of steric constraint of a Gly residue seems likely to be the most important factor, correlating well to the break in helical structure and to the evidence for  $\mu$ s-ms scale dynamics at G261. S263 is sensitive to mutation to either Ala or Ile, with mutation to either residue significantly increasing helical propensity. A good possibility is that the Ala and Ile mutants are sufficiently perturbing to local structure to extend the helical segment beginning at L264 to include S263A or S263I. This would significantly perturb both structure and dynamics in the G261-S263

region.

Interruptions in regular secondary structure have now been documented for NHE1 TM segments IV, VII, IX, and XI (22, 137, 1, 27). It will be important to assess the theme of intermediate time scale motion about a pivot point such as that predicted herein with more detailed chemical exchange information by performing relaxation dispersion experiments (88, 135). Structural and dynamics characterization of mutant TM domains which are known to significantly perturb function are also a potentially valuable tool in terms of understanding these processes. Chemical exchange is entirely consistent with the alternating-access mechanism proposed for exposure of E262 to cytosolic protons during the ion translocation cycle of NHE1, where the residue is bent away from the cytosol in certain conformations (14). An alternating-access mechanism was also proposed for the homologous NhaA protein on the basis of its crystal structure (42).

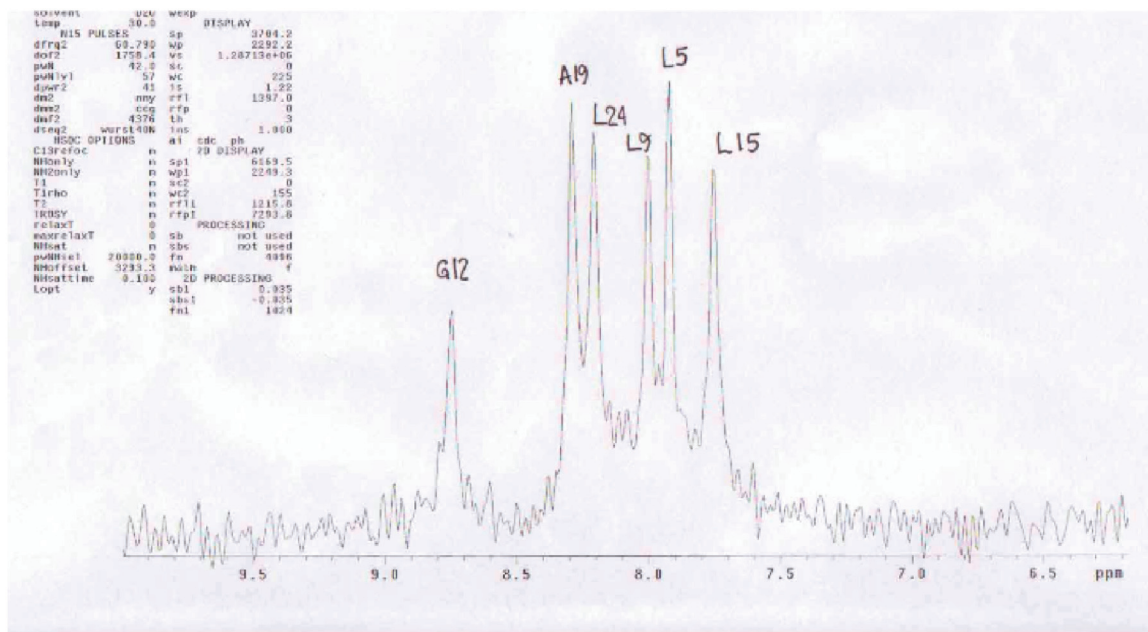


Figure 3.1: The integrity of the NHE1 TM VII sample in DPC micelles was verified by collection of a one-dimensional  $^1\text{H}$ - $^{15}\text{N}$  NMR spectrum on a 600 MHz spectrometer (128 scans). The spectrum is consistent with results from a previous pure sample of NHE1 TM VII (not shown).



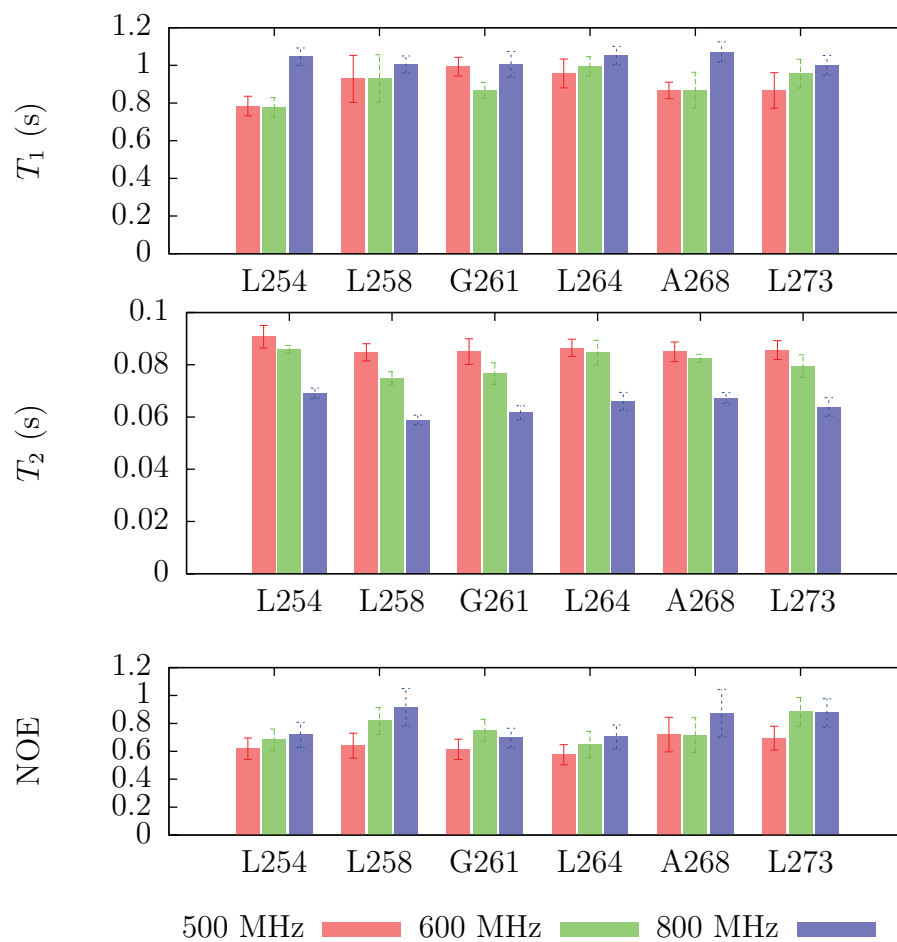


Figure 3.2: The NMR spin-relaxation parameters spin-lattice relaxation time ( $T_1$ ), spin-spin relaxation time ( $T_2$ ), and steady-state NOE are plotted for measurements at three magnetic field strengths for each of the  $^{15}\text{N}$  backbone-labelled positions in the NHE1 TM VII peptide construct. Each residue is numbered according to its position in full-length NHE1.

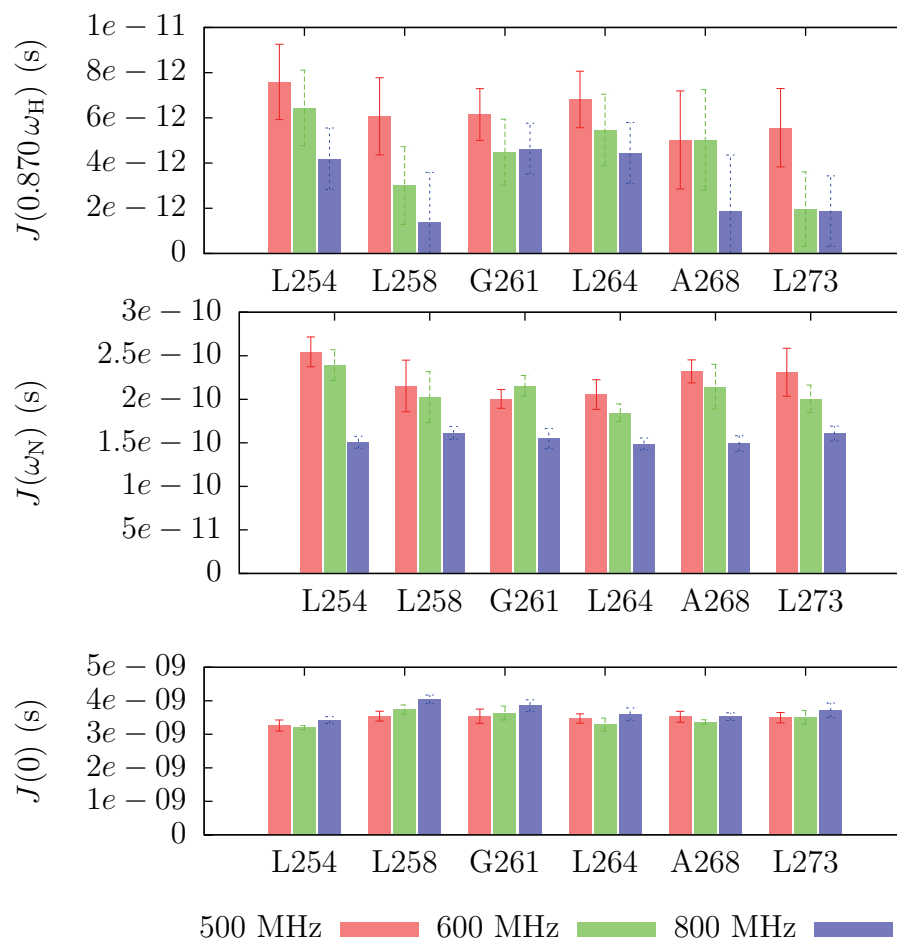


Figure 3.3: Spectral density values at high ( $J(0.870\omega_H)$ ), intermediate ( $J(\omega_N)$ ) and zero ( $J(0)$ ) frequencies are shown for each of the six  $^{15}\text{N}$ -labelled NHE1 TM VII residues for measurements at three magnetic field strengths. The spectral densities represent the population of  $^{15}\text{N}$ -H bond vectors rotating at a given frequency, and the results are similar for each residue at a given field strength within the bounds of uncertainty.

Table 3.1: NMR parameters for  $^{15}\text{N}$  relaxation experiments at 500, 600, and 800 MHz.

Experiment	Spectrometer (MHz)	nt	at (s)	d1 (s)	Relaxation delay [s]; interval (s)
$R_1$	500	1024	0.073	5.0	[0.01,0.05,0.1,0.2,0.3,0.5,0.8]
	600	256	0.061	5.0	[0.01,0.05,0.1,0.2,0.3,0.5,0.8]
	800	256	0.085	10.0	[0.01,0.05,0.1,0.2,0.3,0.5,0.8]
$R_2$	500	1024	0.073	5.0	[0.01-0.11];(0.02)
	600	256	0.061	5.0	[0.01-0.11];(0.02)
	800	256	0.085	10.0	[0.01-0.11];(0.02)
$\{^1\text{H}\}$ - $^{15}\text{N}$ NOE	500	16384	0.073	3.0	-
	600	8192	0.061	3.0	-
	800	8192	0.085	3.0	-

Table 3.2: The full set of longitudinal relaxation rate ( $R_1$ ), transverse relaxation rate ( $R_2$ ), and steady-state NOE  $^{15}\text{N}$  amide relaxation parameters for NHE1 TM VII in DPC micelles.

Residue	$R_1$ ( $\text{s}^{-1}$ )		$R_2$ ( $\text{s}^{-1}$ )		$R_2$ ( $\text{s}^{-1}$ )		NOE		NOE	
	500 MHz	600 MHz	800 MHz	500 MHz	600 MHz	800 MHz	500 MHz	600 MHz	500 MHz	800 MHz
L254	$1.28 \pm 0.08$	$1.29 \pm 0.09$	$0.96 \pm 0.04$	$11.02 \pm 0.52$	$11.65 \pm 0.20$	$14.47 \pm 0.41$	$0.62 \pm 0.08$	$0.68 \pm 0.08$	$0.62 \pm 0.08$	$0.72 \pm 0.09$
L258	$1.08 \pm 0.14$	$1.07 \pm 0.15$	$1.00 \pm 0.04$	$11.80 \pm 0.46$	$13.38 \pm 0.46$	$16.99 \pm 0.53$	$0.64 \pm 0.09$	$0.82 \pm 0.10$	$0.64 \pm 0.09$	$0.91 \pm 0.14$
G261	$1.01 \pm 0.05$	$1.15 \pm 0.06$	$0.99 \pm 0.07$	$11.76 \pm 0.68$	$13.06 \pm 0.71$	$16.23 \pm 0.72$	$0.61 \pm 0.07$	$0.75 \pm 0.08$	$0.61 \pm 0.07$	$0.70 \pm 0.07$
L264	$1.04 \pm 0.08$	$1.00 \pm 0.05$	$0.95 \pm 0.04$	$11.56 \pm 0.44$	$11.81 \pm 0.66$	$15.16 \pm 0.79$	$0.58 \pm 0.07$	$0.65 \pm 0.10$	$0.58 \pm 0.07$	$0.70 \pm 0.09$
A268	$1.15 \pm 0.06$	$1.15 \pm 0.13$	$0.93 \pm 0.05$	$11.77 \pm 0.52$	$12.13 \pm 0.24$	$14.86 \pm 0.47$	$0.72 \pm 0.12$	$0.72 \pm 0.12$	$0.72 \pm 0.12$	$0.87 \pm 0.17$
L273	$1.15 \pm 0.13$	$1.05 \pm 0.08$	$1.00 \pm 0.05$	$11.68 \pm 0.49$	$12.58 \pm 0.69$	$15.66 \pm 0.88$	$0.69 \pm 0.09$	$0.88 \pm 0.10$	$0.69 \pm 0.09$	$0.88 \pm 0.10$

Table 3.3: Spectral density model selections and model-free parameters estimated for each of the  $^{15}\text{N}$ -labelled NHE1 TM VII residues. Large uncertainties in the measurement of  $\tau_e$  are well-documented (69).

Residue	Model	$S^2$	$\tau_e$ (ps)	$R_{ex}$ ( $\text{s}^{-1}$ )
L254	2	$0.78 \pm 0.01$	$40 \pm 11$	
L258	2	$0.89 \pm 0.02$	$37 \pm 229$	
G261	4	$0.65 \pm 0.02$	$16 \pm 5$	$1.9 \pm 0.2$
L264	4	$0.75 \pm 0.02$	$36 \pm 8$	$0.6 \pm 0.3$
A268	4	$0.76 \pm 0.02$	$24 \pm 13$	$0.5 \pm 0.2$
L273	1	$0.85 \pm 0.02$		

# Chapter 4

## Technicalities Of Spitz Preparation

### 4.1 Introduction

Spitz is a TM protein substrate of rhomboid-1 (Figure 4.1), an intramembrane serine protease localized to the Golgi apparatus in the *Drosophila* secretory pathway. Rhomboid is named for the abnormal rhomboid-shaped head skeleton observed in a *Drosophila* mutant (138). It was later found that rhomboid-1 cleavage of spitz is important for the release of this *Drosophila* epidermal growth factor receptor (EGFR) ligand and that proteolytic activity is controlled by Golgi compartmentalization of enzyme and controlled trafficking of substrate by another protein, Star, from the endoplasmic reticulum (ER) to Golgi (139, 140). Indeed, spitz is the principal activating ligand for *Drosophila* EGFR and is similar to mammalian TGF $_{\alpha}$ . While Star is clearly required to chaperone spitz from the ER to the Golgi, it is not involved in the actual cleavage of spitz (139). The cleavage of spitz was clearly established to occur in the Golgi and the luminal fragment is then trafficked to the plasma membrane and released for EGFR signaling. *Drosophila* rhomboid-1 was the first example of an intramembrane serine protease, but a number of others have since been identified across all kingdoms of life (reviewed by (10)). The sequence diversity between rhomboids in different organisms is striking, and only loose sequence requirements have been established for substrate cleavage (141). There is, however, some agreement

that postulated helix-breaking residues are required in the substrate for cleavage to occur.

There are many biologically-relevant examples of rhomboid homologues. The bacterial rhomboid AarA from *Providencia stuartii* cleaves part of the twin-arginine translocase subunit, TatA, and this process is required for the release of a quorum sensing ligand (142). Although TatA oligomerizes to allow for folded bacterial proteins to exit the cell, it is not yet clear if the quorum sensing ligand is directly released by the oligomerization of TatA. Nonetheless, *P. stuartii* is an opportunistic human pathogen, and quorum sensing is involved in its antibiotic resistance (143). Furthermore, apicomplexan rhomboids can cleave cell-surface adhesins, and this appears to be important for invasion of host cells by *Plasmodium falciparum* (144), the parasite responsible for human malaria. Yeast mitochondrial rhomboids are known to regulate membrane remodelling (145). Similarly, the mitochondrial rhomboid PARL (presenilin-associated rhomboid-like) regulates remodelling of cristae and apoptosis in mice (146).

There is clearly a diverse set of rhomboid functions in different organisms, and there are likely many functions yet to be uncovered. Surprisingly, there are no rhomboid-specific inhibitors despite the fact that rhomboids are not phylogenetically related to their soluble serine protease counterparts (reviewed in (10)). Indeed, rhomboids use a Ser-His catalytic dyad unlike their soluble counterparts which require a catalytic triad involving aspartic acid (147). Furthermore, although it has been five years since the first reported rhomboid protease crystal structure, that of *E. coli* GlpG (4), it remains unclear how the substrate enters the active site of the protease—which is surrounded by the helices of the polytopic rhomboid. Some convincing mutagenesis work shows that GlpG TM5 is likely a substrate gate (148), consistent with the variations in TM5 position reported between several different rhomboid crystal structures (reviewed in (149)).

Most structural studies to date on the spitz-rhomboid or homologous systems have focused on the protease rather than the TM substrate. The recent crystallization of *E. coli* GlpG with an irreversible (covalent) mechanism-based isocoumarin inhibitor is a step in the right direction (150), but still does not fully address how a peptide-sized substrate can laterally enter the TM core of the enzyme. To my knowledge, there is no high-resolution structure available for a validated rhomboid substrate. With the biological incentives for understanding rhomboid proteases highlighted above, we endeavoured to produce the first high-resolution structure of a spitz TMD construct. A number of attempts at producing and purifying a variety of spitz and related TMD constructs are described in this chapter. Clearly, the peptides are not very tractable. Although it was possible to collect NMR spectra for one of the preparations (see section 4.4 on page 62), it has not been possible to solve the high-resolution NMR structure yet. Additional studies on the spitz-rhomboid system were performed using coarse-grained molecular dynamics simulations (see chapter 6 on page 191).

## 4.2 Solid-Phase Peptide Synthesis

My primary strategy for production of rhomboid protease substrate constructs was the use of Fmoc (9-fluorenylmethoxycarbonyl) solid-phase peptide synthesis (SPPS). Fmoc-protected amino acids, Fmoc-Lys(Boc)-Wang resin (0.57 mmol/g), Fmoc-Ile Wang resin (0.55 mmol/g), O-benzotriazole-N,N,N',N'-tetramethyl-uronium-hexafluorophosphate (HBTU), and 2-(1H-7-azabenzotriazol-1-yl)-1,1,3,3-tetramethyl uronium hexafluorophosphate methanaminium (HATU) were from aapptec (Louisville, KY). Fmoc-Lys(Boc)-Wang resin (0.19 mmol/g) was from Otwo Biotech (Guangdong, China). Side chain protecting groups on Fmoc amino acids were 2,2,4,6,7-pentamethyl-dihydrobenzofuran (Pbf) for Arg; triphenylmethane (Trt) for Gln and His; tert-butyl (tBu) for Ser; and tert-butoxycarbonyl (Boc) for Lys. N,N-dimethylformamide (DMF, sequencing grade) was obtained from Fisher Scientific (Ottawa, ON). N,N-



diisopropylethylamine (DIEA) was consistently used to facilitate coupling reactions while piperidine was used for Fmoc deprotection reactions. HBTU was used as the coupling reagent for most reactions, but HATU was used for coupling with more costly amino acids (especially those with isotopic labels), and for couplings that follow the incorporation of His or Cys. For problematic coupling reactions, after several rounds of attempted coupling, an acetylation reaction with acetic anhydride was normally performed to cap unreacted sites. The Kaiser (151) and isatin (152) colourimetric tests were used to monitor the completion of reactions.

### **4.3 Production And Purification Of A 34-Residue Spitz TMD Construct (tr-08-1)**

The Fmoc solid-phase peptide synthesis of a 34-residue construct (KRPRPM-LEKASIASGAMCALVFMLFVCLAFYLRK; designated tr-08-1) of the spitz TMD was performed using 0.6 g of Fmoc-Lys(Boc)-Wang resin (0.57 mmol/g loading). After cleavage from the synthesis resin, the crude compound was largely insoluble in a number of solvents: H<sub>2</sub>O, trifluoroacetic acid (TFA), isopropanol, acetonitrile (ACN), and dimethyl sulfoxide (DMSO). After a few days of lyophilization in a mixture of these solvents, ~50 mg of crude peptide was dissolved in 3 mL of 100% TFA. The TFA solution was diluted with 50% H<sub>2</sub>O/ACN, and a ~0.7 mg/mL crude solution was subjected to C3 reverse-phase high performance liquid chromatography (HPLC) (not shown). Mass spectrometry results were consistent with the presence of the target product, but the HPLC profile exhibited substantial impurities.

Many HPLC runs were performed and DTT treatment seemed to improve the traces, so disulfide bond formation may have contributed to the reduced solubility and tractability of the peptide. However, there were persistent inconsistencies in peak elution times in matching runs and we decided to produce the same peptide construct with Cys residues replaced by Ser in an attempt to improve handling (section 4.4 on

page 62). In retrospect, I think it would have been worth performing several rounds of crude lyophilization in the presence of excess H<sub>2</sub>O, as this approach often improved the solubility of subsequent synthetic spitz constructs.

#### **4.4 Production And Purification Of A 34-Residue C→S Spitz TMD Construct (tr-08-2)**

The substantial problems with peptide handling experienced working with the TR-08-1 construct (section 4.3 on page 61) may relate to the formation of disulfide bonds and oligomerization. I repeated the SPPS procedure using Ser in place of both Cys residues. The new construct, labelled TR-08-2, was cleaved from the resin to yield a yellow gel-like crude product that dissolved in 25% H<sub>2</sub>O, 25% ACN, 50% formic acid. Lyophilization over several days with excess water produced a white powder. Initial electrospray ionization mass spectrometry (ESI-MS) results were consistent with the presence of 3+, 4+, 5+, and 6+ charged species of the ~3828 g/mol peptide (not shown).

The purity of the crude sample was further assessed by C18 analytical HPLC (Figure 4.2 on page 74) and there were a number of species present. The MALDI-MS results (not shown) were consistent with the elution of product in fraction 7 (30-35 minutes). TR-08-2 samples were concentrated by optimizing the solvent—optimal reconstitution was achieved in 60% deionized water (DI-H<sub>2</sub>O), 40% ACN. After multiple attempts at HPLC gradient optimization for purification (not shown), one of the most promising fractions produced an encouraging MALDI-MS result (Figure 4.3 on page 75). We were reasonably satisfied with the purity demonstrated by this MALDI spectrum (although in retrospect I'd be more concerned about the low molecular weight species), and decided to purify the entire ~19 mL crude stock. At the time we were limited to a 250 μL sample loop, and I completed 76× 48-minute HPLC runs to purify the crude (see Figure 4.4 on page 76 for a representative HPLC trace).

The pooled TR-08-2 samples produced using the above workflow were lyophilized and further HPLC purification was performed (Figure 4.5 on page 77). Curiously, the MALDI-MS results for fraction 1 (Figure B.35 on page 251) and fraction 2 (Figure B.36 on page 252) similarly reflected a relatively pure sample with major species  $\sim 3830$  g/mol, and the results were also verified by ESI-MS (not shown). Thus, both fractions were collected via many additional rounds of HPLC purification, yielding 0.8 mg and 2.4 mg of fractions 1 and 2 respectively. An NMR sample was prepared with the following composition: 1 mM DSS, 95% H<sub>2</sub>O/5% D<sub>2</sub>O,  $\sim 75$  mM DPC-d<sub>38</sub>,  $\sim 260$   $\mu$ M peptide (TR-08-2 *fraction 1*), and pH 5. A matching sample (but slightly more dilute at  $\sim 100$   $\mu$ M fraction 1, 32 mM DPC) was used for circular dichroism spectropolarimetry (Figure 4.6 on page 78). The peptide appears to be  $\alpha$ -helical, and qualitatively similar results were obtained for a very dilute solution of *fraction 2*. However, TOCSY (total correlation spectroscopy) and HSQC (heteronuclear single quantum correlation) NMR experiments on a 500 MHz spectrometer had insufficient signal, and the described sample may have been too dilute (not shown).

A more concentrated NMR sample was made with TR-08-2 *fraction 2* with the following composition: 0.96 mM peptide,  $\sim 75$  mM DPC-d<sub>38</sub>, 95% H<sub>2</sub>O/5% D<sub>2</sub>O, 1 mM DSS, and pH 5. Although reasonable signal/noise could not be obtained on a 500 MHz spectrometer, high quality NMR spectra were obtained on a 700 MHz magnet (fitted with a cryogenically cooled probe). 2D NOESY (200 ms mixing time, Figure 4.7 on page 79), TOCSY (Figure 4.8 on page 79), and natural abundance <sup>13</sup>C-HSQC spectra were used for resonance assignment. However, as demonstrated in the TOCSY caption, there is substantial resonance overlap and after several weeks of attempted assignment it was deemed unlikely that the homonuclear NMR data would suffice for structure determination. The next step was to produce a construct with isotope labels to facilitate resonance assignment (section 4.5 on page 64).

## 4.5 Production And Purification Of A 21-Residue Spitz TMD Construct (tr-09-1)

A 21-residue spitz TMD peptide construct was designed as outlined in Figure 4.9 on page 80. The Fmoc solid-phase peptide synthesis of this construct (designated tr-09-1) was performed with 0.6 g of Fmoc-Lys(Boc)-Wang resin (0.57 mmol/g) and includes the incorporation of selective fractional  $^{15}\text{N}$  backbone labels. Predicting the isotopic mass of the major expected product requires a ‘tree-branch’ analysis approach (Figure 4.10 on page 81). The latter analysis suggests that major products will incorporate 4-5  $^{15}\text{N}$  backbone labels. Accounting for the natural abundance of  $^{13}\text{C}$  and the incorporation of an additional proton in matrix-assisted laser desorption/ionization (MALDI) mass spectrometry, the expected mass of the product is  $\sim 2249\text{--}2250$  g/mol. I wrote an internet-accessible python program (available at: <http://129.173.89.133/cgi-bin/isotope.cgi.py>) for performing these fractional isotope calculations and the underlying code is included in module A.1 on page 206.

As anticipated from work with previous spitz TMD constructs, there were substantial problems with the solubility of tr-09-1. None of the following solvents could dissolve the crude peptide after cleavage from the resin: ACN/ $\text{H}_2\text{O}$  mixtures, methanol, isopropanol, chloroform, hexane, or DMSO. One possibility is that disulfide cross-linking reduces the solubility of the crude peptide, but a 70%  $\text{H}_2\text{O}$ , 30% ACN, 200 mM dithiothreitol (DTT) mixture with trace TFA could not dissolve the product, even after treatment at 80 °C. While the crude peptide was also insoluble in 50%  $\text{H}_2\text{O}$ /50% TFA, it was finally possible to dissolve the crude peptide in a 100% TFA solution. However, the lyophilization product did not have an improved appearance or consistency—it was a thick, dark yellow gel combined with dark yellow plastic-like flakes. MALDI-MS of the crude sample was consistent with two products in the target mass range—one missing the N-terminal K residue, and the other an almost perfect match to the expected isotopic mass calculated above (Figure 4.11 on page 82).

Having confirmed the presence of the desired product in the crude mixture, we decided to lyophilize the gelatinous crude sample in a large volume of H<sub>2</sub>O in an attempt to remove TFA and other undesirable impurities which may have persisted from the synthesis, cleavage, and lyophilization in other solvents. After several rounds of lyophilization with excess water, the crude sample was a mixture of white and yellow powder. The new crude sample was successfully dissolved in a solution with final composition as follows: ~1.5 mg/mL crude peptide in 1 mL of 50% H<sub>2</sub>O / 50% ACN, 200 mM DTT. The purity of the sample was assessed by HPLC as detailed in Figure 4.12 on page 83. Lyophilized fractions from the crude HPLC run were reconstituted in 50% H<sub>2</sub>O/ACN, either with or without 200 mM DTT (1 hour treatment), and then diluted two-fold with  $\alpha$ -CHC matrix. MALDI results were consistent with elution of the product between 18-30 minutes on the HPLC trace (see summary Table 4.1 on page 102 and selected MALDI mass spectra starting with Figure B.1 on page 217).

The apparently broad elution of the desired product, along with low molecular weight impurities, over a 12 minute HPLC window may result from the rather large 6 minute fraction collection times. Therefore, an HPLC run using the same ACN gradient was repeated with a similar preparation of crude, but using narrower 1 minute fraction collection windows. A broad mass of peaks was again apparent between 18-30 minutes (Figure 4.13 on page 84), and MALDI-MS on the 12 $\times$ 1 minute fractions between 18-30 minutes revealed some promising candidates for additional purification (see summary Table 4.2 on page 103 and selected MALDI spectra starting with Figure B.5 on page 221). While the persistence of low molecular weight impurities and the nearly continuous co-elution of the ~2121 g/mol species (missing the N-terminal K) along with the desired product at ~2250 g/mol should be noted, fraction 5 (22-23 minutes) stands out as the most promising with a favorable 2250:2121 species ratio (assuming a similar MALDI ionization potential).

The MALDI-MS of the most promising C3 (HPLC) column-purified TR-09-1 frac-

tion (#5) contains impurities including a major species at  $\sim 1096$  m/z and the truncated product at  $\sim 2121$  m/z (Figure B.5 on page 221). In an attempt to improve separation, I repeated the same ACN gradient on a C18 semi-preparative HPLC column with the C3 column-purified fraction #5 as starting material (Figure 4.14 on page 85). A promising set of peaks eluted at  $\sim 24$ - $26$  minutes from the C18 column, and the run was repeated to collect 15 second fractions in the region of interest (Figure 4.15 on page 86). The best sample purity yet was achieved in fraction #5 from the C18 column, with the target MALDI peak dwarfing the truncated product and most low molecular weight species (see summary Table 4.3 on page 103 and selected MALDI spectra starting with Figure B.11 on page 227).

For the sake of time and yield, it is desirable to avoid a 2-column HPLC purification scheme as described above. Therefore, I loaded a crude TR-09-1 sample directly to the C18 semi-preparative column to assess the HPLC separation that can be achieved (Figure 4.16 on page 87). The run was repeated (Figure 4.17 on page 88) with collection of fractions near the window of optimal purification observed for the C18 column in the two-part purification scheme. It is encouraging that one of the fractions from this single-column purification scheme exhibited the target product as the major peak (see summary Table 4.4 on page 104 and selected MALDI spectra starting with Figure B.15 on page 231). The purest fraction was offset by  $\sim 15$  seconds relative to the optimal elution in the two-part scheme. The delay may reflect differences in sample viscosity/composition (*i.e.*, C3 column-purified sample would have substantially less impurities than the direct load of crude).

At this stage, it appears that a C18 HPLC column may provide a route to TR-09-1 purification with an optimized ACN gradient. However, I first wanted to verify the purity of fraction # 5 (from the two-column purification scheme) on a C18 analytical column—since this fraction was the purest I had obtained to date and it is desirable to assess the point at which a relatively pure sample elutes from a C18

column. The results were confounded by an impurity present on the C18 analytical column (Figure 4.18 on page 89), and there was no clear candidate peak for elution of the relatively pure sample. Instead of simply repeating the method, I designed a slower ACN gradient for the C18 analytical column in an attempt to achieve better separation of TR-09-1 crude (Figure 4.19 on page 90). However, there were peaks on the tail of the solvent front so I decided to switch to a more conservative starting % ACN and collected fractions for some reasonably well-resolved HPLC peaks near the estimated ~48% ACN elution point for TR-09-1 (Figure 4.20 on page 91). Unfortunately, MALDI-MS results did not show evidence of TR-09-1 in any of the analytical fractions (not shown). I reloaded the fractions to the C18 analytical column and the results did not convincingly demonstrate that I had in fact captured the target peaks with the assumed ~97.5 second delay at 0.48 mL/min (not shown).

Assured delay by HPLC re-calibration allowed me to return to the original challenge—purification of TR-09-1. I decided to aim for bulk purification of TR-09-1 on a C18 semi-preparative column since the C18 results had generally been better than C3, and because another group reported successful purification of similar spitz TMD constructs using a C18 column (153). The entire remaining TR-09-1 stock was dissolved in ~15 mL of 50% ACN/H<sub>2</sub>O, 0.1% TFA, 156 mM DTT solution. The mass of crude prior to dissolution was not known, but the latter solution was quite viscous. The first two bulk purification test runs on the C18 semi-preparative column are shown in Figure 4.21 on page 92. The fractions were collected in too narrow a time window in the first run based on the MALDI results (summarized in Table 4.5 on page 104 and see selected MALDI spectra starting with Figure B.19 on page 235). I shifted the fraction collection to a later time window (on the basis of crude viscosity and the latter MALDI results) in the second run, and while the product was more prominent in some of the fractions, some product was present in virtually all of the fractions over the 7 minute collection window (see summary Table 4.6 on page 105 and selected

MALDI spectra starting with Figure B.21 on page 237). The breadth of product elution based on MALDI results is not surprising given the outrageously broad HPLC profiles for crude TR-09-1.

The 2-100% ACN gradient used for the bulk purifications above may ramp too rapidly and result in incomplete separation, especially in the context of a concentrated sample. I tried a set of more conservative gradients in C18 semi-preparative HPLC purifications of the crude—starting at 40% ACN (Figure 4.22 on page 93) or 30% ACN (Figure 4.23 on page 94). Depressingly, TR-09-1 eluted over (practically) the full 26 minute range of collected fractions from the replicate starting at 40% ACN (see summary Table 4.7 on page 105 and selected MALDI spectra starting with Figure B.24 on page 240). Expansion of the collection window in the following run (starting at 30% ACN) revealed a 42 minute elution window (see summary Table 4.8 on page 106 and selected MALDI spectra starting with Figure B.26 on page 242). Some fractions were promising, but this is clearly not a desirable separation.

The ninth fraction (Figure B.26 on page 242) from the HPLC run detailed in Figure 4.23 (page 94) is promising despite the presence of the impurity at  $\sim 1096$  m/z. I repeated this HPLC gradient for bulk purification of several mL of crude TR-09-1, collecting the equivalent of the ninth fraction and also collecting subsequent eluent up to 52 minutes because of the broad elution profile described above. A subset of these purification HPLC runs are compared with the original run in Figure 4.24 on page 95. All remaining TR-09-1 crude was purified in this fashion. However, it was not possible to achieve purification of sufficient material to perform NMR spectroscopy after several additional rounds of purification (not shown).

## **4.6 Production And Purification Of A 16-Residue TatA TMD Construct (tr-10-2)**

*P. stuartii* TatA is a substrate of the rhomboid homologue AarA and I targeted a



TM portion of TatA for manual solid-phase peptide synthesis because of the demand on our automated synthesizer and the lack of success producing spitz constructs in an automated context. The Fmoc-based synthesis of the 16-residue TatA construct (ESTIATAAFGSPWQLI), designated TR-10-2, is based on a reported manual synthesis procedure (154) and was produced starting with 100 mg of Fmoc-Ile Wang resin (0.55 mmol/g). Remarkably, all of the colourimetric tests for completion of reactions were successful. The MALDI spectrum of the crude sample reconstituted in ACN produced a promising result that includes prominent peaks for the Na<sup>+</sup> and K<sup>+</sup> product adducts (starting with Figure 4.25 on page 96).

To further characterize the purity of the crude TR-10-2 preparation and to assess the prospects for separation, I performed C18 analytical HPLC on the sample (Figure 4.27 on page 98 and Figure 4.28 on page 99). The results were encouraging, with a dominant peak in the traces at 210 nm and 280 nm—the latter consistent with the presence of a W residue in the desired product. Furthermore, the target product was only present in two fractions near the expected elution time of the major peak (see MALDI summary Table 4.9 on page 107 and representative MALDI spectra starting with Figure B.33 on page 249). Impurities persist in the promising fractions but this is not surprising given the width of the HPLC collection windows. Although several attempts were made to collect higher resolution HPLC fractions in the region of interest, a number of problems with HPLC plumbing and possible contamination of HPLC solvents were highly problematic (not shown). Nonetheless, I think this particular TatA construct and manual synthesis are both promising ideas for the production of a spitz homologue.

## **4.7 Production Of A 37-Residue Gurken TMD Construct (tr-10-1)**

Gurken is another *Drosophila* substrate of Rhomboid, and I attempted automated

solid-phase peptide synthesis of a 37 residue construct of gurken (designated TR-10-1) based on the construct outlined in (141). I incorporated a number of partial  $^{15}\text{N}$ -backbone isotope labels (KKRKV[50]RM[50]A[50]HIV[50]F[25]SF[20]PV[50]L[50]-LM[25]LSSL[15]YVL[25]F[10]A[75]A[100]VF[25]ML[50]RKKK, with label % following residues) during the synthesis, which used 0.7 g of Fmoc-Lys(Boc)-Wang resin (loading of 0.19 mmol/g). Colourimetric tests for completed reactions were often not successful, and it is therefore not surprising that crude peptide MALDI-MS characterization did not exhibit any prospective product peaks (not shown). It is also noteworthy that the target product mass  $\sim 4405$  g/mol is calculated using a procedure similar to that outlined in Figure 4.10 on page 81 for fractional isotope incorporation.

## 4.8 Production And Purification Of A Spitz Construct By Expression In *E. coli*

Clearly it is difficult to produce spitz TMD constructs by SPPS, and purification by HPLC is even more problematic. Overexpression of a spitz TMD construct in *E. coli* allows for the incorporation of a hexahistidine tag for Ni-affinity purification, a FLAG tag for affinity chromatography, and does not involve the production of truncated products with similar chemical and physical properties (which is observed in SPPS because of failed reactions). Thus, there are options to circumvent HPLC purification or at least simplify the work-up by avoiding truncated species.

A 2.7 mg crude yield was obtained by expression of the construct outlined in Figure 4.29 on page 99 and subsequent Ni-affinity purification. C3 semi-preparative HPLC assessment of the crude produced a promising trace (Figure 4.30 on page 100), and ESI-MS characterization of the major peak  $\sim 26$  minutes was consistent with the mass of the (N-terminal Met-deficient) unlabelled product (not shown). However, subsequent matching HPLC runs were not consistent, and expression of the  $^{13}\text{C}$ ,  $^{15}\text{N}$ -labelled construct resulted in an extremely poor yield.

A number of expressions were attempted with different strategies for target protein isolation (*i.e.*, use of inclusion bodies), but there were always problems with sample solubility and the apparent presence of oligomers. One MALDI-MS result confirmed that ~50%  $^{15}\text{N}$  incorporation was achieved because of the (accidental) exposure of bacteria to both labelled and unlabelled nutrient sources (Figure 4.31 on page 101). In subsequent spitz expressions there were persistent problems with the presence of dimers, which could not be disrupted by exposure to 8 M urea and/or the reducing agents DTT, TCEP, or  $\beta$ -mercaptoethanol (see representative Figure 4.32 on page 102). Furthermore, MALDI and SDS-PAGE results were both consistent with the formation of higher order oligomers, and this may account for inconsistent elution times in matching HPLC runs. For now, spitz expression constructs are not any more tractable than constructs produced by SPPS and the yields are very low for isotope-labelled expression employing minimal media.

## 4.9 Conclusions

In this chapter I've described several different strategies and constructs for the production of a spitz or homologous rhomboid substrate with the ultimate objective of determining a high-resolution NMR structure. Although the latter goal was not achieved, it is certainly clear that the production and purification of these TMD constructs is extremely challenging. In light of the difficulties I had assigning resonances in homonuclear spitz TMD NMR spectra (see section 4.4 on page 62), I would strongly suggest that incorporation of isotope labels will be important for future efforts. It is still not clear if cysteine residues are forming disulfide bridges responsible for the extensive oligomerization issues reported in section 4.5 (page 64), but it is not necessarily a bad idea to substitute with Ser as described for the TR-08-2 construct.

It is particularly disturbing that successful HPLC purification of similar spitz-based TMD constructs was reported on a C18 column with no apparent difficulties

(153). I have some concern that I often overloaded the column because of the broad traces that were often observed. Conversely, the poor solubility of the samples largely excluded the possibility of loading very high concentrations of crude, but may also account for in-column retention/precipitation. Furthermore, it can hardly be considered desirable to further dilute a sample that required  $76\times$  48-minute HPLC runs for purification (section 4.4 on page 62).

Frustrated by the poor peptide tractability described in this chapter, my remaining spitz-rhomboid studies were performed *in silico* with coarse-grained molecular dynamics simulations (chapter 6 on page 191).

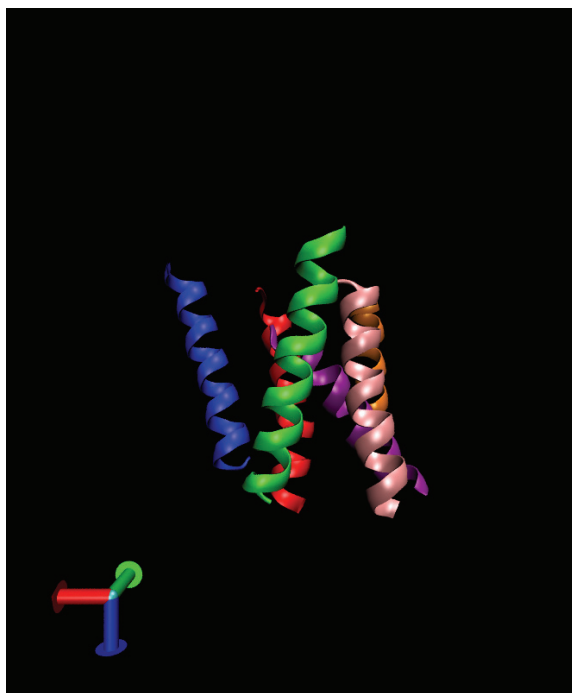


Figure 4.1: A cartoon representation of the TM segments from the ecGlpG structure (PDB: 2IC8; (4)). The top of this representation would represent the extracellular environment, the loops between helices have been excluded, and helix colouring follows the scheme: TM 1 (dark blue), TM 2 (red), TM 3 (green), TM 4 (purple), TM 5 (orange), TM 6 (pink). Note that TM 4, which includes the catalytic serine residue, is protected from the exterior of the protein by the ring of TM helices.

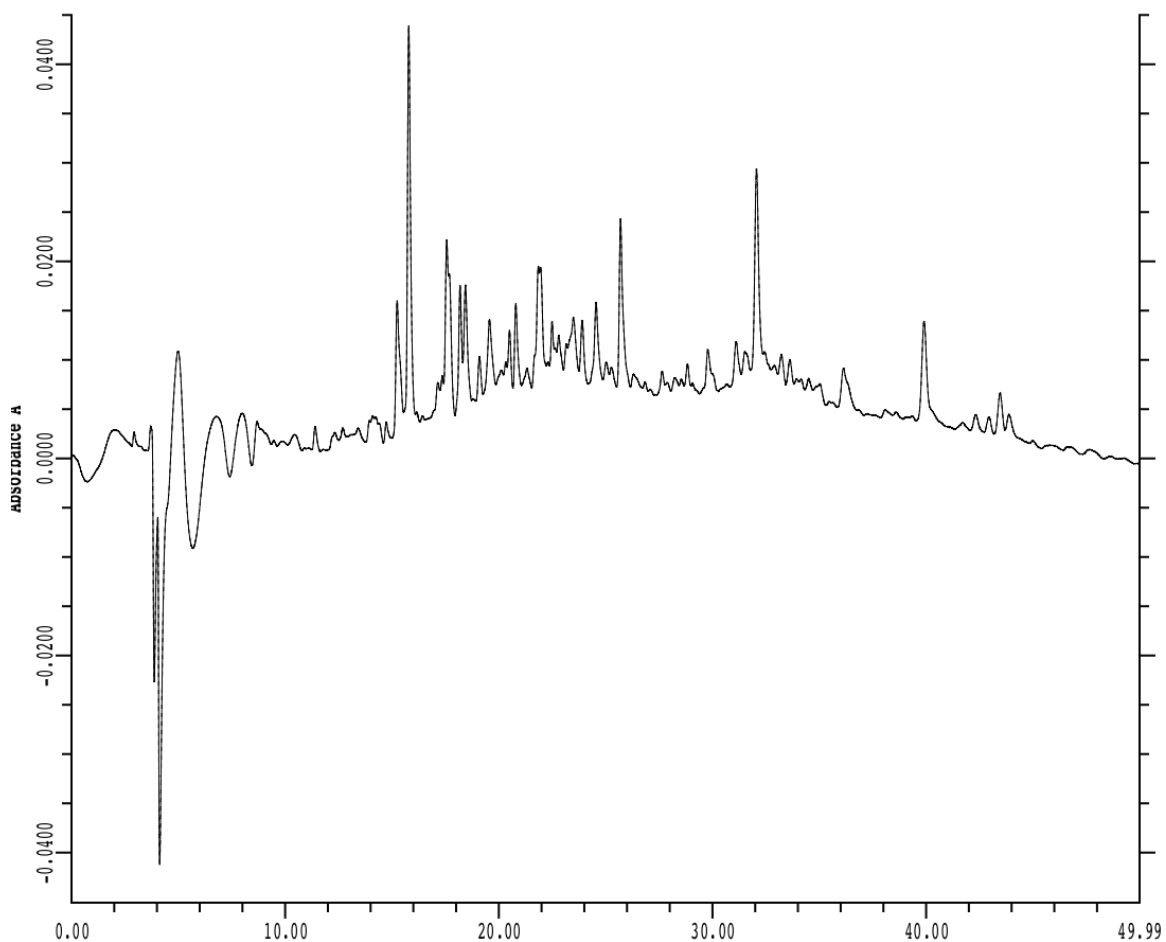


Figure 4.2: 30  $\mu\text{L}$  of a crude TR-08-2 preparation was loaded to a C18 analytical HPLC column and subject to a 0.8 mL/min gradient: 1) 2% ACN pre-run, 2) +2%/minute ACN up to 100%. Fractions were collected in 5-minute windows between 0 and 50 minutes and this trace is monitored at 210 nm. A preceding blank run with DI- $\text{H}_2\text{O}$  did not reveal any substantial column-retained impurities (not shown).

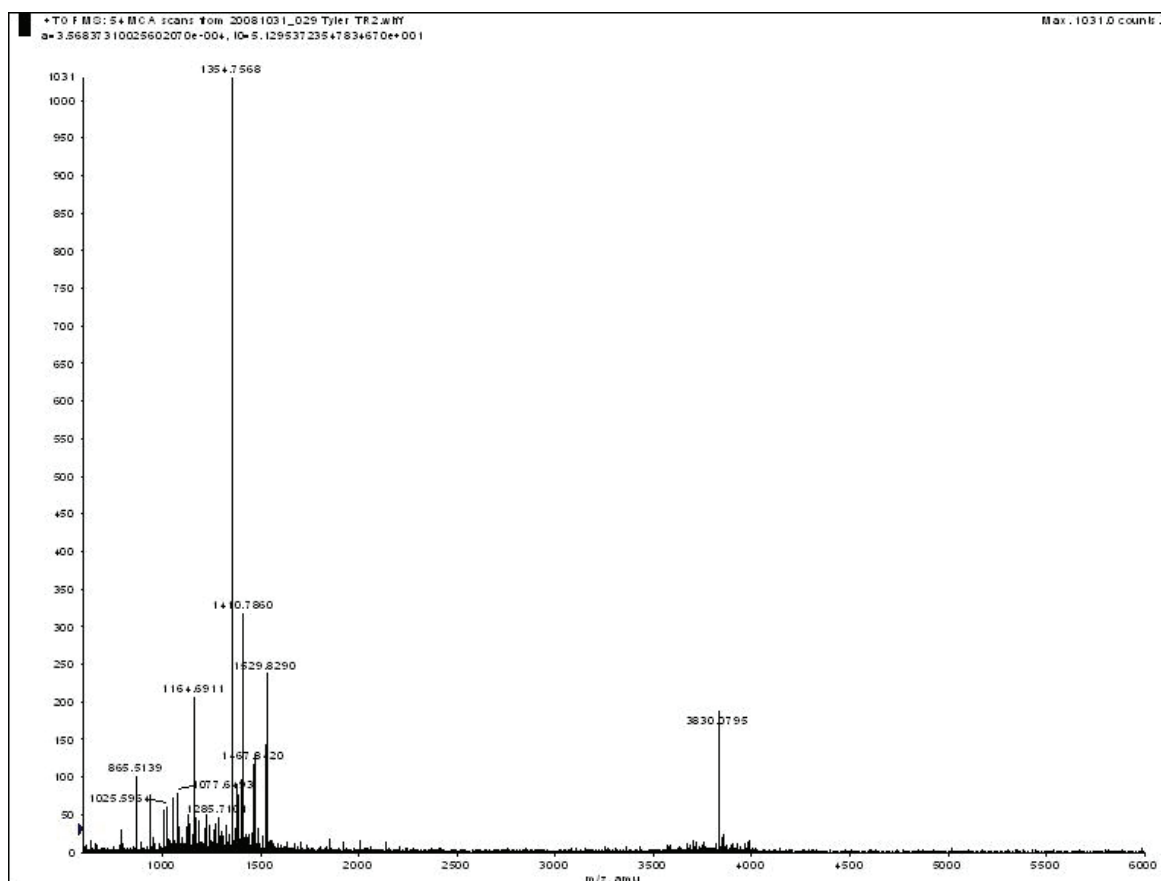


Figure 4.3: The most promising TR-08-2 fraction following several iterative rounds of HPLC optimization produced this MALDI spectrum with a clear candidate peak for the target product  $\sim 3830$  m/z with the monoisotopic theoretical mass  $\sim 3828$  g/mol. There do, however, appear to be some low molecular weight impurities.

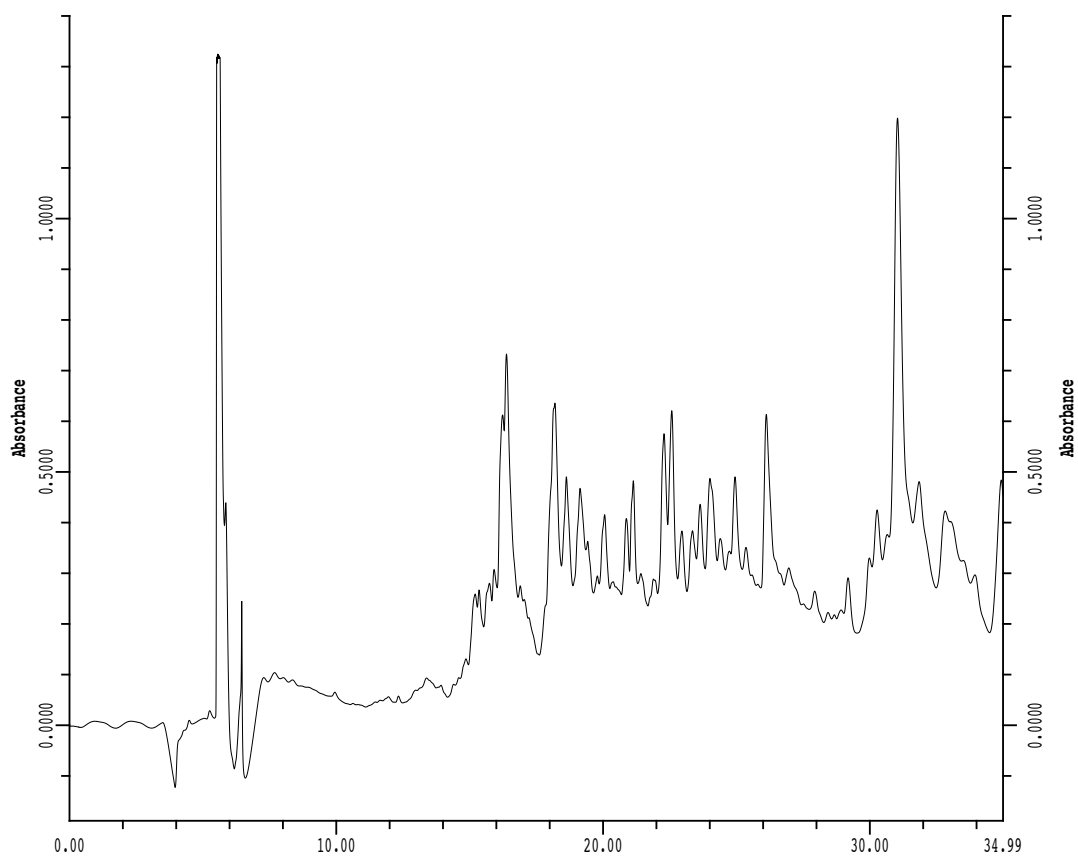


Figure 4.4: This is one of the 76 representative TR-08-2 bulk purification HPLC runs monitored at 210 nm. For each replicate, I collected between 28.5 and 32.3 minutes based on the MALDI results in Figure 4.3 on page 75 for a matching run. The gradient starts at 2% ACN and ramps by 2%/minute.



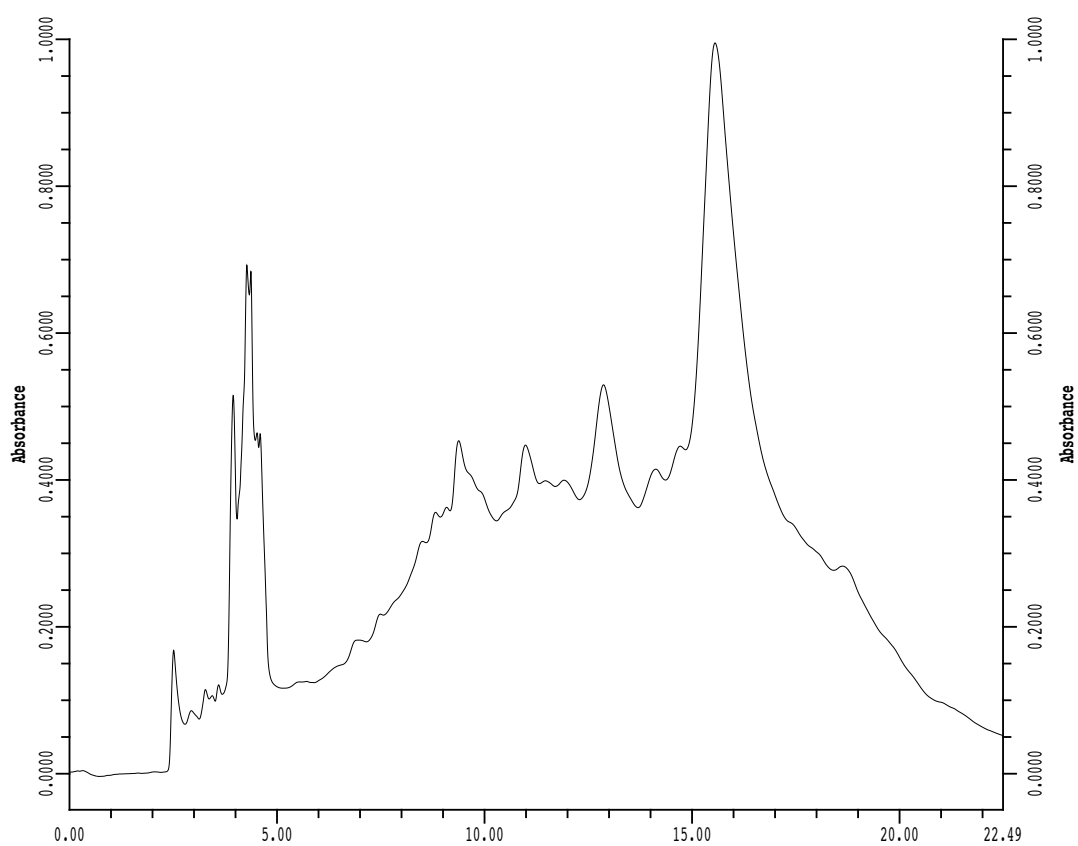


Figure 4.5: Fractions collected as detailed in Figure 4.4 (page 76) were subjected to additional semi-preparative HPLC purification and this is a representative trace monitored at 210 nm. Two fractions were collected between 12.2-14.2 minutes (fraction 1) and 14.8-16.6 minutes (fraction 2).

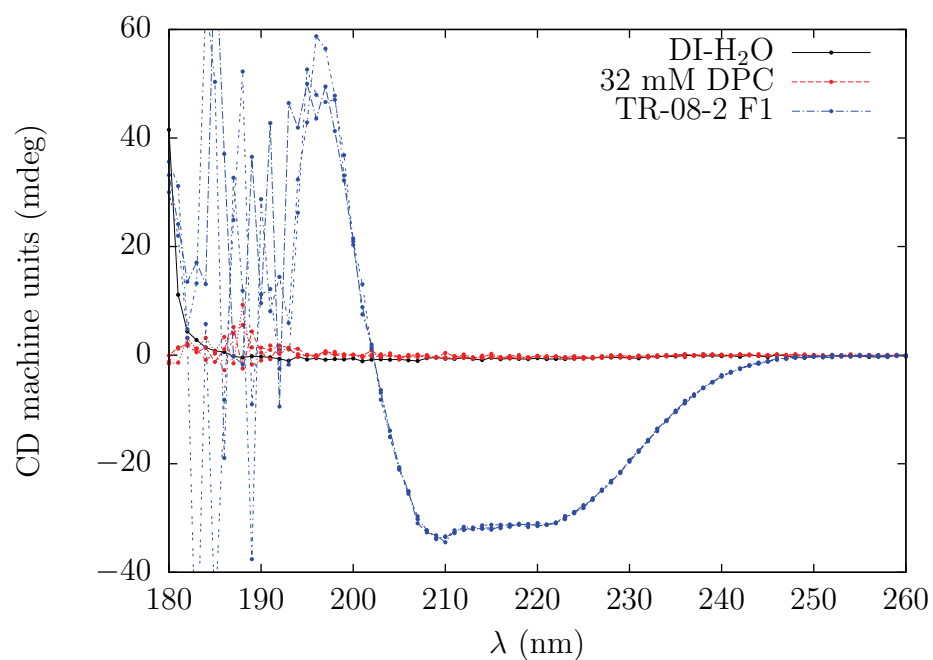


Figure 4.6: Circular dichroism (CD) spectropolarimetry was performed in triplicate for TR-08-2 preparations (100  $\mu$ M peptide, 32 mM DPC, pH 5) and a 32 mM DPC control solution at matching pH. A single blank run was also performed with deionized water. Measurements were collected at 37°C with a 20 nm/min scan rate through a 1 mm path length cell. Measured dips near 208 nm and 222 nm are characteristic of  $\alpha$ -helical secondary structure.

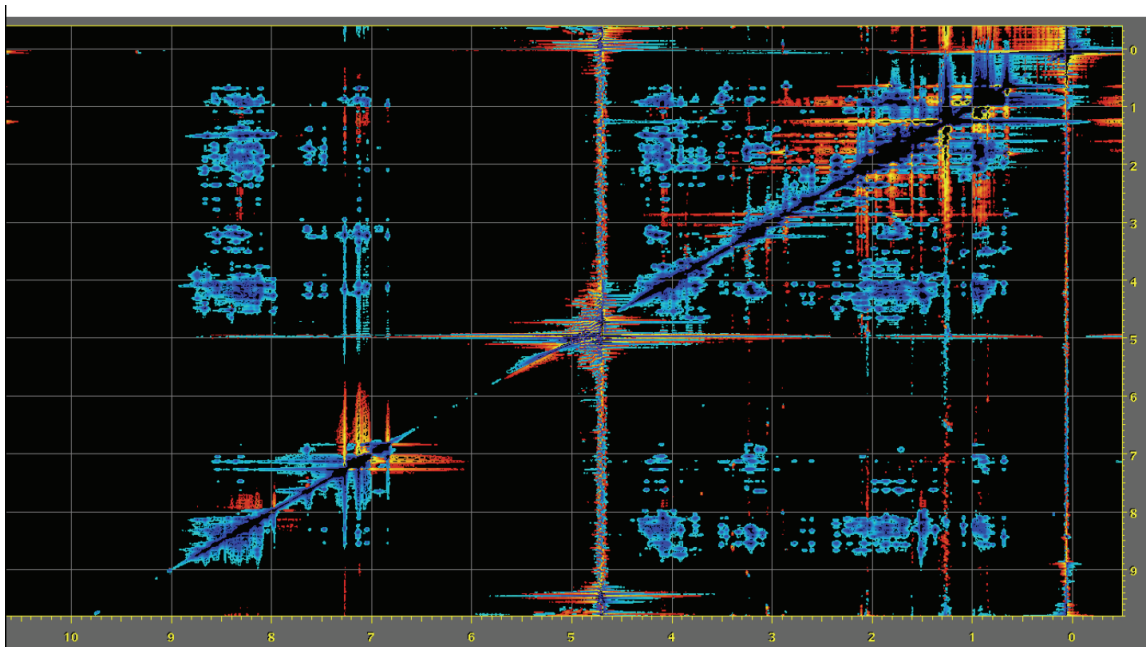


Figure 4.7: Overview of 2D  $^1\text{H}$ - $^1\text{H}$  NOESY (200 ms mixing time) collected for TR-08-2 on a 700 MHz magnet fitted with a cold probe. Water suppression was performed with excitation sculpting, and the spectrum was collected at 310.15 K.

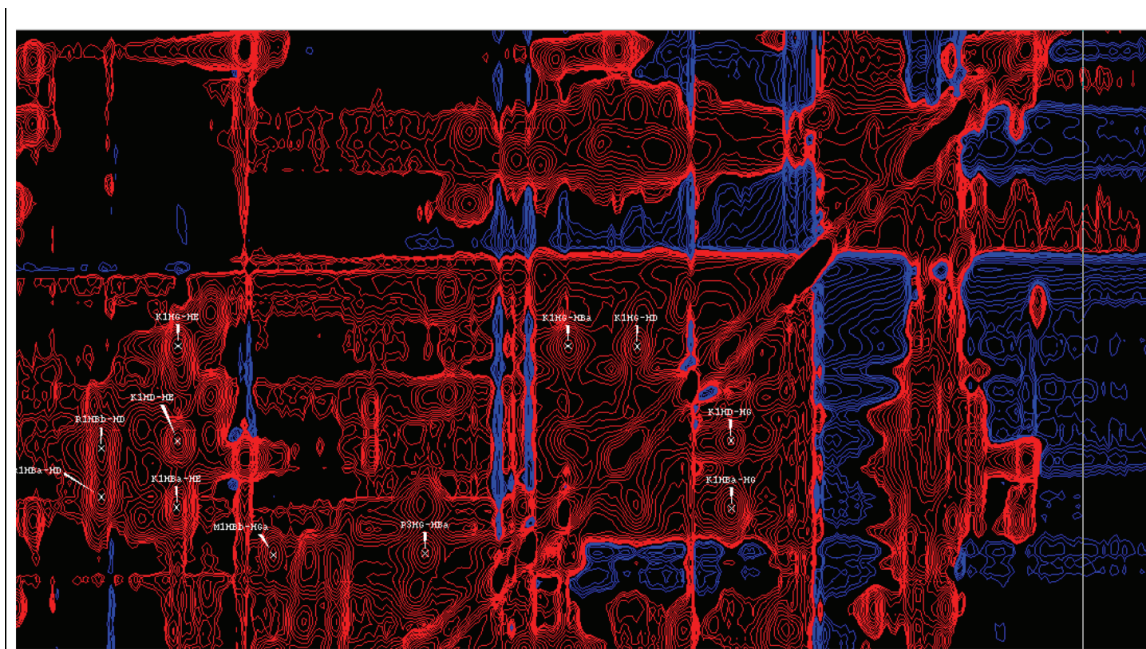


Figure 4.8: A zoom-in view of 2D  $^1\text{H}$ - $^1\text{H}$  TOCSY collected for TR-08-2 on a 700 MHz magnet fitted with a cryogenically cooled probe. Water suppression was performed with excitation sculpting, and the spectrum was collected at 310.15 K. Although some assignments are visible in this caption, it is also clear that there is substantial resonance overlap.

KRPRPMLEKASIASGAMSALVFMLFVSLAFYLRK  
KEKASIASGAMCALVFMLFV  
KEKASIASGAMCALVFMLFVK

Figure 4.9: Comparing spitz TMD constructs for solid-phase peptide synthesis. The 34-residue construct with 2 Cys→Ser substitutions (*top*) was problematic for structure determination by NMR spectroscopy. The *middle* construct is known to be cleaved by a Rhomboid homologue and was successfully produced by solid-phase peptide synthesis by another group (153). I produced and purified the *bottom* 21-residue construct because of its similarity to the published construct.

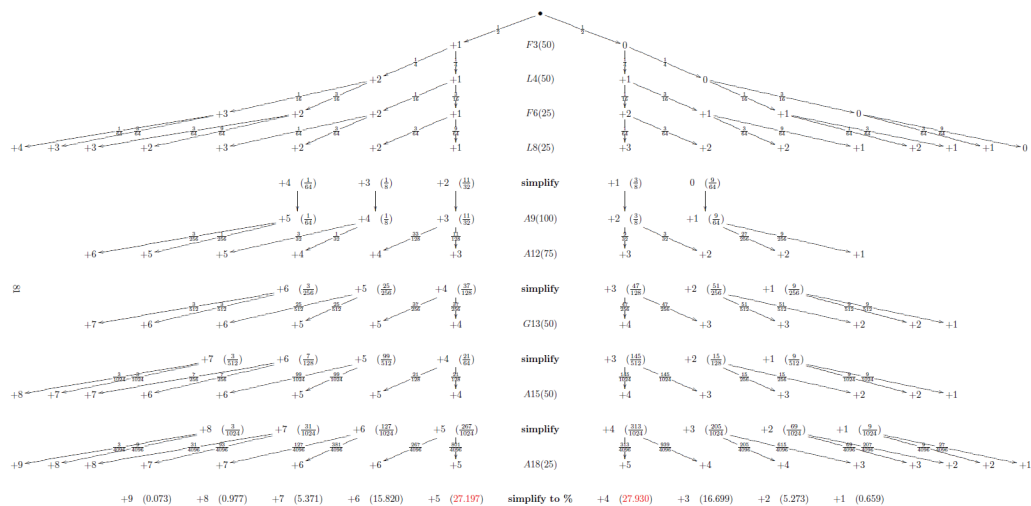


Figure 4.10: There are nine  $^{15}\text{N}$  fractional selective isotope label positions in the tr-09-1 construct, and properly predicting the adjustment to the monoisotopic mass requires a ‘tree-branch’ probability approach as outlined here. The coupling amino acid (proceeding  $\text{C}\rightarrow\text{N}$  as synthesized) and the isotope % are both indicated near the middle of the row that results from a given coupling. The arrow or parenthetical fractions indicate the proportion of all peptide species that have the resulting atomic mass adjustment (ignoring impurities that form from failed reactions). It is clear that between 4 and 5 atomic mass units should be added to the predicted monoisotopic mass to obtain the most abundant  $^{15}\text{N}$ -enriched species.

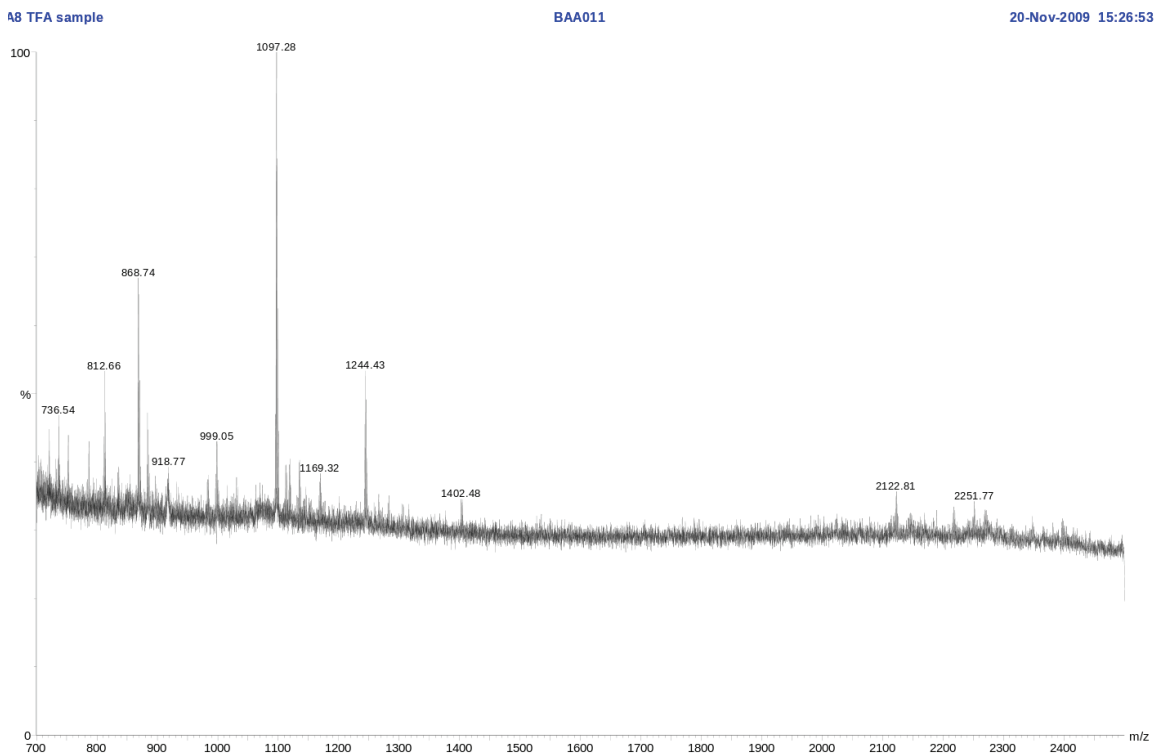


Figure 4.11: A 2.9 mg/mL crude TR-09-1 solution in 100% TFA was diluted with a MALDI matrix solution, and 1  $\mu\text{L}$  was spotted to a MALDI plate. The mass spectrum was captured in reflectron mode. The peak at  $\sim 2252$  m/z is consistent with the predicted mass of the desired product while the peak at  $\sim 2123$  most likely corresponds to the desired product missing the N-terminal K residue.

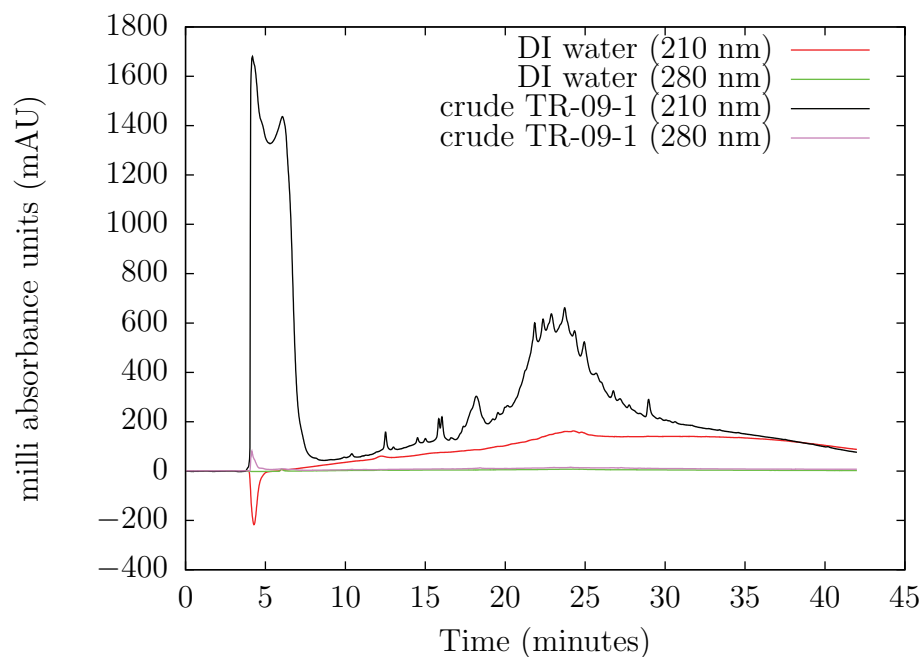


Figure 4.12: 1 mL of 1.5 mg/mL TR-09-1 crude peptide in 50% H<sub>2</sub>O / 50% ACN, 200 mM DTT, was loaded to a C3 RP-HPLC semi-preparative column and monitored at 210 nm and 280 nm using a gradient that proceeded from 2-100% ACN in 40 minutes, followed by 100-2% ACN in 2 minutes. The blank was an equivalent volume of deionized water (DI-H<sub>2</sub>O). Seven fractions from the crude peptide run were collected in six minute windows—F1 (0-6 minutes), F2 (6-12 minutes), F3 (12-18 minutes), F4 (18-24 minutes), F5 (24-30 minutes), F6 (30-36 minutes), F7 (36-42 minutes). The HPLC dead time at 3 mL/minute is ~28 seconds.

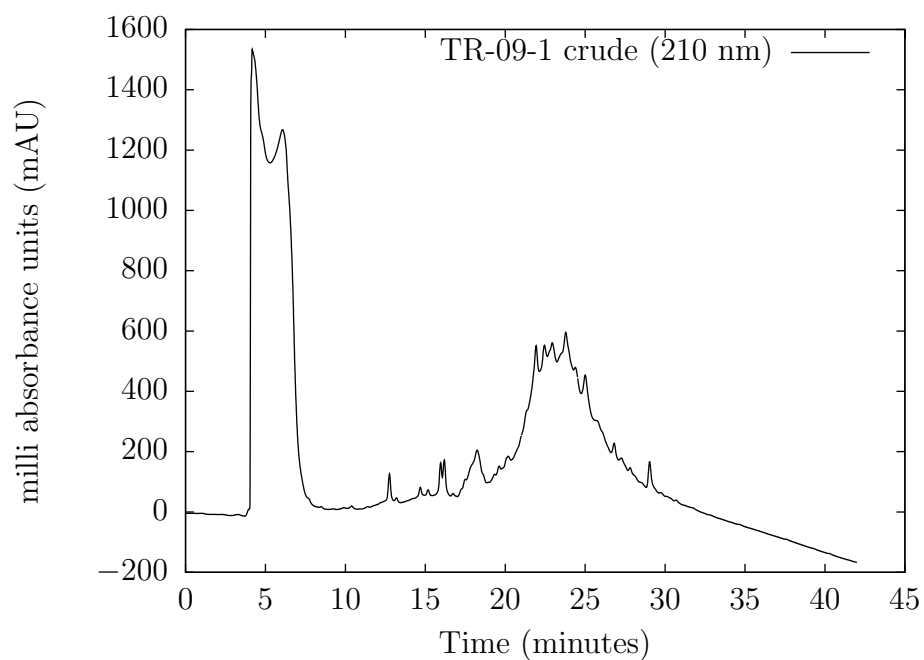


Figure 4.13: 1 mL of a 1.9 mg/mL TR-09-1 crude solution dissolved in 50% H<sub>2</sub>O/ACN was treated with 200 mM DTT for 1 hour and loaded to a C3 semi-preparative column. The gradient and flow rate match the conditions detailed in Figure 4.12 on page 83, but in this case narrower 1 minute fractions were collected between 18 and 30 minutes run time. There were no substantial impurities in a preceding blank run with 1 mL of deionized water (not shown).



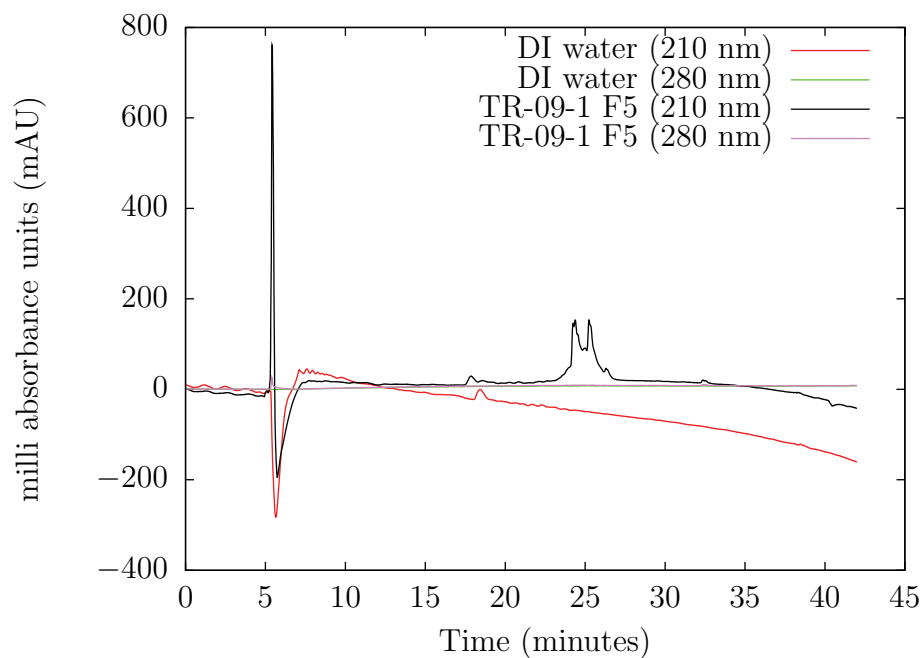


Figure 4.14: 1 mL of a C3 HPLC column-purified fraction (#5) from the run detailed in Figure 4.13 (page 84) was loaded to a C18 semi-preparative HPLC column and elution was monitored at 210 nm and 280 nm during a 3 mL/min gradient involving 2-100% ACN in 40 minutes followed by 100-2% ACN in 2 minutes. A matching volume of deionized water was used as a blank in a preceding HPLC run with the same gradient and flow rate.

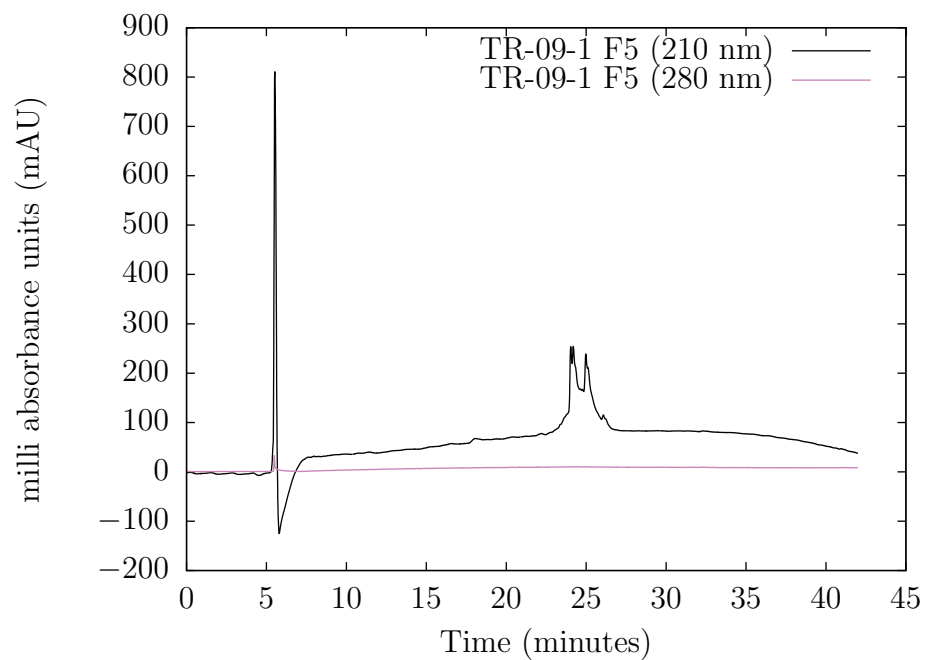


Figure 4.15: A repeat of the C18 semi-preparative HPLC run involving TR-09-1 fraction #5 (from a C3 column) as detailed in Figure 4.14 on page 85. However, in this case  $8 \times 15$  second fractions were collected starting at 24:16 run time. A preceding blank run with a matching volume of deionized water did not reveal any substantial impurities (not shown).

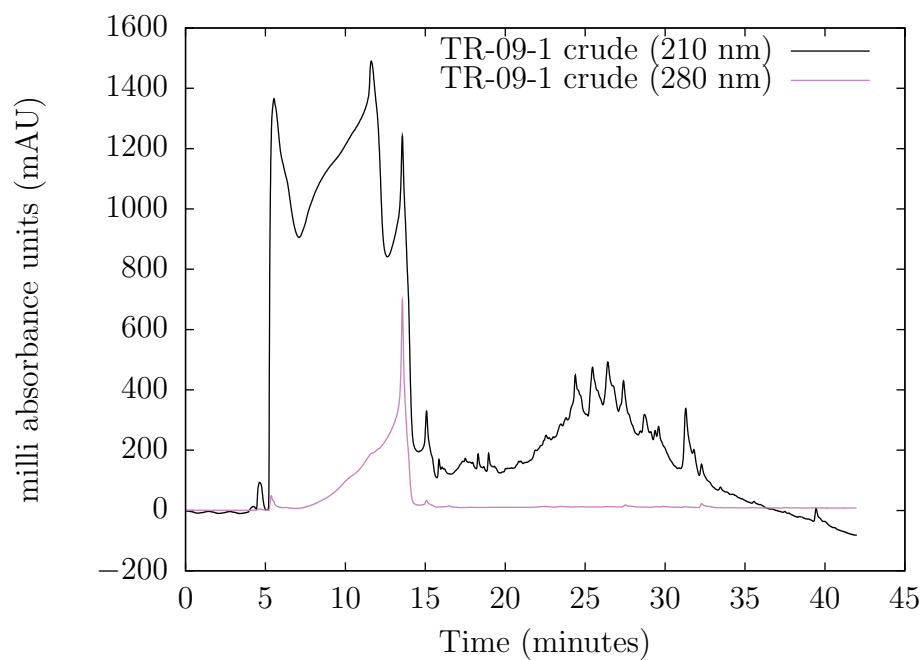


Figure 4.16: 1 mL of 1.3 mg/mL TR-09-1 crude sample in 50% H<sub>2</sub>O/ACN and 200 mM DTT was loaded to a C18 semi-preparative column and subjected to the same method described in Figure 4.14 on page 85. A matching volume of deionized water was used in a preceding blank run with no substantial impurities present (not shown).

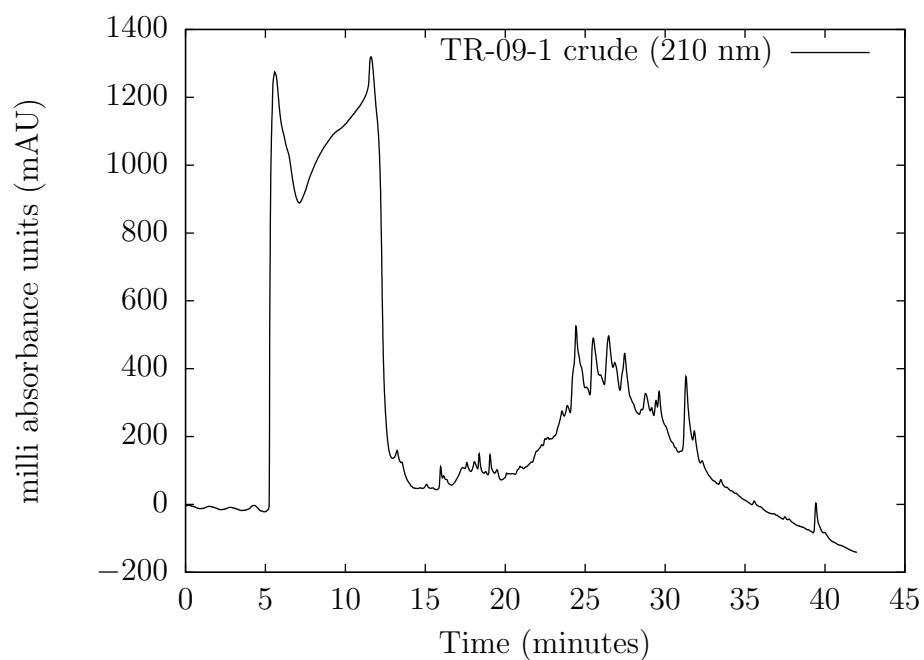


Figure 4.17: 1 mL of 2.1 mg/mL crude TR-09-1 in 50% H<sub>2</sub>O/ACN and 200 mM DTT was loaded to a C18 semi-preparative column in a repeat of the run described in Figure 4.16 (page 87), but in this case three fractions were collected to match the promising results from a C18 column summarized in Table 4.3 (page 103): F1 (25:01-25:16), F2 (25:16-25:31), and F3 (25:31-25:46). A matching volume of deionized water was used to confirm the absence of major column impurities in a preceding blank run.

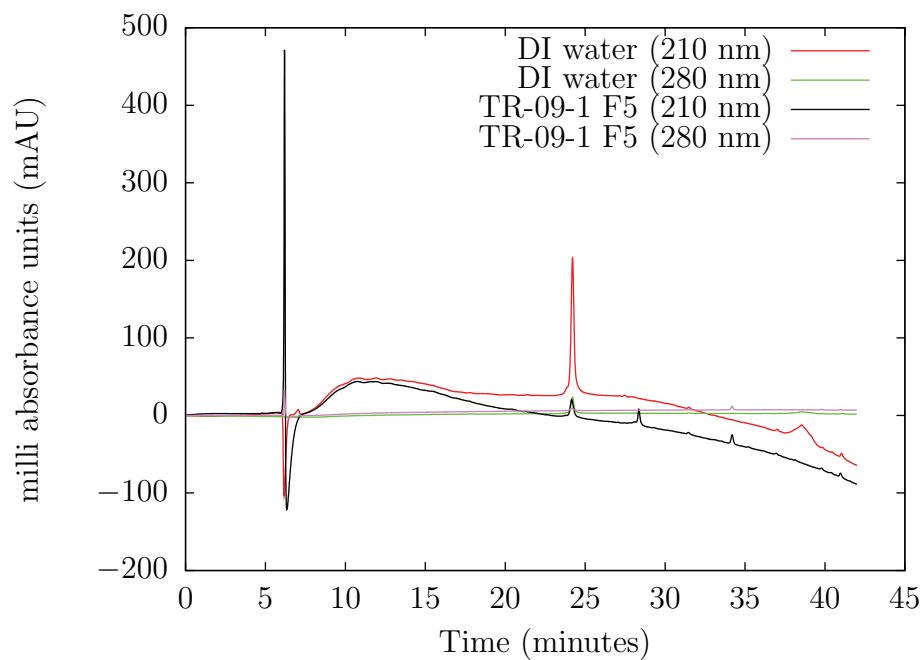


Figure 4.18: 20  $\mu\text{L}$  of the purest TR-09-1 fraction (# 5 from the two-column HPLC purification scheme) was loaded to a C18 analytical column and subject to an ACN gradient at 0.48 mL/min: 2-100% ACN in 40 minutes, followed by 100-2% ACN in 2 minutes. A matching volume of deionized water was used as a blank in a preceding run, and unfortunately a residual impurity confounds the results of the test trace.

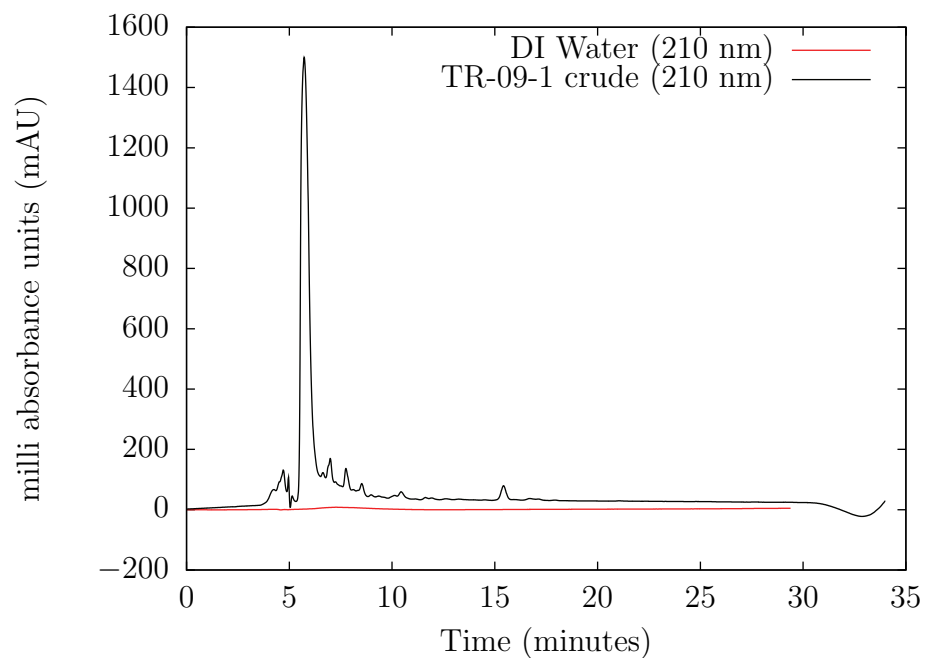


Figure 4.19: A 0.8 mg/mL preparation of TR-09-1 crude in 50% H<sub>2</sub>O/ACN and 200 mM DTT was loaded to a C18 analytical column using a 20  $\mu$ L sample loop. The method consisted of a 0.48 mL/min flow rate and a multi-step gradient: 1) Start at 53.9% ACN, 2) 53.9%-66.15% ACN from 0-25 minutes, 3) 66.15%-100% ACN from 25-27 minutes, 4) 100%-53.9% ACN from 27-28 minutes. The peak between 15-16 minutes on the crude trace was collected assuming a  $\sim$ 97.5 second delay based on the flow rate and system plumbing.

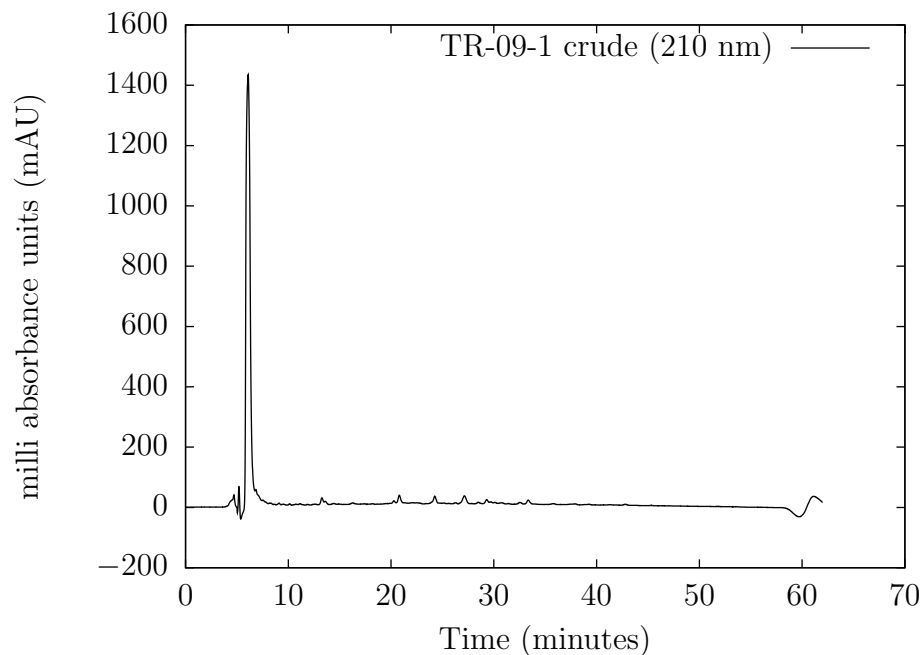


Figure 4.20: The crude TR-09-1 preparation described in Figure 4.19 (page 90) was again loaded to a C18 analytical column using a 20  $\mu\text{L}$  sample loop, but in this case using a more conservative gradient (still at 0.48 mL/min): 1) 40% ACN pre-run, 2) 40%-66% ACN in 53 minutes, 3) 66%-100% ACN in 1 minute, 4) 100%-40% ACN in 1 minute, 5) Continue monitoring run until  $\sim 62$  minutes. Three sets of peaks (20-22 minutes, 23-25 minutes, 26-28 minutes) were collected assuming a  $\sim 97.5$  second delay at this flow rate. A preceding blank run with deionized water demonstrated an absence of impurities on the column (not shown).

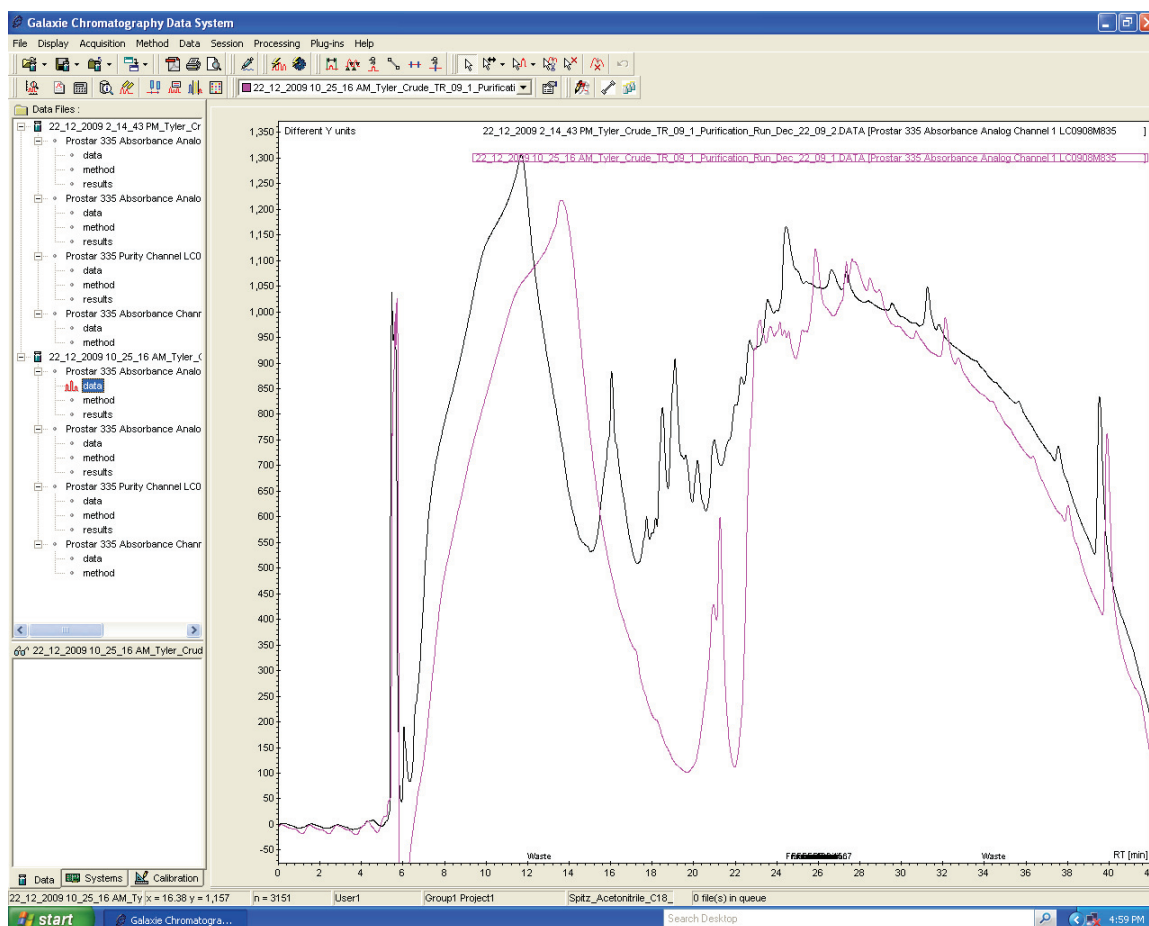


Figure 4.21: The first (pink) and second crude TR-09-1 bulk purification runs monitored at 210 nm. In both cases, 1 mL of crude (dissolved in 50% H<sub>2</sub>O/ACN, 0.1% TFA, 156 mM DTT) was loaded to a 2mL sample loop (bottom injector) and subject to a 3 mL/min gradient on a C18 semi-preparative column similar to those described for previous C18 purifications. 7×15 second fractions were collected between 25:10-26:55 from the first replicate, and 14×30 second fractions were collected between 26:55-33:55 from the second replicate.



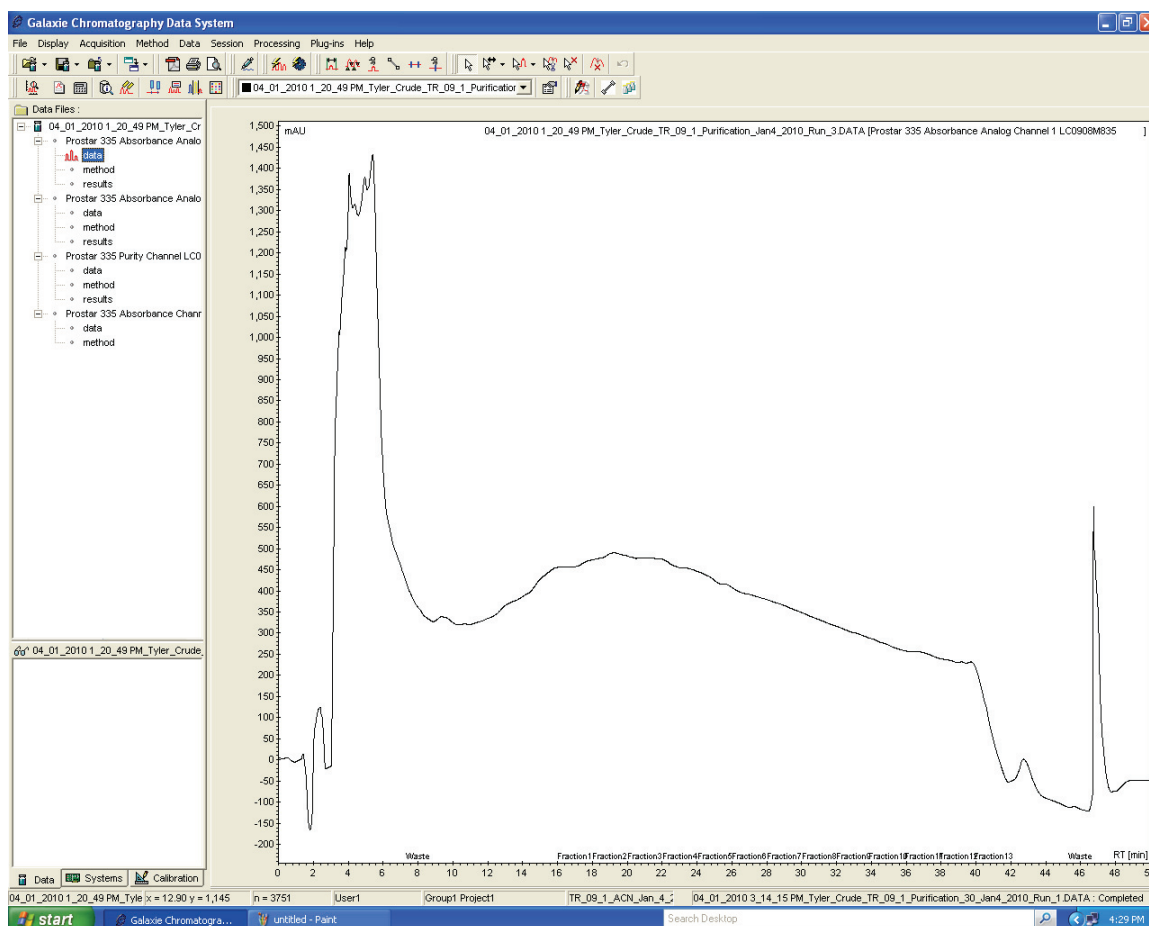


Figure 4.22: ~500  $\mu\text{L}$  of crude TR-09-1 was loaded to a C18 semi-preparative HPLC column via a 2 mL sample loop (bottom injector). The 3 mL/min gradient involved a few steps: 1) 40% ACN pre-run, 2) 40-75% ACN in 35 minutes, 3) 75-100% ACN in 1 minute, 4) hold 100% ACN for 5 minutes, 5) 100-40% ACN in 1 minute. 2 minute fractions were collected between 16-42 minutes. This trace is monitored at 210 nm, and a matching run with deionized water demonstrated an absence of column-retained impurities.

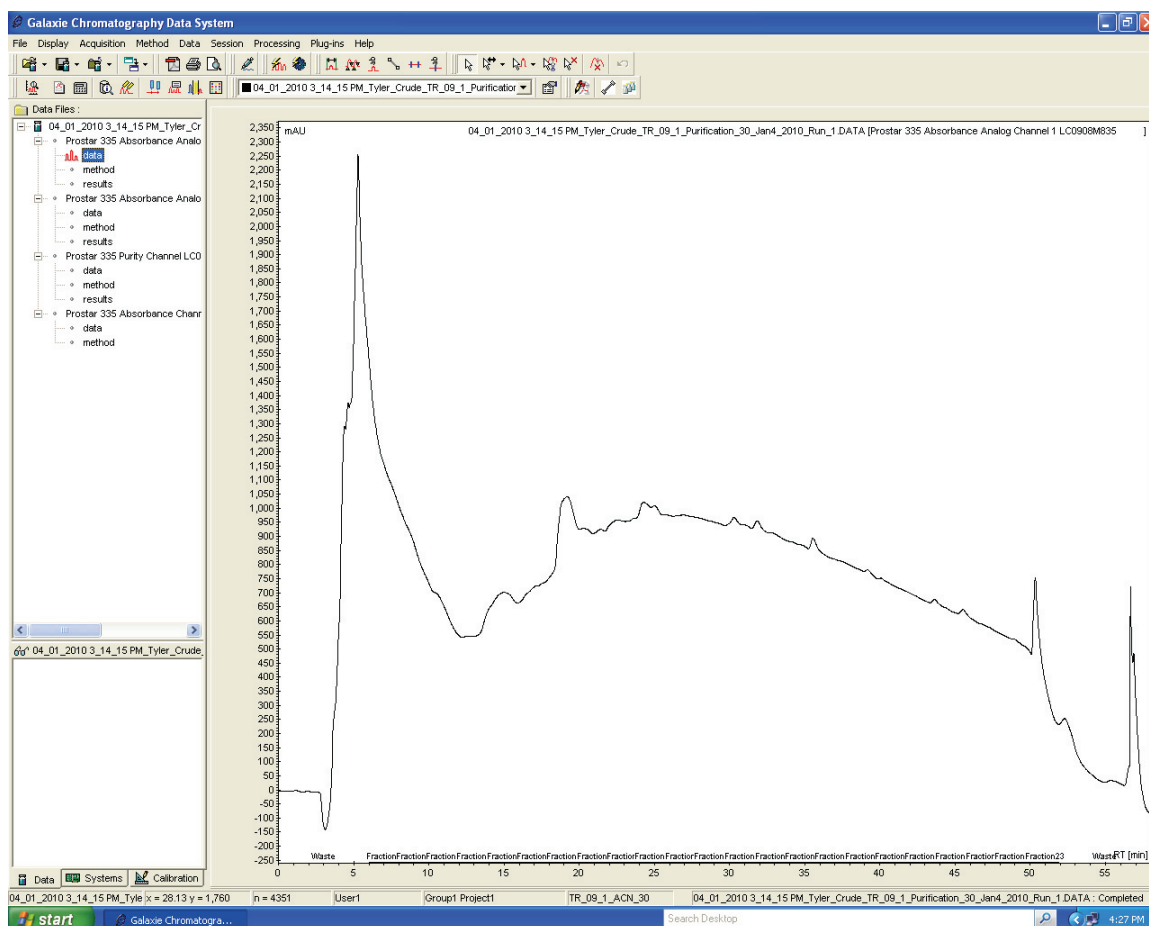


Figure 4.23: 1 mL of crude TR-09-1 was loaded to a C18 semi-preparative HPLC column via a 2 mL sample loop (bottom injector). The 3 mL/min gradient was modified from the previous run (Figure 4.22 on page 93) to start at a lower % ACN: 1) 30% ACN pre-run, 2) 30-75% ACN from 0-45 minutes, 3) 75-100% ACN from 45-46 minutes, 4) hold 100% ACN from 46-51 minutes, 5) 100-30% ACN from 51-52 minutes. This trace is monitored at 210nm.

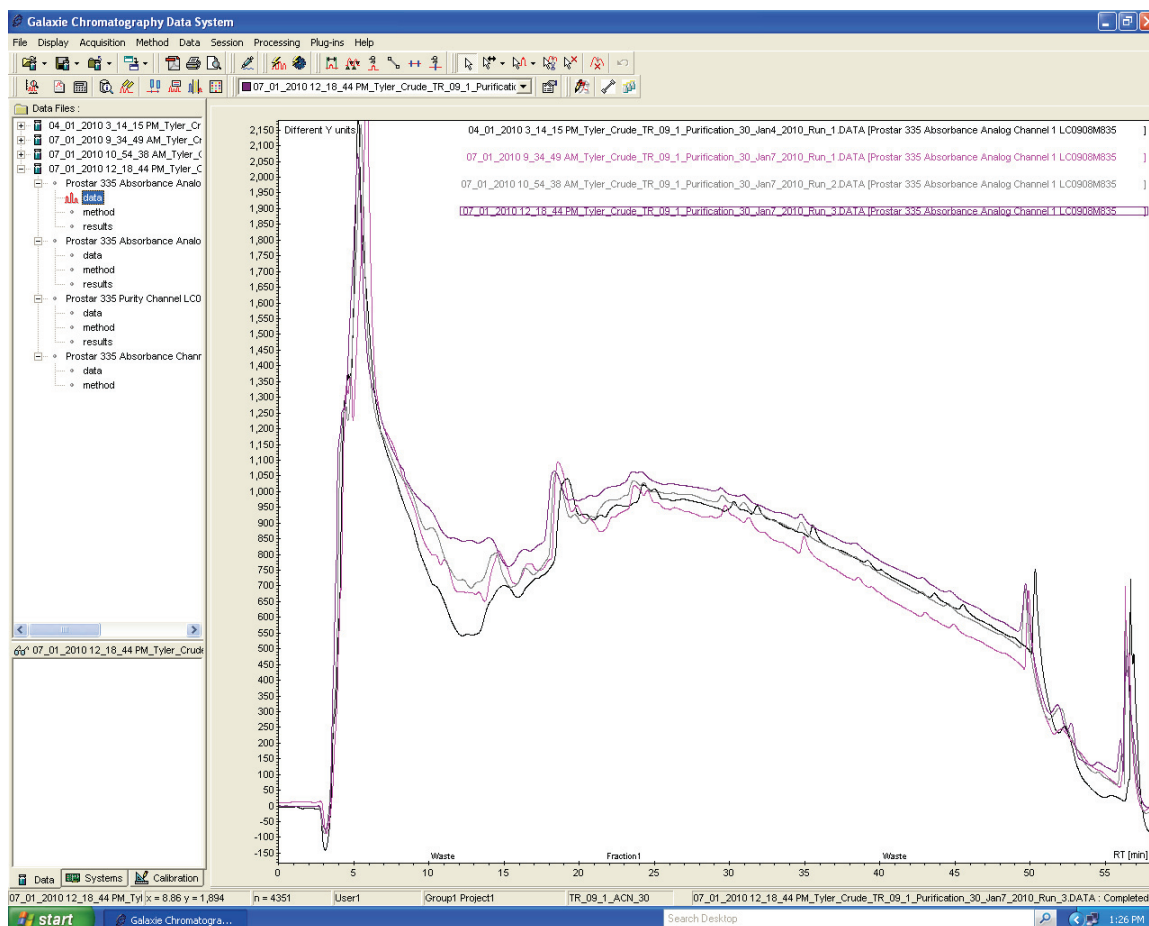


Figure 4.24: 1 mL of crude TR-09-1 was loaded to a C18 semi-preparative HPLC column and subjected to the same gradient described in Figure 4.23 on page 94. Based on the latter (original) HPLC run, the new purification runs targeted collection of the promising fraction that elutes between 22-24 minutes, but the remaining eluent up to 52 minutes was also collected because the product exhibits a very broad elution profile (see Table 4.8 on page 106). The 210 nm traces shown here compare the original HPLC run (in black) with the more recent bulk purifications. Unfortunately, the more recent runs accidentally employed a 1 mL sample loop instead of the 2 mL loop used in the original run, causing an earlier elution profile.

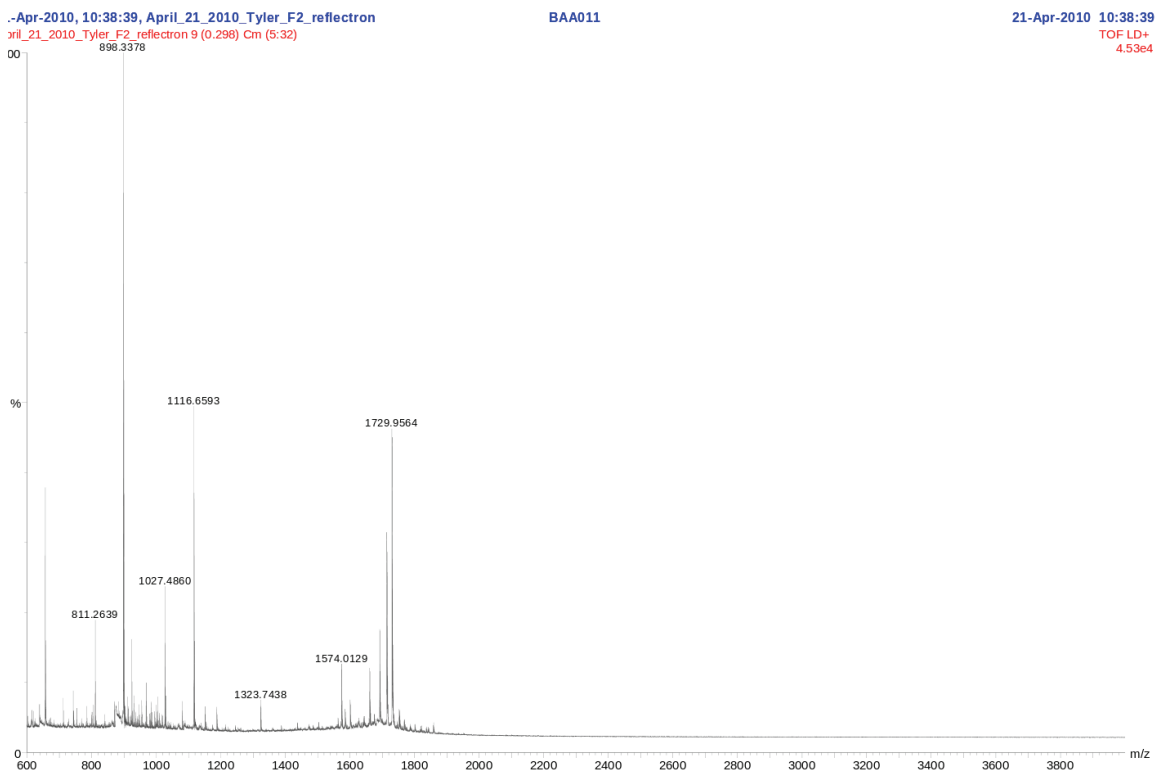


Figure 4.25: The TR-10-2 peptide construct was cleaved from the synthesis resin, lyophilized with excess water, and the crude was reconstituted with 50% ACN/ $\alpha$ -CHC matrix and the displayed reflectron-mode MALDI spectrum was collected. The target product should be  $\sim 1691$  g/mol, and the large peak at  $\sim 1729$  g/mol is almost certainly the  $K^+$  adduct. While this is a promising result for the first attempt at manual solid-phase peptide synthesis in our lab (and for a zoom-in version see Figure 4.26 on page 97), there are certainly some low molecular weight impurities.

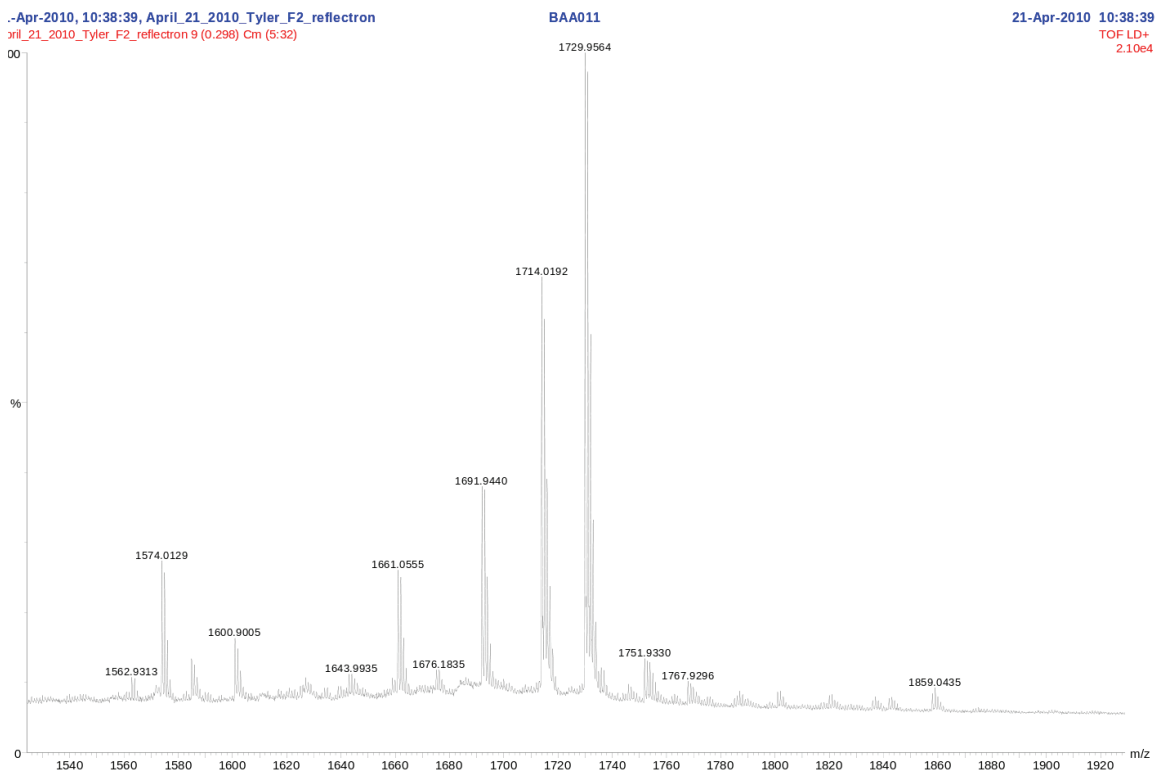


Figure 4.26: A zoom-in view of the reflectron-mode MALDI spectrum detailed in Figure 4.25 on page 96. The TR-10-2 target product is  $\sim 1691$  m/z, while the  $\text{Na}^+$  and  $\text{K}^+$  adducts are also clearly visible  $\sim 1714$  m/z and  $\sim 1729$  m/z respectively.

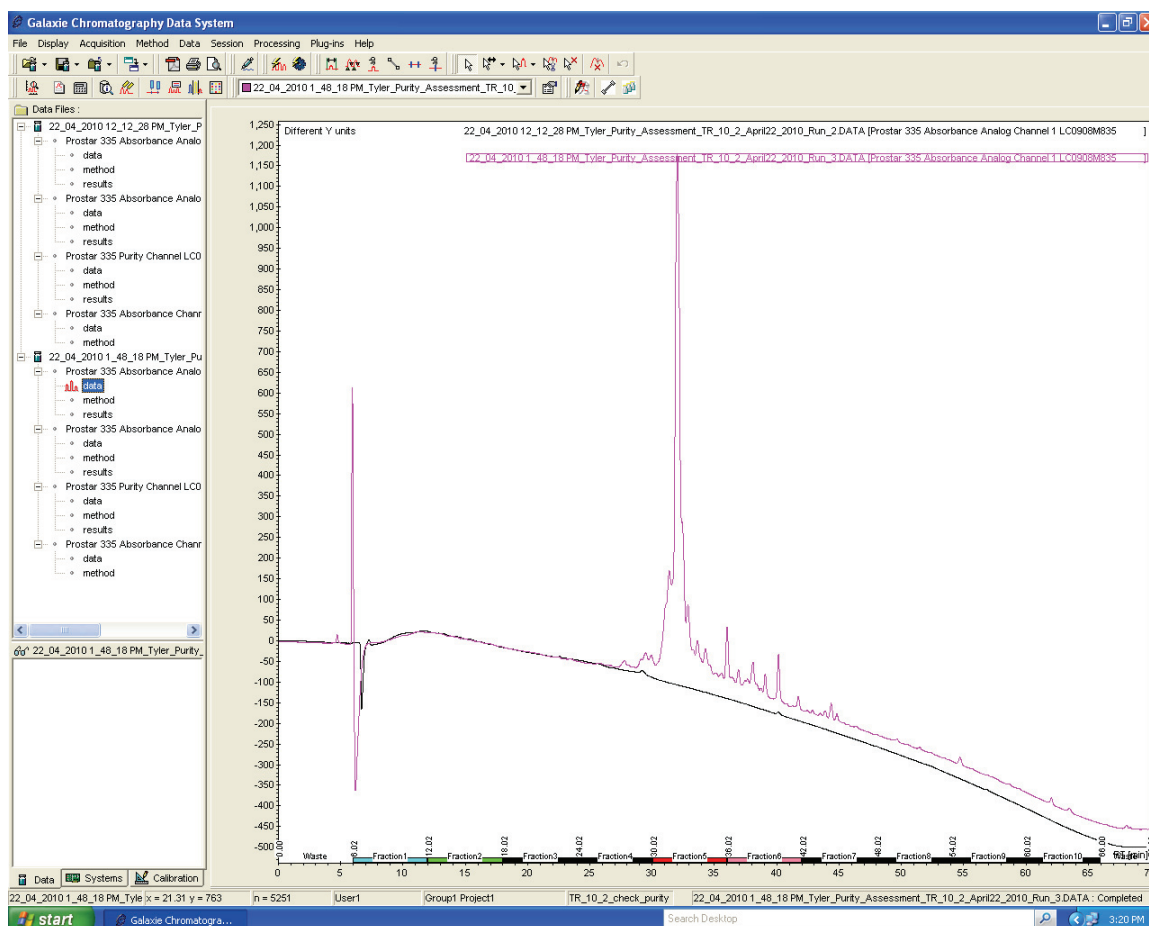


Figure 4.27: A crude preparation of TR-10-2 dissolved in ACN was loaded to a C18 analytical column via a 20  $\mu$ L sample loop (top injector) and subject to a 0.48 mL/min gradient: 1) 2-100% ACN in 60 minutes, 2) hold 100% ACN for 5 minutes, 3) ramp back 100-2% ACN in 1 minute. The trace shown here was monitored at 210 nm for the test run (pink) and a matching blank run with DI-H<sub>2</sub>O (black) and fractions were collected in 6 minute windows between 6 and 66 minutes. The trace monitored at 280 nm is shown in Figure 4.28 on page 99.

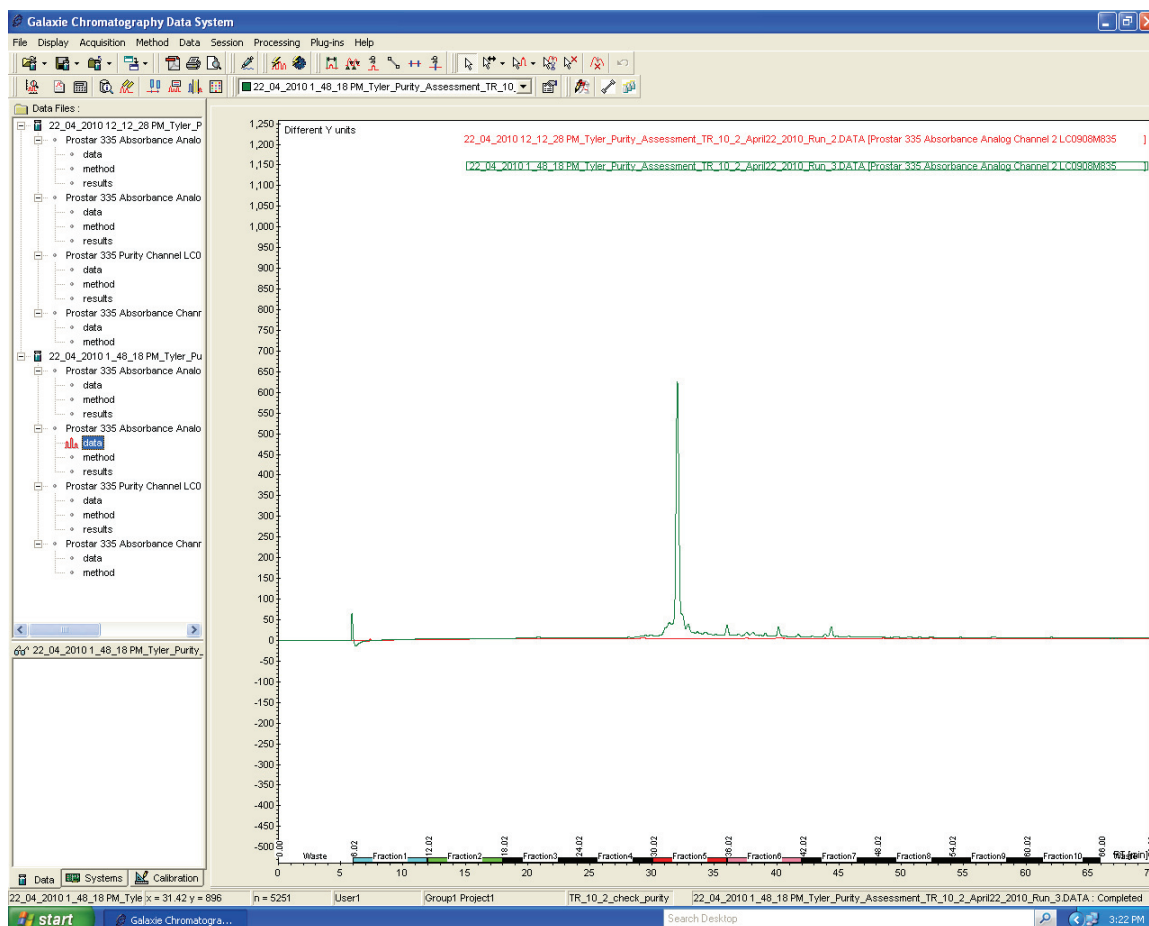


Figure 4.28: This is the TR-10-2 C18 analytical HPLC purity assessment run detailed in Figure 4.27 (page 98) monitored at 280 nm for the crude peptide (green) and the DI-H<sub>2</sub>O blank (red).



Figure 4.29: Expected (82-residue) spitz fusion peptide produced by expression in pEXP5-NT/TOPO vector with TEV protease cleavage site highlighted in red. There is an N-terminal His tag and a C-terminal FLAG tag. The N-terminal Met is normally removed during production, shifting the expected average isotopic mass from ~9524 g/mol to ~9393 g/mol.

NAME	CHN	LEV	REP	TYPE	DIRECTORY	TIME	DATE
COLLECTION DATA	30122413	A	1	1	0119 C:\GOLD\SYSTEM2\DATA\	INJECTION	12:24:13 30 JUN 2009
METHOD	0-100%BDE				C:\GOLD\SYSTEM2\	ANALYSIS	13:11:13 30 JUN 2009
REPORT DATA	SP130	A	1	1	0119 C:\GOLD\SYSTEM2\DATA\	REPORT	15:46:28 24 JUL 2009
METHOD	0-100%BDE				C:\GOLD\SYSTEM2\		

SAMPLE TABLE none  
SYSTEM 2: SYSTEM2

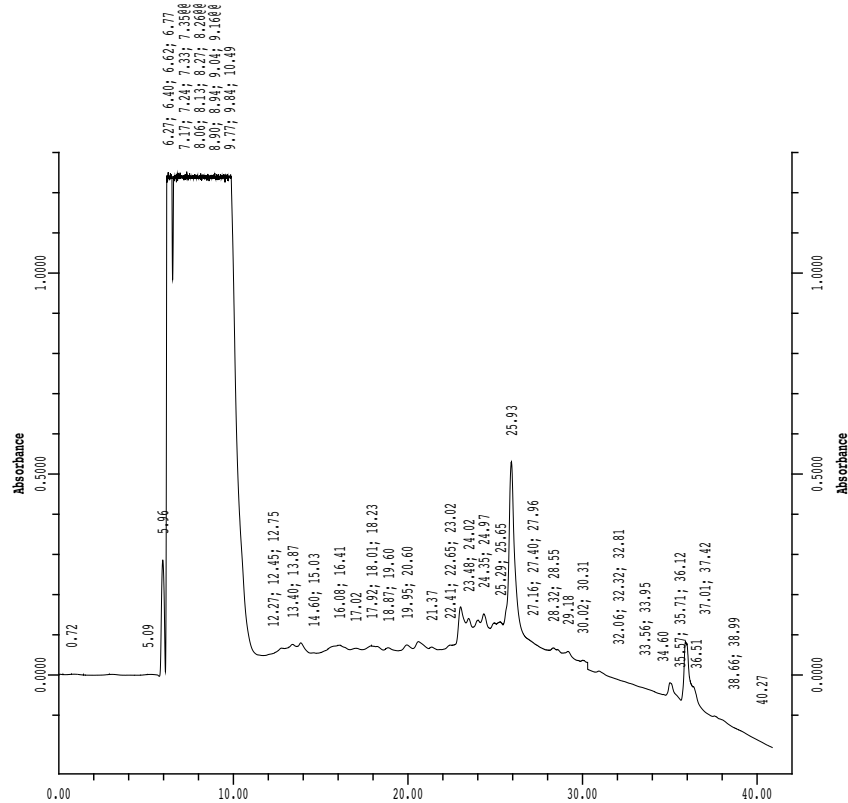


Figure 4.30: 250  $\mu$ L of  $\sim$ 1 mg/mL semi-crude (Ni-purified) expressed spitz construct in 50% H<sub>2</sub>O/ACN was loaded to a C3 semi-preparative HPLC column and subject to a 2 mL/min gradient that ramps from 0-100% ACN at 2.5%/min. The trace shown here is monitored at 210 nm, and encouragingly the major peak  $\sim$ 26 minutes was confirmed by ESI-MS to correspond to a species with average molecular weight consistent with the target for the construct (not shown).



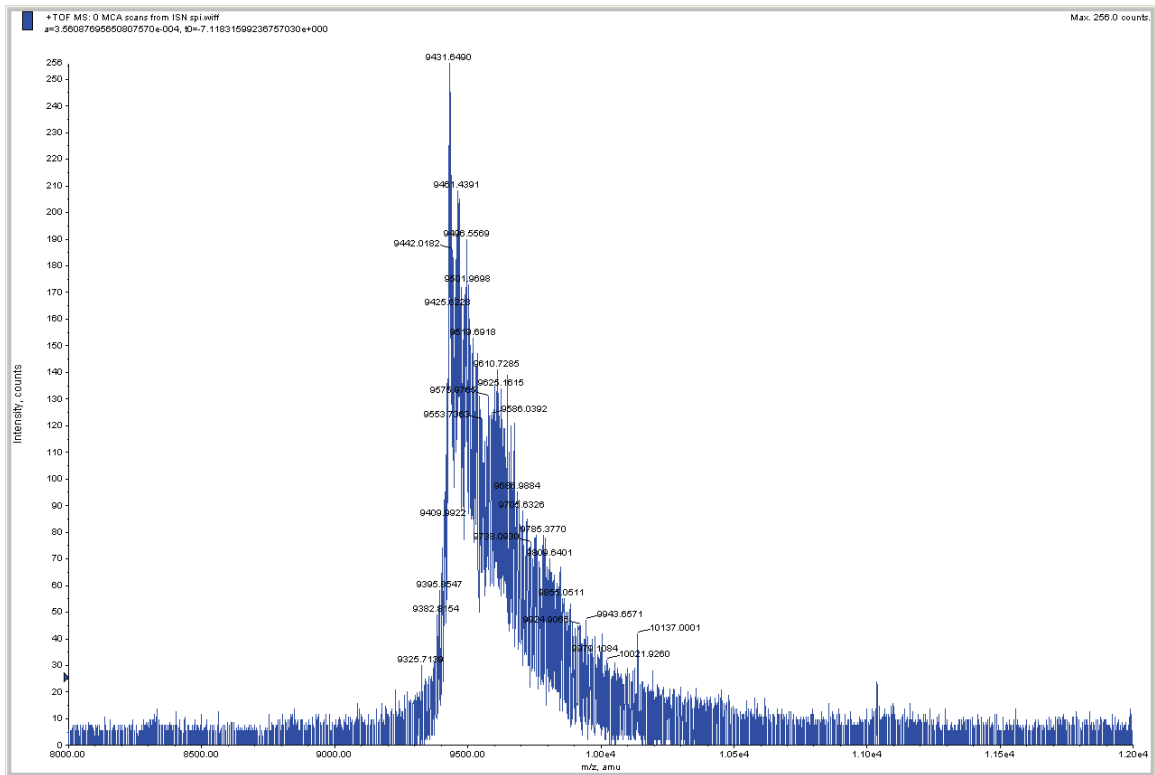


Figure 4.31: MALDI spectrum for the expressed spitz construct detailed in Figure 4.29 on page 99. The mass of the major peak is roughly consistent with a  $\sim 50\%$   $^{15}\text{N}$  incorporation. This is sensible as the bacteria were (accidentally) exposed to both isotopically enriched and unenriched nutrient sources.

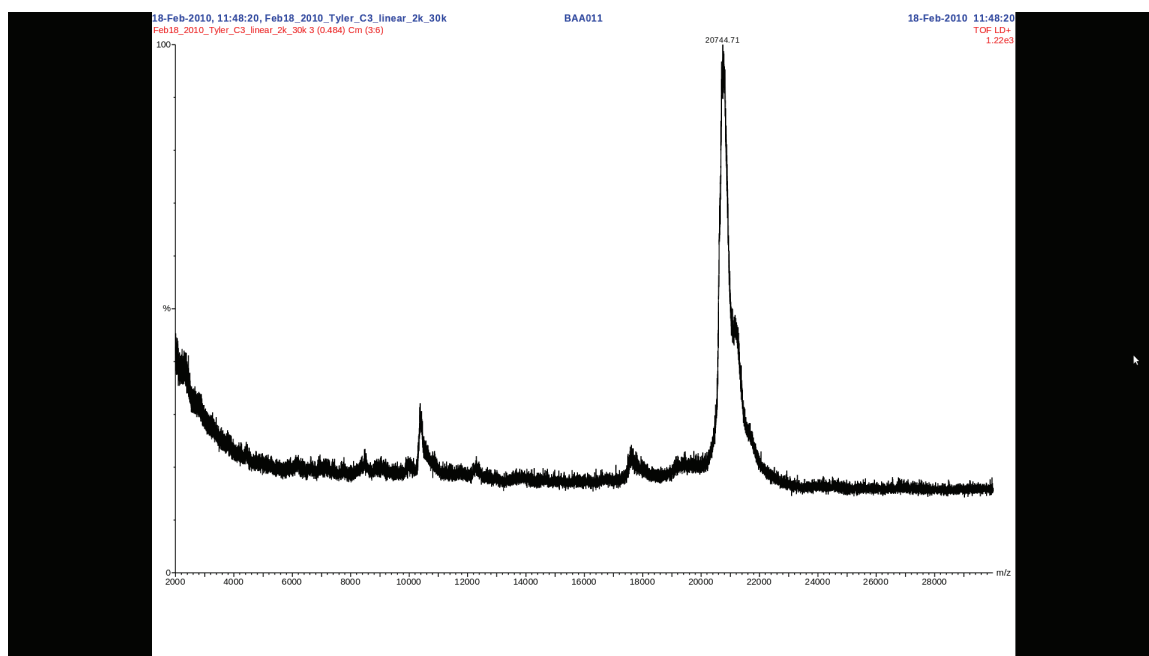


Figure 4.32: A crude expressed spitz construct dissolved in 50% H<sub>2</sub>O/ACN (0.1% TFA) was treated with immobilized TCEP, separated from the reducing agent by centrifugation, diluted two-fold with sinapic acid (matrix solution), and the resulting linear-mode MALDI spectrum is shown here. After accounting for the linear-mode calibration, a potential dimer peak ~20.6 kDa dwarfs a peak consistent with the monomer ~10.3 kDa.

Table 4.1: Summary of MALDI-MS results for the HPLC fractions collected during the TR-09-1 crude peptide run described in Figure 4.12 on page 83. Samples were diluted two-fold in  $\alpha$ -CHC matrix and the spectra were collected in reflectron mode.

Fraction#	HPLC window (minutes)	200 mM DTT	Peak ~2250 (m/z)
F2	6-12	yes	—
F3	12-18	yes	—
F4	18-24	yes	+
F5	24-30	yes	+
F6	30-36	yes	—
F2	6-12	no	—
F3	12-18	no	—
F4	18-24	no	+
F5	24-30	no	+
F6	30-36	no	—

Table 4.2: 1 minute fractions collected during the TR-09-1 crude HPLC run detailed in Figure 4.13 (page 84) were combined with  $\alpha$ -CHC matrix and MALDI spectra were collected without DTT treatment in reflectron mode. A '+' in the table indicates the presence of the desired product ( $\sim 2250$  m/z), which clearly exhibits a broad elution profile.

Fraction#	HPLC window (minutes)	Peak $\sim 2250$ (m/z) <sup>a</sup>
F1	18-19	–
F2	19-20	–
F3	20-21	– (2218 present)
F4	21-22	– (2217 present)
F5	22-23	++ (2250 major/2121 minor)
F6	23-24	+ (2122 major)
F7	24-25	+ (2122/2251 minor)
F8	25-26	+ (2121/2250 minor)
F9	26-27	+ (2122 present)
F10	27-28	+ (2121 present)
F11	28-29	+ (2122 present)
F12	29-30	+ (2122 present)

<sup>a</sup>while the focus is on peaks near the target mass, low molecular weight impurities do persist in many of these fractions

Table 4.3: A C3 HPLC column-purified fraction was further purified on a C18 column and 15 second fractions were collected in a region of interest on the trace detailed in Figure 4.15 on page 86. The latter fractions were diluted two-fold with  $\alpha$ -CHC matrix and MALDI spectra were collected in reflectron mode. This table summarizes the results using a '+' to indicate the presence of the desired spitz construct  $\sim 2250$  m/z. Fraction 5 represents the highest purity TR-09-1 sample obtained to date.

Fraction#	HPLC window (minutes)	Peak $\sim 2250$ (m/z) <sup>a</sup>
F1	24:16-24:31	–
F2	24:31-24:46	– (2089 present)
F3	24:46-25:01	– (2090 present)
F4	25:01-25:16	+ (1097>>2249>2121)
F5	25:16-25:31	++ (2249 major <sup>b</sup> >>1097>2121)
F6	25:31-25:46	+
F7	25:46-26:01	+ (2250 minor)
F8	26:01-26:16	+ (2122 / 2250 minor)

<sup>a</sup>while the focus is on peaks near the target mass, low molecular weight impurities do persist in many of these fractions

<sup>b</sup>Na<sup>+</sup> and K<sup>+</sup> adducts also present

Table 4.4: Crude TR-09-1 was directly loaded to a C18 semi-preparative HPLC column and fractions were collected as described in Figure 4.17 (page 88). The fractions were diluted two-fold with  $\alpha$ -CHC matrix and MALDI spectra were collected in reflectron mode. A ‘+’ is used to indicate the presence of the desired TR-09-1 peptide  $\sim$ 2250 m/z in this summary table.

Fraction#	HPLC window (minutes)	Peak $\sim$ 2250 (m/z)
F1	25:01-25:16	+ (1096 >> 2250)
F2	25:16-25:31	+ (1096 >> 2250)
F3	25:31-25:46	++ (2250 > 1096)
Control <sup>a</sup>	—	no major peaks

---

<sup>a</sup>50% H<sub>2</sub>O/ACN, 0.1% TFA combined with  $\alpha$ -CHC matrix

Table 4.5: TR-09-1 fractions were collected from the first replicate HPLC run detailed in Figure 4.21 on page 92, and diluted two-fold with  $\alpha$ -CHC matrix prior to collection of MALDI spectra in reflectron mode. A ‘+’ in the table reflects the presence of the desired product  $\sim$ 2250 m/z.

Fraction#	HPLC window (minutes)	Peak $\sim$ 2250 (m/z)
F1	25:10-25:25	—
F2	25:25-25:40	+ (minor)
F3	25:40-25:55	+ (minor)
F4	25:55-26:10	— (2218 present)
F5	26:10-26:25	—
F6	26:25-26:40	+ (minor)
F7	26:40-26:55	+ (minor)
R	recovery <sup>a</sup>	+

---

<sup>a</sup>solution contains remaining eluent in the range of 22-30 minutes

Table 4.6: TR-09-1 fractions were collected from the *second* replicate HPLC run detailed in Figure 4.21 on page 92, and diluted two-fold with  $\alpha$ -CHC matrix prior to collection of MALDI spectra in reflectron mode. A ‘+’ in the table reflects the presence of the desired product  $\sim 2250$  m/z.

Fraction#	HPLC window (minutes)	Peak $\sim 2250$ (m/z)
F1	26:55-27:25	+
F2	27:25-27:55	+
F3	27:55-28:25	+
F4	28:25-28:55	++ (2121 present)
F5	28:55-29:25	++ (2121 present)
F6	29:25-29:55	+ (2121 present)
F7	29:55-30:25	+ (2121 present)
F8	30:25-30:55	++ <i>excellent</i> (2121 present)
F9	30:55-31:25	++ <i>excellent</i> (2121 present)
F10	31:25-31:55	++ <i>excellent</i> (2121 present)
F11	31:55-32:25	– (low ion count)
F12	32:25-32:55	+ (low ion count) (2122 present)
F13	32:55-33:25	+ (low ion count)
F14	33:25-33:55	+ (low ion count)

Table 4.7: Fractions collected from a bulk purification of TR-09-1 (detailed in Figure 4.22 on page 93) were diluted two-fold with  $\alpha$ -CHC matrix and this table summarizes the reflectron-mode MALDI spectra. A ‘+’ in the table reflects the presence of the desired product  $\sim 2250$  m/z.

Fraction#	HPLC window (minutes)	Peak $\sim 2250$ (m/z) <sup>a</sup>
F1	16-18	+ (2250 major)
F2	18-20	+
F3	20-22	+
F4	22-24	+ (2249 minor)
F5	24-26	+
F6	26-28	+
F7	28-30	+
F8	30-32	+
F9	32-34	+
F10	34-36	–
F11	36-38	+ (minor)
F12	38-40	+ (minor)
F13	40-42	+

<sup>a</sup>most of the fractions had rather low ion counts

Table 4.8: Fractions collected from a bulk purification of TR-09-1 (detailed in Figure 4.23 on page 94) were diluted two-fold with  $\alpha$ -CHC matrix and this table summarizes the reflectron-mode MALDI spectra. A ‘+’ in the table reflects the presence of the desired product  $\sim 2250$  m/z.

Fraction#	HPLC window (minutes)	Peak $\sim 2250$ (m/z)
F1	6-8	–
F2	8-10	–
F3	10-12	+ (2249 minor)
F4	12-14	–
F5	14-16	–
F6	16-18	–
F7	18-20	+ (2249 minor)
F8	20-22	+ (2250 minor)
F9	22-24	+ <i>excellent</i>
F10	24-26	+
F11	26-28	+
F12	28-30	+ <sup>a</sup>
F13	30-32	+
F14	32-34	+ <sup>b</sup>
F15	34-36	+
F16	36-38	+
F17	38-40	+
F18	40-42	+ (2250 major)
F19	42-44	+
F20	44-46	+ <i>excellent</i>
F21	46-48	+ <i>excellent</i>
F22	48-50	+
F23	50-52	+

---

<sup>a</sup>substantial variety of peaks in  $\sim 2000$  m/z range

<sup>b</sup>substantial variety of peaks in  $\sim 2000$  m/z range

Table 4.9: TR-10-2 fractions collected from the HPLC run detailed in Figure 4.27 (page 98) were diluted two-fold with  $\alpha$ -CHC matrix and this table summarizes the collected reflectron-mode MALDI results. A ‘+’ symbol is used to indicate the presence of the target product  $\sim 1691$  m/z (or one of its salt adducts).

Fraction#	HPLC window (minutes)	Peak $\sim 1691$ (m/z)
1	6-12	–
2	12-18	–
3	18-24	–
4	24-30	–
5	30-36	+
6	36-42	+
7	42-48	–
8	48-54	–
9	54-60	–
10	60-66	–

# Chapter 5

## FGFR3 Simulation

### 5.1 Fibroblast Growth Factor Receptors

Fibroblast growth factor receptors (FGFRs) are a family of four FGF-activated receptor tyrosine kinase (RTK) transmembrane (TM) glycoproteins (155, 156, 157). All FGFRs have three extracellular immunoglobulin (Ig)-like domains, and alternative splicing of the proximal Ig-like domain in FGFRs 1-3 produces receptor isoforms with different ligand-binding specificities (158, 159, 160). The signaling complexity of the FGFR family is further compounded by the existence of at least 19 different FGFs (158, 161). Ligand-binding at the proximal Ig-like domain likely stabilizes the active dimer, with rearrangements in the TM and cytoplasmic tyrosine kinase domains driving autophosphorylation and signal transduction. FGFRs are of medical interest because many skeletal dysplasias and cancers are associated with mutations in FGFRs 1-3.

### 5.2 Achondroplasia: Specific Relevance Of FGFR3

Achondroplasia, the most common form of human dwarfism, was mapped to the short arm of chromosome 4 (162), and shortly thereafter a G380R mutation resulting from a transition or transversion in the FGFR3 gene was specifically identified as the major cause of the phenotype (163, 164). Indeed, G380R (in the FGFR3 TM



domain) is the underlying cause of achondroplasia in  $\sim 99\%$  of cases (163, 164, 165). Achondroplasia is autosomal dominant with complete penetrance (166, 167), and in 80-90% of cases results from a spontaneous paternal germ line mutation (there is a correlation with increased paternal age) (168, 169, 170). While intelligence is normal in individuals with achondroplasia, there is increased mortality in the first four years of life and in the late fourth to fifth decades of life (171).

There is experimental evidence for increased signaling by the mutant FGFR3 with resultant negative regulation of bone growth (172). However, free energies of dimerization measured by fluorescence resonance energy transfer (FRET) in 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine (POPC) for wild-type and G380R TMD peptides match within the bounds of experimental uncertainty (173). Neutron diffraction and oriented circular dichroism spectroscopy indicate a 5 Å shift of the mutant peptide away from a POPC bilayer center (11). There appears to be a conceptual disconnect between the latter perturbation in FGFR3 mutant membrane topology relative to WT and the similar dimerization propensity to dimerize. Because of the medical relevance, there is a substantial motivation to gain insight into the influence of the G380R mutation on the dimerization properties of FGFR3.

In this chapter I describe and analyze coarse-grained molecular dynamics (CG-MD) simulations of the FGFR3 WT and G380R mutant TMDs in lipid bilayers. The analysis involves simulations for the WT homodimer, the heterodimer (because achondroplasia is autosomal dominant), and the mutant homodimer (section 5.4 on page 111). In section 5.5 (page 126) I focus specifically on the properties of the lipid bilayer proximal and distal to the peptide TMDs. Monomer simulations were also performed to control for the behaviour of the individual peptide constructs, and a preliminary analysis of these results is presented in section 5.6 (page 134).

## 5.3 Coarse-Grained Simulations And Glycophorin A Control

Ideal  $\alpha$ -helical atomistic starting structures for FGFR3 constructs were built in PyMOL (174) based on the primary sequences of published experimental constructs (11) and the amino acids were coarse-grained as described previously for lipids (175). An approximate 4:1 mapping of heavy atoms (*i.e.*, non-hydrogen atoms) to CG particles is performed to produce several specific particle types: polar (P), mixed polar/apolar (N), hydrophobic apolar (C), and charged (Q). A few additional particle subtypes allow fine tuning of Lennard-Jones potentials to reflect hydrogen-bonding propensities. Details of the amino acid to CG particle mapping process have been described elsewhere (176, 177). Lipid and water molecules were parametrized as described previously (175), and all simulations were performed using GROMACS (178). The MARTINI force field (179, 180) was employed for all simulations except for the FGFR3 monomer simulations (section 5.6.2 on page 135) which employ the Bond force field (176). Lennard-Jones interactions were shifted to zero between 9 and 12 Å, and electrostatics were shifted to zero between 0 and 12 Å. All simulations were performed at constant temperature, pressure, and number of particles. The temperature coupling of system components was performed independently for each component using the Berendsen algorithm at 323 K (181). The system pressure was semiisotropically coupled in the x, y, and z directions using the Berendsen algorithm. The time step for integration was 40 fs. VMD (182) was used for visualization and MDAnalysis (183) for parsing of trajectories. Parallel simulations and analyses were performed for glycophorin A (GpA) as described for previous CG-MD simulations (184). GpA is a well-characterized single-pass TMD protein which serves as a control for FGFR3—a protein for which there is no high-resolution structure available.

It should be noted that there are, of course, limitations to CG-MD simulations. The coarse-grained nature of the system provides improved performance at the cost

of reduced accuracy and reduced similarity to the true biological system. Molecules diffuse at least four times faster than they would in a more realistic atomistic context, the involvement of water in many processes is often not well characterized because of the much larger size of CG-MD water particles, there are no hydrogen atoms or hydrogen bonds (only crude approximations and restraints), and the screening of electrostatic interactions is only approximated and frequently cut-off in a crude fashion.

## **5.4 Analysis Of FGFR3 And GpA *Dimer* Trajectories**

### **5.4.1 Tracking The Distance Between Helices In A Trajectory**

The initial separation between glycoporphin A (GpA) and FGFR3 helices in the dimer simulations was  $\sim 55$  Å, the same value used in previous GpA coarse-grained (CG) molecular dynamics (MD) simulations (184). Tracking the separation between helices in the POPC bilayer is useful for testing a number of properties. Perhaps most obvious is the assessment of the rate of dimer formation—if the separation between helices decreases and reaches the final dimer interhelix separation rapidly then dimer formation occurs quickly for a given construct. The stability of the dimer can also be assessed because increased interhelix separation after initial dimer formation may indicate that the dimer has dissociated, and frequent association and dissociation events may indicate a weak dimerization affinity. Performing many replicate simulations (see Table 5.1 on page 189) may also allow for the determination of whether the rates of dimerization and the dimer stability are consistent for a given dimer construct or vary stochastically between simulations.

My specific strategy was to track the closest interhelix approach using only  $C_\alpha$  particles from each of the peptides. In each frame of a given simulation, a dis-

tance matrix for all possible interhelical  $C_\alpha$  combinations was calculated with the minimum value defined as the closest approach distance. Although I coded a few MDAnalysis-dependent (183) python functions for these calculations, the most efficient function is `closest_contacts_efficient()` from the python module/library `dimer_geometric_tools.py` D.17 on page 356.

The closest interhelix approach is monitored during the first replicate trajectory of the GpA wild-type (WT) homodimer control condition in Figure 5.1 on page 138. Dimerization occurs very rapidly and the dimer is stable once formed. In contrast, dimerization takes  $> 1 \mu\text{s}$  in the FGFR3 WT replicate 3 simulation (Figure 5.2 on page 139), although the dimer is still stable once formed. The ninth FGFR3 heterodimer replicate simulation requires almost  $4 \mu\text{s}$  for dimerization (Figure 5.3 on page 139), but the dimer is again stable once formed. The second FGFR3 mutant homodimer replicate exhibits extremely fast dimerization and the dimer remains stable for the duration of the simulation (Figure 5.4 on page 140).

The outlined results may lead the reader to believe that the FGFR3 heterodimer forms more slowly than the WT or the mutant homodimer. However, it is important to consider the closest approach distance results across all of the replicate simulations (Figure 5.5 on page 141). It is clear from the latter plot that the variation in dimerization rates *within* any of the FGFR3 constructs is considerable. Thus, there appears to be a stochastic component to the rate of dimerization with no particular trend for FGFR3 WT, heterodimer or mutant homodimer conditions. However, in all cases, GpA and FGFR3 dimers are stable once formed in the self-assembled CG POPC bilayers used in these simulations since no dissociations were observed in 34 replicate simulations. The varied dimerization rates are consistent with the previously reported time required for WT GpA helix association in DPPC—varying from 0.5 to  $3 \mu\text{s}$  (with a  $60 \text{ \AA}$  initial separation) (185). Note that the fifth GpA replicate simulation does not exhibit dimer dissociation—one of the helices adopted an orienta-

tion perpendicular to the membrane normal (and interacted with the other TM helix) before finally returning to a transmembrane orientation and forming a stable dimer.

### 5.4.2 Relative Helical Motion

All GpA and FGFR3 dimer constructs are stable once formed, but there is a stochastic component to the rate of dimerization. What are the helices doing during the variable simulation time prior to dimerization? One way to simplify the problem is to look at the motion of one helix in the fixed reference frame of the other. Specifically, in each frame of a (GpA or FGFR3) dimer simulation the coordinates of the geometric center of the first helix were subtracted so that its position was always at the origin. For consistency, the same subtraction per frame was applied to the coordinates of the geometric center of the second helix. Tracking the motion of the second helix in this fashion has been done previously for GpA dimer simulations in DPPC (184). The authors highlight one caveat—the dimerization interface will not be specific because the reference helix is free to rotate about its axis at the origin (as long as the geometric center is fixed).

The MDAnalysis-based `relative_helical_motion()` python function I wrote in the `dimer_geometric_tools` module D.17 (on page 356) performs the described adjustment of the trajectory reference frame and printing of new helix 2 coordinates. In the case of the first GpA control replicate, helix 2 follows a relatively straightforward path from its starting position to dimerization with helix 1 (Figure 5.6 on page 142). The case for FGFR3 wild-type replicate 7 is more convoluted, with helix 2 moving through several periodic boundaries before dimerizing with a mirror image of the reference helix (Figure 5.7 on page 142). The ninth heterodimer FGFR3 replicate requires nearly 4  $\mu$ s for dimerization (Figure 5.3 on page 139) and accordingly helix 2 moves through a large distance, including many periodic boundaries, before dimerization with a mirror image of helix 1 (Figure 5.8 on page 143). The rapid dimerization of the second FGFR3 mutant homodimer replicate (Figure 5.4 on page 140) is con-

sistent with the straightforward path followed by helix 2 for dimerization with helix 1 in this simulation (Figure 5.9 on page 143). Thus, the variety of dimerization rates is reflected in the relative ‘wandering’ motions of the helices.

### 5.4.3 Helix Crossing Angle

Sections 5.4.1 (page 111) and 5.4.2 (page 113) demonstrate a wide variety of helix dimerization rates and wandering distances before dimerization, but a stable dimer once it has formed. The observations are consistent with an unbiased simulation setup. Had the helices been placed too close together at the start of the simulation, the dimerization rates would likely have been more consistent with no opportunity for relative wandering motion. Given the formation of stable and unbiased dimers we now focus on the details of the dimer proper.

It has been well established that GpA dimers exhibit a strong preference for right-handed helix crossing angles from experimental high-resolution structures (186, 187) and computational studies in DPPC (184). The first GpA replicate simulation in POPC is consistent with this behaviour (Figure 5.10 on page 144). There is currently no high-resolution experimental structure of the FGFR3 TM domain (we employ an ideal  $\alpha$ -helix), so it is important to establish that the dimer configuration for the GpA control is consistent with previous experimental and computational findings. Unlike GpA, the first FGFR3 WT homodimer replicate exhibits dimerization with a bimodal helix crossing angle distribution, and perhaps a slight preference for a left-handed crossing angle (Figure 5.11 on page 145). The ninth replicate FGFR3 heterodimer simulation appears to exhibit a preference for a right-handed helix crossing angle (Figure 5.12 on page 146), but there is a small number of sampled dimer frames because this trajectory required so long for dimerization (Figure 5.3 on page 139). The first replicate of the FGFR3 mutant homodimer construct exhibits a bimodal helix crossing angle distribution (Figure 5.13 on page 147) similar to that observed for the wild-type replicate above.

While there is value in analyzing replicate simulations on an individual basis, it should also be clear that it is important to perform many replicates and study the merged results to establish overall trends. In particular, the helix crossing angle results merged over all the replicate trajectories indicate that all FGFR3 constructs (including heterodimer) exhibit bimodal helix crossing angle distributions, in contrast to GpA which exhibits an overall preference for a right-handed helix crossing angle (Figure 5.14 on page 148). The bimodal helix crossing angle distribution for FGFR3 constructs might be explained by alternation between configurations as the helices oscillate closer and farther apart in the dimer. However, helix crossing angles were independent of the closest interhelical approach between  $C_\alpha$  particles for all FGFR3 constructs (Figure 5.15 on page 149).

#### 5.4.4 Correlated Helical Motion

There is stochastic wandering prior to dimerization in the GpA and FGFR3 replicate trajectories, and the dimers are stable once formed, but the helix crossing angle distribution for all FGFR3 constructs is curiously bimodal. It is not clear how the FGFR3 helices move relative to one another in the context of the dimer since both left- and right-handed crossing angles are sampled in a given trajectory. However, since the dimers are stable we would still expect them to move together in the POPC bilayer. To test for coordinated helical motion, I first tracked the Z coordinate (along the bilayer normal) of the geometric center of the  $C_\alpha$  particles of each helix relative to the center of the bilayer. The center of the bilayer was defined as the average of the center of mass coordinates of the two leaflet phosphate CG particle populations, and the MDAnalysis-dependent python function I wrote for this calculation (`geo_Z_tracking_relative_to_bilayer()`) is stored in the python library `dimer_geometric_tools.py` D.17 on page 356. An example of the results of this analysis for the first GpA replicate simulation are shown in Figure 5.16 on page 150. While there is certainly evidence for coordinated excursions of the helical geometric

centers away from the center of the bilayer, it is rather cumbersome to quantify the degree of coordinated motion by inspection.

To quantify the degree of coordinated helical motion in GpA and FGFR3 dimer simulations, I calculated the absolute correlation coefficient ( $|R|$ ) between the Z coordinates of the geometric centers of the helices before and after dimerization. The python `split_Z_file()` function in the `dimer_geometric_tools.py` library (D.17 on page 356) splits the helix geometric center Z coordinates into pre- and post-dimerization files and produces the corresponding linear correlation coefficients. The results for individual replicates are summarized in Figure 5.17 on page 151, and for merged data that includes all replicates per condition in Figure 5.18 on page 152. For all constructs, as expected, helical motion is more strongly correlated following dimerization. The larger correlations observed for GpA dimers relative to FGFR3 constructs is consistent with previous results indicating that GpA forms much stronger dimers than FGFR3 (188).

#### 5.4.5 Identification Of Predominant Interhelical Contacts

The unbiased GpA and FGFR3 simulations exhibit formation of stable dimers which move together in the membrane. In addition, FGFR3 has no apparent preference for left- or right-handed helix crossing angles. However, these are fairly broad properties, and I would now like to focus on more specific aspects of the dimer configuration. Which residues feature most prominently at the dimer interface for FGFR3? Without a high-resolution structure of the FGFR3 TM domain available, the identification of important dimer interface residues via simulation may provide insight into the molecular mechanism of pathology in achondroplasia and other skeletal dysplasias.

I devised a simple method to parse a CG-MD trajectory for the predominant interhelical dimer contacts. For each frame of a simulation (if the helices are within 7 Å), a distance matrix of all possible interhelical  $C_\alpha$ - $C_\alpha$  combinations is calcu-



lated. The five smallest distances for each frame are recorded along with the corresponding residue identifiers. The probability for each residue to appear in the five closest dimer contacts is then calculated based on the total number of these contacts in which the residue is found divided by the total number of contacts (which is  $5 \times$  the number of dimer frames in the simulation). Separate close contact probabilities are calculated for matching residues in each helix in order to assess symmetry. I have written MDAnalysis-dependent python functions to calculate the residue close contact probabilities for each helix in: a single GpA trajectory (`top_five_closest_residues_GpA()`), a single FGFR3 trajectory (`top_five_closest_residues_FGFR3()`), and aggregate results for each of the FGFR3 constructs across all replicate trajectories (`merged_top_five_FGFR3()`). The functions are all stored in the library module D.17 (`dimer_geometric_tools.py`) on page 356. In the case of aggregate probabilities, the head script (`analyze_FGFR3_dimer_simulations.py`) D.15 on page 338 must also call a second library function (`parse_overall_FGFR3_top_five_data()`) to parse the larger set of data.

The results of the residue close contact probability analysis for the first replicate GpA simulation are plotted in Figure 5.19 on page 152. The predominant residues are representative of the other GpA replicates with G79, G83, and T87 among the most likely residues in the closest interhelical contacts. Other candidates (L75, I91) are spaced at 4 residue intervals, which is consistent with the identification of a helical face. G79, G83, and T87 have previously been identified as experimentally crucial interfacial contacts (189). The analysis procedure I have employed therefore produces computational predictions of dimer interface residues that are consistent with experiment for GpA, and this provides confidence for the application of this method to the FGFR3 TM domain.

Three residues stand out from the equivalent aggregate analysis over all trajectories for each of the three FGFR3 conditions—G370, A374, R397 (Figure 5.20 on

page 153). Other prominent close contact residues are spaced at 3-4 residue intervals, which is consistent with the identification of a specific helical face of FGFR3. It is noteworthy that G380/R380 (the site of the achondroplasia mutation) does not feature prominently as a close contact residue, nor does A391 (mutation to E results in Crouzon syndrome) (190). Does it make sense that residues mutated in disease are not located directly at the dimer interface? Strikingly, G370 (one of the most prominent contacts) is mutated to C in type 1 thanatophoric dysplasia, which has a more severe phenotype than achondroplasia and is normally neonatal lethal (191). It is possible that the severity of the phenotype correlates with the proximity of the mutated residue to the dimer interface. Furthermore, G370 was recently localized to the FGFR3 dimer interface by site-specific infrared dichroism (192). The propensity for disulfide formation in FGFR3 follows the trend Cys370 > Cys371 > Cys375 (193), which is also consistent with an interfacial position for residue 370. R397, another of the prominent interfacial contacts, is part of the C-terminal CRLR tetrapeptide, which can be removed to increase the dimerization affinity of FGFR3 to match that of GpA (192).

#### 5.4.6 Identification Of Dimer Interfaces

The experimental and clinical support for the proposed FGFR3 dimer interface residues is encouraging. Despite identification of a small set of candidate interfacial residues in the FGFR3 dimers, it is not clear if the broad distribution of allowable helix crossing angles (section 5.4.3 on page 114) results from sampling of different dimer interfaces. To test the possibility that more than one dimer interface can be sampled in the FGFR3 dimer, I employed a simplified reference frame. For all three FGFR3 conditions, a reference structure corresponding to the first helix in the first frame of the first WT FGFR3 simulation was used. The first helix in all frames of all FGFR3 replicate simulations was rmsd-fixed to match the configuration of that reference structure. Applying the same coordinate transformation to the second helix

allows for the assessment of the relative position of helix 2 while helix 1 is fixed in a single configuration. The geometric center of helix 2 was tracked in this adjusted coordinate system to test for the presence of multiple dimer interfaces. This is similar to a strategy previously employed to study the GpA dimer interface (184). I have written python MDAnalysis-dependent functions for calculating the positional probability of helix 2 in the rmsd-fixed reference frame of helix 1 for individual GpA and FGFR3 trajectories (`fixed_helix_thermal()`) and for results merged across all replicates for a given FGFR3 condition (`fixed_helix_thermal_merged()`). While the former function contains its own data-binning routine for outputting the helix 2 positional probabilities, the latter function is called prior to a separate binning routine (`thermal_bins()`). All three functions are stored in the python library `dimer_geometric_tools.py` D.17 on page 356, and controlled from the head script `analyze_FGFR3_dimer_simulations.py` D.15 on page 338.

The result for the first control replicate GpA trajectory (representative of the other GpA replicates) is shown in Figure 5.21 on page 154. The positional probability of helix 2 is consistent with a primary dimer interface at the ‘bottom right’ of helix 1 and a secondary interface at the ‘top right.’ Since ten replicate simulations were performed for each of the FGFR3 dimer conditions (Table 5.1 on page 189), the aggregate helix 2 positional probability results across all replicates were calculated for each condition (Figure 5.22 on page 155). All three FGFR3 constructs have a primary dimer interface at the ‘bottom left’ of helix 1. While the WT has no major dimer interaction at the ‘bottom right’ of helix 1, there is a progressively greater likelihood of helix 2 interacting with helix 1 at the ‘bottom right’ when looking at the WT→ heterodimer→ mutant homodimer results. Thus, a secondary dimer interface progressively appears in the FGFR3 heterodimer and mutant homodimer simulation conditions.

### 5.4.7 Dimer Interface Transitions

There is evidence for primary and secondary dimer interfaces for FGFR3 (and GpA) dimers (section 5.4.6 on page 118). It is not immediately clear how the helices transition between the interfaces. One possibility is a continuous sampling of the two interfaces with a bias toward the primary interface. However, a less frequent interface transition scheme is also possible—there may be discrete periods of the simulation at each of the interfaces with a single or a few transition points. To distinguish between these transition schemes I employed a simplified reference frame with rmsd-fixing of helix 1, as outlined in section 5.4.6 (page 118). However, instead of calculating the positional probability of helix 2, I simply tracked its geometric center during the trajectory by using a third plotting dimension—the frame (simulation time). The MDAnalysis-dependent python function `frame_abstracted_relative_position()` in the `dimer_geometric_tools.py` module D.17 (page 356) was used to parse individual GpA and FGFR3 trajectories in the described fashion.

An example of the result of this analysis on a representative GpA simulation trajectory is shown in Figure 5.23 on page 156. The stable helix 2 position at the ‘bottom right’ of helix 1 is apparent, with only one substantial excursion to the secondary dimer interface late in the simulation. The excursion is clearly discrete—there is a single point of exit from the primary interface and a single point of return to the primary interface. The other GpA replicates exhibit similar discrete (rather than continuous sampling) transitions between the primary and secondary dimer interfaces.

The FGFR3 mutant homodimer replicate 4 simulation is an interesting target for this analysis because there is a roughly equivalent positional probability for helix 2 at the primary and secondary dimer interfaces (Figure 5.24 on page 157). Tracking the geometric center of helix 2, it is apparent that helix 2 starts the simulation at the secondary interface (‘bottom right’ of helix 1), and exhibits a very large amplitude of motion at this interface (Figure 5.25 on page 158). However, roughly half way

through the trajectory there is a transition to the primary interface (‘bottom left’ of helix 1) where the position of helix 2 is much more stable. This is a representative result of discrete interface transitions in the mutant homodimer FGFR3 condition.

#### 5.4.8 Identification Of Representative Dimer Interface Structures And Contacts

The FGFR3 mutant homodimer (and to some extent the heterodimer) has both primary and secondary dimer interfaces which are occupied for discrete periods of the simulations (section 5.4.7 on page 120). It would be informative to pull out a representative structure for each of the interfaces or to determine which interfacial residue contacts predominate each of the configurations. In principle, it should be possible to extract structures from simulation frames chosen manually from the plots in section 5.4.7. However, the 3D perspective view of these plots (*i.e.*, Figure 5.25 on page 158) makes it cumbersome to extract appropriate frames to represent a particular interface.

One way to simplify the plots is to remove one of the dimensions. It is crucial to track the simulation time or frame number because we need to know the exact frame that corresponds to a coordinate (interface position) for helix 2 in the reference frame of rmsd-fixed helix 1. However, instead of tracking both x and y coordinate positions it is possible to convert the Cartesian coordinates to a single polar angle  $\theta$  (in the two-dimensional polar coordinate system).

An example plot of  $\theta$  and  $\Omega$  (helix crossing angle) tracked as a function of simulation frame number is shown in Figure 5.26 on page 159 for the first replicate GpA simulation. The excursions of helix 2 to the secondary dimer interface are discrete but brief for GpA, and there is apparently no substantial change in  $\Omega$  for GpA dimer interface transitions. This is not surprising given the strong preference of GpA dimers for right-handed helix crossing angles (section 5.4.3 on page 114). In contrast, there appears to be a change in  $\Omega$  as the FGFR3 dimer transitions from the secondary

to the primary interface in the fourth replicate mutant homodimer simulation (Figure 5.27 on page 160). Initially, at the secondary interface,  $\Omega$  is primarily negative (right-handed) and then transitions to a bimodal distribution about 0 as the dimer assumes the primary configuration. It is also notable that there is a much larger amplitude of motion for FGFR3 helix 2 at the secondary interface when compared with its relatively stable position at the primary interface, consistent with observations in section 5.4.7 (page 120).

The demonstrated procedure for selection of representative dimer interface structures (simulation frames) from the helix 2 polar angle in the rmsd-fixed frame of helix 1 is ‘manual,’ and may not be the ideal way to address the selection of a representative structure across the population of all simulation replicate frames. To address the quality of the manually selected representative frames I have highlighted them in the context of a direct correlation between helix 2 polar  $\theta$  and  $\Omega$  for all frames in a given FGFR3 simulation (Figure 5.28 on page 160). The calculations were performed using the `correlate_helixcrossing_polar_theta()` function in the `dimer_geometric_tools.py` module D.17 (page 356). Even in the simple case of the FGFR3 WT there is an obvious complication with the manual selection procedure—the selection of a single representative structure will always, by chance, pull out a left- or right-handed helix crossing configuration even though both are possible at the WT (primary) interface. The same problem crops up for selection of the representative primary interface in the FGFR3 heterodimer and mutant homodimer, and is potentiated by the comparison with the secondary interface in these conditions. Thus, while it is apparent that the FGFR3 secondary dimer interface represents mostly right-handed helix crossing configurations, the bimodal helix crossing angle distribution at the primary interface means that comparison of the representative primary and secondary interfaces would depend on the stochastic selection of a left- or right-handed dimer from the primary interface.

With the above limitations in mind, the FGFR3 dimer structures corresponding to the representative frames highlighted in Figure 5.28 (page 160) are shown in Figure 5.29 on page 161. By inspection, the FGFR3 secondary dimer interfaces are similar when compared between heterodimer and mutant constructs while there appear to be more substantial differences between representative primary interface structures. This is consistent with the wider variety of helix crossing angles at the primary interface and the stochastic component to manual structure selection described above.

The closest contacts by residue for each helix in the representative constructs were calculated using the `closest_approach_representative()` function in the `dimer_geometric_tools.py` module (D.17 on page 356). Plots of the reciprocal (for visualization purposes) closest contact distances for each residue in the constructs are shown in Figure 5.30 on page 162. The WT ‘secondary’ interface refers to a ‘top left’ position for helix 2 in Figure 5.22 on page 155, and is distinct from the secondary dimer interface which develops in the heterodimer and mutant. Nonetheless, there are some noticeable differences in the WT contacts between the two interfaces and R397 in particular has a much closer contact in the primary interface. The heterodimer interfaces differ strikingly at the N-terminus (including G370 and A374) while R397 and other C-terminal residue contacts are more similar. Less striking differences are observed between the interface contacts for the mutant homodimer, consistent with selection of representative constructs that have the same helix crossing angle (compare heterodimer and mutant selections in Figure 5.28 on page 160).

#### **5.4.9 Population-Based Dimer Interface Classification**

The previous section (5.4.8 on page 121) highlights the limitations with selection of single representative dimer interface structures for FGFR3. The next step is to use a clustering or population-based approach to classify groups of frames falling into representative categories to avoid the stochastic problems with selection of single

representative frames. GROMACS tools has a built-in clustering utility, `g_cluster`, which served as the starting point for this analysis. I used the `g_cluster` single linkage and gromos methods for generating the major clusters in each of the replicate simulations. Single linkage will add a frame to a cluster as long as it is within a specified rmsd-cutoff of *any* frame within that cluster, while the gromos algorithm is more sophisticated, using a neighbour clustering cut-off technique as described in (194). Plots correlating the % of trajectory structures in the major cluster with the rmsd of the major cluster are shown for the single linkage (Figure 5.31 on page 163) and gromos (Figure 5.32 on page 164) algorithms, as well as a direct comparison of the two (Figure 5.33 on page 165), using a 0.4 nm rmsd cutoff. The gromos algorithm is clearly more stringent—producing major clusters with lower rmsd values and therefore lower % frame incorporation into the major cluster. It is also notable that GpA trajectories consistently have less variation (lower rmsd), despite a similar or higher % incorporation, within their major clusters compared with FGFR3. This is consistent with the formation of a stronger dimer by GpA (188).

While the above GROMACS-based clustering methods produce sensible differences between GpA and FGFR3 trajectories, it would be preferable to incorporate the polar angle ( $\theta$  of helix 2 in the reference frame of rmsd-fixed helix 1) into the population-based clustering method because  $\theta$  (a change in position of helix 2) is the basis of the dimer interface classification. The basic idea is to select a population of frames within a certain set of  $\theta$  bounds rather than a single frame within the interface boundary and then analyze the members of each interface population as a whole. One of the challenges for automated classification of simulation frames within populations of interfaces are the large-amplitude and short-lived excursions of FGFR3 away from the secondary interface and occasionally the primary interface. I do not want every brief excursion back and forth between primary and secondary interfaces to be treated as a stable transition from one interface to the other. Instead, I smoothed



the data for each FGFR3 replicate trajectory using a weighted moving average and a ‘spike-filter.’ Specifically, the `simple_moving_average_polar_theta()` function in the `dimer_geometric_tools.py` module D.17 (page 356) performs a few analysis steps: 1) For each replicate simulation, select only frames/ $\theta$  values that correspond to dimer configurations (frames after helix-helix closest  $C_\alpha < 6\text{\AA}$ ). 2) Subject each data point to a ‘spike filter:’ if the current  $\theta$  differs by more than 1.0 radians from the average  $\theta$  values of *both* the 20 preceding and 20 following frames, its instantaneous  $\theta$  is replaced by the average of the previous 10 data points. It is important to check *both* preceding and following  $\theta$  trends because if  $\theta$  only deviates substantially on one side of the current data point, this may reflect a stable transition between interfaces that is not an intended target for attenuation of a sudden change in  $\theta$ . Note that using a large window for the weighted average instead of this kind of spike filter is also not desirable because it will produce a ‘lag’ in the data with the large window tail. 3) Use the standard python `numpy.convolve()` function to calculate a linear weighted moving average with a window size of 10 on the current spike-filtered data set. The simplification of the automated classification of frames to representative dimer interfaces as a result of the smoothed  $\theta$  values is apparent with the attenuated fluctuations in the new data, as highlighted in Figure 5.34 on page 165.

Finally, the resulting data is parsed for the predominant interfacial residue contacts (as described in section 5.4.5 on page 116) at each of the interfaces, which are defined based on classification of the ‘smoothed’ polar  $\theta$  values:

Primary interface (rad):	$-3.0 < \theta < -1.5$
Secondary interface (rad):	$-1.0 < \theta < 1.0$
Other interfaces (rad):	remaining $\theta$

The specific filtering and analysis implementation is available in the `interface-`

`_filtered_merged_top_five_FGFR3` and `interface_filtered_parse_overall_FGFR3_top_five_data()` functions in the `dimer_geometric_tools.py` module D.17 (page 356). The predominant contacts, across all 30 FGFR3 dimer simulations (WT, heterodimer and mutant), for each of the FGFR3 dimer interfaces are shown in Figure 5.35 on page 166. There is a striking loss of contact symmetry at G370 (and to some extent R397) at the secondary interface compared with the primary interface. The other interface(s) exhibit a broad contact profile, which may simply reflect the fact that more than one discrete alternative interface exists outside the primary and secondary. Since G370 and R397 are candidates as important dimer contacts (section 5.4.5 on page 116), the present evidence is consistent with changes at critical dimer interface residues between primary and secondary interfaces and the existence of distinct configurations at each interface.

## 5.5 Analysis Of Lipid Bilayer In GpA And FGFR3 Simulations

The previous section (5.4 on page 111) concerns the analysis of the peptide components of the GpA and FGFR3 dimer trajectories. However, the peptides are not simulated in a vacuum and I will now focus on the analysis of the POPC lipid bilayer surrounding the TM segments in these simulations. One of the challenges with lipid bilayer analysis is the categorization of individual phospholipids to a particular leaflet. MDAnalysis propagates a network of connections between selected particles (*i.e.*, phosphates in lipid headgroups) using a cut-off distance which determines the leaflet groupings. This may cause problems if any lipid in one leaflet approaches a lipid in the other by the cut-off distance because the network of connections could then propagate through both leaflets and their separate definitions would be lost (Figure 5.36 on page 167). To simplify the tracking of lipid leaflets, I defined the leaflets in the first frame of each simulation and assigned each phosphate particle permanently

to a specific leaflet. In this manner, if a phospholipid moves closer to the center of the bilayer it will not abrogate the leaflet definitions because each phosphate is assigned permanently to one leaflet from the first frame (there is no propagation step in each frame).

### 5.5.1 Testing For Phospholipid ‘Flip-Flop’

The method for leaflet selection outlined above handles bilayer pinching/narrowing more gracefully than a network propagation procedure in each frame. However, my method could produce misleading results if a phospholipid ‘flip-flops’ between leaflets and is treated as a member of the incorrect leaflet in a calculation. To test for lipid ‘flip-flop’ I wrote the `flip_flop_tracker()` function in the `dimer_geometric_tools.py` module (D.17 on page 356), which performs the following steps: 1) Selection of phosphate particles and their assignment to a particular leaflet, in the first frame of the simulation, by network propagation using the MDAnalysis built-in `LeafletFinder` function. The latter uses a networking cut-off optimized by the built-in `optimize_cut-off` function, which is called by the `optimize_leaflet_selection_cutoff` function in the `dimer_geometric_tools.py` module (D.17 on page 356) 2) With the absolute leaflet assignments complete, iterate through each frame of the simulation and track the largest and smallest Z (along bilayer normal) coordinates for each set of leaflet phosphates. 3) Calculate the average Z coordinate of all phosphates in the system for each frame to provide an estimate of the center of the bilayer.

This way, any replicate simulation can be tracked for phosphates which cross the center of the bilayer and enter the other leaflet, and I will simply discard those replicates with flip-flop activity from any leaflet-based analyses. One of the five GpA replicate simulations was discarded from bilayer analysis because of the in-plane orientation assumed by one of the peptides during this simulation, while the other four did not show evidence of lipid flip-flop. A representative example from the fourth GpA replicate simulation is shown in Figure 5.37 on page 167. Interestingly,  $\frac{2}{10}$

replicates in the FGFR3 WT condition exhibited lipid flip-flop between leaflets despite the same bilayer lipid (POPC). An example of a phosphate headgroup crossing the bilayer leaflet boundary, in the first replicate FGFR3 WT simulation, is shown in Figure 5.38 on page 168. None of the ten FGFR3 heterodimer replicates exhibited lipid flip-flop activity (*i.e.*, Figure 5.39 on page 169), nor did any of the ten FGFR3 mutant homodimer replicates (*i.e.*, Figure 5.40 on page 170).

The number of replicate simulations discarded because of lipid flip-flop is low, and the absolute assignment of lipid phosphates to a particular leaflet in the first frame of each simulation is therefore a robust method which can serve as the basis for subsequent bilayer parameter measurements.

### 5.5.2 Protein-Local And -Distal Bilayer Thickness

The previous section (5.5.1 on page 127) dealt with the limitations of my lipid bilayer leaflet selection procedure. With a robust method for leaflet selection established, the focus now switches to determination of the bilayer thickness near (local to) the TM monomers or dimers in comparison to the distal bilayer thickness (away from the TM segments). I have written the MDAnalysis-dependent python function `analyze_leaflets()` in the `dimer_geometric_tools.py` module D.17 (page 356) to perform a number of analysis steps on each of the replicate trajectories: 1) Assign top and bottom leaflet phosphate populations in the first frame as described above. 2) Select all phosphate particles in the top and bottom leaflets within a 16 Å shell of either TM segment. Also, select all phosphate particles in the top and bottom leaflets that are more than 16 Å away from both TM segments. 3) Calculate the difference between the Z coordinates of the centers of geometry for the respective phosphate particle populations selected in the previous step. These are estimates of bilayer thickness in protein-local and protein-distal regions of the system.

The interphosphate bilayer thickness tracked in protein-local and protein-distal regions is shown in a representative plot from the fourth GpA replicate results (Fig-

ure 5.41 on page 171). By inspection, there is at least a 4 Å thinning of the bilayer in protein-local versus -distal regions. While some protein-local bilayer thinning is also observed in the case of the FGFR3 WT, it is not nearly as substantial (*i.e.*, Figure 5.42 on page 172). Similar results were observed for the FGFR3 heterodimer replicates (*i.e.*, Figure 5.43 on page 173). One of the FGFR3 mutant homodimer replicates exhibited a dip in protein-local bilayer thickness roughly half way through the simulation (Figure 5.44 on page 173), but this was not observed for the majority of the mutant replicates.

The substantial POPC bilayer thinning observed near GpA may relate to hydrophobic mismatch with this 23 residue TM construct, while the moderate bilayer thinning in proximity to each of the FGFR3 constructs may be explained by the longer TM segments (33 residues). However, there is an additional concern with the bilayer thickness analysis presented in this section—are the 16 Å protein-local shells including a sufficient number of lipid phosphates? For example, if typically only two lipids are within the defined local shell of each peptide, there is an undesirably small sample size for the measurement. This is addressed in the next section.

### 5.5.3 Protein-Local And -Distal Lipid Shell Counts

The previous section (5.5.2 on page 128) includes analysis consistent with protein-local bilayer thinning in GpA and (to a lesser extent) in FGFR3 simulations. However, it is important to determine whether a sufficient number of protein-local lipid phosphates were captured in the defined 16 Å shells around the  $C_{\alpha}$  particles of each TM segment. The `count_lipids_in_local_shell()` function in the `dimer_geometric_tools.py` module D.17 (page 356) was designed to count the number of phosphates in the top and bottom leaflets within 16 Å of either peptide in the dimer simulations as well as the total top and bottom leaflet phosphate counts for positions farther than 16 Å from either TM segment. Representative results for GpA are shown in Figure 5.45 on page 174. As expected, there are considerably more protein-distal than protein-

local lipids in each leaflet. It is noteworthy that the peptide dimerization event is clearly observable as a synchronized increase in the number of protein-distal lipids, and this is sensible because the protein-local shells overlap following dimerization. There are more lipids in the distal bottom leaflet relative to the distal top leaflet, but the reason for this is not clear and may simply relate to a partially random distribution of lipids during the POPC bilayer self-assembly process. A closer look at the peptide-local lipid shell counts reveals roughly 10 lipids per leaflet near a given GpA monomer (Figure 5.46 on page 174).

The FGFR3 WT also has a larger lipid population in the distal bottom leaflet versus the top (*i.e.*, Figure 5.47 on page 175). Again, there is a clear indicator of peptide dimerization as the distal leaflet lipid counts increase in unison, and the local lipid shell counts appear to synchronize at this time. The local shell count synchronization is especially apparent in the closer view (Figure 5.48 on page 175), which also reveals that 10-15 lipids are included per leaflet within 16 Å of a given FGFR3 monomer. In the pre-dimer state, one of the monomers fluctuates to some lower lipid shell counts.

The gap in distal leaflet lipid populations is much smaller in the FGFR3 heterodimer replicates (*i.e.*, Figure 5.49 on page 176), which is consistent with a stochastic component to leaflet population distribution during the bilayer self-assembly process. The ninth heterodimer replicate simulation tracked in Figure 5.49 again exhibits a synchronized increase in distal leaflet lipid populations coinciding with peptide dimerization (the extended time required for dimerization of this replicate is consistent with observations in section 5.4.1 on page 111). The local lipid shell counts also synchronize upon dimerization (Figure 5.50 on page 176), and 10-15 lipids are included per leaflet within 16 Å of a given FGFR3 monomer. Again, in the pre-dimer state, one of the monomers fluctuates to some lower lipid shell counts.

The behaviour of the FGFR3 mutant homodimer is very similar to the heterodimer

(*i.e.*, Figure 5.51 on page 177 and Figure 5.52 on page 177). Overall, the 10-15 lipids per leaflet counted in the protein-local shells for GpA and FGFR3 replicates provide confidence that the 16 Å local shell definition is appropriate. A larger number of lipids (~ 90) was reported in a defined local shell around another simulated protein in the literature (195), but the rhomboid protease in question is substantially larger than the dimers reported here. With verification of the local lipid shell definition complete, the next section returns to the issue of tracking bilayer thickness, with an emphasis on the effect of dimerization on local bilayer thickness.

#### 5.5.4 Overall Bilayer Thickness Analysis For GpA And FGFR3 Simulations

With added confidence in the protein-local and -distal lipid selection methodology (section 5.5.3 on page 129), and evidence for protein-local bilayer thinning in the CG simulations for GpA and FGFR3 (section 5.5.2 on page 128), it is desirable to analyze the data gathered across all replicate simulations for average bilayer thickness values before and after peptide dimerization. The function `bilayer_thickness_average_results()` in the `dimer_geometric_tools.py` module D.17 (page 356) performs a few tasks in this analysis: 1) Using the previously calculated closest interhelical  $C_\alpha$  approach (per frame) during each of the trajectories (section 5.4.1 on page 111), determine the frame number at which dimerization occurs (defined as  $d_{C_\alpha-C_\alpha} < 6$  Å). 2) Parse the previously calculated protein-local and -distal bilayer thickness results (section 5.5.2 on page 128) and split the protein-local data in to pre- and post-dimerization lists, combining results from each monomer. The protein-distal bilayer thickness values are not split on the dimerization frame. In total, this produces three separate lists for each of the GpA, FGFR3 WT, FGFR3 heterodimer, and FGFR3 mutant homodimer conditions.

Finally, the `bilayer_stats()` function in the same module is called separately by the head script (`analyze_FGFR3_dimer_simulations.py` on page 338) to deter-

mine the average and standard deviation values for the global lists produced by `bilayer_thickness_average_results()`. The overall results for GpA and the FGFR3 conditions are summarized in Figure 5.53 on page 178. While it is clear that GpA and FGFR3 TM peptides cause local bilayer thinning, it is not clear within one standard deviation that any substantial local bilayer thinning occurs following dimerization. The latter observation may be consistent with a stable dimerization process because additional local bilayer thinning following dimerization may come with an entropic lipid rearrangement penalty.

### **5.5.5 DPPC: The Effect Of Phospholipid Type On FGFR3 Dimer Behaviour**

In addition to the manually setup and executed dimer simulations summarized in Table 5.1 (page 189), I also submitted FGFR3 WT, heterodimer and mutant homodimer configurations to the high-throughput GROMACS-based SIDEKICK automated simulation program (196). SIDEKICK performs the setup and execution of a large number of coarse-grained replicate simulations while abstracting the details from the user, and performed a total of 98 FGFR3 WT replicates, 96 FGFR3 heterodimer replicates, and 92 FGFR3 mutant homodimer replicates, with each replicate consisting of a 0.5  $\mu$ s trajectory. One limitation of this high-throughput framework is that only DPPC may be used as the bilayer lipid (because it has been sufficiently well characterized for coarse-grained membrane-based simulations to be used in an automated context). I've used POPC for the manually executed simulations for consistency with experimental FGFR3 studies (11, 173). However, I have performed a preliminary analysis on the SIDEKICK results to probe the effect of DPPC versus POPC on FGFR3 dimer behaviour.

#### **5.5.5.1 FGFR3 Dimer Stability In DPPC**

All of the GpA and FGFR3 constructs I have analyzed from CG simulations in POPC bilayers have formed stable dimers which do not dissociate once formed (see



section 5.4.1 on page 111). However, WT and mutant GpA TM constructs have been observed to dissociate in DPPC-based CG simulations (184). In contrast, more recent DPPC-based CG simulations of WT and mutant GpA TM peptides do not exhibit any dissociation behaviour (185), and the authors suggest the discrepancy with previous work is related to the more thoroughly calibrated MARTINI force field they employ.

FGFR3 WT, heterodimer, and mutant homodimer constructs all exhibited dissociation behaviour in DPPC bilayer-based SIDEKICK CG simulations (Figure 5.54 on page 179). The large number of replicate simulations were parsed by the `closest_contacts_efficient_SIDEKICK()` function in the `dimer_geometric_tools.py` module (page 356), which was controlled from a head script designed for the high-throughput data organization (`analyze_sidekick_FGFR3_dimer_simulations.py`). The reduced dimerization propensity is probably not the expected consequence of switching to a more saturated phospholipid, but given the variety of reported results within DPPC alone, it is not surprising that different phospholipids encourage varied dimerization behaviour.

### 5.5.5.2 FGFR3 Dimerization Interface In DPPC

In section 5.4.6 (page 118) I described a method for assessing the number of dimer interfaces explored by helix 2 when helix 1 is rmsd-fixed to a reference frame. For FGFR3, this reference frame is the configuration of helix 1 in the first frame of the first WT replicate simulation. Using the same (POPC-based) reference configuration for DPPC simulations, I have once again tracked the positional probability of helix 2, in this case to provide a direct comparison of the DPPC and POPC dimer interfaces for the three FGFR3 conditions. It is especially useful that the sample sizes are comparable between the SIDEKICK (DPPC) and manual (POPC) data sets. There were  $10 \text{ replicates} \times 5 \frac{\mu\text{s}}{\text{replicate}} = 50 \mu\text{s}$  total for each of the manual FGFR3 conditions. The 98 WT, 96 heterodimer, and 92 mutant FGFR3  $0.5 \mu\text{s}$  replicates accumulate 49,

48, and 46  $\mu s$  of total simulation time, respectively. Although each FGFR3 condition can be compared over roughly 50  $\mu s$  of total simulation time, it is noteworthy that the much larger number of SIDEKICK replicates results in more simulation *restarts*, and therefore less time spent in the dimer configuration.

The `fixed_helix_thermal_merged_SIDEKICK()` and `thermal_bins_SIDEKICK()` functions in the `dimer_geometric_tools.py` module (page 356) were called in sequence by the head script (`analyze_sidekick_FGFR3_dimer_simulations.py`) to produce the probability map for the position of helix 2 in the reference frame of rmsd-fixed helix 1. The comparison between FGFR3 dimer interfaces in POPC and DPPC is shown in Figure 5.55 on page 180. Clearly, the primary dimer interface is still located at the ‘bottom left,’ which is encouraging for validation as a stable interface. However, the secondary dimer interface is located at the ‘top right’ in DPPC rather than the ‘bottom right’ observed for POPC. Curiously, the secondary dimer interface in DPPC is more prominent in the WT than the heterodimer, and then reappears in the mutant homodimer.

## 5.6 FGFR3 Monomer Simulations

I performed a set of FGFR3 monomer simulations in POPC bilayers (see Table 5.2 on page 190) to control for the individual behaviour of the FGFR3 WT and G380R mutant TM peptides used in the dimer simulations. I wrote the preliminary analysis functions for the monomer replicates in the `monomer_geometric_tools.py` module (page 460), which is controlled by the head script `analyze_FGFR3_monomer_simulations.py` (page 457).

### 5.6.1 Helix Tilt Angle

I wrote the `helix_tilt_vs_bilayer_normal()` function in the `monomer_geometric_tools.py` module (page 460) to track the tilt of the FGFR3 WT or mutant helices with respect to the bilayer normal in each of the replicate monomer trajectories. The

helical axis was defined as the first eigenvector of the  $C_\alpha$  backbone and the bilayer normal was defined as the third eigenvector of the POPC phosphate headgroups. Both the WT (*i.e.*, Figure 5.56 on page 181) and mutant (*i.e.*, Figure 5.57 on page 182) monomers exhibited frequent fluctuations in helical tilt angles, mostly between  $0^\circ$  and  $30^\circ$ . This is a wider range of fluctuation than predicted by oriented circular dichroism and neutron diffraction of FGFR3 peptides in POPC bilayers ( $0^\circ$  to  $20^\circ$ ) (11).

### 5.6.2 FGFR3 SIDEKICK Monomer Simulations

In addition to the manually executed monomer replicates, the SIDEKICK high-throughput simulation framework (described in section 5.5.5 on page 132) was used to conduct 100 replicate simulations (of  $0.1 \mu\text{s}$  duration) for each of the WT and G380R FGFR3 TM constructs in POPC bilayers. In contrast to the case for dimer simulations, SIDEKICK allows for the use of POPC and automatically generates a gallery of analysis plots. The peptide bilayer burial depth distributions are plotted for the WT (Figure 5.58 on page 183) and mutant (Figure 5.59 on page 184), and it is clear that the center of the mutant peptide is displaced upward from the center of the bilayer relative to the WT (by at least  $1 \text{ \AA}$ ). This is consistent with the introduction of the R residue in the mutant TM segment and the movement of this residue away from the hydrophobic core of the bilayer. Experimental results for FGFR3 peptide constructs in POPC bilayers were consistent with a  $5 \text{ \AA}$  upward (N-terminal) displacement of the R380 residue relative to G380 in the WT (11).

SIDEKICK also automatically produced plots for the FGFR3 WT (Figure 5.60 on page 185) and mutant (Figure 5.61 on page 186) helix tilt angle distributions over all the replicate simulations. Both WT and mutant constructs exhibited similar helix tilt angle distribution modes  $\sim 38^\circ$ . This angle is larger than the upper value of the range predicted by experimental constraints from oriented circular dichroism and neutron diffraction of FGFR3 peptides in POPC bilayers ( $0^\circ$  to  $20^\circ$ ) (11).

The final analysis completed by SIDEKICK is the distribution of helix rotation

angles (about the helical axis itself). While the WT FGFR3 helix clearly assumes a single favoured rotation angle in the bilayer (Figure 5.62 on page 187), the G380R mutant exhibits a bimodal distribution of rotation angles with a roughly 180° rotation between the preferred configurations (Figure 5.63 on page 188). This added ‘flexibility’ may contribute to the secondary dimer interface behaviour observed in heterodimer and mutant homodimer (section 5.4.6 on page 118).

## 5.7 Summary And Conclusions

I started this chapter by describing the medical relevance of the G380R mutation in the TMD of FGFR3—in ~99% of cases it is the underlying cause of achondroplasia (163, 164, 165). CG-MD simulations of the FGFR3 WT homodimer, heterodimer, and mutant homodimer TMDs in lipid bilayers were analyzed in detail (section 5.4 on page 111). I demonstrated that the initial 55 Å separation between helices was sufficient to ensure that there was no bias in the dimer formation process and that the dimers did not dissociate once formed (section 5.4.1 on page 111). A substantial amount of stochastic wandering can occur in the bilayer prior to dimer formation (section 5.4.2 on page 113), and this is also consistent with unbiased dimer formation. FGFR3 dimer constructs all exhibited bimodal helix crossing angles (section 5.4.3 on page 114), but still moved together in the membrane after dimer formation (section 5.4.4 on page 115).

Having confirmed stable and unbiased dimer formation, I parsed the GpA and FGFR3 trajectories for the closest interfacial contacts in the dimers (section 5.4.5 on page 116). I first tested my analytical approach on the GpA constructs and the major contacts parsed from the simulations were consistent with experimental results. This provided confidence for the application of the analysis to FGFR3 and I identified at least three major interfacial residues—G370, A374, and R397. The latter residues were consistent with experimental and medical findings, and are suggested as candi-

date interfacial residues for FGFR3 dimers (for which no high-resolution structures are currently available). The three FGFR3 dimer constructs all have a consistent primary dimer interface, but also a secondary dimer interface that appears progressively in the heterodimer and is most prominent in the mutant homodimer (section 5.4.6 on page 118). I demonstrated that the FGFR3 helical positions are more stable at the primary interface than the secondary interface, and that there is a discrete time spent at each interface rather than a continuous sampling between the two interfaces (section 5.4.7 on page 120). Attempts to manually select representative dimer interface structures based on the polar  $\theta$  of helix 2 in the rmsd-fixed reference frame of helix 1 were problematic because there is a stochastic component to the manual selection procedure that can select a representative primary interface structure that has either a left- or right-handed helix crossing angle (section 5.4.8 on page 121).

For a rigorous classification of FGFR3 dimer interfaces, I parsed across the frames of all trajectories to define *populations* of simulation frames that fall into the primary, secondary, and ‘other’ interfaces based on moving average (sliding window) polar  $\theta$  and a filter for sudden short-lived fluctuations in the data (section 5.4.9 on page 123). The results were consistent with a loss of symmetry at G370 and R397 at the secondary interface compared with the primary interface. These interfacial differences are at important contacts (based on simulation and experiment) and are consistent with rotation of one helix relative to the other—a potential molecular mechanism for pathology. In support of this model, activation of ErbB2 occurs by a 120° rotation of the TMD monomers relative to each other after ligand binding to the RTK (197). In addition, the oncogenic V664E TMD mutation in Neu (rat homologue of ErbB2) prevents conformational switching between the active and inactive states (198). Thus, activating RTK TMD mutations may encourage or restrict the rotation of TMDs depending on the specific structure of the RTK.

In section 5.5 (page 126) I analyzed some properties of the POPC bilayer local

and distal to the FGFR3 TMD peptides. There was peptide-local bilayer thinning for GpA and (to a lesser extent) for FGFR3. Curiously, I found that all three FGFR3 dimer constructs dissociated in DPPC but not in POPC, and that the secondary dimer interface and its trends are different in DPPC (section 5.5.5 on page 132).

FGFR3 monomer simulations in POPC were consistent with increased vertical (N-terminal) displacement of the mutant monomer from the center of the bilayer (relative to FGFR3 WT) (section 5.6.2 on page 135), similar helix tilt angle distributions (relative to the bilayer normal), and the presence of two preferred rotation (about the helical axis) angles for the mutant constructs (and only a single preferred rotation angle for the WT). The latter observation may relate to the observed secondary dimer interface for dimer constructs involving G380R and may in particular explain the ability of this helix to rotate in the membrane between the two interfaces.

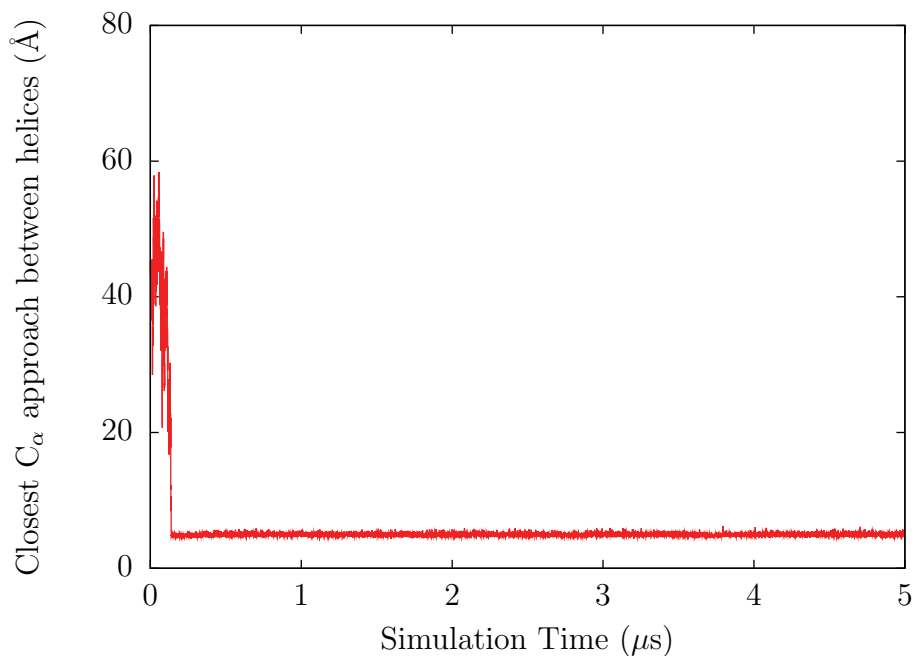


Figure 5.1: Tracking closest  $C_\alpha$  helix-helix approach for the first GpA coarse-grained simulation.

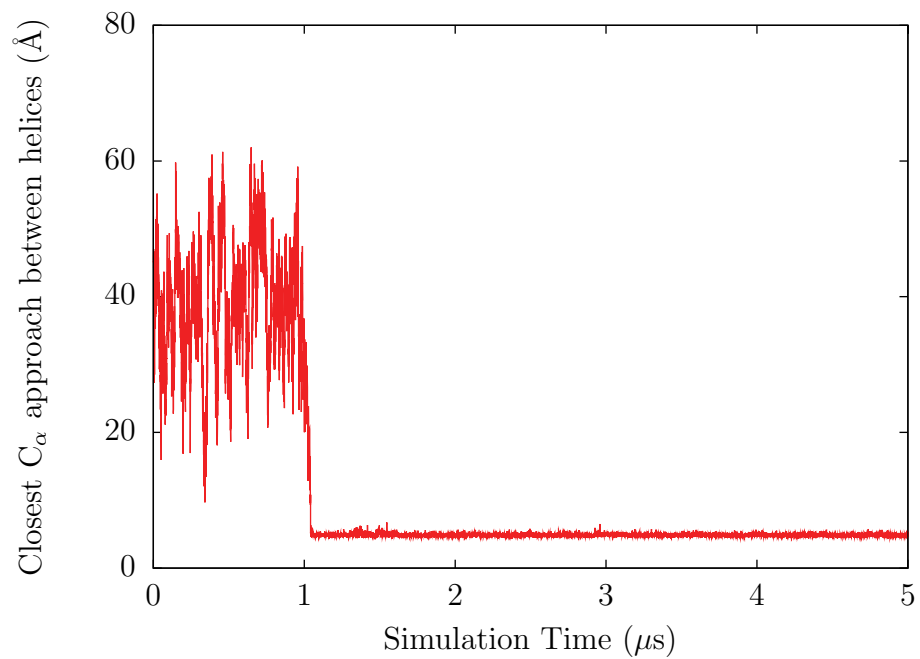


Figure 5.2: Tracking closest C<sub>α</sub> helix-helix approach for the third WT FGFR3 coarse-grained simulation.

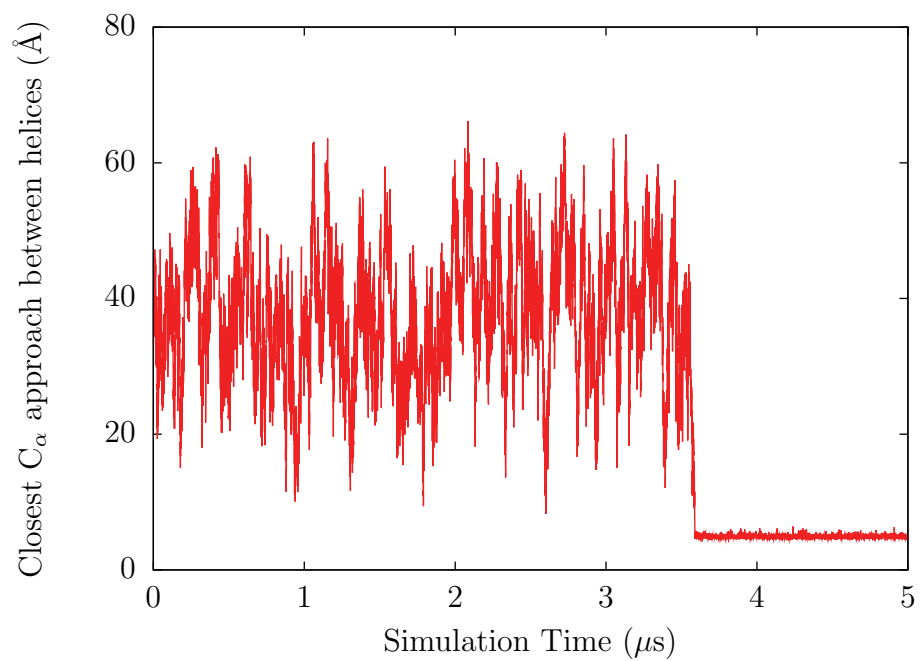


Figure 5.3: Tracking closest C<sub>α</sub> helix-helix approach for the ninth FGFR3 heterodimer coarse-grained simulation.

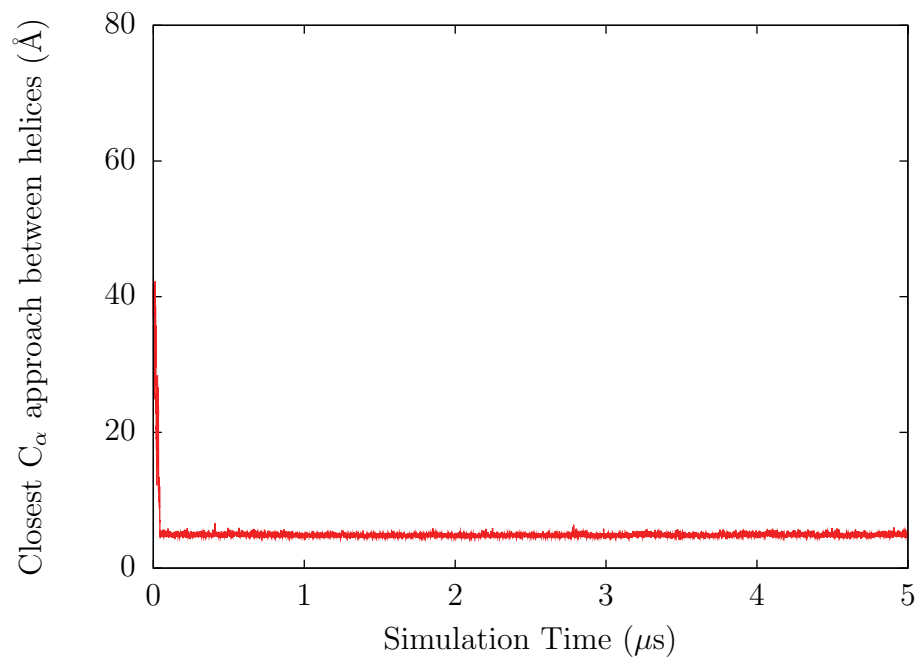


Figure 5.4: Tracking closest  $C_\alpha$  helix-helix approach for the second FGFR3 mutant homodimer coarse-grained simulation.



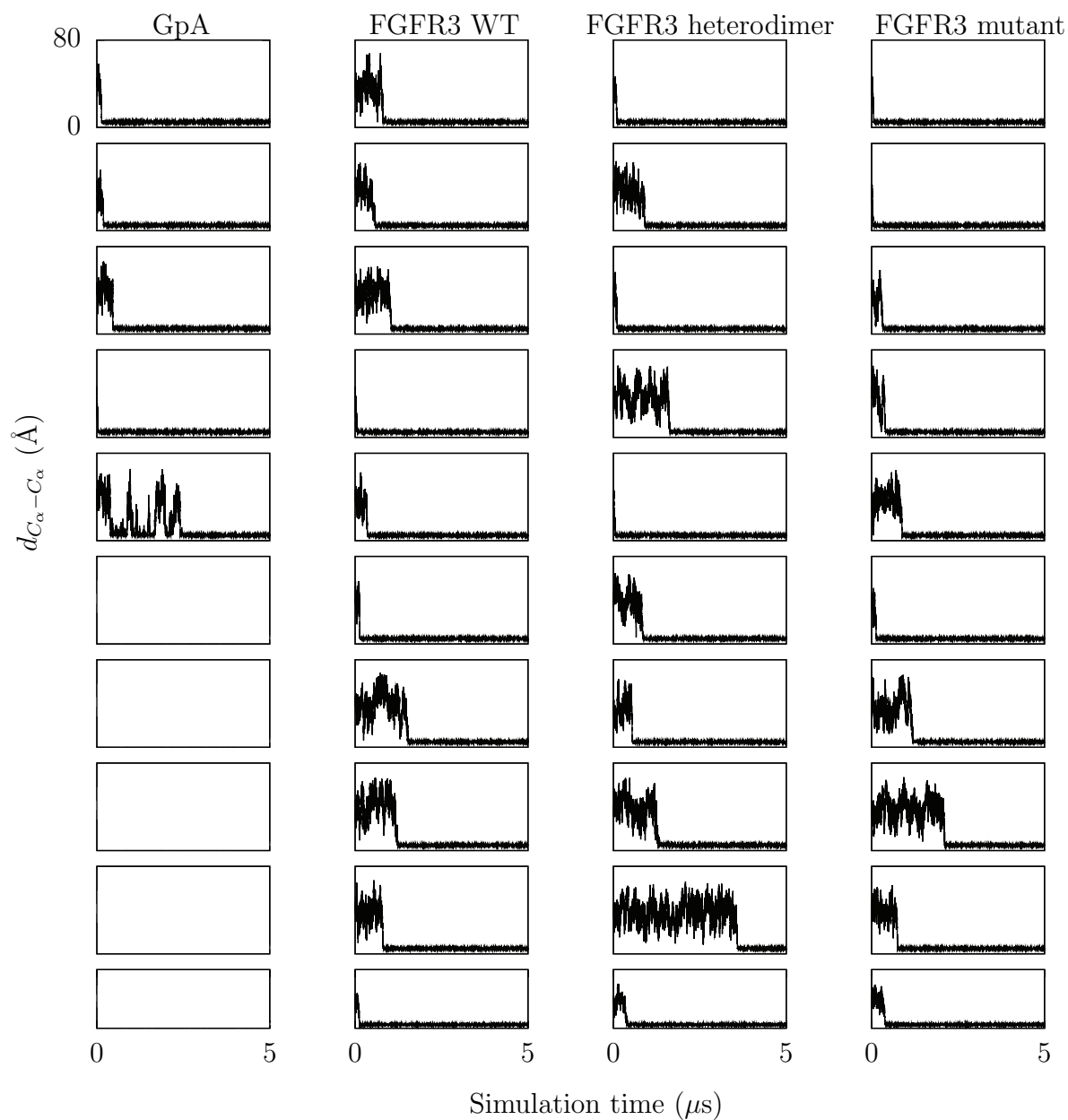


Figure 5.5: The closest interhelix approach between  $C_\alpha$  particles is monitored during the 5  $\mu\text{s}$  coarse-grained simulations. Five replicates were conducted for GpA, while ten replicates were completed for each of FGFR3 wild-type, G380R heterodimer, and G380R mutant homodimer conditions.

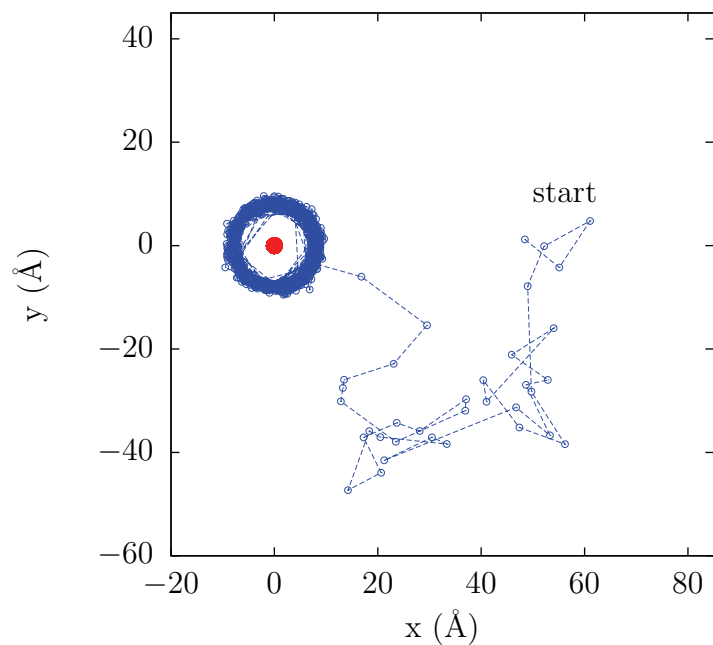


Figure 5.6: Relative motion of the center of geometry of helix 2 (*blue*) in the reference frame of the center of geometry of helix 1 (*red*) during the first wild-type homodimer GpA coarse-grained simulation (every 10<sup>th</sup> frame).

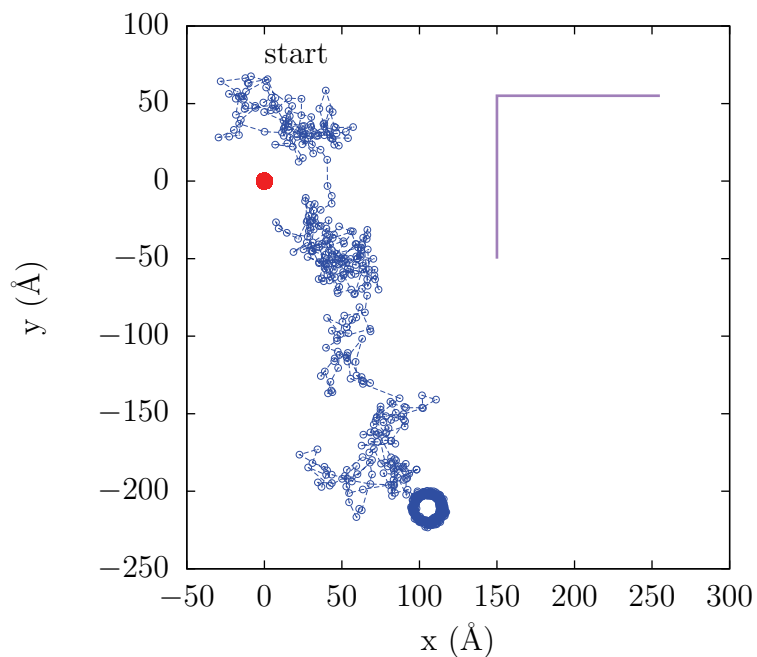


Figure 5.7: Relative motion of the center of geometry of helix 2 (*blue*) in the reference frame of the center of geometry of helix 1 (*red*) during the seventh FGFR3 WT homodimer replicate coarse-grained simulation (every 10<sup>th</sup> frame). The approximate size of the simulation box in the x-y plane is shown in *purple*.

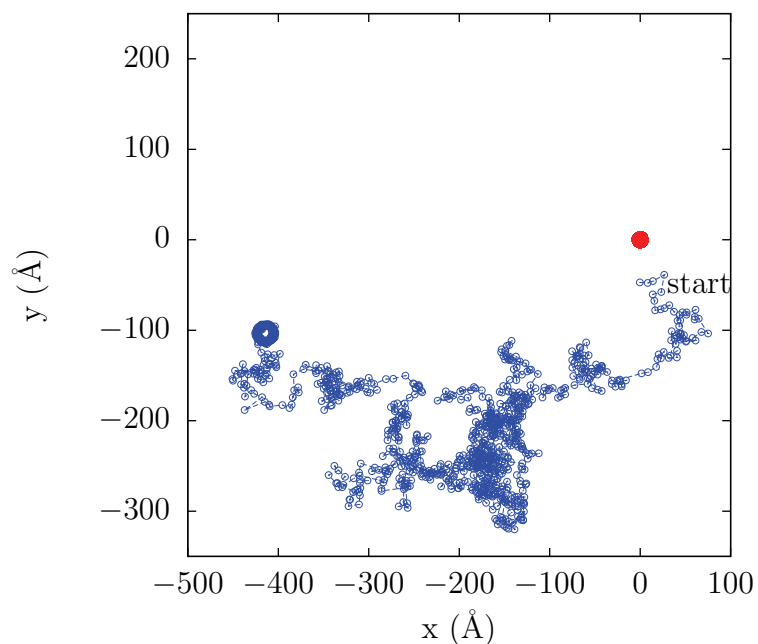


Figure 5.8: Relative motion of the center of geometry of helix 2 (*blue*) in the reference frame of the center of geometry of helix 1 (*red*) during the ninth FGFR3 heterodimer replicate coarse-grained simulation (every 10<sup>th</sup> frame).

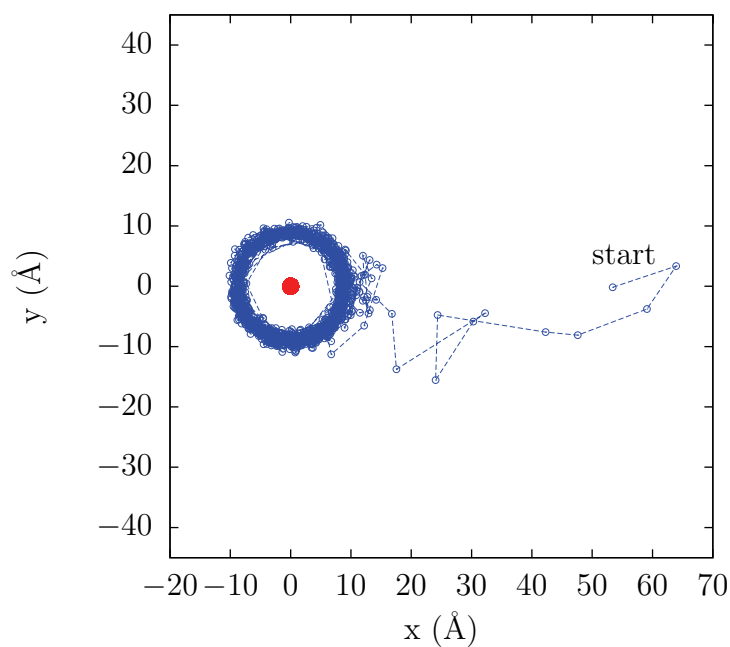


Figure 5.9: Relative motion of the center of geometry of helix 2 (*blue*) in the reference frame of the center of geometry of helix 1 (*red*) during the second FGFR3 mutant homodimer replicate coarse-grained simulation (every 10<sup>th</sup> frame).

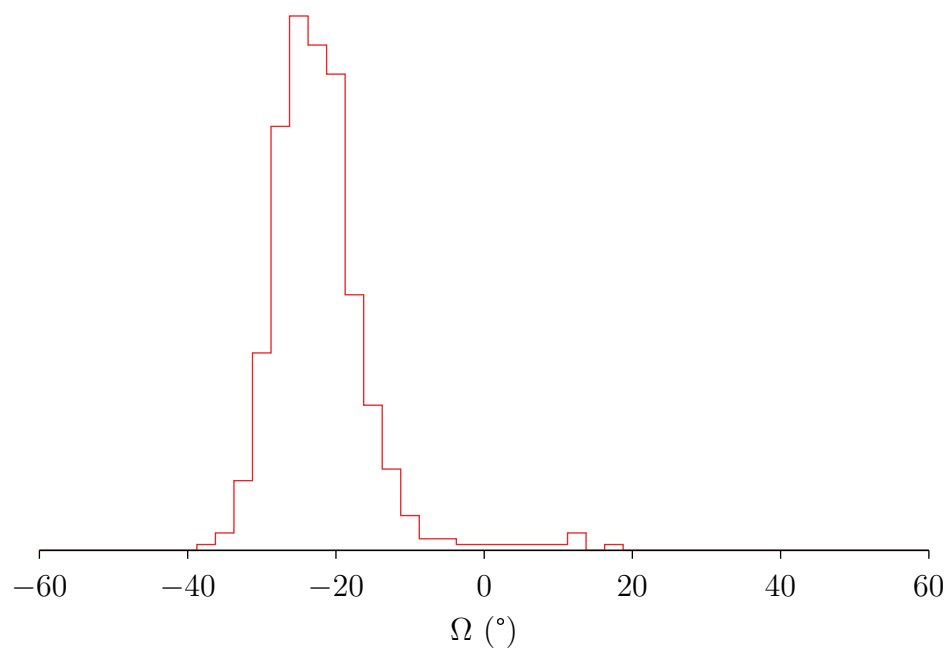


Figure 5.10: Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the GpA WT homodimer construct. Every 25<sup>th</sup> frame of the trajectory was parsed and included in the histogram only if the helices were dimerized. The area under the curve sums to unity.

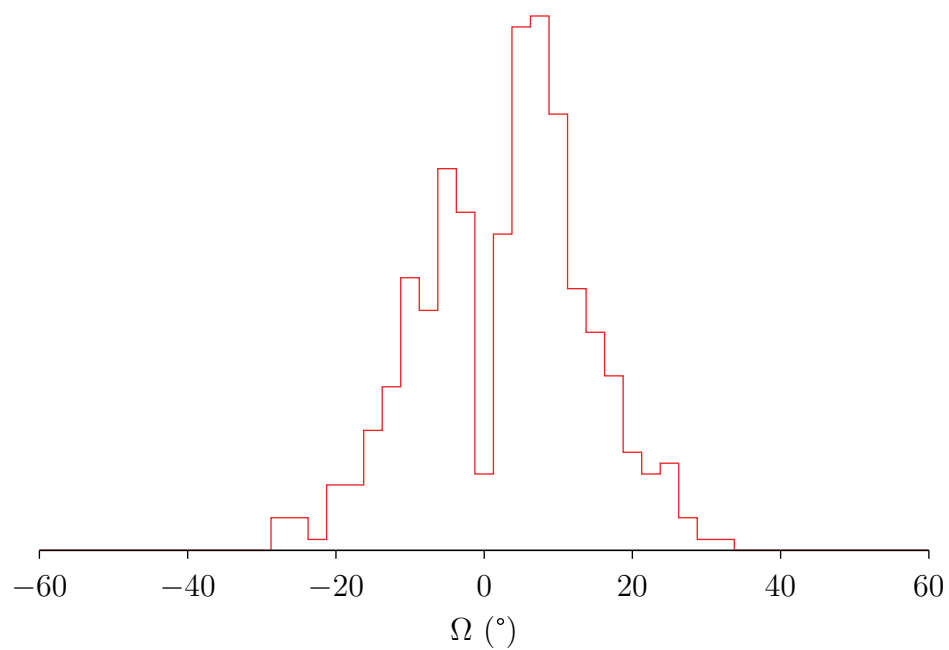


Figure 5.11: Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the FGFR3 WT homodimer construct. Every 25<sup>th</sup> frame of the trajectory was parsed and included in the histogram only if the helices were dimerized. The area under the curve sums to unity.

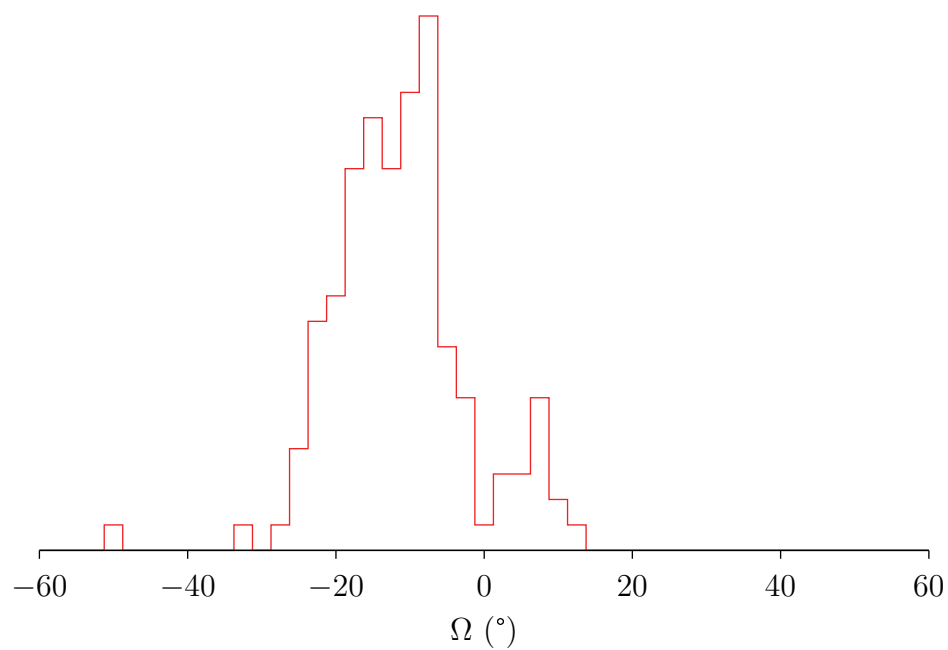


Figure 5.12: Helix crossing angle ( $\Omega$ ) distribution for the ninth replicate of the FGFR3 heterodimer construct. Every 25<sup>th</sup> frame of the trajectory was parsed and included in the histogram only if the helices were dimerized. The area under the curve sums to unity.

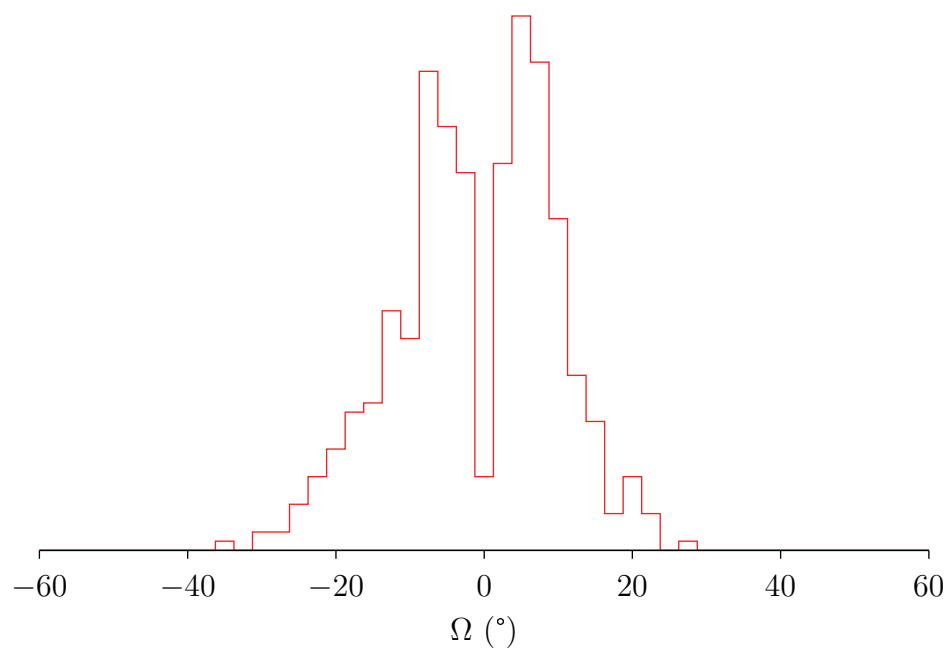


Figure 5.13: Helix crossing angle ( $\Omega$ ) distribution for the first replicate of the FGFR3 mutant homodimer construct. Every 25<sup>th</sup> frame of the trajectory was parsed and included in the histogram only if the helices were dimerized. The area under the curve sums to unity.

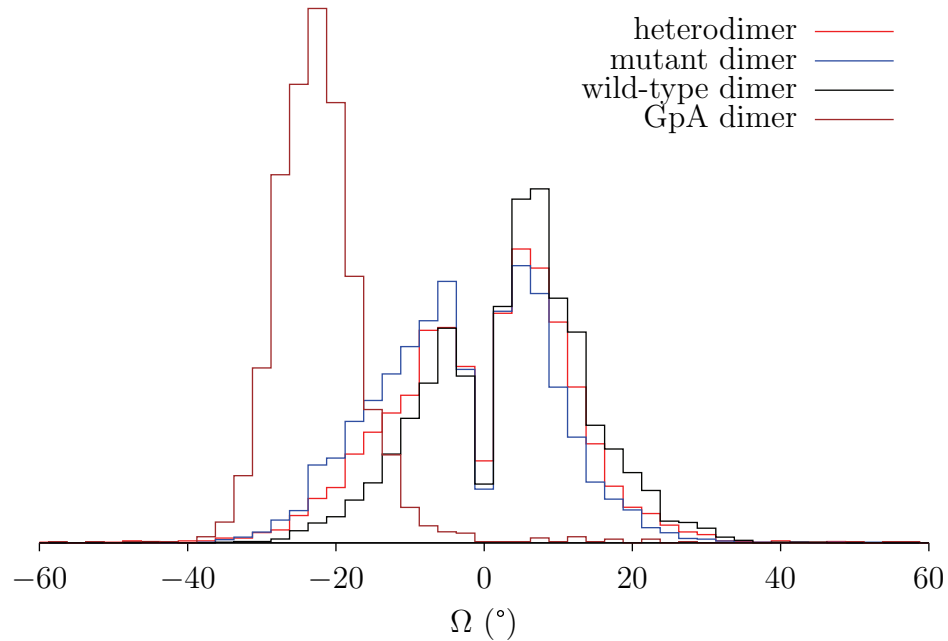


Figure 5.14: Helix crossing angle ( $\Omega$ ) distribution merged over all replicates of GpA and FGFR3 constructs. Every 25<sup>th</sup> frame of the constituent trajectories was parsed and included in the histograms only if the helices were dimerized. Helix crossing angles are shown for GpA (*brown*), FGFR3 wild-type (*black*), FGFR3 heterodimer (*red*), and FGFR3 mutant homodimer (*blue*). The area under each curve sums to unity.



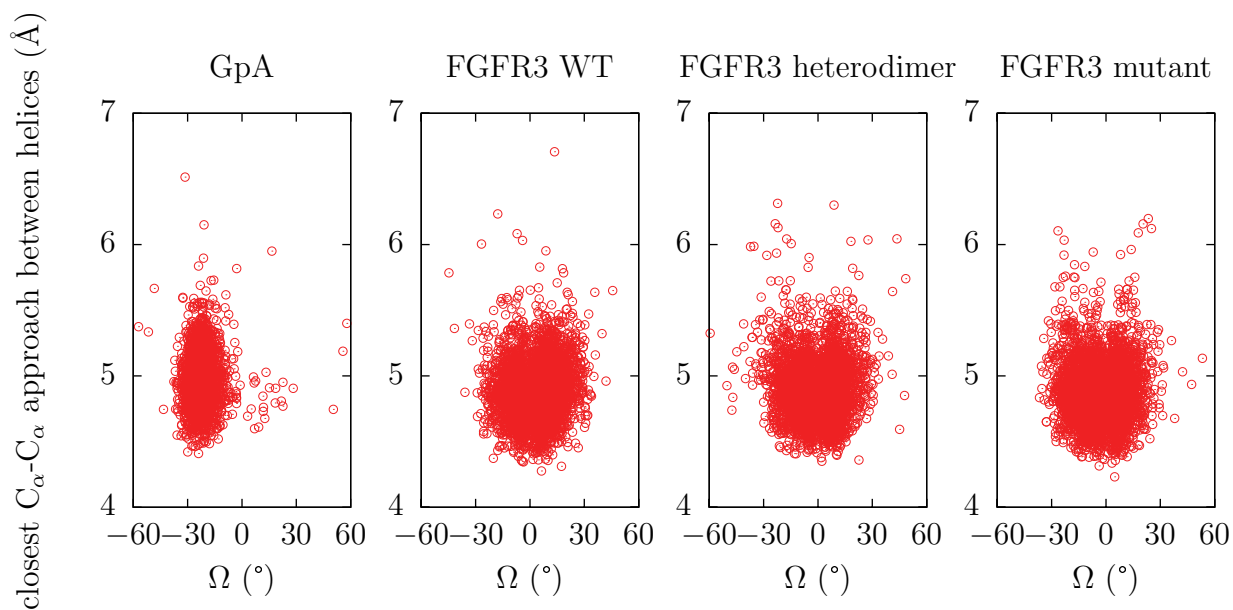


Figure 5.15: Correlating the closest  $C_{\alpha}$  interhelical approach with helix crossing angle ( $\Omega$ ) for GpA (5 replicates), FGFR3 WT (10 replicates), FGFR3 heterodimer (10 replicates), and FGFR3 mutant homodimer (10 replicates) constructs. The plotted results are merged across all available replicate simulations as indicated.

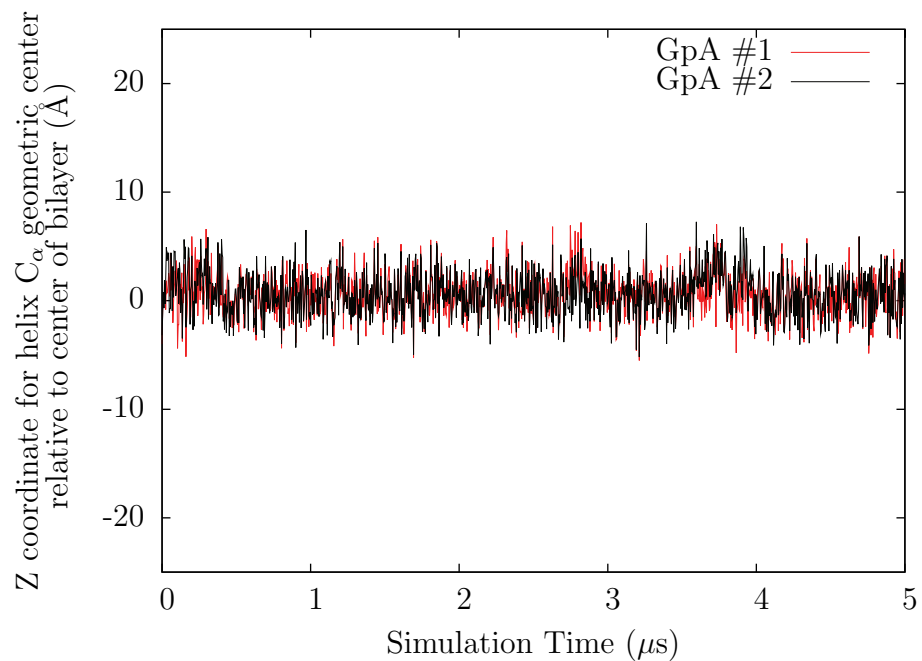


Figure 5.16: Tracking Z coordinate (along bilayer normal) for each helix C<sub>α</sub> geometric center relative to center of bilayer during the first GpA replicate simulation. Every 10<sup>th</sup> frame was parsed in a non-centered trajectory.

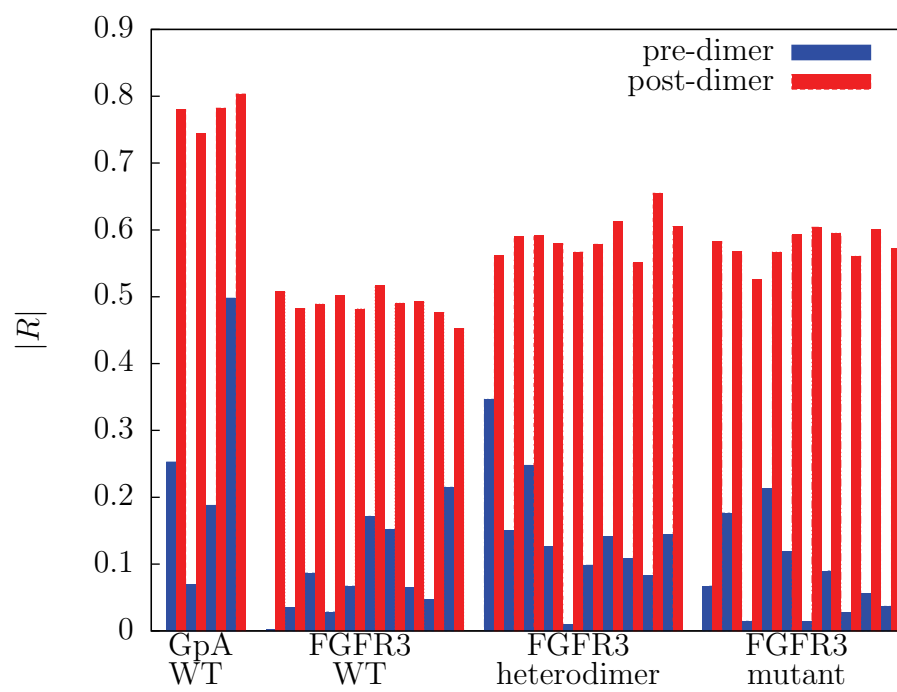


Figure 5.17: The absolute correlation coefficients ( $|R|$ ) between the Z coordinates of the geometric centers of the GpA or FGFR3 helices in each replicate simulation before and after dimerization. The dimerization distance is defined as a closest interhelical  $C_\alpha$  approach  $< 6 \text{ \AA}$ .

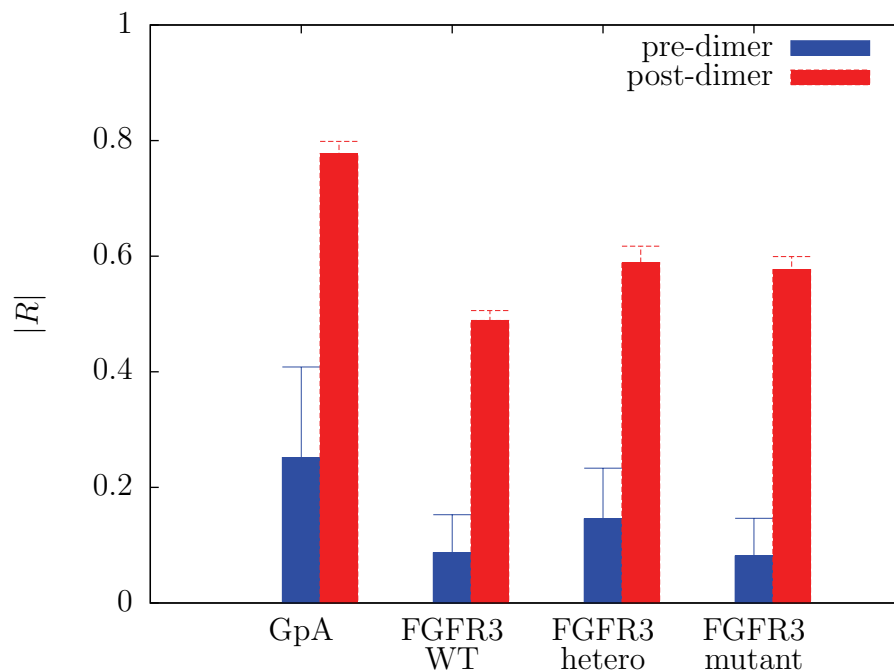


Figure 5.18: Average and standard deviation (across all replicates) for the absolute correlation coefficient ( $|R|$ ) between helical geometric center Z coordinates for GpA and FGFR3 dimer simulation constructs before and after dimerization. The dimerization distance is defined as a closest interhelical  $C_\alpha$  approach  $< 6 \text{ \AA}$ .

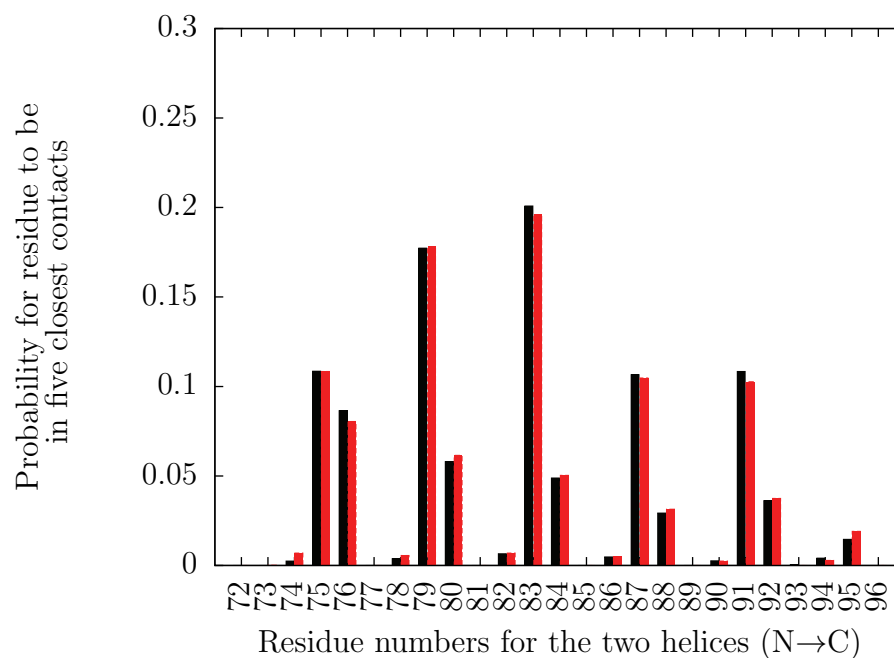


Figure 5.19: The normalized frequency of occurrence for a residue in the five closest contacts between helix 1 (*black*) and helix 2 (*red*) when the GpA helix-helix  $C_\alpha$  separation is within  $7 \text{ \AA}$  in the first replicate coarse-grained simulation.

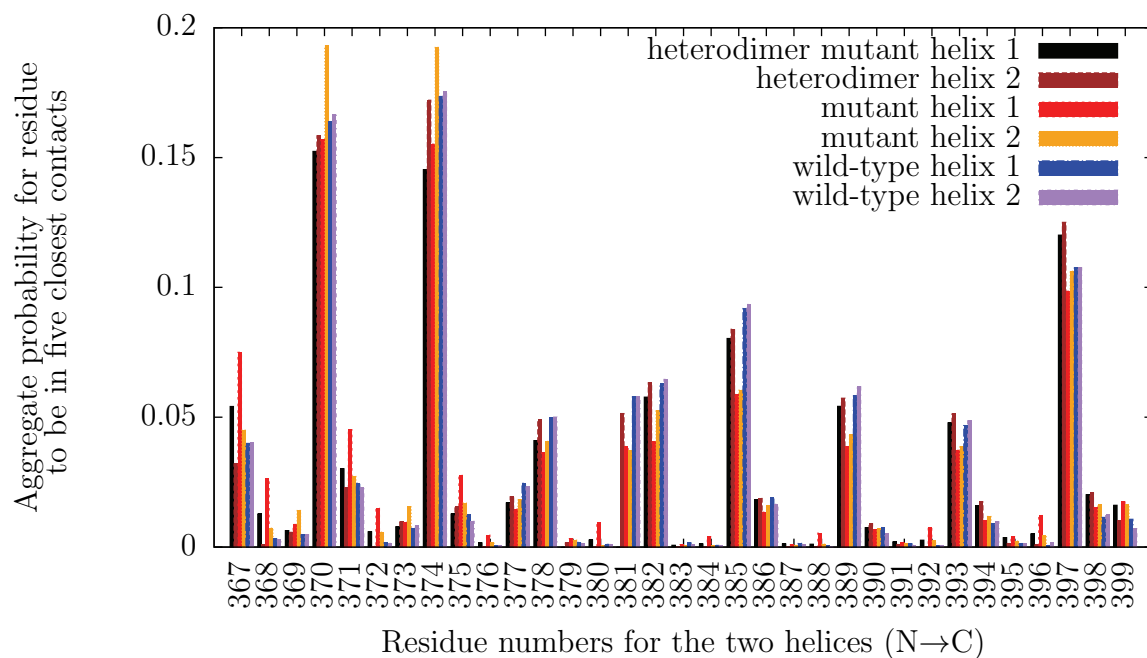


Figure 5.20: Normalized frequency of occurrence for a residue in the five closest contacts between helices when interhelix  $C_{\alpha}$  separation is within  $7 \text{ \AA}$  over *all* the CG replicate simulations for each of FGFR3 wild-type (helix 1 *blue*, helix 2 *purple*), heterodimer (G380R helix 1 *black*, WT helix 2 *brown*), and mutant homodimer (helix 1 *red*, helix 2 *yellow*) conditions.

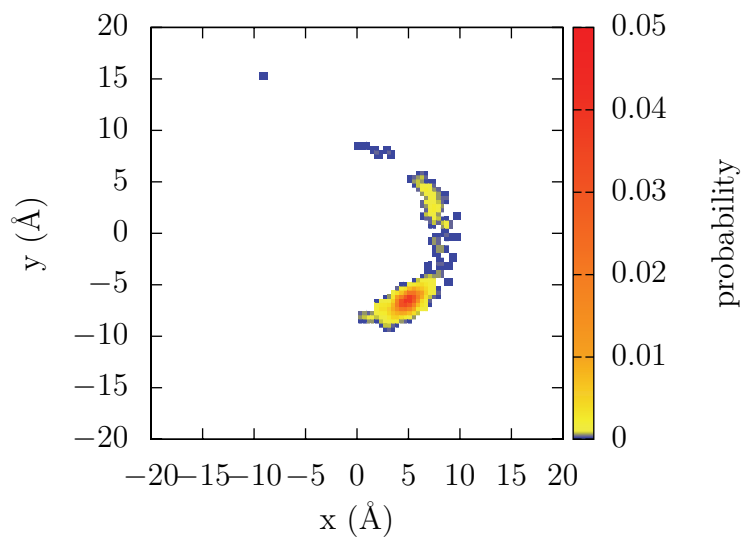


Figure 5.21: A contour plot of the positional probability of GpA helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) in the first GpA replicate simulation. The non-linear probability scale is indexed as  $P = 0.0$  *white*,  $P = 0.000001$  *blue*,  $P = 0.001$  *yellow*,  $P = 0.01$  *orange*,  $P = 0.05$  *red*.

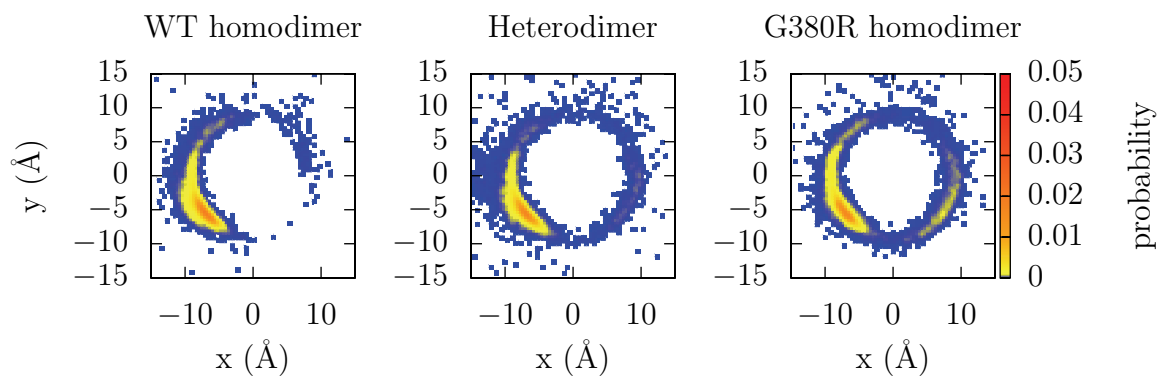


Figure 5.22: A contour plot of the positional probability of helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) calculated over all ten replicate trajectories for each of the FGFR3 dimer conditions. The non-linear probability scale is indexed as  $P = 0.0$  *white*,  $P = 0.000001$  *blue*,  $P = 0.001$  *yellow*,  $P = 0.01$  *orange*,  $P = 0.05$  *red*.

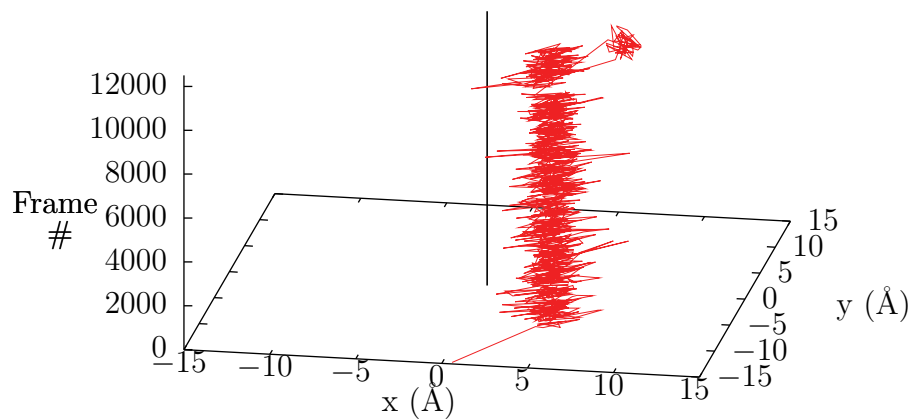


Figure 5.23: The coordinates of the geometric center of helix 2 (*red line*) are tracked in the reference frame of rmsd-fixed helix 1 (central *black line*) as the simulation progresses (with frame number along the Z-axis). This is the fourth replicate GpA CG simulation and is representative of results for other GpA replicates.



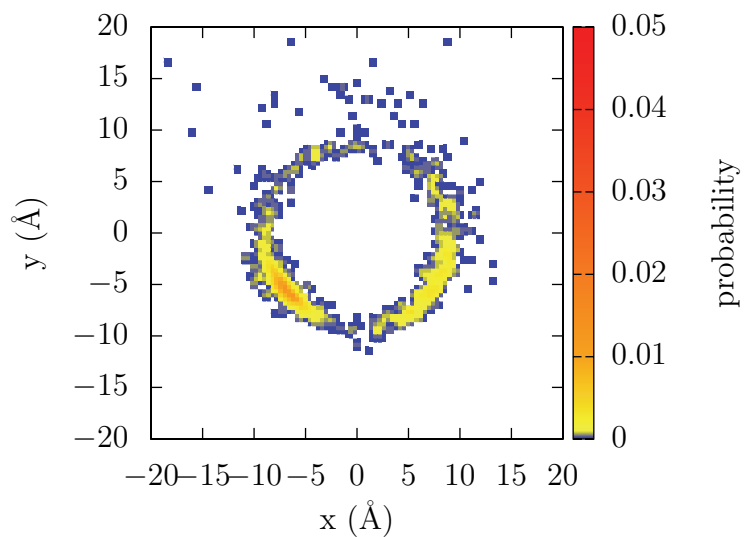


Figure 5.24: A contour plot of the positional probability of helix 2 in the reference frame of rmsd-fixed helix 1 (centered at the origin) in the fourth mutant homodimer FGFR3 simulation. The non-linear probability scale is indexed as  $P = 0.0$  *white*,  $P = 0.000001$  *blue*,  $P = 0.001$  *yellow*,  $P = 0.01$  *orange*,  $P = 0.05$  *red*.

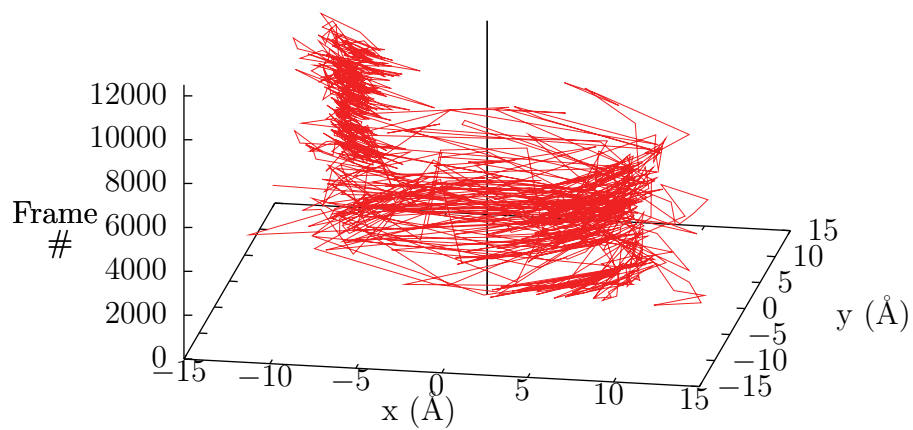


Figure 5.25: The coordinates of the geometric center of helix 2 (*red line*) are tracked in the reference frame of rmsd-fixed helix 1 (central *black line*) as the simulation progresses (with frame number along the Z-axis). This is the fourth replicate FGFR3 mutant homodimer CG simulation and is representative of results for other mutant replicates.

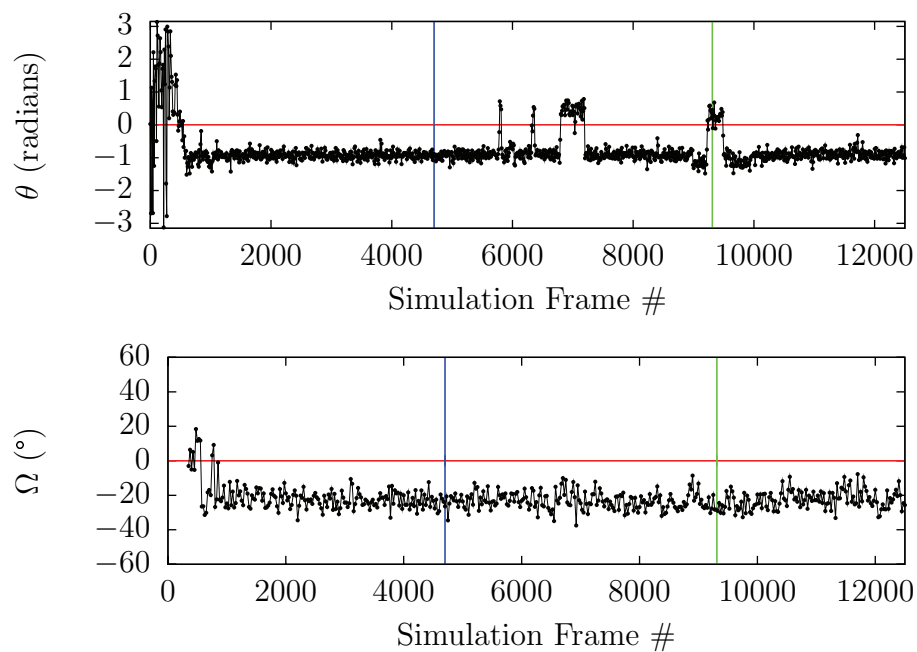


Figure 5.26: Polar angle of helix 2 ( $\theta$ , top) in the rmsd-fixed frame of helix 1 and helix crossing angle ( $\Omega$ , bottom) are tracked during the first GpA replicate simulation. Vertical lines are drawn in the respective plots to highlight frames that represent the primary (*blue*, frame 4701) and secondary (*green*, frame 9311) dimer interfaces based on their polar angles. The excursions to the secondary dimer interface are discrete but brief for GpA.

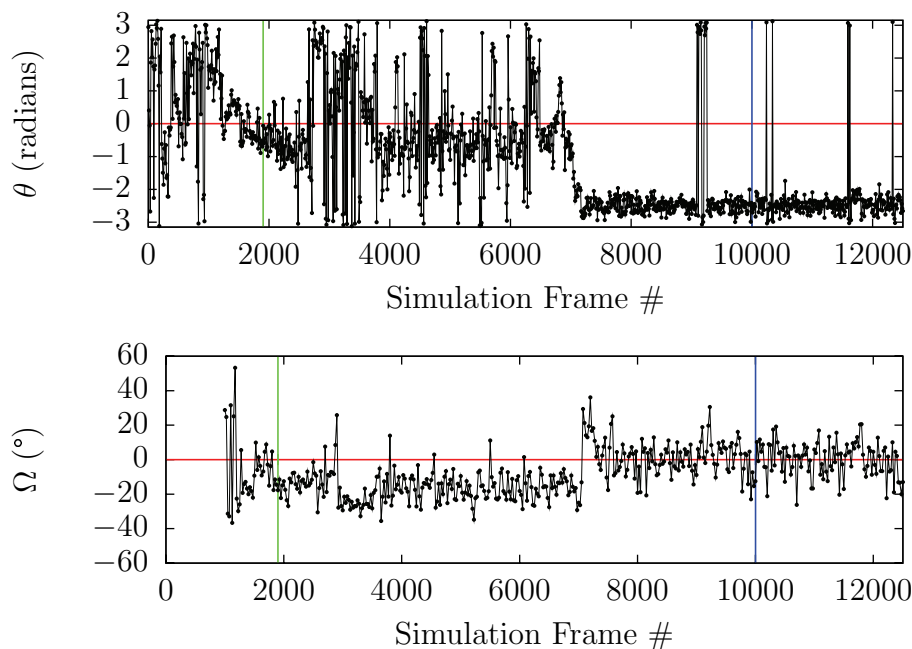


Figure 5.27: Polar angle of helix 2 ( $\theta$ , top) in the rmsd-fixed frame of helix 1 and helix crossing angle ( $\Omega$ , bottom) are tracked during the fourth FGFR3 mutant homodimer replicate simulation. Vertical lines are drawn in the respective plots to highlight frames that represent the primary (*blue*, frame 10001) and secondary (*green*, frame 1901) dimer interfaces based on their polar angles. There appears to be a transition in  $\Omega$  that coincides with the dimer interface transition.

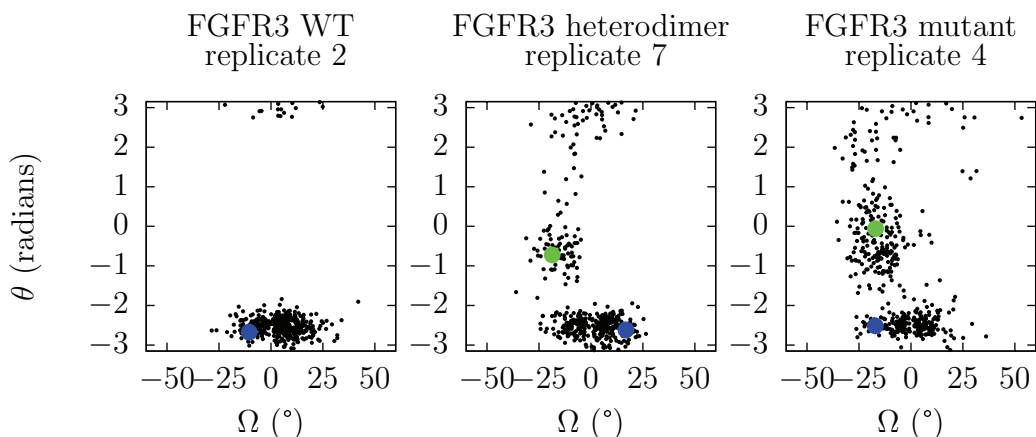


Figure 5.28: Direct correlation of polar angle ( $\theta$ ) and helix crossing angle ( $\Omega$ ) for the frames of representative FGFR3 simulations. The specific primary (*blue*) and secondary (*green*) representative dimer interface frames selected by the manual procedure are highlighted as appropriate.

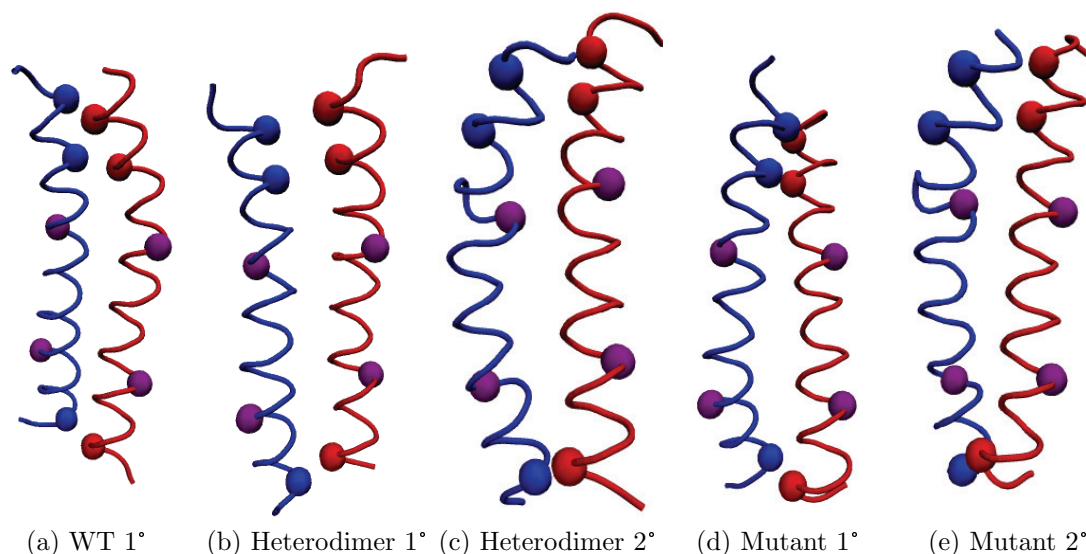


Figure 5.29: Representative coarse-grain  $C_{\alpha}$  structures (N-terminus top) for the FGFR3 primary and secondary dimer interfaces. G370, A374, and R397 (which are among the predominant interfacial contacts in Figure 5.20 on page 153) are shown in van der Waals representation, and disease-target residues G/R380 and A391 are in *purple*. Similar secondary interface structures for heterodimer (c) and mutant (e) reflect right-handed crossing angles while differences between primary interface structures reflect (in part) variation in helix crossing angles (which have a bimodal distribution at the FGFR3 primary interface as highlighted in Figure 5.28 on page 160)

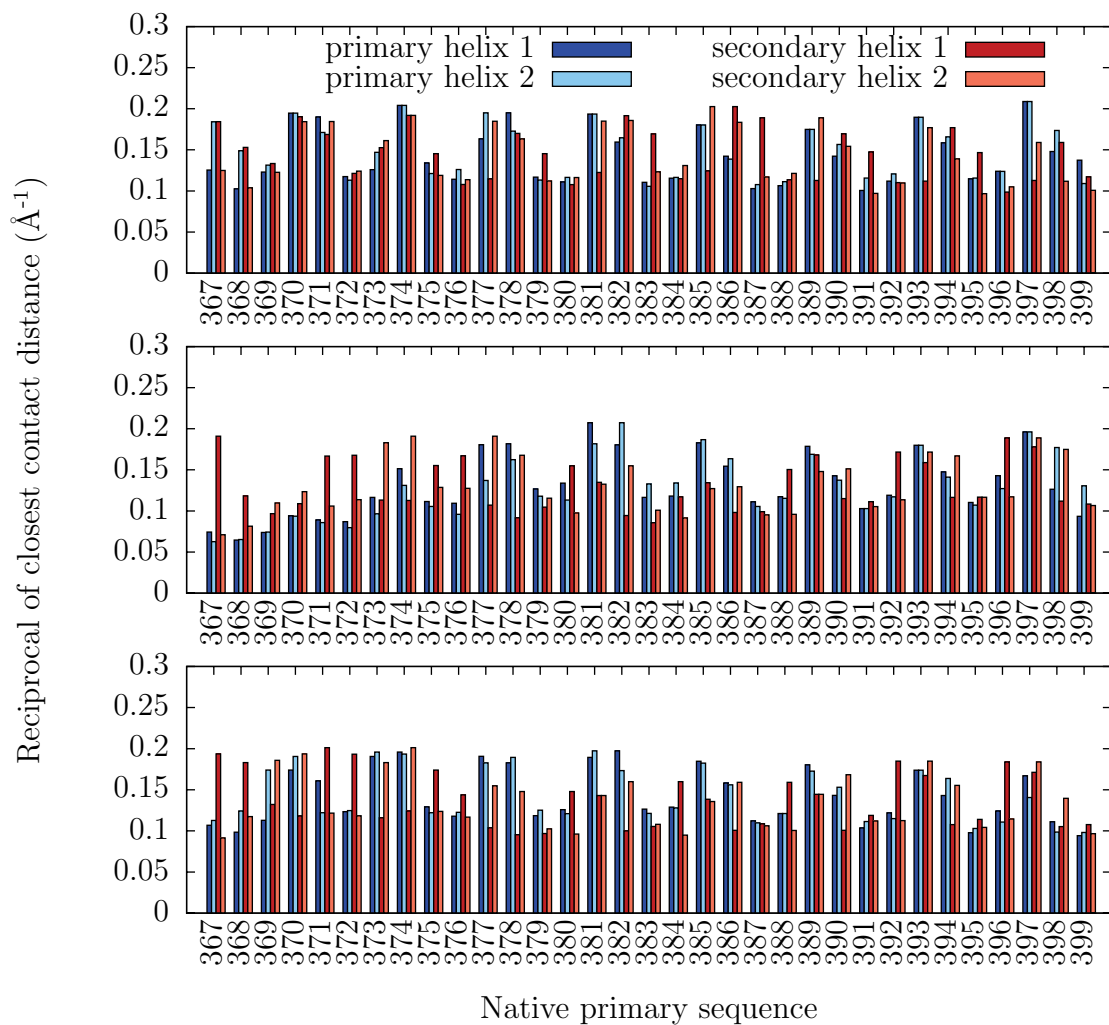


Figure 5.30: FGFR3 WT (top), heterodimer (middle), and mutant homodimer (bottom) reciprocal closest contact distances for each residue in each helix of a representative dimer interface structure.

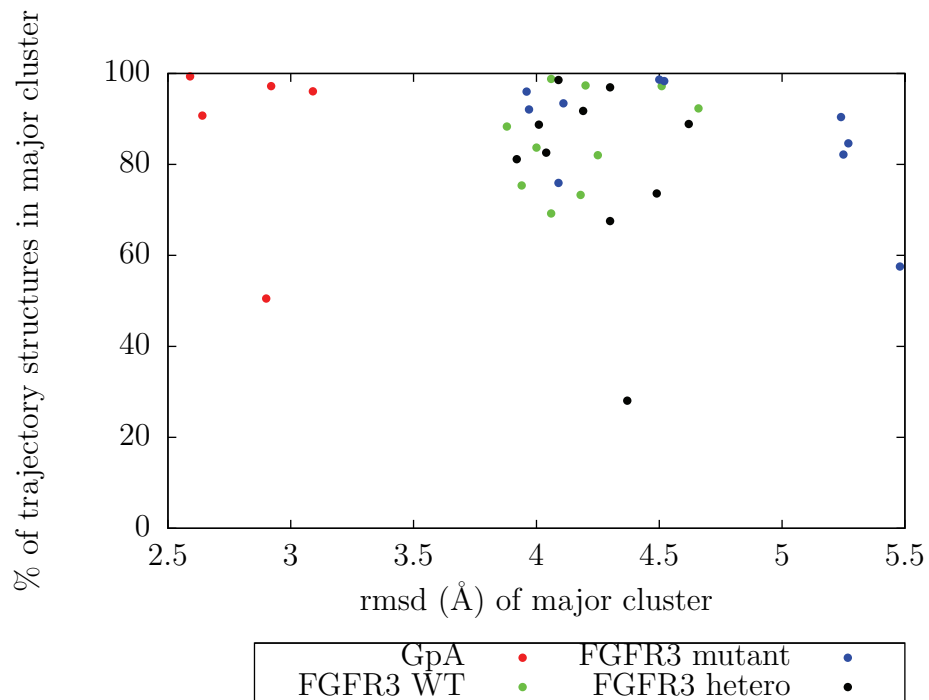


Figure 5.31: Comparing coarse-grained MD simulation trajectory structural clustering by the single linkage method (0.4 nm cutoff; every 10<sup>th</sup> frame parsed). The fifth replicate GpA simulation has a smaller % incorporation to the major cluster because one of the helices assumes an in-plane orientation during part of that simulation. An even smaller % incorporation is observed for the 9<sup>th</sup> replicate FGFR3 heterodimer simulation because of the nearly 4  $\mu$ s time required for dimerization.

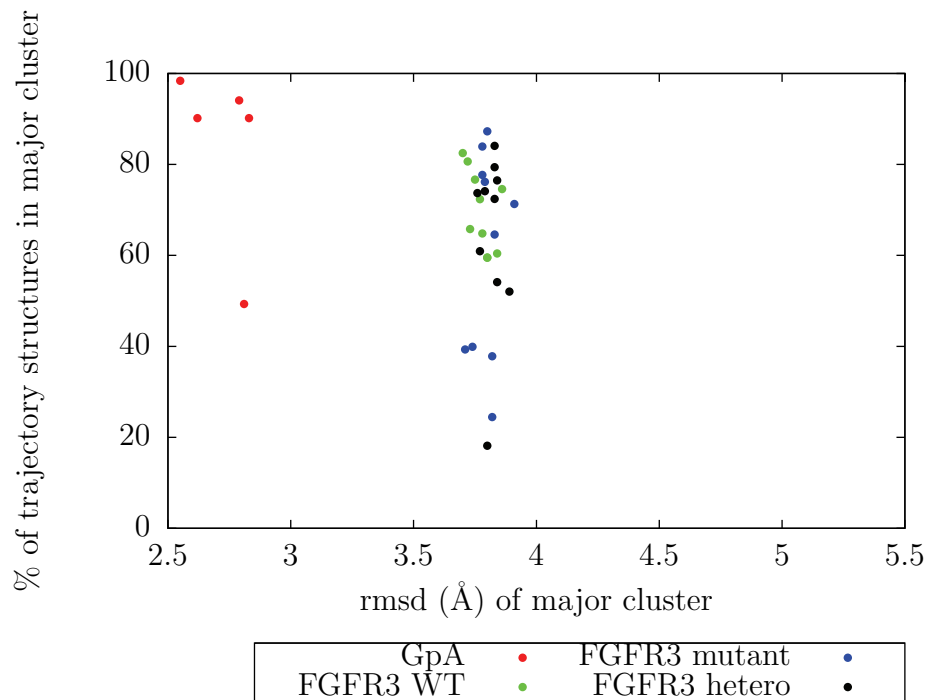


Figure 5.32: Comparing coarse-grained MD simulation trajectory structural clustering by the gromos algorithm (0.4 nm cutoff; every 10<sup>th</sup> frame parsed). The fifth replicate GpA simulation has a smaller % incorporation to the major cluster because one of the helices assumes an in-plane orientation during part of that simulation. An even smaller % incorporation is observed for the 9<sup>th</sup> replicate FGFR3 heterodimer simulation because of the nearly 4  $\mu$ s time required for dimerization.



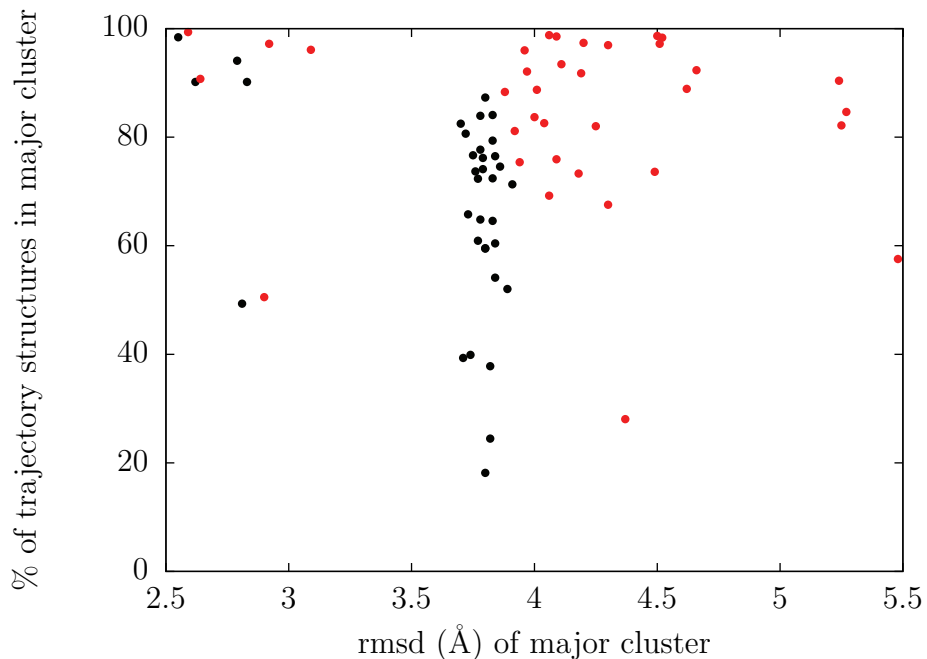


Figure 5.33: Comparing gromos (*black*) and single linkage (*red*) algorithms for clustering 35 dimer simulations involving FGFR3 or GpA (0.4 nm cutoff; every 10<sup>th</sup> frame).

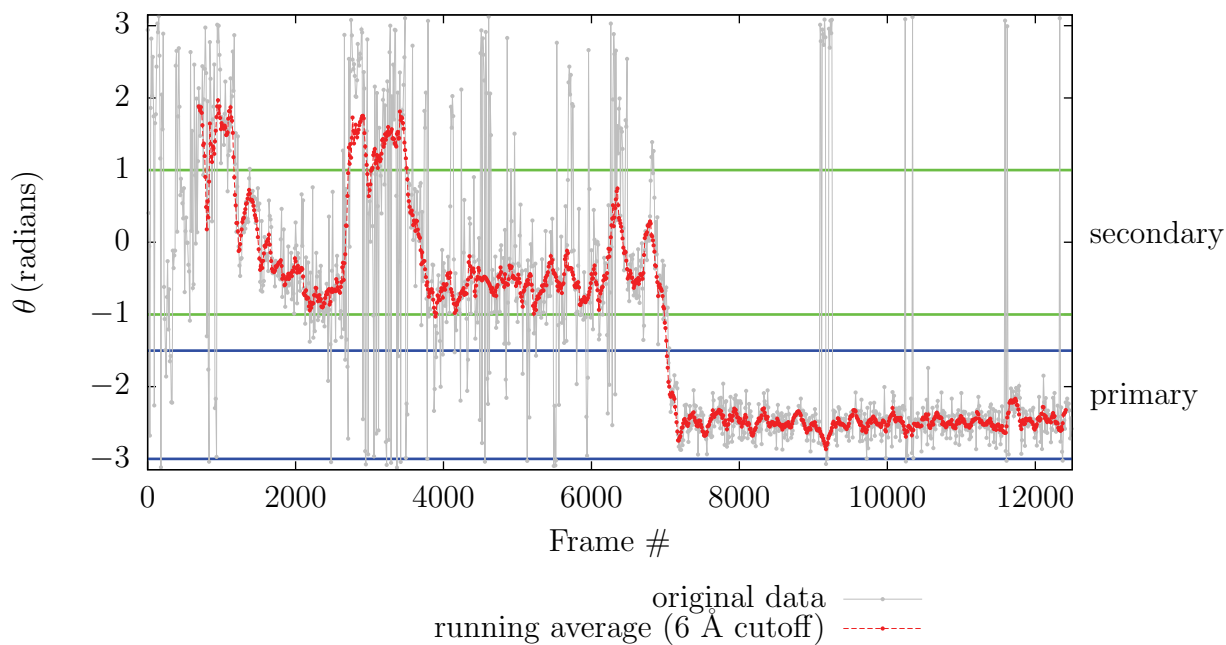


Figure 5.34: Using a weighted moving average and spike-filter to classify frames according to helix 2 COM  $\theta$  in rmsd-fixed reference frame of helix 1 in the fourth replicate FGFR3 mutant simulation.

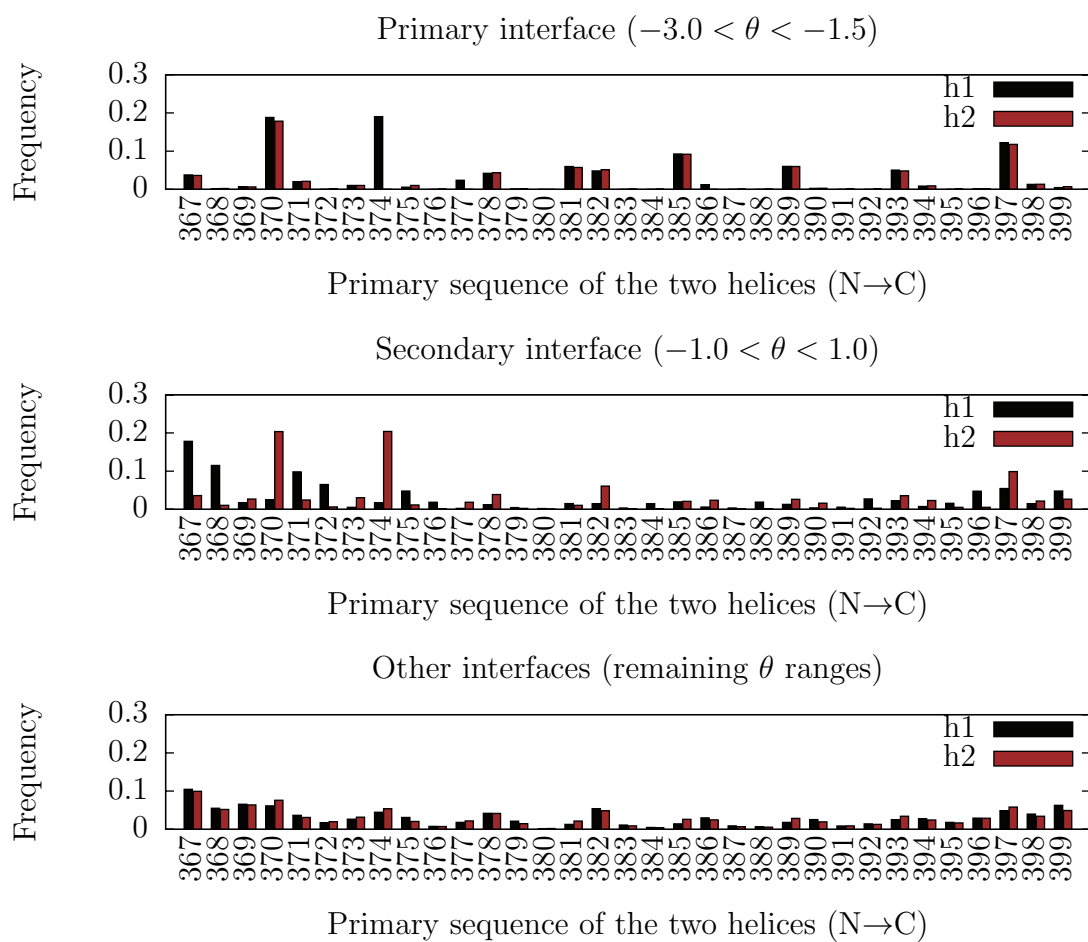


Figure 5.35: Normalized frequency of occurrence for a residue in the five closest contacts between helices when interhelix ( $C_{\alpha}$ ) separation is within 6 Å for primary, secondary and ‘other’ dimer interface populations from *all* 30 FGFR3 CG replicate simulations.

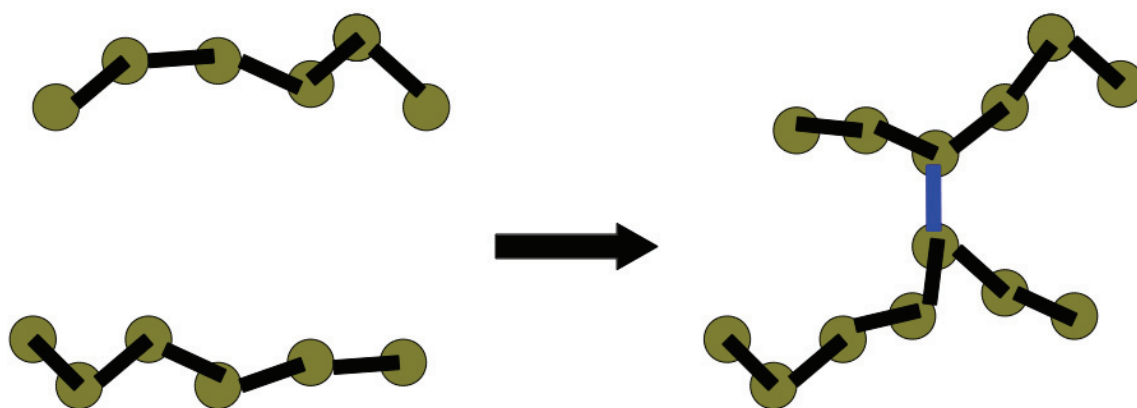


Figure 5.36: Using an MDAnalysis `networkx`-based utility, a set of phosphate particles (circles) are connected within a certain cut-off boundary to produce two clearly separated leaflets (*left*) or there is a failure to identify separate leaflets because of phosphate headgroups with an intermediate position (*right*). The latter is a concern because it may occur during a subset of frames in a simulation when a particular lipid particle is substantially perturbed away from the regular surface of the leaflet, causing failure of any measurement that depends on the identification of two separate leaflets.

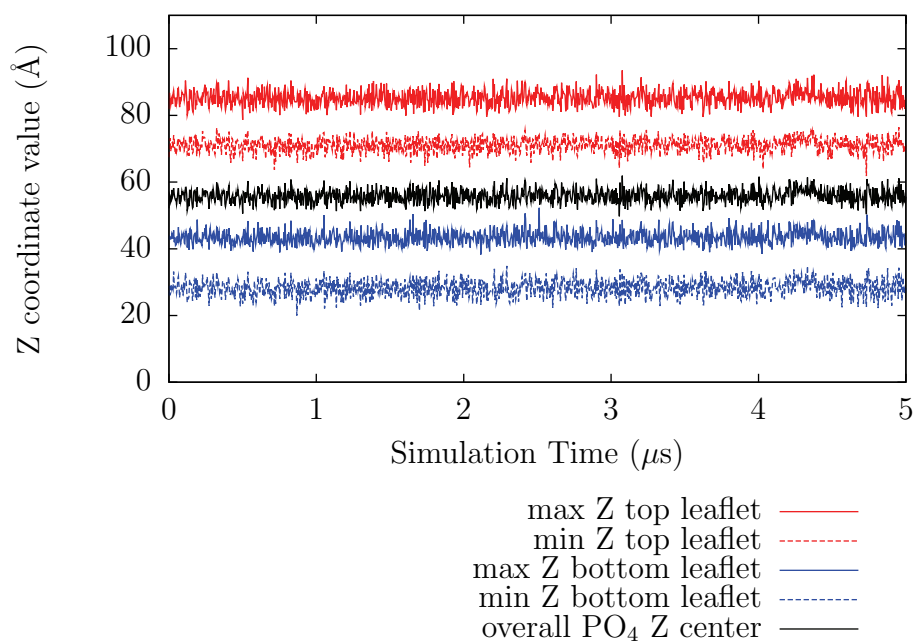


Figure 5.37: Checking for phospholipid ‘flip-flop:’ tracking POPC leaflet minimum and maximum phosphate Z coordinates for the fourth coarse-grained GpA simulation (every 10<sup>th</sup> frame). It is apparent that no phosphate particle crosses the approximate center boundary of the bilayer.

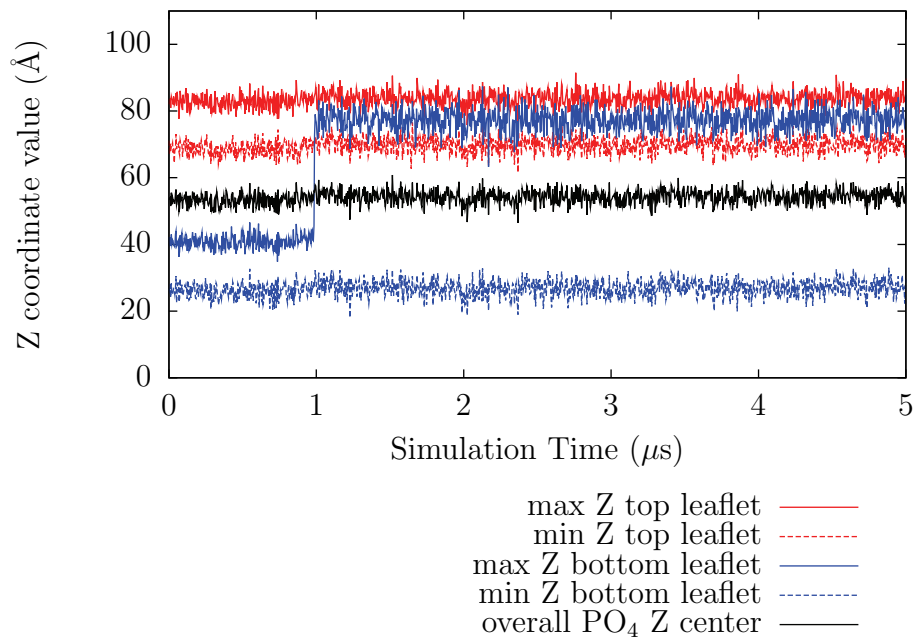


Figure 5.38: Checking for phospholipid ‘flip-flop:’ tracking POPC leaflet minimum and maximum phosphate Z coordinates for the first coarse-grained FGFR3 WT simulation (every 10<sup>th</sup> frame). It is apparent that a phosphate particle crosses the approximate center boundary of the bilayer.

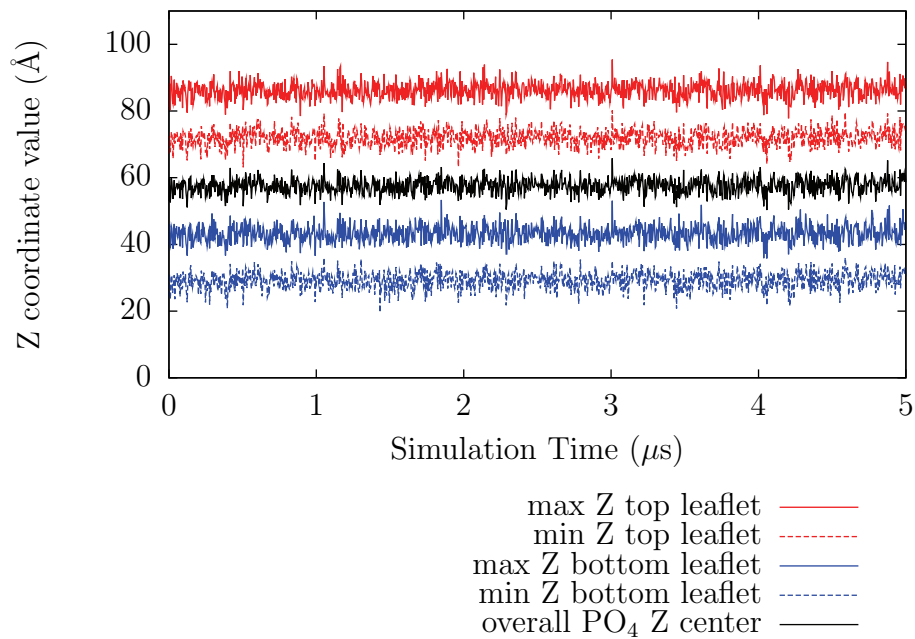


Figure 5.39: Checking for phospholipid ‘flip-flop.’ tracking POPC leaflet minimum and maximum phosphate Z coordinates for the tenth coarse-grained FGFR3 heterodimer simulation (every 10<sup>th</sup> frame). It is apparent that no phosphate particle crosses the approximate center boundary of the bilayer.

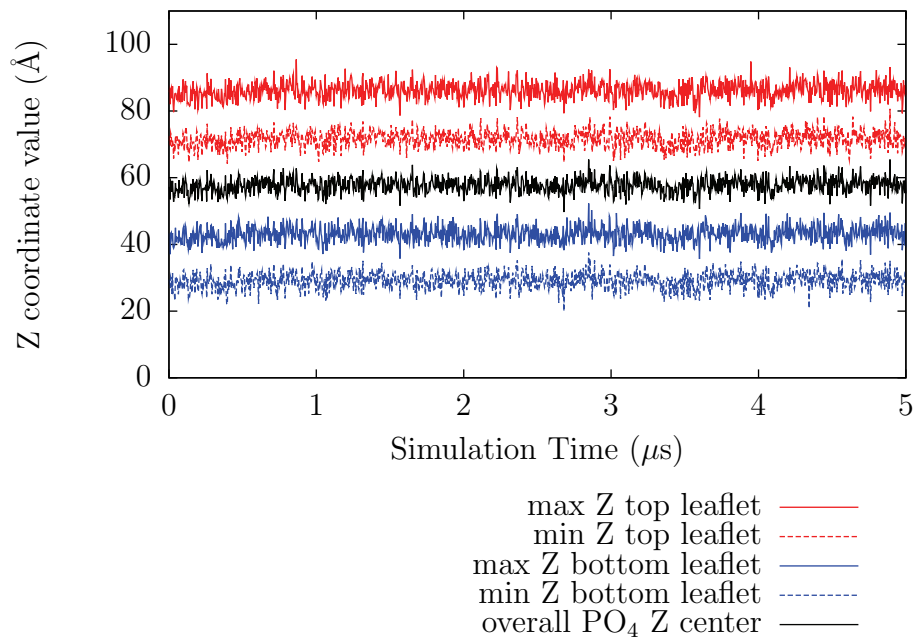


Figure 5.40: Checking for phospholipid ‘flip-flop.’ tracking POPC leaflet minimum and maximum phosphate Z coordinates for the tenth coarse-grained FGFR3 mutant homodimer simulation (every 10<sup>th</sup> frame). It is apparent that no phosphate particle crosses the approximate center boundary of the bilayer.

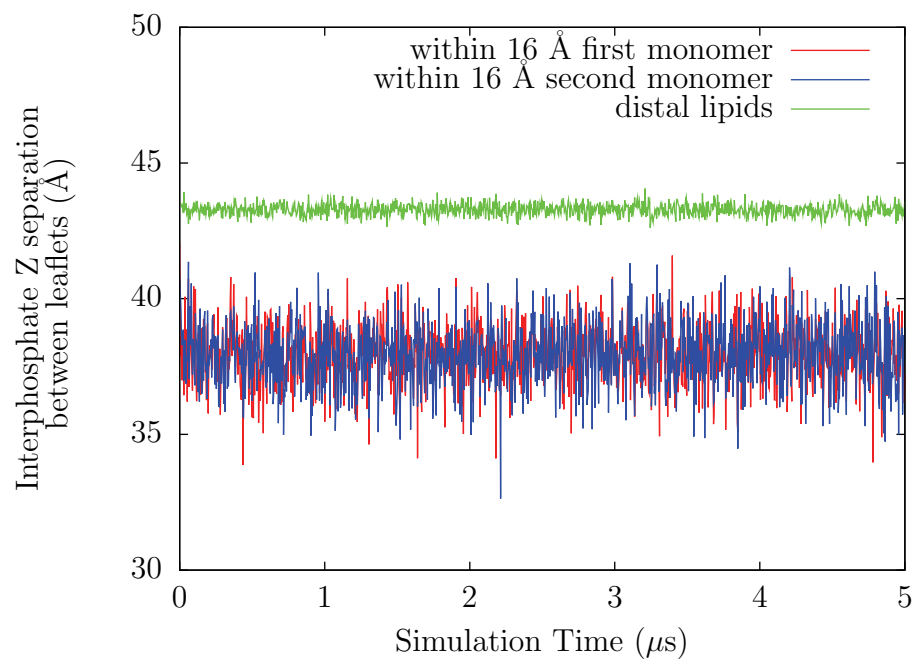


Figure 5.41: Comparing leaflet POPC interphosphate distance (bilayer thickness) for protein-local and -distal regions in the fourth coarse-grained GpA simulation (every 10<sup>th</sup> frame).

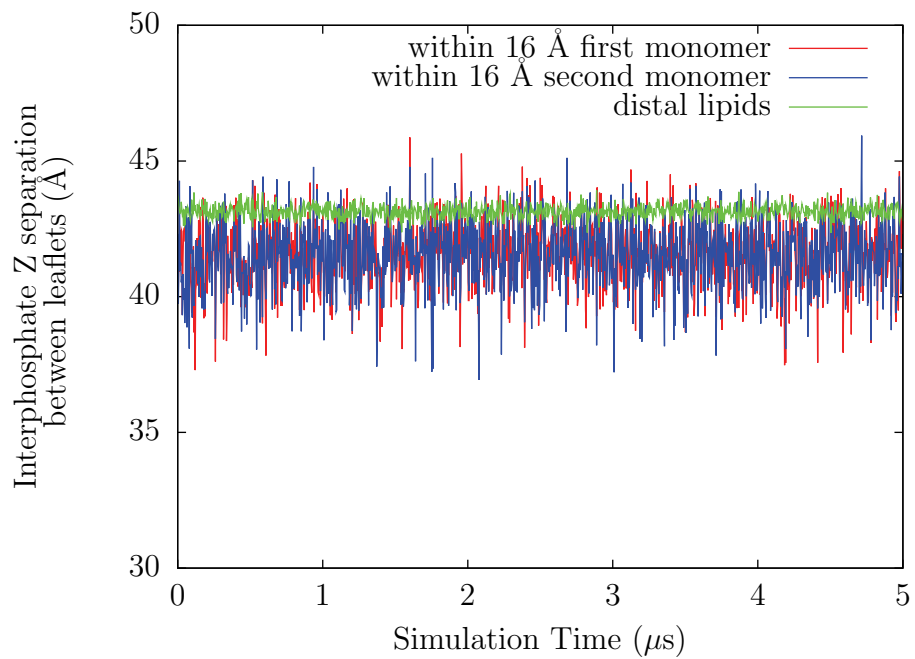


Figure 5.42: Comparing leaflet POPC interphosphate distance (bilayer thickness) for protein-local and -distal regions in the tenth coarse-grained FGFR3 WT simulation (every 10<sup>th</sup> frame).



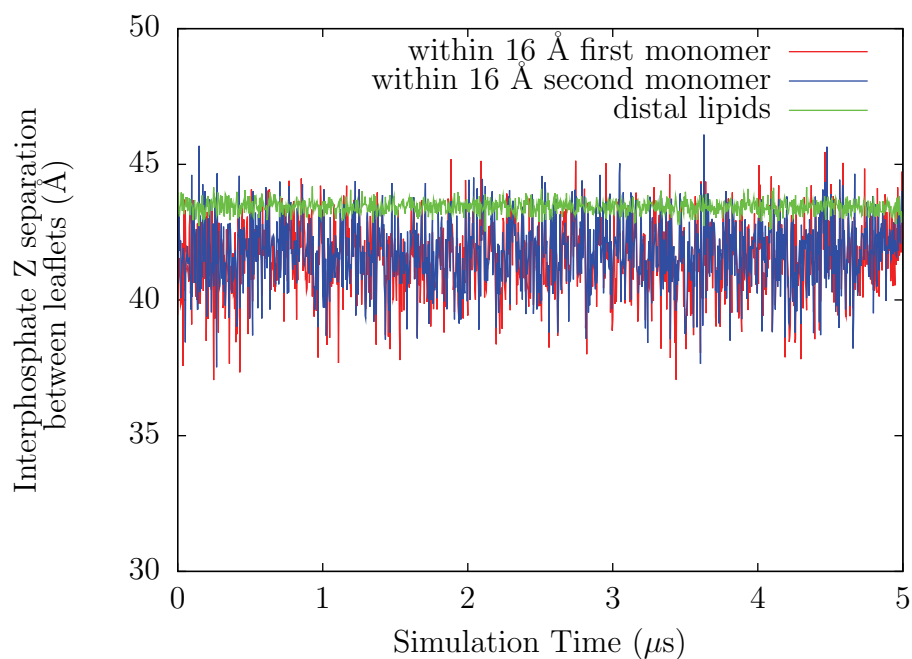


Figure 5.43: Comparing leaflet POPC interphosphate distance (bilayer thickness) for protein-local and -distal regions in the tenth coarse-grained FGFR3 heterodimer simulation (every 10<sup>th</sup> frame).

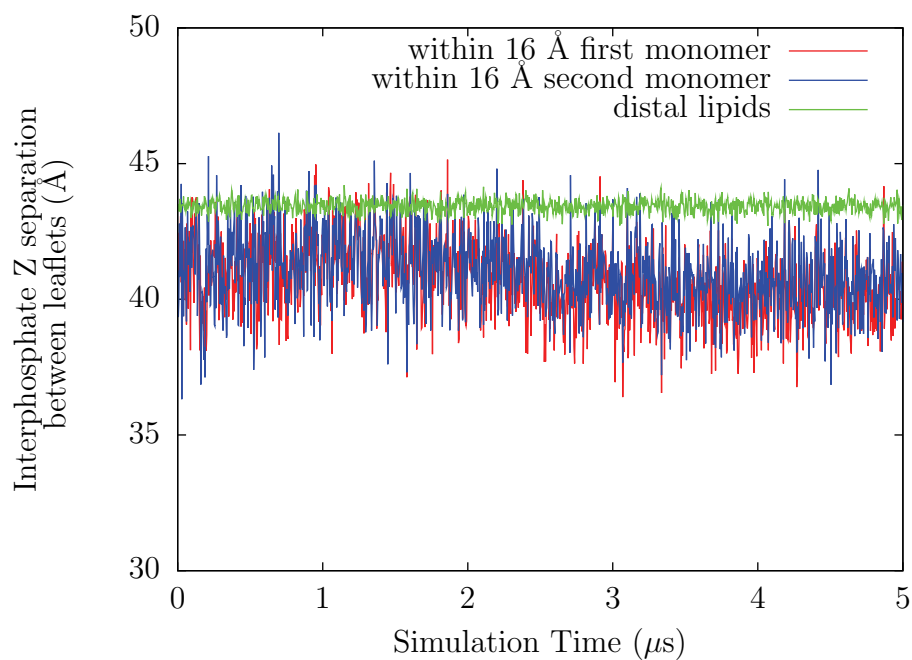


Figure 5.44: Comparing leaflet POPC interphosphate distance (bilayer thickness) for protein-local and -distal regions in the ninth coarse-grained FGFR3 mutant homodimer simulation (every 10<sup>th</sup> frame).

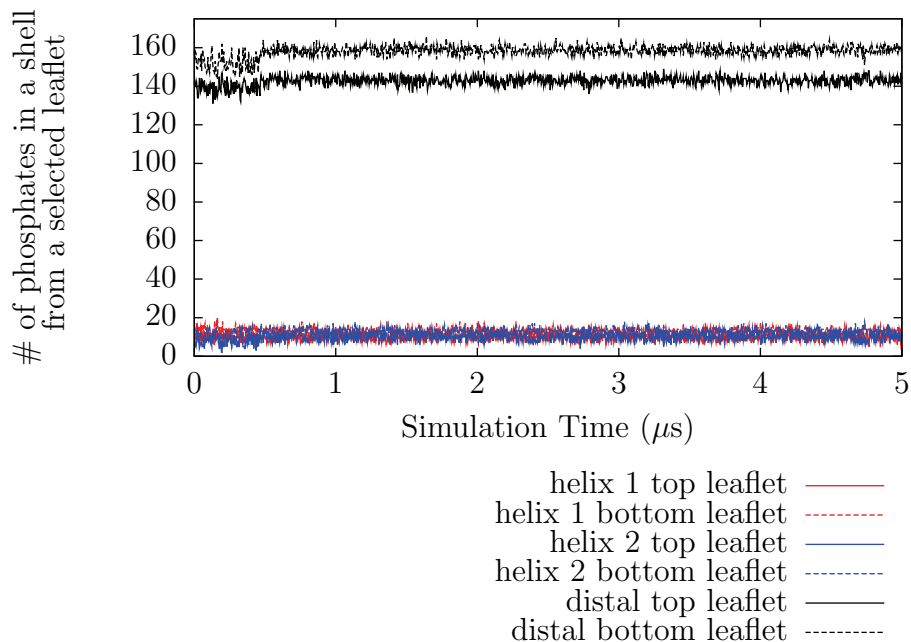


Figure 5.45: Tracking the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the third replicate GpA CG simulation (every 10<sup>th</sup> frame). The remaining (protein-distal) lipid phosphate counts are also tracked.

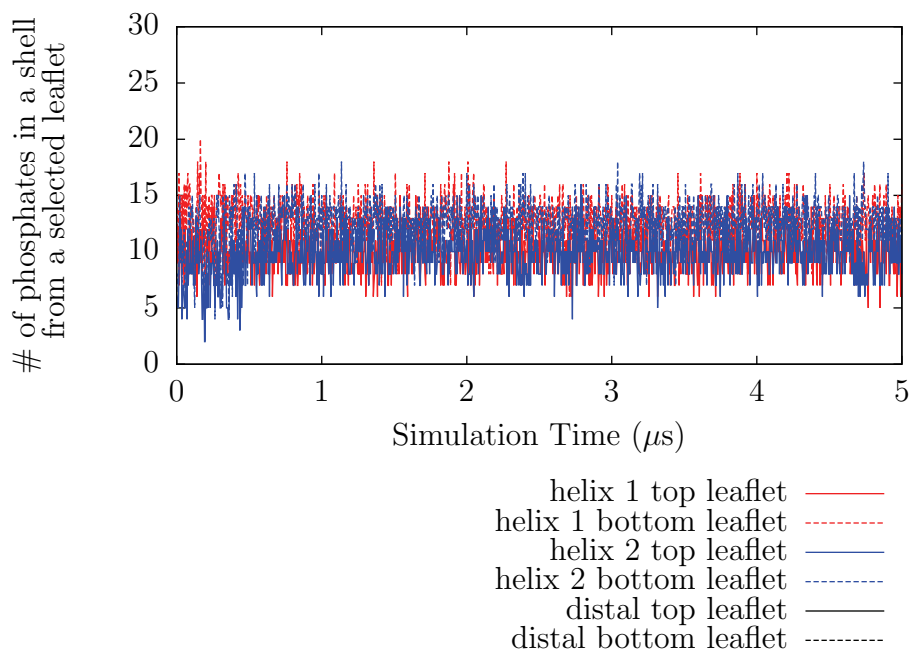


Figure 5.46: This is a zoom-in version of Figure 5.45 (page 174) focusing on the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the third replicate GpA CG simulation.

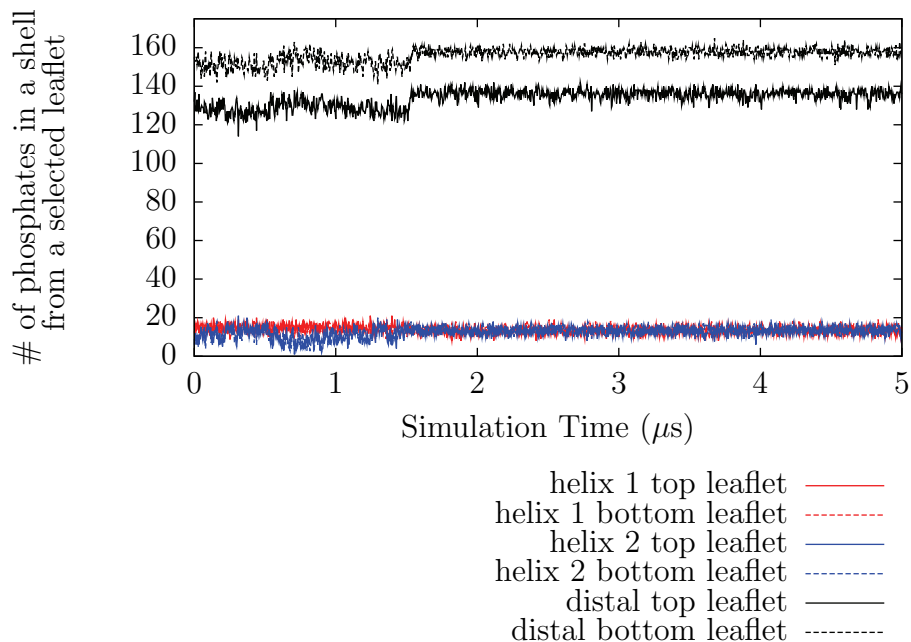


Figure 5.47: Tracking the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the seventh replicate FGFR3 WT CG simulation (every 10<sup>th</sup> frame). The remaining (protein-distal) lipid phosphate counts are also tracked.

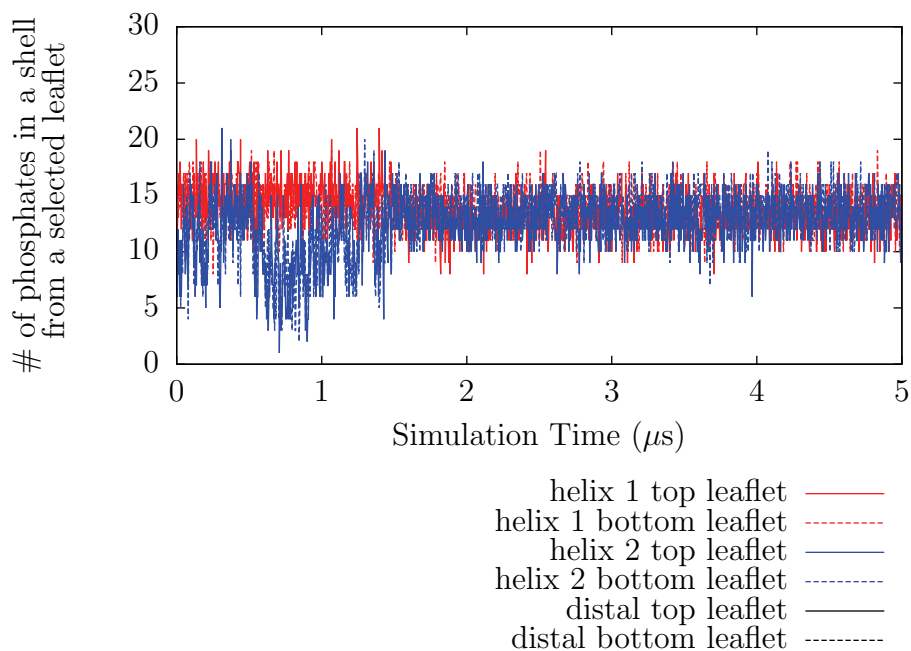


Figure 5.48: This is a zoom-in version of Figure 5.47 (page 175) focused on the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the seventh replicate FGFR3 WT CG simulation.

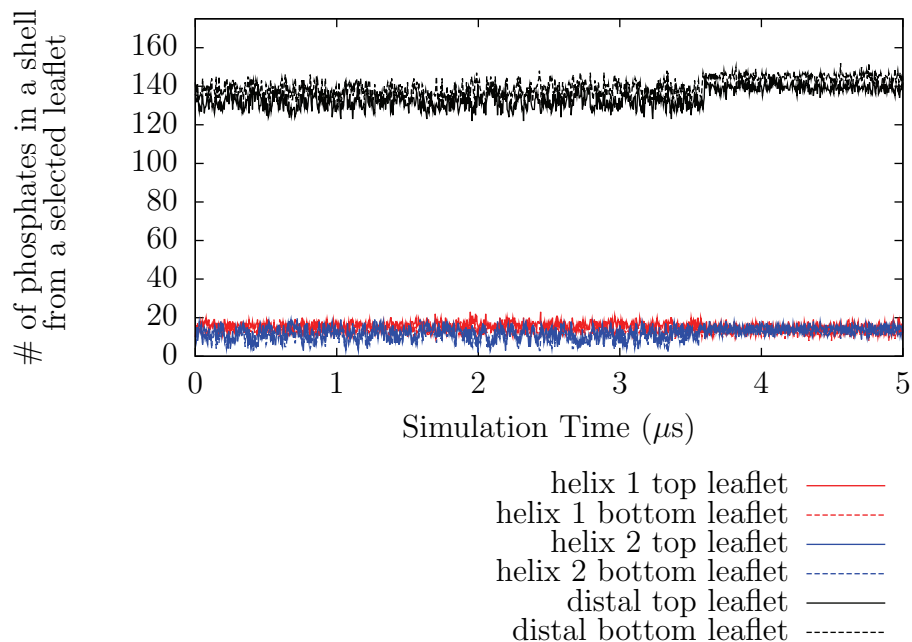


Figure 5.49: Tracking the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the ninth replicate FGFR3 heterodimer CG simulation (every 10<sup>th</sup> frame). The remaining (protein-distal) lipid phosphate counts are also tracked.

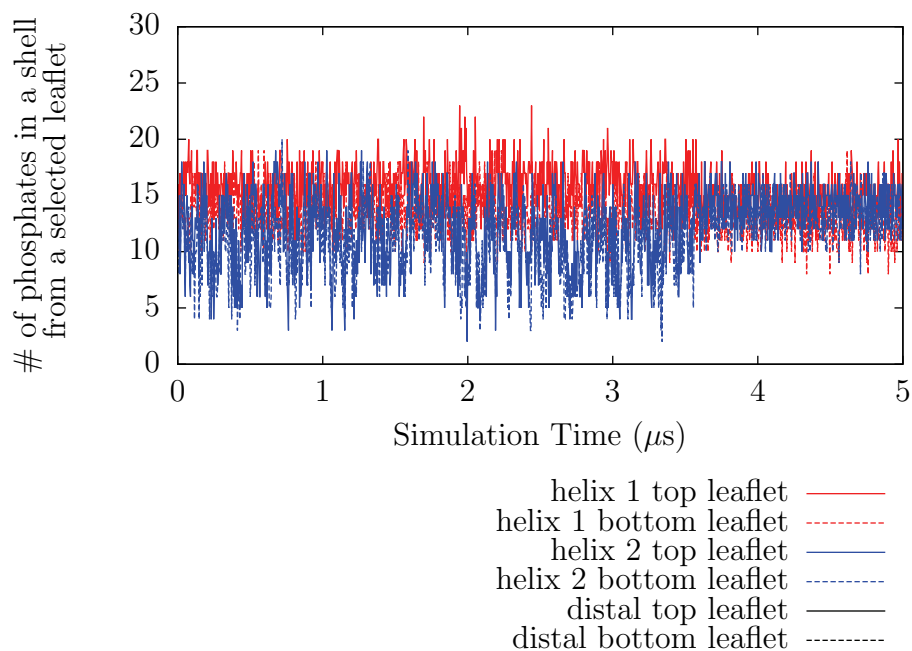


Figure 5.50: This is a zoom-in version of Figure 5.49 (page 176) focused on the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the ninth replicate FGFR3 heterodimer CG simulation.

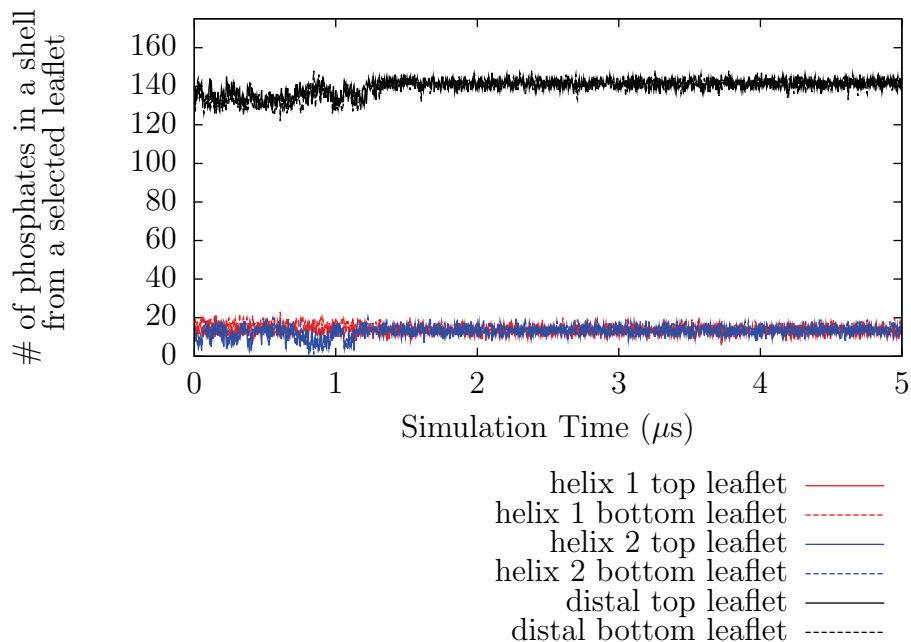


Figure 5.51: Tracking the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the seventh replicate FGFR3 mutant homodimer CG simulation (every 10<sup>th</sup> frame). The remaining (protein-distal) lipid phosphate counts are also tracked.

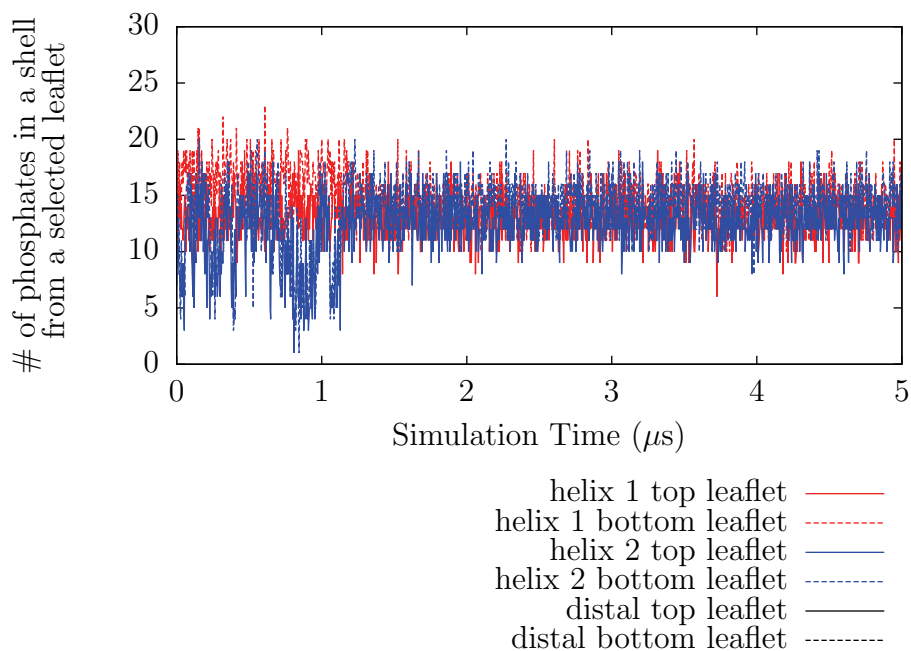


Figure 5.52: This is a zoom-in version of Figure 5.51 (page 177) focused on the number of lipid phosphates within 16 Å of either TM peptide (set of  $C_\alpha$  particles) for each bilayer leaflet in the seventh replicate FGFR3 mutant homodimer CG simulation.

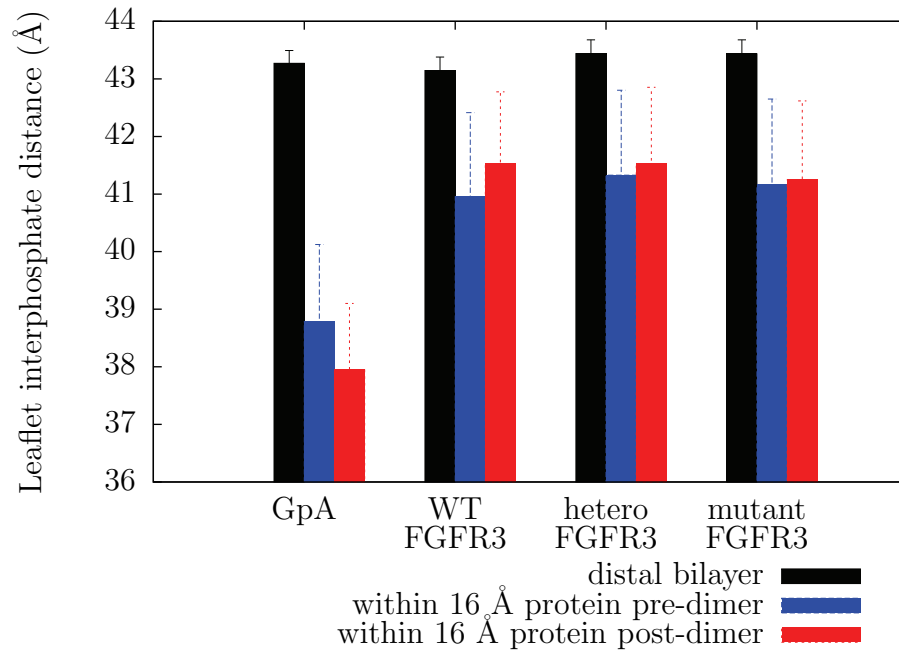


Figure 5.53: Average and standard deviation values for interphosphate bilayer thickness in GpA and FGFR3 simulation conditions. The protein-local thickness values are measured within 16 Å of the peptides and the pre- and post-dimerization divisions are based on a closest interhelical  $C_\alpha$  approach of  $< 6$  Å.

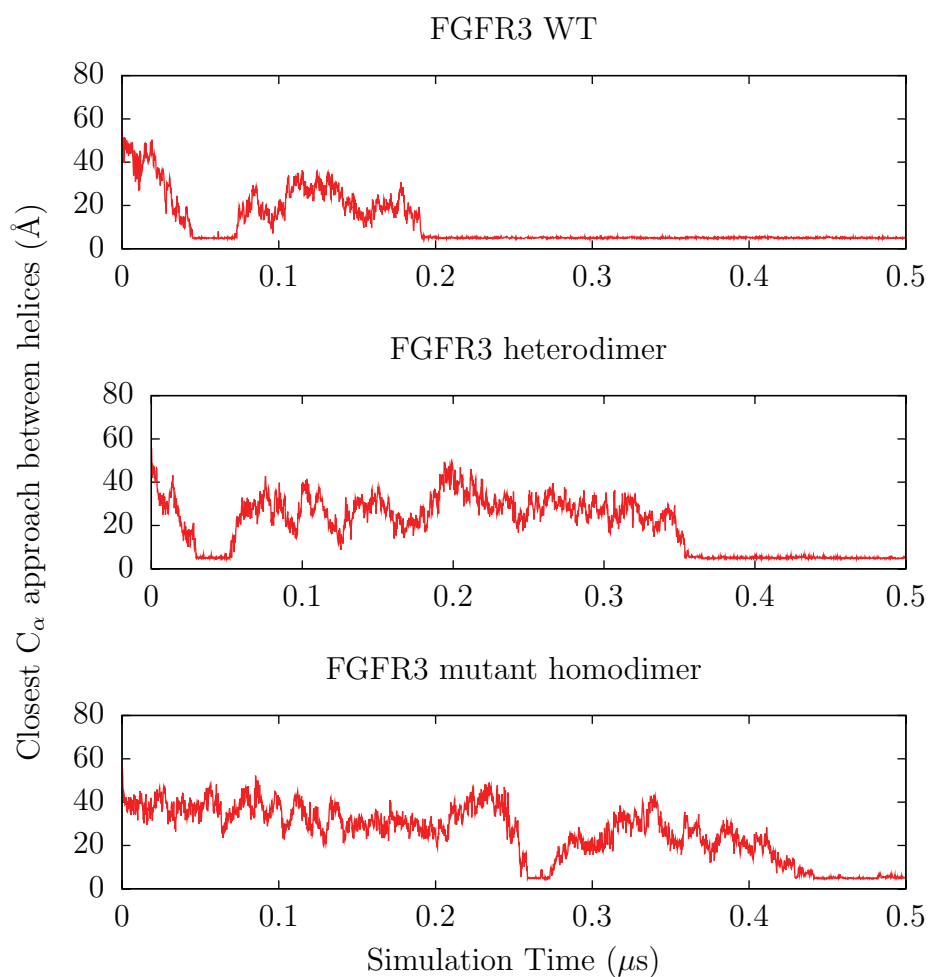


Figure 5.54: Tracking the closest  $C_\alpha$  interhelical distance for representative SIDEKICK-based CG simulations in DPPC bilayers. Dimer dissociation events are conspicuous, in contrast to the results with POPC bilayers in section 5.4.1 (page 111).

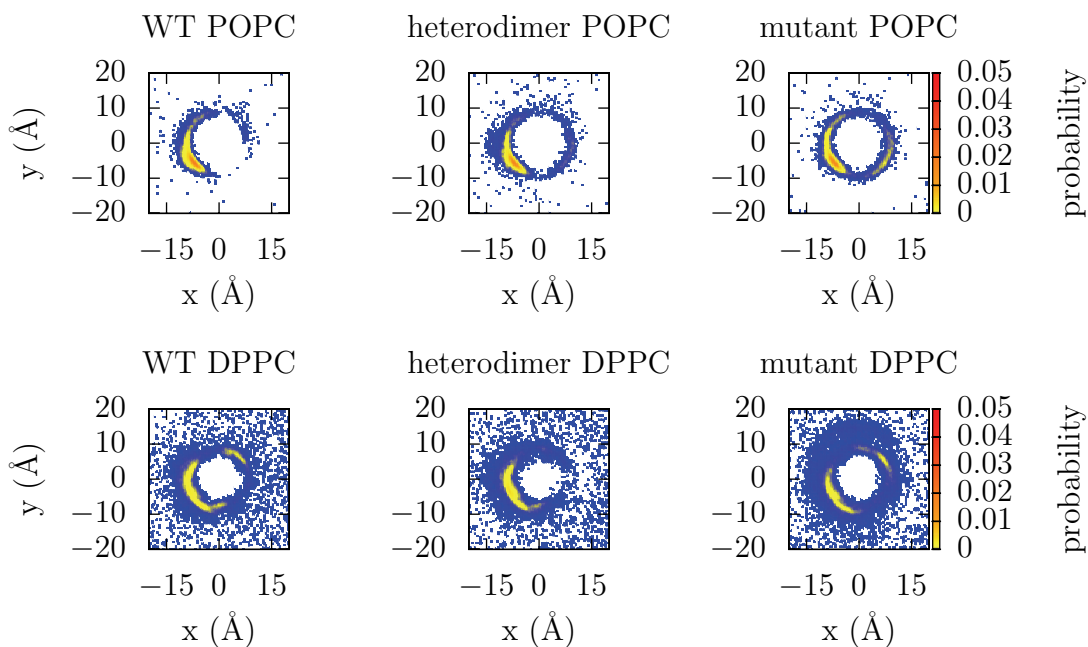


Figure 5.55: Probability map for the position of helix 2 in the rmsd-fixed reference frame of helix 1 compared for FGFR3 CG simulations in POPC (*top*) and DPPC (*bottom*). The reference structure for rmsd alignment is always the configuration of helix 1 in the first frame of the first WT POPC simulation. The non-linear probability scale is indexed as  $P = 0.0$  *white*,  $P = 0.000001$  *blue*,  $P = 0.001$  *yellow*,  $P = 0.01$  *orange*,  $P = 0.05$  *red*.



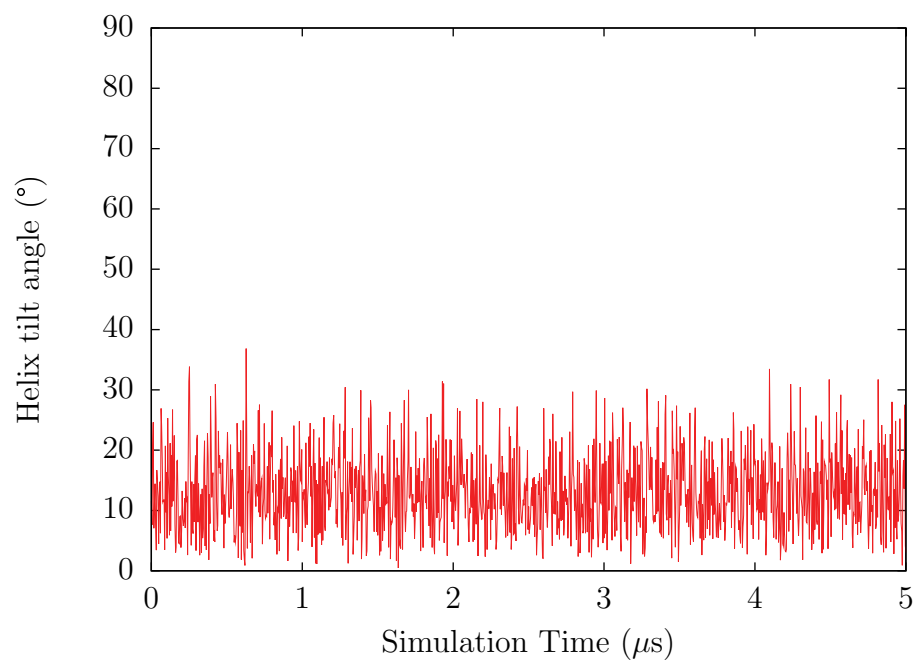


Figure 5.56: Representative plot tracking the helix tilt angle of the FGFR3 WT monomer (replicate 1) relative to the bilayer normal. The helix axis was defined as the first eigenvector of the  $C_{\alpha}$  backbone and the bilayer normal was calculated as the third eigenvector of the POPC phosphate population.

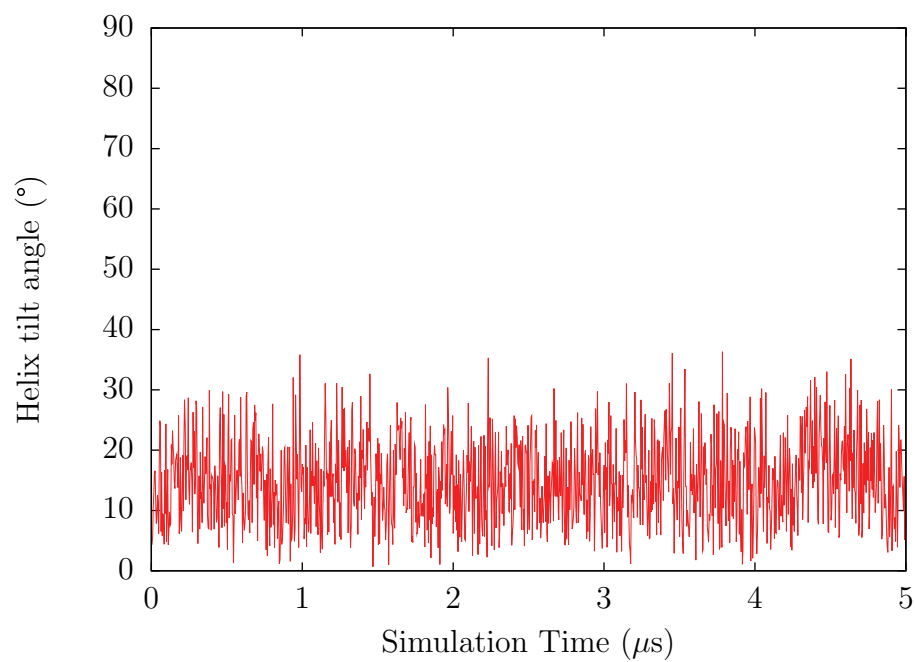


Figure 5.57: Representative plot tracking the helix tilt angle of the FGFR3 mutant monomer (replicate 1) relative to the bilayer normal. The helix axis was defined as the first eigenvector of the  $C_{\alpha}$  backbone and the bilayer normal was calculated as the third eigenvector of the POPC phosphate population.

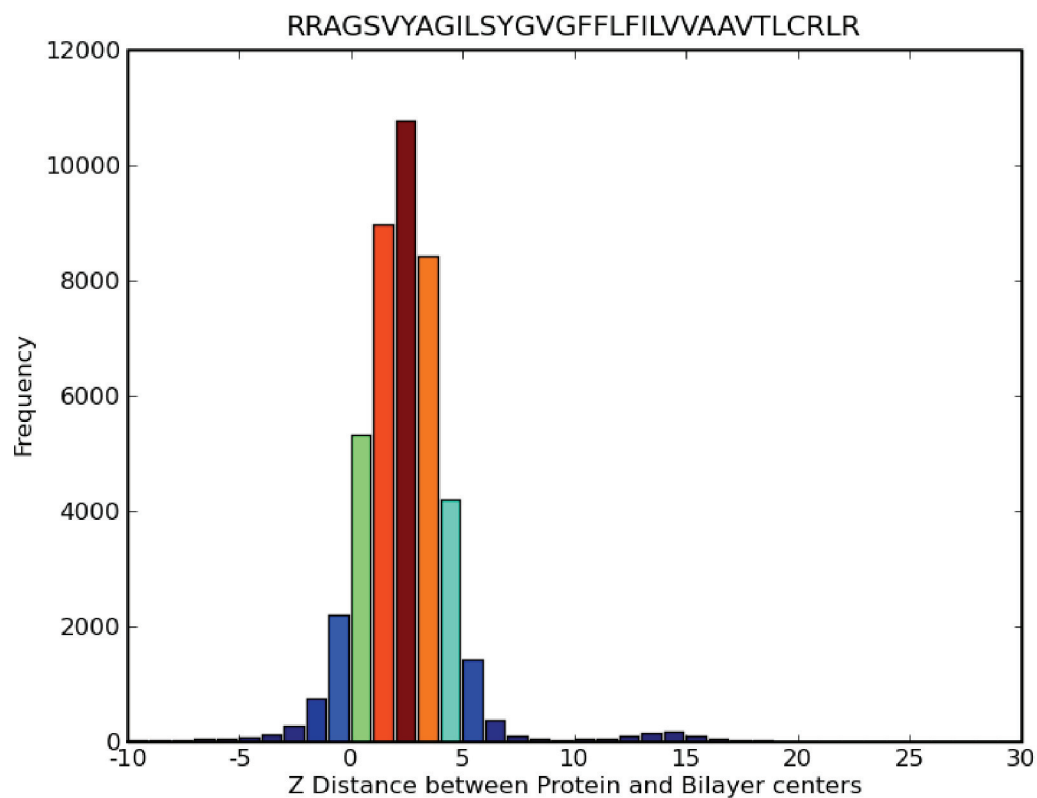


Figure 5.58: The Z separation (along POPC bilayer normal, in Å) between the center of the FGFR3 WT peptide and the center of the bilayer was recorded across 100 (0.1  $\mu$ s) replicate simulations and the frequency distribution is plotted here. The mode of the distribution is an upward (N-terminal) vertical displacement of  $\sim 3$  Å.

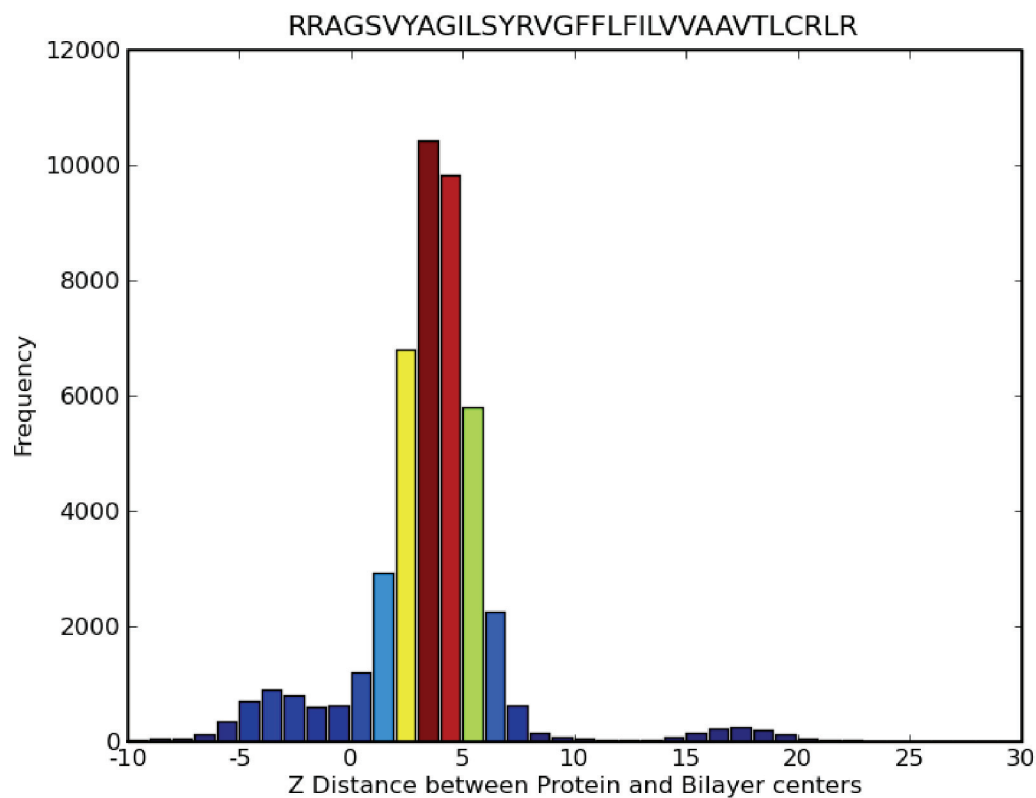


Figure 5.59: The Z separation (along POPC bilayer normal, in Å) between the center of the FGFR3 G380R mutant peptide and the center of the bilayer was recorded across 100 (0.1  $\mu$ s) replicate simulations and the frequency distribution is plotted here. The mode of the distribution is an upward (N-terminal) vertical displacement of  $\sim 4$  Å.

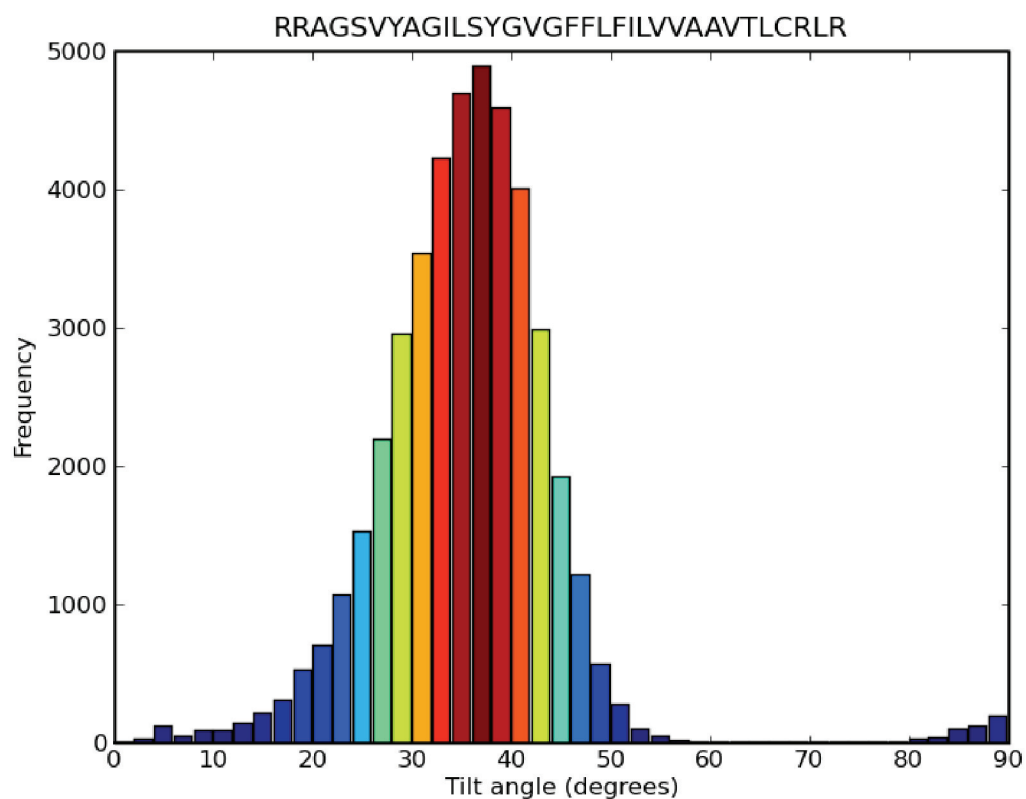


Figure 5.60: The distribution of FGFR3 WT helix tilt angles relative to the bilayer normal is plotted as frequencies accumulated over all the SIDEKICK monomer replicate simulations. The mode of the distribution is a helix tilt angle of  $\sim 38^\circ$ .

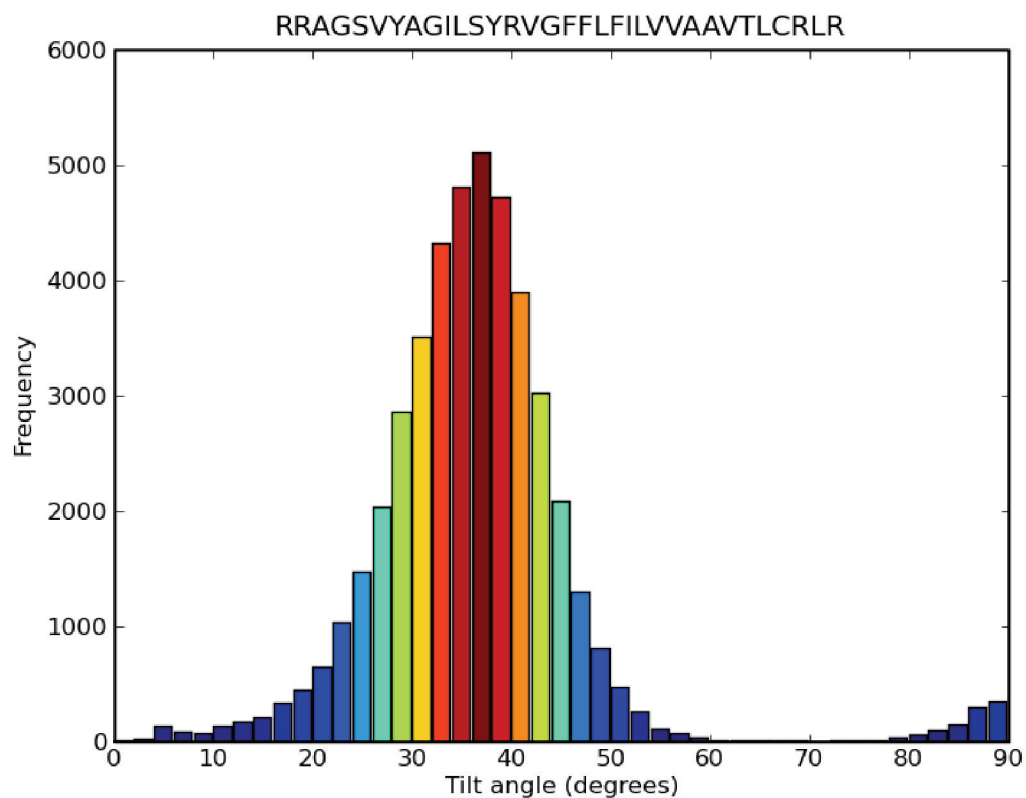


Figure 5.61: The distribution of FGFR3 G380R mutant helix tilt angles relative to the bilayer normal is plotted as frequencies accumulated over all the SIDEKICK monomer replicate simulations. The mode of the distribution is a helix tilt angle of  $\sim 38^\circ$ .

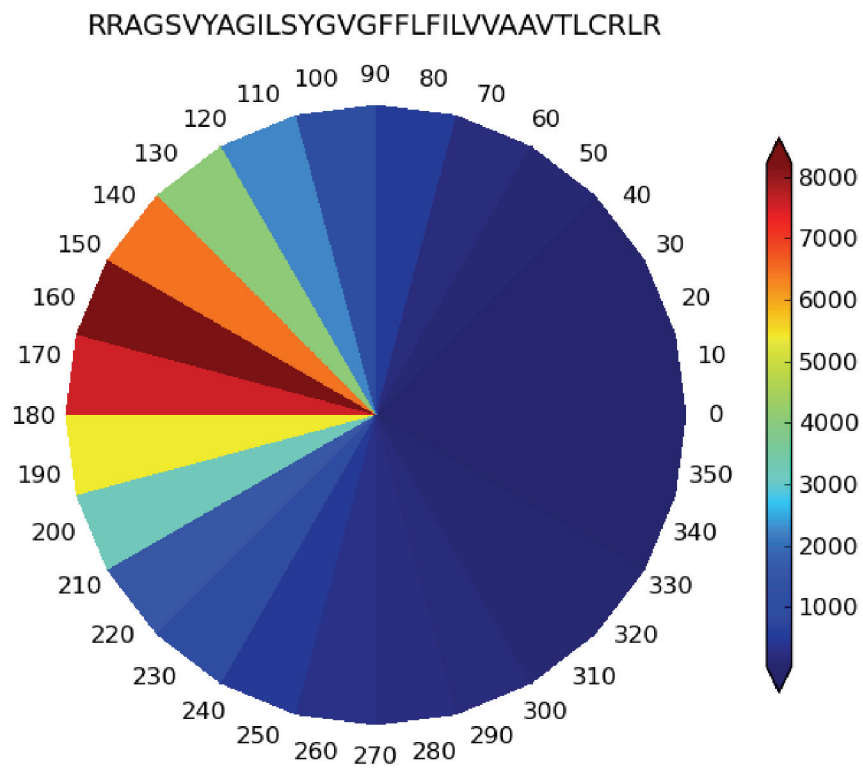


Figure 5.62: The distribution of FGFR3 WT helix rotation angles (about the helical axis) is plotted as frequencies accumulated over all the SIDEKICK monomer replicate simulations.

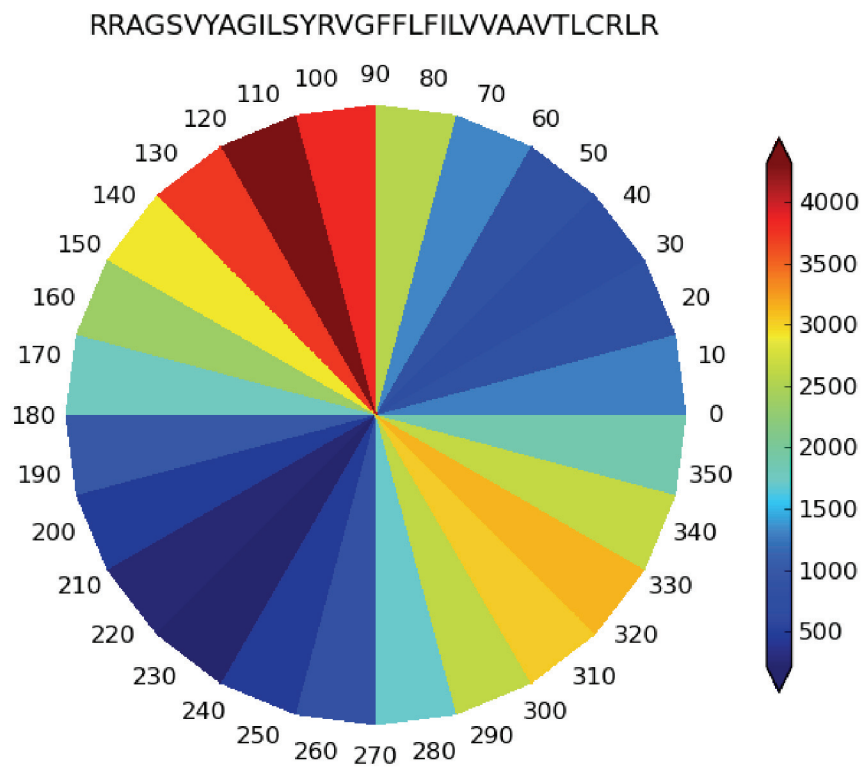


Figure 5.63: The distribution of FGFR3 G380R mutant helix rotation angles (about the helical axis) is plotted as frequencies accumulated over all the SIDEKICK monomer replicate simulations.



Simulation # (cluster used)	Description <sup>a</sup>	gen_seed <sup>b</sup>
15 (neuron at SBCB)	WT homodimer replicate #1	117
16 (neuron at SBCB)	WT homodimer replicate #2	7692
17 (neuron at SBCB)	WT homodimer replicate #3	3971
18 (neuron at SBCB)	G380R homodimer replicate #1	38
19 (neuron at SBCB)	G380R homodimer replicate #2	41916
20 (neuron at SBCB)	G380R homodimer replicate #3	5009
21 (neuron at SBCB)	Heterodimer replicate #1	4135
22 (neuron at SBCB)	WT homodimer replicate #4	19109
23 (neuron at SBCB)	WT homodimer replicate #5	6751094
24 (neuron at SBCB)	G380R homodimer replicate #4	43638
25 (mahone at ACEnet)	G380R homodimer replicate #5	13389
26 (mahone at ACEnet)	Heterodimer replicate #2	998120
27 (mahone at ACEnet)	Heterodimer replicate #3	1734
28 (mahone at ACEnet)	Heterodimer replicate #4	581905
29 (mahone at ACEnet)	Heterodimer replicate #5	133786
30 (mahone at ACEnet)	Heterodimer replicate #6	5599878
31 (mahone at ACEnet)	Heterodimer replicate #7	1991
32 (mahone at ACEnet)	Heterodimer replicate #8	2036784
33 (mahone at ACEnet)	Heterodimer replicate #9	556809289
34 (mahone at ACEnet)	Heterodimer replicate #10	13391339
35 (fundy at ACEnet)	G380R homodimer replicate #6	51561673
36 (fundy at ACEnet)	G380R homodimer replicate #7	4666283
37 (fundy at ACEnet)	G380R homodimer replicate #8	4636837
38 (fundy at ACEnet)	G380R homodimer replicate #9	3618616616
39 (fundy at ACEnet)	G380R homodimer replicate #10	447009371
40 (glooscap at ACEnet)	WT homodimer replicate #6	15242205
41 (fundy at ACEnet)	WT homodimer replicate #7	1404083761
42 (glooscap at ACEnet)	WT homodimer replicate #8	255957890
43 (fundy at ACEnet)	WT homodimer replicate #9	89887165
44 (glooscap at ACEnet)	WT homodimer replicate #10	667000009
57 (fundy at ACEnet)	GpA WT control replicate #1	395918
58 (glooscap at ACEnet)	GpA WT control replicate #2	1170019
59 (fundy at ACEnet)	GpA WT control replicate #3	669661
60 (fundy at ACEnet)	GpA WT control replicate #4	454977977
61 (fundy at ACEnet)	GpA WT control replicate #5	393105

Table 5.1: The full set of FGFR3 and GpA *dimer* replicate simulations as they were tracked during production

<sup>a</sup>all simulations involve two (33-residue FGFR3 or 23-residue GpA) peptides in POPC bilayer and water

<sup>b</sup>GROMACS starting velocity parameter

Simulation # (cluster used)	Description <sup>a</sup>	gen_seed <sup>b</sup>
47 (fundy at ACEnet)	WT monomer replicate #1	556611119
48 (fundy at ACEnet)	WT monomer replicate #2	23332
49 (fundy at ACEnet)	WT monomer replicate #3	56668990001
50 (fundy at ACEnet)	WT monomer replicate #4	2222913331
51 (fundy at ACEnet)	WT monomer replicate #5	36327
52 (neuron at SBCB)	G380R monomer replicate #1	770091336
53 (neuron at SBCB)	G380R monomer replicate #2	363695
54 (neuron at SBCB)	G380R monomer replicate #3	16443305
55 (neuron at SBCB)	G380R monomer replicate #4	17231355
56 (neuron at SBCB)	G380R monomer replicate #5	6609199

Table 5.2: The full set of FGFR3 *monomer* replicate simulations as they were tracked during production

---

<sup>a</sup>all simulations involve a single 33-residue FGFR3 peptide in POPC bilayer and water

<sup>b</sup>GROMACS starting velocity parameter

# Chapter 6

## Spitz-Rhomboid Simulation

### 6.1 Introduction

The spitz-rhomboid system was described in section 4.1 (page 58) as a prelude to describing production and purification of spitz (or homologous) TMD peptide constructs. The latter project resulted in several successfully synthesized peptides that proved extremely difficult to purify. While it was possible to collect NMR spectra for one construct, it was not possible to fully assign resonances and determine the high-resolution structure. In this chapter I use coarse-grained molecular dynamics simulations to study the interaction of *E. coli* GlpG (ecGlpG) and a spitz construct in a POPE bilayer. POPE has been used because ecGlpG is active when reconstituted in PE lipids but not in PC lipids (199). The simulation setup and analysis is very similar to that described in chapter 5 (page 108) for FGFR3. Because the methodology and source code are detailed in the previous chapter, I will not be as comprehensive in describing similar methods employed in this chapter. A simple objective of these CG-MD spitz-rhomboid studies is to determine if there is a preferential interaction face between enzyme and substrate (*i.e.*, do they preferentially interact near TM5—the putative substrate gate (148)?) Atomistic simulations have been reported for ecGlpG in POPC and POPE bilayers, but the time scales were short and substrate was not included (195). The CG-MD simulations analyzed in this chapter allow for

$\mu$ s timescales that include both enzyme and substrate in a lipid bilayer. In total, 20 replicates were performed—two sets of  $10 \times 5 \mu$ s replicates with different spitz starting positions relative to ecGlpG to discourage any association bias.

## 6.2 Tracking Enzyme-Substrate Separation

A CG representation of the 34-residue spitz TMD construct detailed in section 4.3 (page 61) was placed 70 Å from the geometric center of coarse-grained ecGlpG (PDB: 2IC8, (4)) and the closest  $C_\alpha$  interprotein separation between the two constructs was monitored. While there were replicates where spitz rapidly associated with TM5 (*i.e.*, Figure 6.1 on page 196), this was not normally observed (*i.e.*, Figure 6.2 on page 197). Thus, the spitz TMD construct does not appear to preferentially associate with TM5 despite evidence that TM5 serves as the substrate gate (148).

## 6.3 Position Of Spitz In Fixed Rhomboid Reference Frame

The previous section (6.2 on page 192) suggests that spitz does not preferentially associate with TM5 when it first encounters rhomboid, but it is cumbersome to assess the preferred position of spitz over all replicate simulations using the described plots. To gauge the position of spitz relative to each ecGlpG TM segment over all replicate simulations, the configuration of ecGlpG was rmsd-fixed to a reference structure and the positional probability of the geometric center of the spitz TMD construct was monitored in this reference frame. To avoid bias, this analysis was performed for ten replicate simulations where spitz was placed nearer to ecGlpG TMs 1 and 3, or in the opposite corner and nearer TM5 but the same distance from the geometric center of ecGlpG (Figure 6.3 on page 198). Although placing spitz nearer TM5 did increase the likelihood of association at that location, the preferred location of interaction was near ecGlpG TM1 for both starting positions. It may be possible that initial interaction (capture) between enzyme and substrate occurs near TM1 even if the actual gate is

on the other side of the enzyme. In addition, both starting configurations clearly allow for interaction of spitz with various rhomboid TM segments, and this sampling is consistent with an unbiased simulation setup.

## 6.4 Analysis Of The POPE Lipid Bilayer

The previous section (6.3 on page 192) provides evidence for unbiased simulation conditions and a potential enzyme-substrate interaction site at ecGlpG TM1. However, it is not clear how the lipid bilayer is influenced by the presence of the enzyme, the substrate, and their mutual interaction. As described above, ecGlpG activity is sensitive to the lipid headgroup type when reconstituted, and ecGlpG atomistic simulations suggest a  $\sim 4$  Å thinning of the bilayer in the vicinity of the enzyme (but not the substrate) (195). Before investigating the protein-local and -distal bilayer thickness, I tested for spontaneous phospholipid flip-flop between bilayer leaflets to ensure that I could use the strategy for bilayer thickness analysis detailed in section 5.5 on page 126 for FGFR3. None of the ten tested replicates (spitz starting near ecGlpG TMs 1 and 3) exhibited flip-flop activity (not shown). I also validated that counting lipids in a local shell within 16 Å of the proteins captured a sufficiently large number of phosphates to measure the leaflet interphosphate bilayer thickness proximal to the proteins (not shown). The average protein-local (within 16 Å) and -distal bilayer thickness values are summarized in Figure 6.4 on page 199. The results are reasonably consistent with the  $\sim 4$  Å proximal bilayer thinning reported for ecGlpG in atomistic bilayers (195), while less bilayer thinning is observed near spitz until it associates with rhomboid. The larger standard deviation observed for spitz-proximal bilayer thickness relative to ecGlpG is consistent with a smaller number of captured (local) lipids in the former versus the latter case because of their vast size difference. There is no additional thinning of the bilayer near ecGlpG after association with spitz, despite the suggestion in (195) that this may occur.

## 6.5 Identification Of Predominant Interprotein Contacts

There is additional confidence in the quality of the CG simulations after confirming that they are unbiased and that the CG lipids reflect the protein-local behaviour observed in an atomistic context. I began a more in-depth analysis of the interaction between the spitz construct and ecGlpG by parsing out the predominant interprotein contacts from a set of replicate simulations as detailed for FGFR3 in section 5.4.5 (page 116). The results are summarized for each of the spitz construct residues from the first ten replicate simulations (spitz starting near ecGlpG TMs 1 and 3) in Figure 6.5 on page 200. The predominant contacts are located at the N- and C-termini and around the putative **ASIASGA** consensus sequence (200). It should be noted that the substrate consensus sequence for cleavage is now known to be substantially less specific (141).

Having identified spitz construct residues which interact with ecGlpG, the reciprocal question arises—which residues on ecGlpG interact with the substrate? Using the same methodology for the much larger rhomboid construct produces a cumbersome set of contact probability plots for each of the TM segments and loops (not shown). A more natural approach includes a third dimension to indicate the bilayer burial depth of the residue because this information helps gauge the topological (rather than merely sequential) context within ecGlpG. A representative contact plot for the first ecGlpG helix is shown in Figure 6.6 on page 201, and it is a common theme among most ecGlpG TM segments that the N- and C-terminal residues are predominantly involved in close contacts with the substrate. This is not surprising given that the spitz construct mirrors—with N- and C-terminal residues primarily involved. It was also striking that helix 4 (which contains the catalytic Ser) had almost no contacts with the substrate (not shown), and this is consistent with its protected position in the center of the protease. The full set of ecGlpG top contacts (with the spitz TMD

construct) are summarized in the structure in Figure 6.7 (page 202). It is clear that the predominant interfacial contacts are located at the N- and C-terminal ends of TM segments and in loops between TM segments.

## 6.6 Conclusions

Despite recent evidence that ecGlpG TM5 serves as the substrate gate (148, 149), the CG-MD simulations indicate that a preferential interaction may occur between spitz and ecGlpG TM1. The spitz TMD construct was able to sample several ecGlpG interaction faces—consistent with an unbiased simulation setup regardless of the precise starting configuration. Protein-lipid interactions were investigated because of the sensitivity of the enzyme to headgroup type, and the  $\sim 4$  Å bilayer thinning we observe near ecGlpG is consistent with reported atomistic simulations (195), providing confidence in the retention of crucial behaviours despite the simplifications introduced by the CG model. The CG approach provides the unique opportunity to probe the interactions between enzyme and substrate, and my analysis suggests that the predominant interfacial contacts are at the N- and C-terminal ends of ecGlpG TM segments and in loops connecting the TM segments. The latter results were consistent with predominant interfacial residues on the substrate located at the N- and C-termini. The initial capture of substrate by ecGlpG may thus depend on interactions in the juxtamembrane region rather than more central TM residues, and this is consistent with substrate cleavage sequence competency defined near the N-terminal juxtamembrane region (of type I TM proteins) (141). Additional analysis will include the contact probability filtered by polar angle (see section 5.4.9 on page 123), adaptive Poisson-Boltzmann Solver (APBS) electrostatic analysis, and assessment of ecGlpG rmsd/rmsf during the CG simulations.

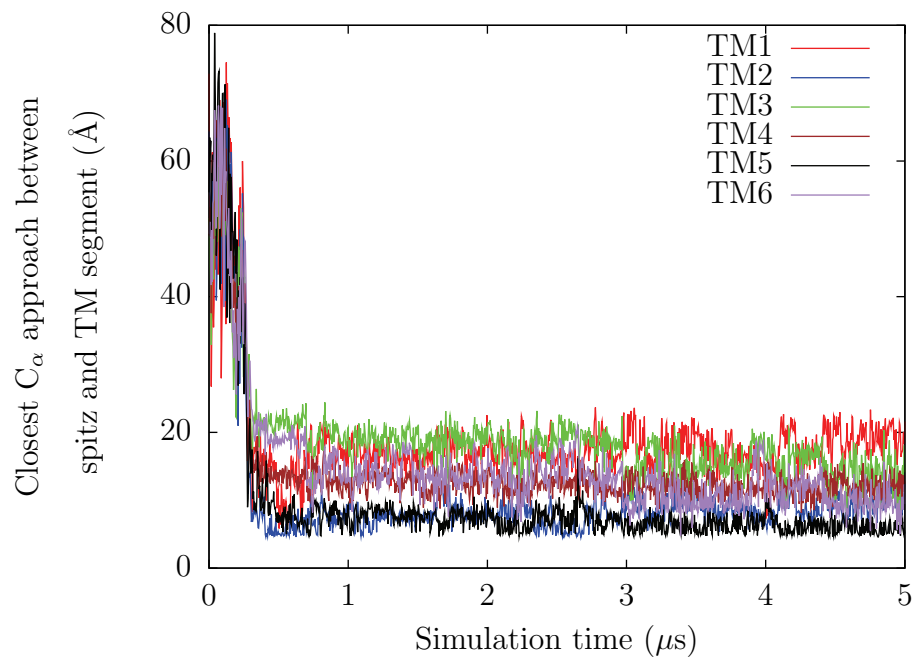


Figure 6.1: The closest interprotein  $C_\alpha$  separation monitored between spitz and ecGlpG TM segments (as defined in the crystal structure reported in (4)) during the fourth replicate coarse-grained simulation. In this case, spitz associates with TM5 early in the simulation, but this is certainly not observed in most replicates.



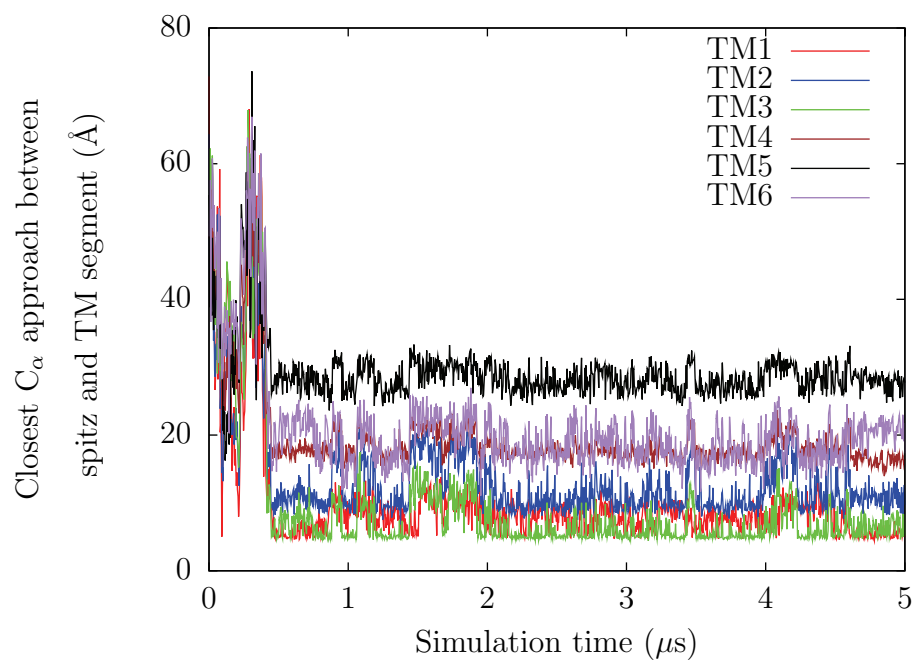
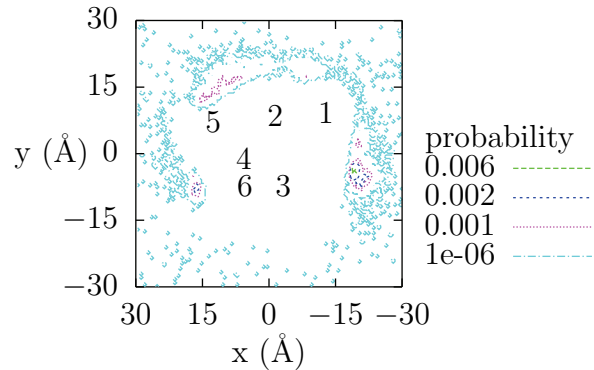


Figure 6.2: The closest interprotein  $C_{\alpha}$  separation monitored between spitz and ecGlpG TM segments (as defined in the crystal structure reported in (4)) during the first replicate coarse-grained simulation. In this case, spitz does *not* associate with TM5 early in the simulation, as observed in most replicates.

spitz starting near rhomboid TMs 1 and 3:  $10 \times 5 \mu\text{s}$  replicates total



spitz starting near rhomboid TM 5:  $10 \times 5 \mu\text{s}$  replicates total

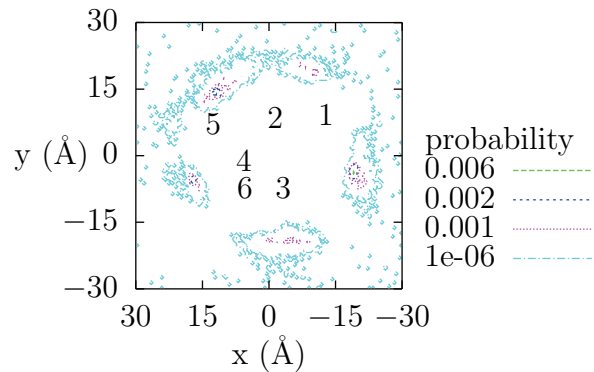


Figure 6.3: The positional probability of the geometric center of the spitz TMD construct is monitored in an rmsd-fixed reference frame (that of rhomboid in the first frame of the first replicate simulation). The ecGlpG TM segment geometric centers are indicated. To avoid any association bias, the starting configuration of spitz was nearer TMs 1 and 3 (top) or TM5 (bottom). Although placing spitz nearer TM5 at the start of the CG simulations appears to increase the likelihood of association at that location, the preferred location of interaction is consistently near TM1. Furthermore, in each case the spitz construct is clearly able to sample multiple interaction faces with rhomboid, and this is consistent with an unbiased interaction between the constructs.

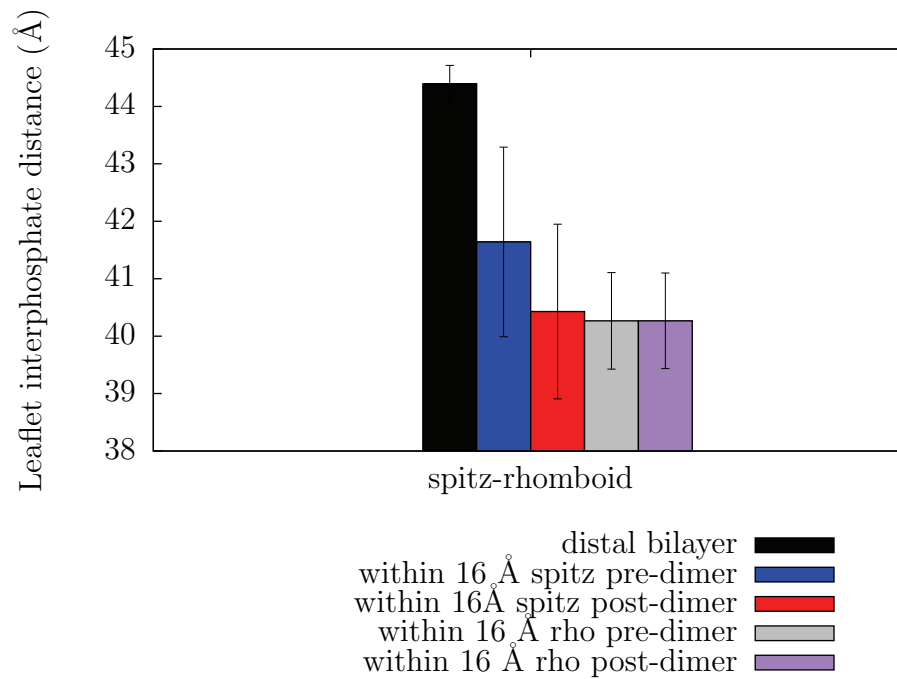


Figure 6.4: Average protein-local and -distal bilayer thickness before and after spitz-rhomboid association (6 Å cutoff) for the ten replicate simulations where spitz starts near ecGlpG TMDs 1 and 3. Error bars display one standard deviation from the mean.

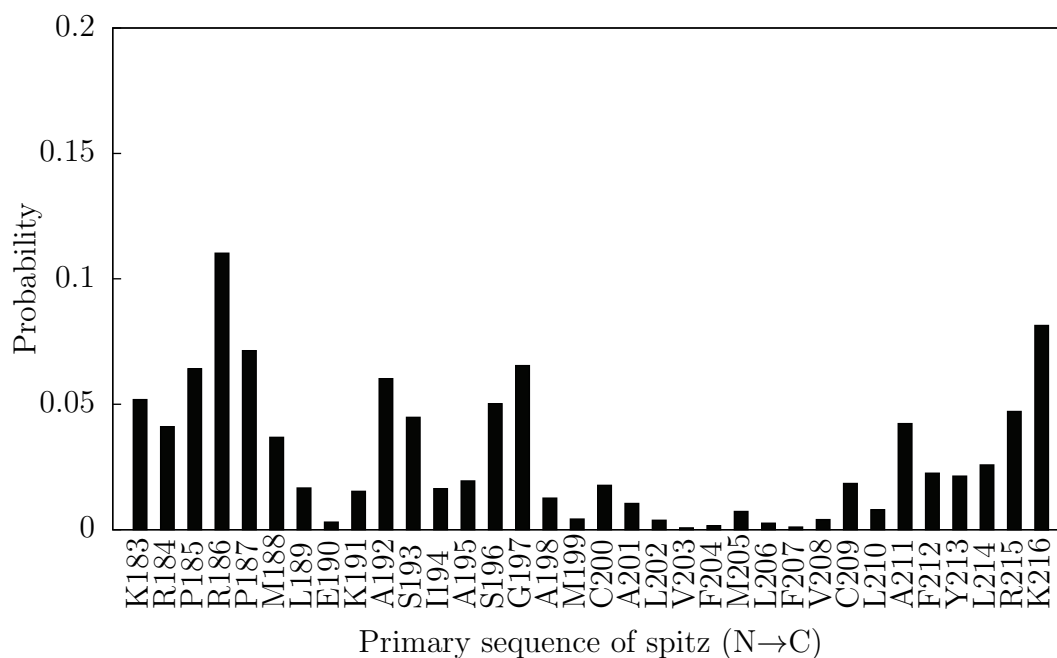


Figure 6.5: The probability for each spitz (TMD construct) residue to reside in the closest contacts with ecGlpG. The exact methodology is described in detail in section 5.4.5 (page 116). The predominant contacts appear to occur at the N- and C-termini, and also around the proposed cleavage consensus sequence (ASIASGA) (200).

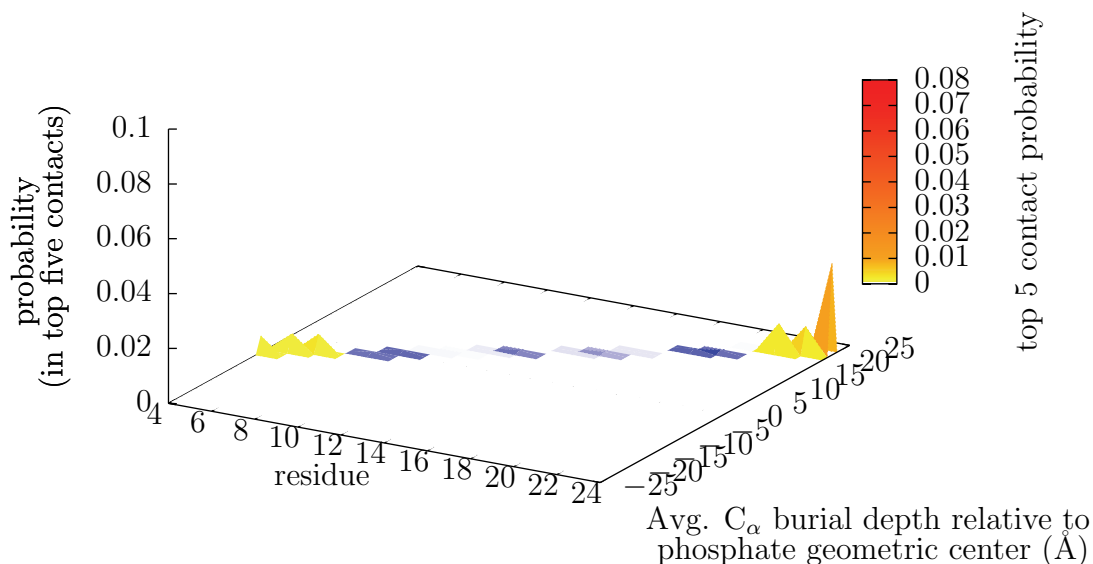


Figure 6.6: The predominant contacts between ecGlpG *helix 1* (shown here) and the spitz TMD construct are highlighted and sorted by POPE bilayer burial depth. These results are representative of most ecGlpG helix contact plots, with the N- and C-terminal residues predominantly involved in substrate interaction. The results are aggregated from the first 10 replicate simulations (spitz starting near TMs 1 and 3), and the exact methodology for assessing the predominant contacts is described in detail in section 5.4.5 (page 116)

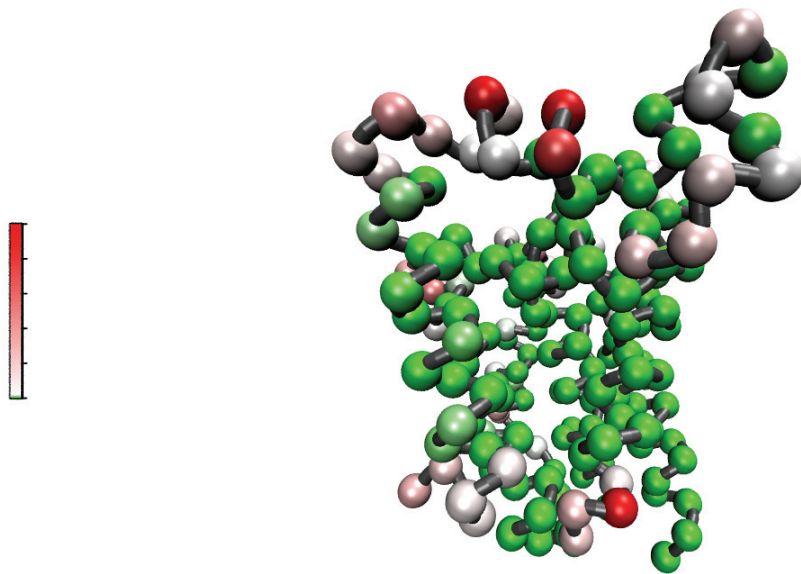


Figure 6.7: The coarse-grained representation of ecGlpG has been simplified to include only  $C_\alpha$  particles which are coloured based on their close contact probability (see section 5.4.5 on page 116) with the substrate. Water and the lipid bilayer are also excluded for clarity. The probability colour map runs from low (green)→ medium (white)→ high (red), with values ranging from 0→80% (of total close contacts) and is based on the first ten replicate simulations (spitz starting near ecGlpG TM segments 1 and 3). The cytosolic side of ecGlpG is at the bottom and the periplasmic side at the top.

# Chapter 7

## Conclusions

NMR spectroscopy was used to determine an ensemble of NHE1 TM IX structures in DPC micelles that featured a disruption in helicity near functionally critical residues. A common approach to extend beyond structural information is to probe the flexibility of peptides using NMR spin relaxation experiments. I performed the latter class of experiments for NHE1 TM VII in DPC micelles, and report  $\mu$ s-ms timescale fluctuations over a critical segment of the peptide. The structural and dynamics results suggest an importance for flexibility in NHE1 TM segments, a theme which has been emphasized in a number of NHE1 TM segment studies (22, 27, 201, 137). It is, however, not yet possible to unambiguously assign the membrane-spanning topology of each NHE1 TM segment to either of the two proposed topologies (13, 14). This will likely require the crystal structure of full-length NHE1.

The poor tractability of spitz or spitz-related rhomboid protease substrate TMD constructs was quite clear. Although many of the constructs were successfully produced by SPPS, purification by HPLC was problematic. When others attempted to produce spitz constructs using expression techniques in *E. coli*, there were also substantial difficulties with yield and purification. In the case of an apparently pure spitz peptide construct, homonuclear NMR spectra were not sufficient to unambiguously assign resonances. Thus, it is worthwhile to include isotope labels in the peptide production process, but there is no clear route to simplified production and purification.

The rhomboid protease system was investigated from another angle using CG-MD simulations, and the initial results suggest that a preferential interaction between enzyme and substrate occurs near TMs 1 and 3 rather than near the proposed substrate gate, TM 5 (148). Furthermore, ecGlpG and the spitz TMD appear to preferentially interact at the terminal ends of helices rather than within the hydrocarbon core of the bilayer.

An extensive analysis of CG-MD simulations of the FGFR3 dimerization process was presented. This includes an integrated discussion of the algorithms used to parse the simulation trajectories, and the appendix includes extensively-documented source code for these analyses. There is no high-resolution structure of FGFR3 available, and these simulation studies provide insight into residues near the dimer interface and the effect of the G380R mutation in the FGFR3 TMD—which causes achondroplasia (202). Strikingly, residue 380 does not feature prominently at the dimer interface, while G370 is one of the closest contacts in the dimers and is mutated to Cys in type 1 thanatophoric dysplasia, a much more severe skeletal phenotype (191). Thus, the phenotypic severity of an FGFR3 mutation may correlate with the proximity of the mutated residue to the dimer interface. I have also described a secondary dimer interface which progressively appears in the heterodimer and mutant homodimer FGFR3 constructs. The rotation of one helix relative to the other may increase signaling activity which causes the phenotype, a conclusion supported by helix-rotation effects reported to affect the activity of receptor tyrosine kinases (197, 198).

Overall, I have employed complementary techniques from structural biology and computational biochemistry to gain insight into biologically relevant membrane proteins.



# Appendix A: Source Code For Fractional Isotope Incorporation In Peptide Mass Calculations

## A.1 Introduction

This appendix contains the Python source code I wrote for peptide mass calculations involving fractional isotope labels incorporated during solid-phase synthesis. The website employing this code is available at <http://129.173.89.133/cgi-bin/isotope CGI>. My code has also been incorporated in the back-end of other online mass calculation tools (see <http://structbio.biochem.dal.ca/jrainey/mspep/>).

## A.2 Source Code Proper

Listing A.1: CGI module for predicting peptide mass resulting from  $^{15}\text{N}$  backbone fractional isotope incorporation during solid-phase peptide synthesis.

```
1 #!/Library/Frameworks/Python.framework/Versions/Current/bin/python
2 #need to add in counting isotopes like 13C when lots of atoms
3 import cgi
4 import cgitb; cgitb.enable()
5 import sys
6
7 print "Content-Type: text/html"      # HTML is following
8 print                                # blank line, end of headers
9
10 def print_form(retained_sequence='', retained_fractions='', retained_pure_15N='',
11               retained_checked_monoisotopic=''): #provide empty defaults to these vars in arguments
12     so they are not referenced before assignment on first load of script
13     if form.getfirst("client_sequence"): #if the user entered something into the AA
14         sequence text area
15         retained_sequence=form.getfirst("client_sequence") #retain that
16         information
17     if form.getfirst("fraction_list"): #if the user entered something into the
18         fraction list text area
19         retained_fractions=form.getfirst("fraction_list")
20     if form.getfirst("pure_isotopes"): #if the user entered something into the pure
21         15N isotope text area
22         retained_pure_15N=form.getfirst("pure_isotopes")
23     if form.getfirst("monoisotopic"): #if they checked the monoisotopic box, keep
24         it that way after hitting submit
25         retained_checked_monoisotopic='checked'
26
27     print '''
```



#Will therefore have to add the molecular weight of water to the total mass of the peptide or protein b/c of the terminal residues

monoisotopic\_D={

"A":71.03711,

"R":156.10111,

"N":114.04293,

"D":115.02694,

"C":103.00919,

"E":129.04259,

"Q":128.05858,

"G":57.02146,

"H":137.05891,

"I":113.08406,

"L":113.08406,

"K":128.09496,

"M":131.04049,

"F":147.06841,

"P":97.05276,

"S":87.03203,

"T":101.04768,

"W":186.07931,

"Y":163.06333,

"V":99.06841}

average\_D={

"A":71.0788,

"R":156.1875,

"N":114.1038,

"D":115.0886,

"C":103.1388,

41

43

45

47

49

51

53

55

57

59

61

63

65

67

69

```

71 "E":129.1155,
72 "Q":128.1307,
73 "G":57.0519,
74 "H":137.1411,
75 "I":113.1594,
76 "L":113.1594,
77 "K":128.1741,
78 "M":131.1926,
79 "F":147.1766,
80 "P":97.1167,
81 "S":87.0782,
82 "T":101.1051,
83 "W":186.2132,
84 "Y":163.1760,
85 "V":99.1326}

86 def pure_15N_mass(pure_15N_form_input=0): #default is zero pure 15N residues
87     '''Error checking and accounting for pure <sup>15</sup>N contribution to total
88     mass based on user input'''
89     if pure_15N_form_input: #if there is text in this box
90         try:
91             num_pure_15N_residues=int(pure_15N_form_input)
92             return num_pure_15N_residues
93         except:
94             print '<b>Number of pure <sup>15</sup>N residues must be an
95             integer value</b>\'
96             sys.exit()
97         else:
98             return 0

```

```

109 def table_printer(working_list,i,maximum_mass_change,pure_15N_mass,pure_15N_form_input,
110 total_mass):
111     '''Prints out the isotope table after the user hits the submit button'''
112
113     print '<tr>' + '<td>' +str(working_list[i]) + '</td>' + '<td>' + '<center>' + '+' +str(
114         maximum_mass_change-i+pure_15N_mass(pure_15N_form_input))\
115         + '</center>' + '</td>' + '<td>' +str(total_mass+(maximum_mass_change-i)+
116         pure_15N_mass(pure_15N_form_input))\
117         + '</td>' + '<td>' +str(total_mass+(maximum_mass_change-i)+pure_15N_mass(
118         pure_15N_form_input)+1) + '</td>' \
119         + '<td>' +str(total_mass+(maximum_mass_change-i)+pure_15N_mass(
120         pure_15N_form_input)+23) + '</td>' \
121         + '<td>' +str(total_mass+(maximum_mass_change-i)+pure_15N_mass(
122         pure_15N_form_input)+39) + '</td>' + '</center>'
123
124 def print_isotope_list(fractions_list,total_mass,pure_15N_form_input):
125
126     maximum_mass_change=1
127
128     working_list=[]
129     first_fraction=fractions_list[0]
130     working_list.append(first_fraction) #gains +1 amu
131     working_list.append(1.0-first_fraction) #gain 0; maintains monoisotopic mass
132     print '''<center><table border="1">
133         <tr>
134             <th>Fraction of <br/>synthesis yield</
135             th>
136             <th>Additional mass from<br/> <sup
137             >15</sup>N isotopic enrichment</th>
138             <th>Total mass</th>

```

```

121 <th>MALDI <br/>H<sup>+</sup></th><sup>&nbsp;</sup>&nbsp;</th>
122 Adduct</th>
123 <th>MALDI <br/>Na<sup>+</sup></th><sup>&nbsp;</sup>&nbsp;</th>
124 Adduct</th>
125 <th>MALDI <br/>K<sup>+</sup></th><sup>&nbsp;</sup>&nbsp;</th>
126 Adduct</th>
127 </tr>'''
128
129 if len(fractions_list)==1: #special case of a single element list stops here
130     for i in range(2):
131         table_printer(working_list,i,maximum_mass_change,pure_15N_mass,
132                       pure_15N_form_input,total_mass)
133     else: #more than 1 fractional isotope position
134         for i in range(2, len(fractions_list)+1):
135             maximum_mass_change+=1
136             new_list=[]
137             fraction=fractions_list[i-1]
138
139             for element in working_list:
140                 new_list.append(element*fraction)
141                 new_list.append(element*(1-fraction))
142
143             new_internal_list=[]
144             for x in range(0, len(new_list[1:-1]),2):
145                 new_internal_list.append(new_list[1:-1][x]+new_list
146                                       [1:-1][x+1])
147
148             new_list[1:-1]=new_internal_list
149             working_list=new_list[:]
150
151             for i in range(0, len(working_list)):

```

```

145 table_printer(working_list,i,maximum_mass_change,pure_15N_mass,
146     pure_15N_form_input,total_mass)
147     #add together contiguous pairs of list elements (because they
148     have the same total amu) excluding the first and last
149     elements
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167

```

```

def sum_amino_acids(dictionary_used,dictionary_name,pure_15N_form_input):
    '''Add together the amino acids input by the client and print output'''

    #will have to incorporate average mass of water at some point

    total_mass=0
    try: #try the iteration over AA sequence input

        for amino_acid in form.getfirst("client_sequence", "").upper():
            total_mass+=dictionary_used[amino_acid]
            total_mass+=18 #include water for the terminal residues

        #now check for fractional isotope incorporation

        if form.getfirst("fraction_list"): #if there's a non-empty 15N mass
            fraction_list

            try:
                fraction_input_list=[float(number) for number in form.
                    getfirst("fraction_list").split(',')]
            except: #if it's not a list of decimal numbers separated by a
                comma

```



```
169     print '<center><b>Only decimal input values separated
170         by a comma are accepted</b></center>'
171     sys.exit()
172
173     else: #the fractional 15N input was valid; produce a list of
174           isotope distributions
175           print_isotope_list(fraction_input_list,total_mass,form.
176                             getfirst('pure_isotopes'))
177
178           print '<center>'+ dictionary_name + '</center>'
179
180           else: #no fractional 15N positions; simply add together AAs
181           print '<center>'+ dictionary_name + ':' + str(total_mass+
182                 pure_15N_mass(pure_15N_form_input)) + '</center>'
183
184           except KeyError: #KeyError could be any text input that is a character other
185                             than a letter representing one of the common AAs
186           print '<center><b> Only the 20 common amino acids are accepted as input
187                 .</b></center>'
188
189           form = cgi.FieldStorage()
190           print_form() #call function to print blank or data-entry retained html form
191
```

```
193
195
197
199
201
203
205
207
209
211
213
215
217

if form.getfirst("client_sequence", ""): #if it's not an empty sequence
    if form.getfirst("monoisotopic"): #if the user selects monoisotopic
        sum_amino_acids(monoisotopic_D, "Monoisotopic mass", form.getfirst('
            pure_isotopes'))
    else: #if the user leaves checkbox empty average masses are used
        sum_amino_acids(average_D, "Average isotopic mass", form.getfirst('
            pure_isotopes'))
else: # if it's an empty sequence
    total_mass=0

#proper compact code sample for CGI from python std library:

#import cgi
#form = cgi.FieldStorage()
#user = form.getfirst("user", "").upper() # This way it's safe.
#for item in form.getlist("item"):
#    do_something(item)

#note the use of getfirst because the user might try to enter multiple values in some
tricky way, or there might be more than one value
#so can use getlist to generalize without having to write code to deal with single
versus multiple values
```

```
#&nbsp;<a href="url" target="_blank"> More info </a>
```

219

# Appendix B: Additional Data From Spitz Peptide/Protein Production And Purification

## B.1 Mass Spectrometry Results

### B.1.1 TR-09-1 Construct

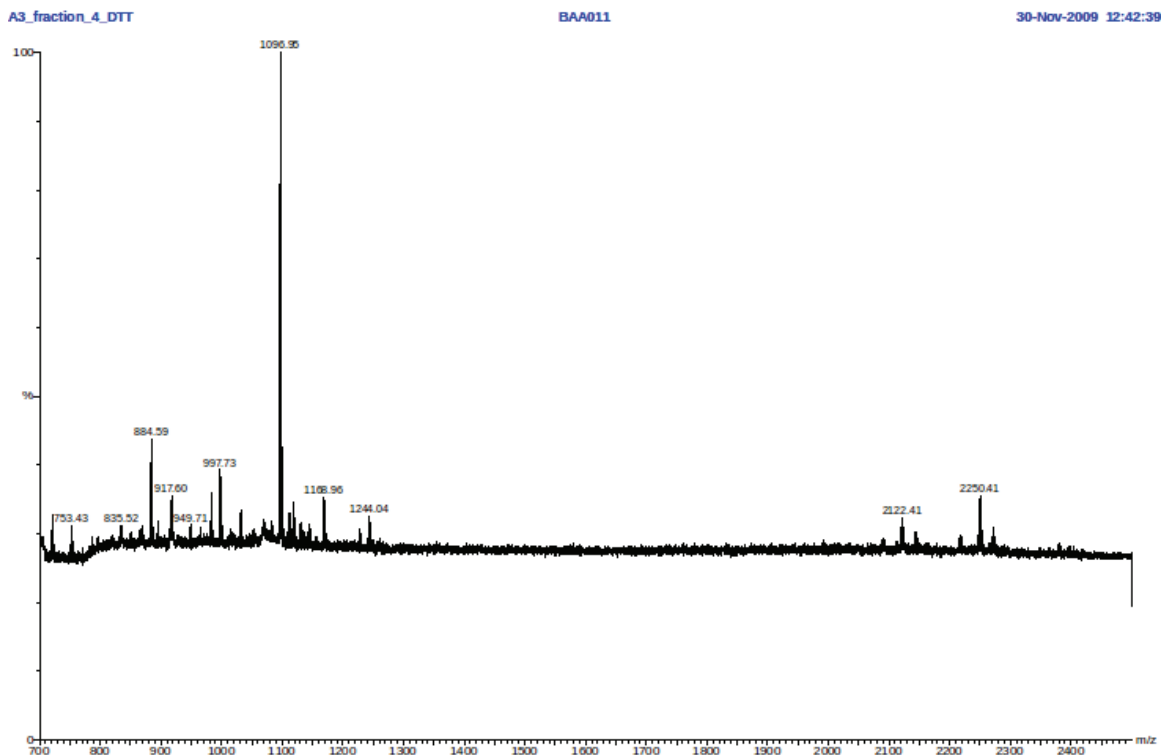


Figure B.1: Fraction 4 from the TR-09-1 crude HPLC run described in Figure 4.12 on page 83 was treated with 200 mM DTT for 1 hour, and then diluted with  $\alpha$ -CHC matrix before collecting this reflectron-mode MALDI spectrum. The peak at 2122 m/z is the product missing the N-terminal K while the peak at 2250 m/z corresponds to the desired product. The lower molecular weight compound at 1096 m/z persists in many synthetic spitz fractions.

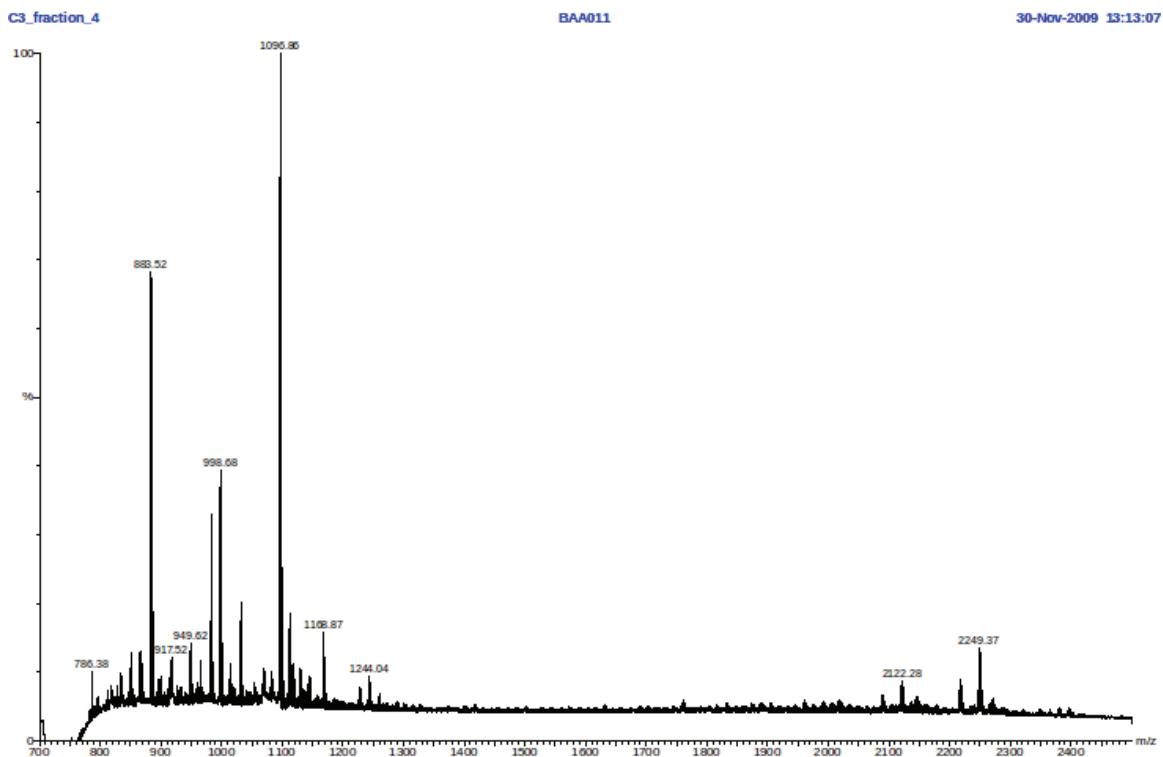


Figure B.2: Similar results are obtained for the TR-09-1 HPLC fraction (#4) used in Figure B.1 (page 217) when there is no DTT pre-treatment.

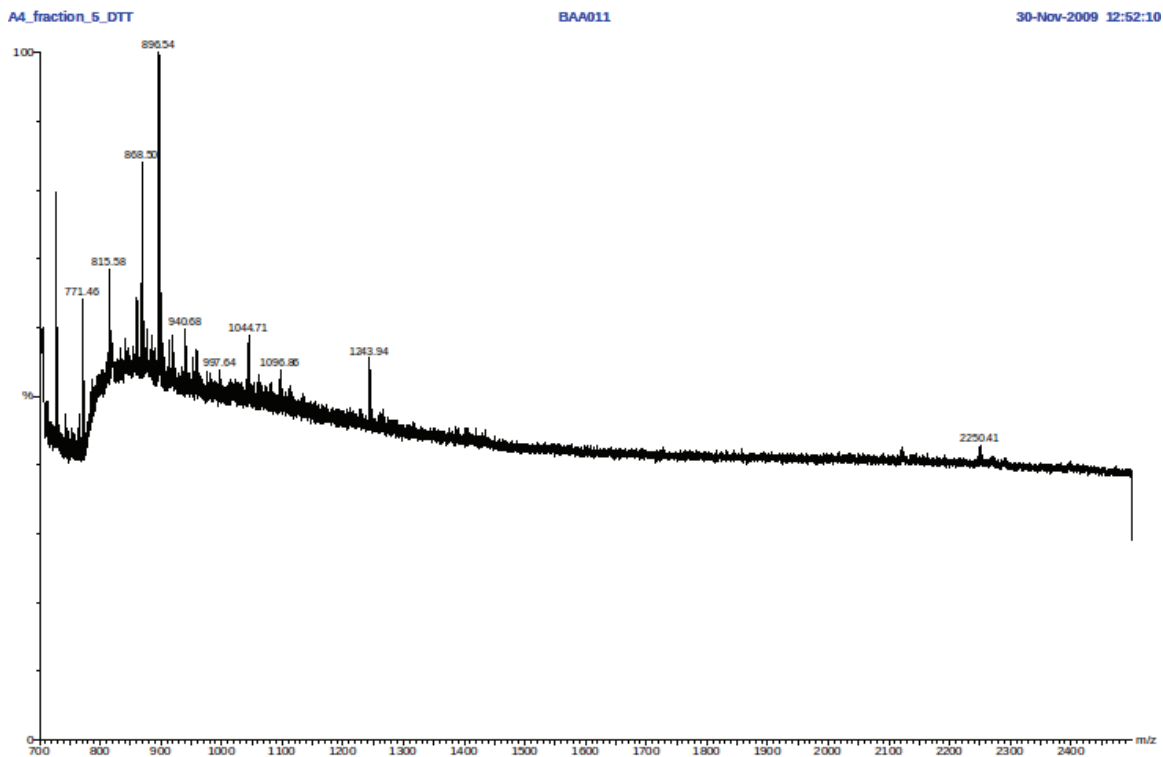


Figure B.3: Fraction 5 from the TR-09-1 crude HPLC run described in Figure 4.12 on page 83 was treated with 200 mM DTT for 1 hour, and then diluted with  $\alpha$ -CHC matrix before collecting this reflectron-mode MALDI spectrum. The peak at 2250 m/z corresponds to the desired product, but there are a number of lower molecular weight impurities visible.

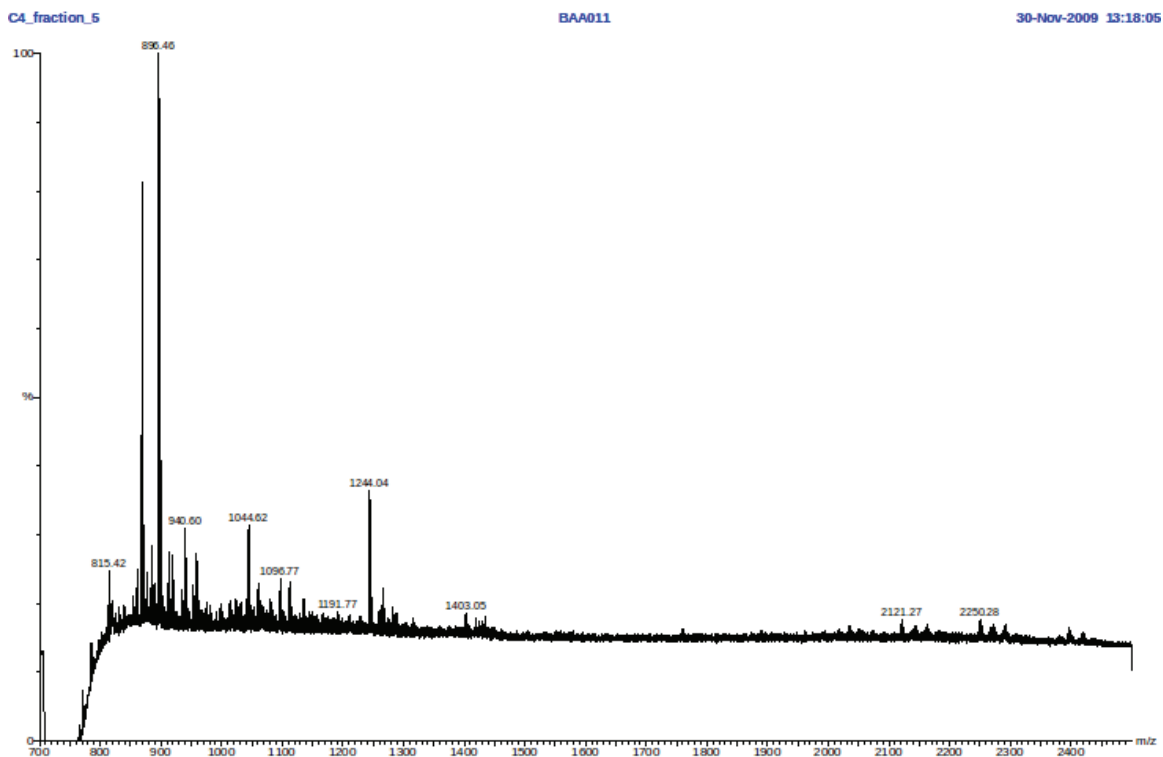


Figure B.4: Similar results are obtained for the TR-09-1 HPLC fraction (#5) used in Figure B.3 (page 219) when there is no DTT pre-treatment.



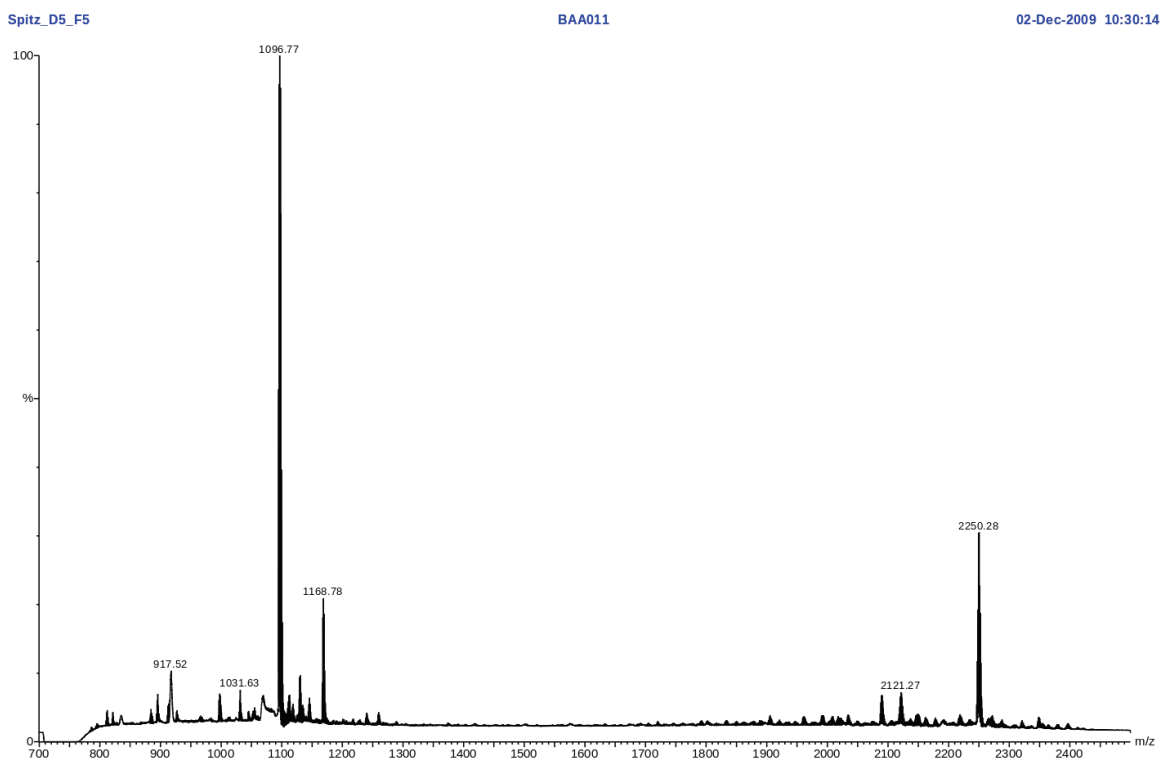


Figure B.5: Reflectron-mode MALDI mass spectrum for fraction 5 (collected between 22-23 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. This is the most promising fraction from the latter HPLC run on the basis of the ratio of the desired product ( $\sim 2250$  m/z) to the product missing the N-terminal K at  $\sim 2121$  m/z (assuming the species have similar ionization potentials).

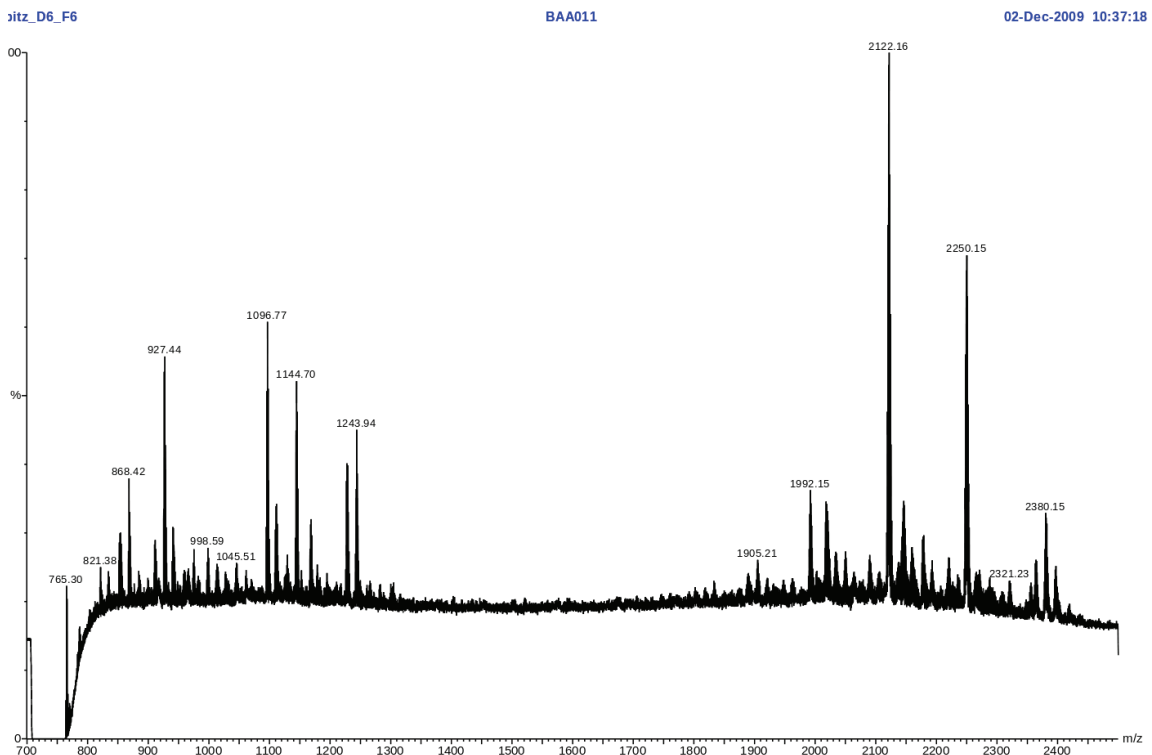


Figure B.6: Reflectron-mode MALDI mass spectrum for fraction 6 (collected between 23-24 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. The desired product is at  $\sim 2250$  m/z while the compound at  $\sim 2122$  m/z is missing the N-terminal K.

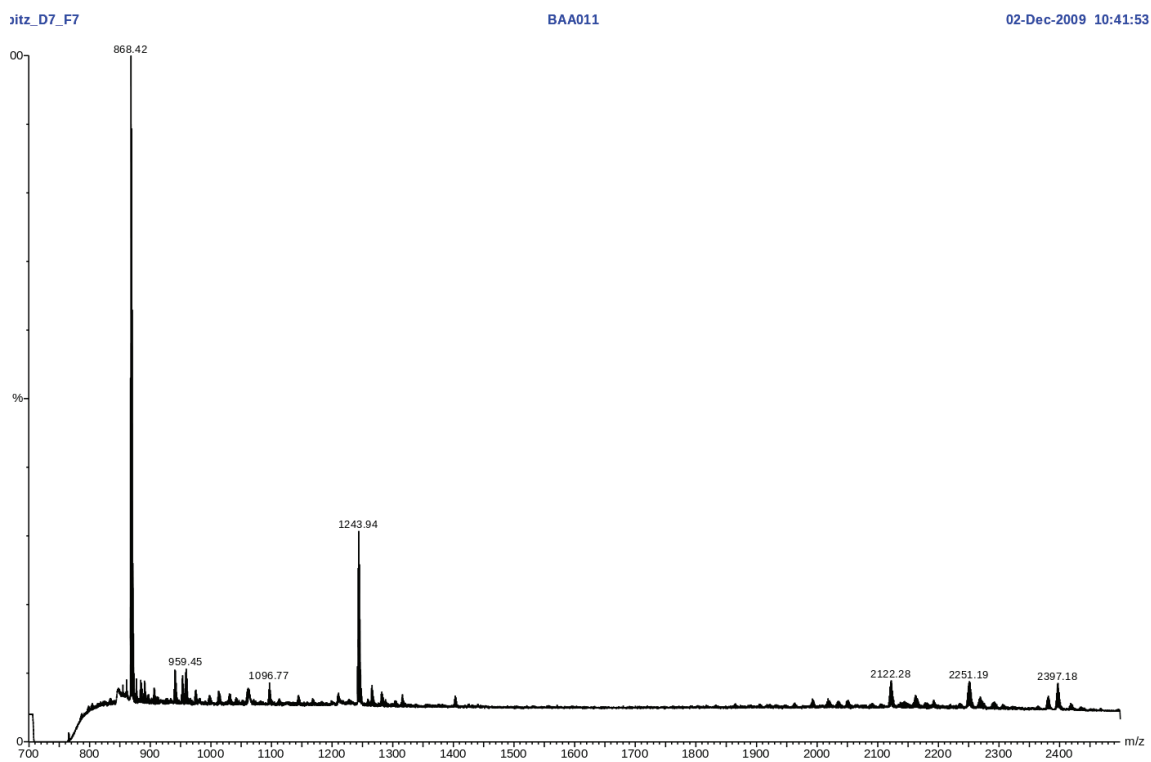


Figure B.7: Reflectron-mode MALDI mass spectrum for fraction 7 (collected between 24-25 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. The desired product is at ~2250 m/z while the compound at ~2122 m/z is missing the N-terminal K.

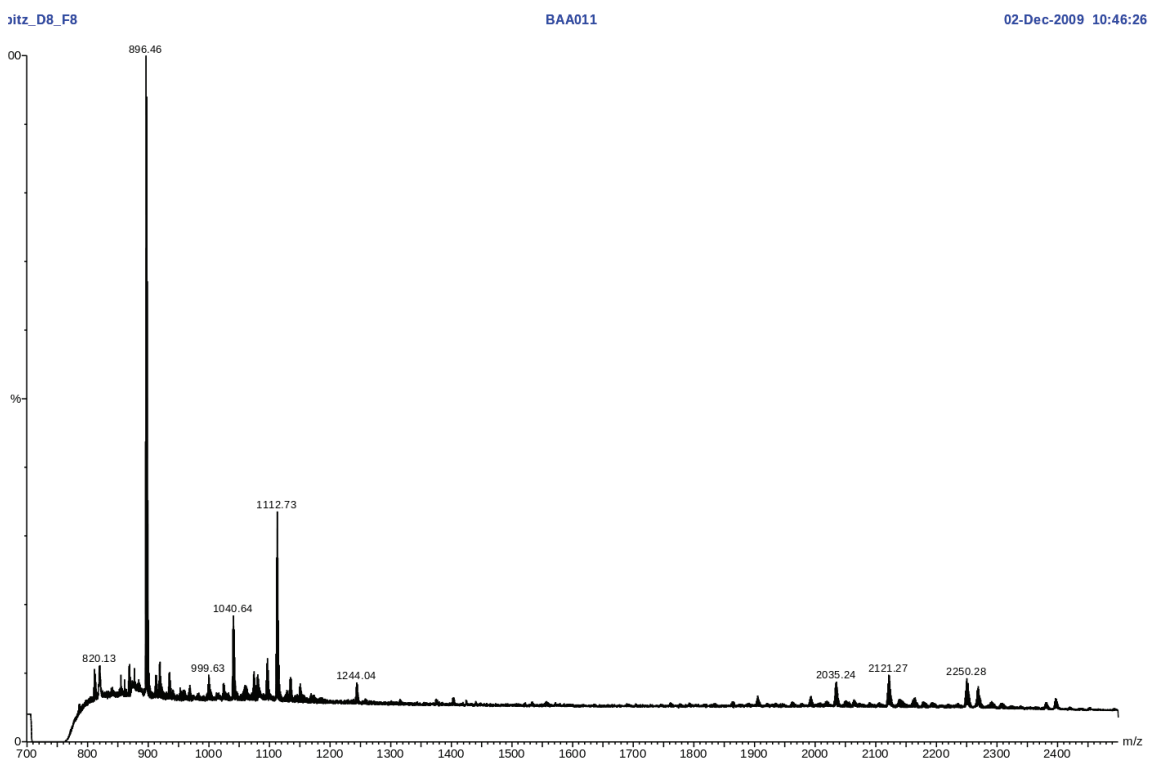


Figure B.8: Reflectron-mode MALDI mass spectrum for fraction 8 (collected between 25-26 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. The desired product is at ~2250 m/z while the compound at ~2122 m/z is missing the N-terminal K.

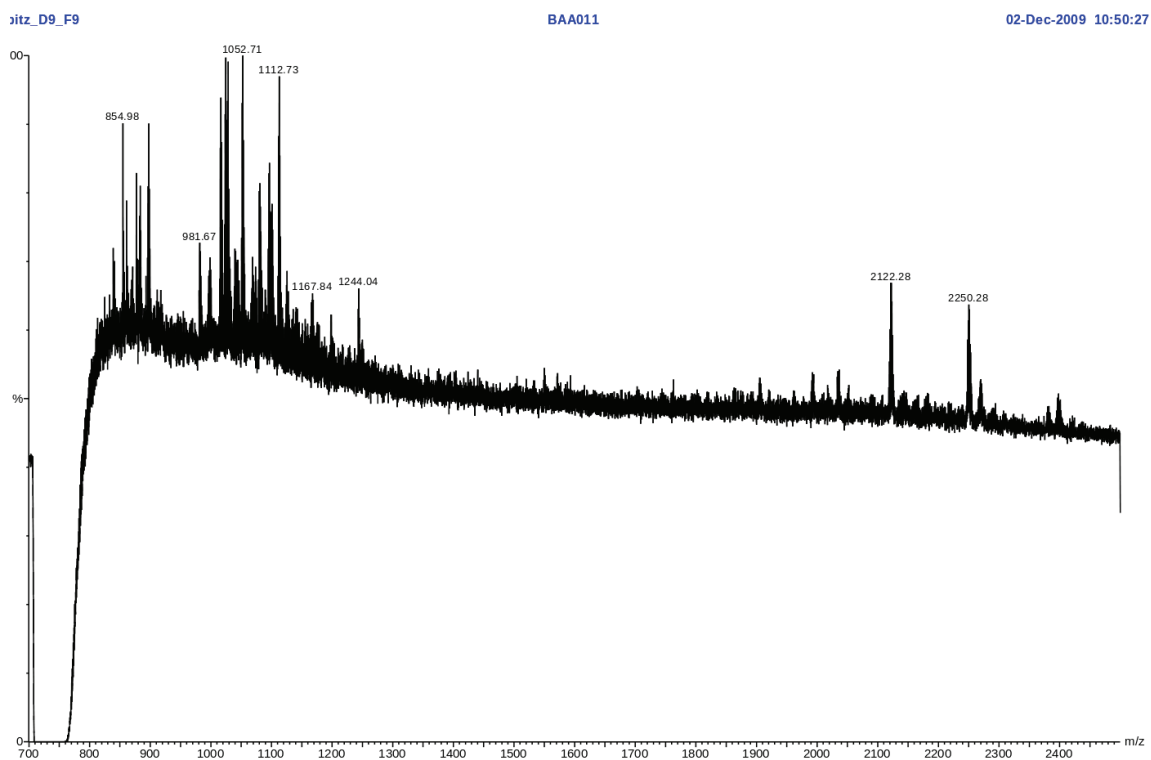


Figure B.9: Reflectron-mode MALDI mass spectrum for fraction 9 (collected between 26-27 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. The desired product is at  $\sim 2250$  m/z while the compound at  $\sim 2122$  m/z is missing the N-terminal K.

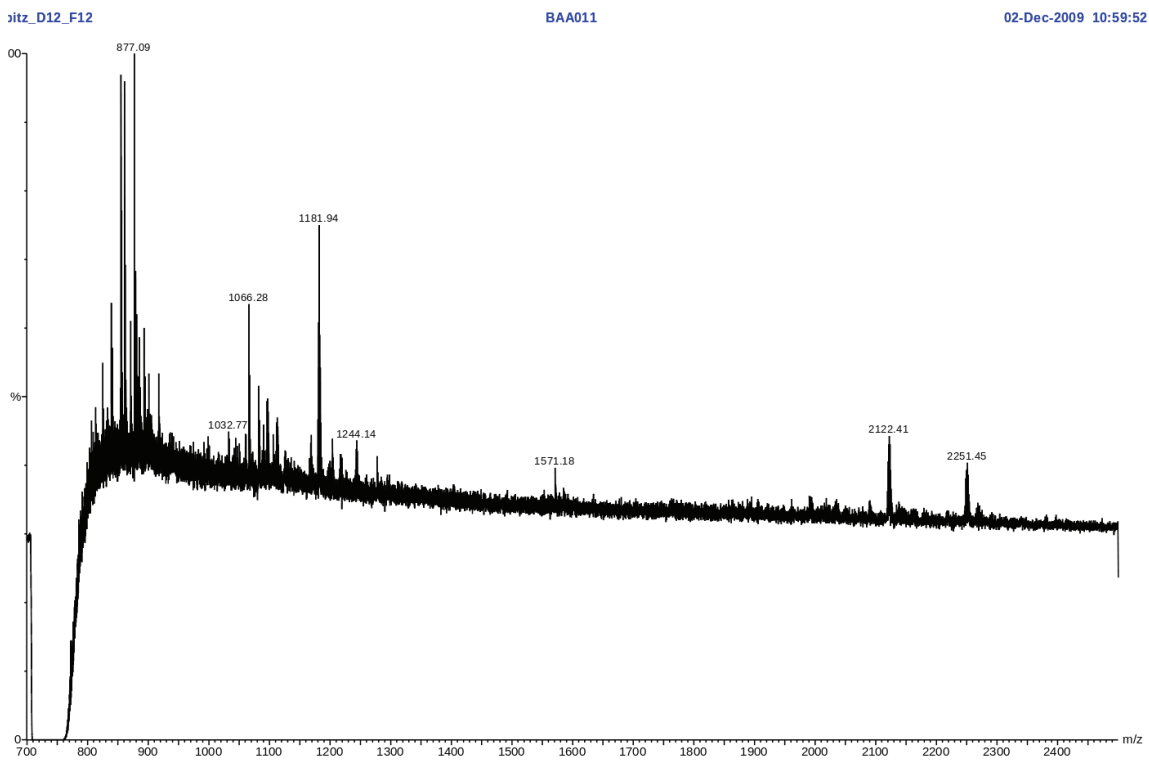


Figure B.10: Reflectron-mode MALDI mass spectrum for fraction 12 (collected between 29-30 minutes) from the crude TR-09-1 HPLC run detailed in Figure 4.13 on page 84. The desired product is at  $\sim 2250$  m/z while the compound at  $\sim 2122$  m/z is missing the N-terminal K. The product is still eluting 12 minutes after initial detection from the HPLC trace.

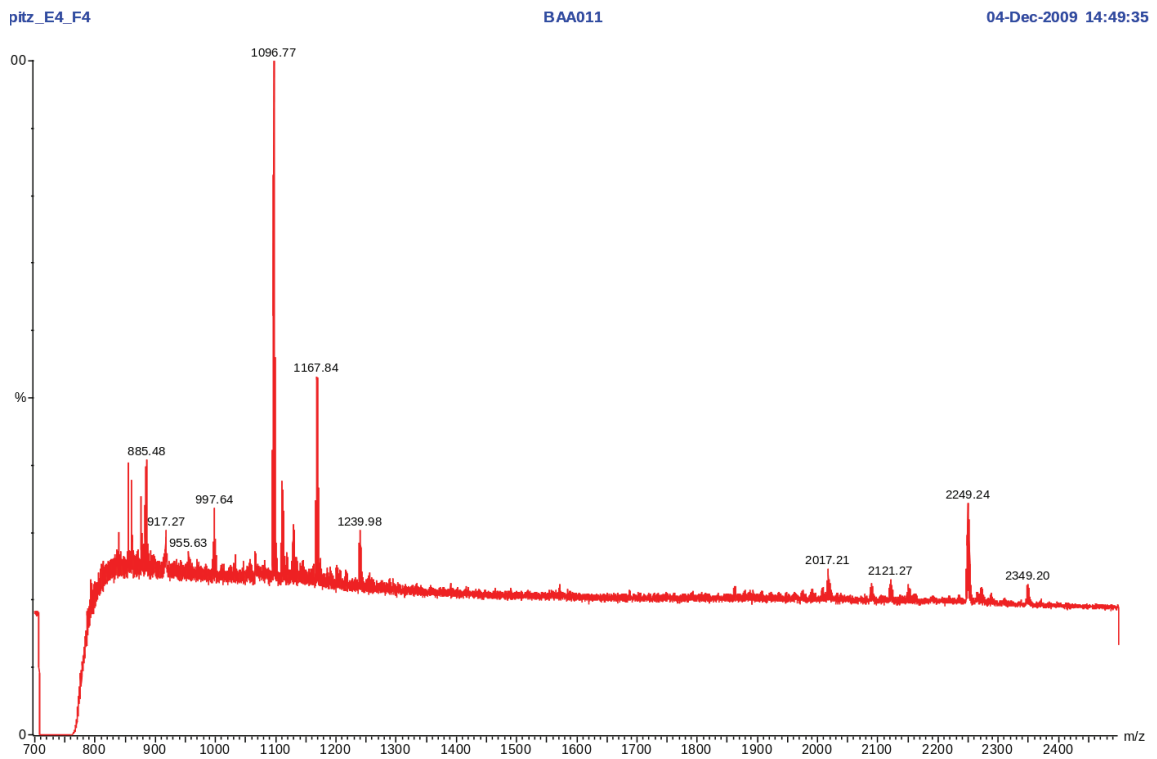


Figure B.11: Fraction #4 (25:01-25:16) from the TR-09-1 C18 HPLC run detailed in Figure 4.15 on page 86 was diluted two-fold with  $\alpha$ -CHC matrix and a MALDI spectrum was collected in reflectron mode.

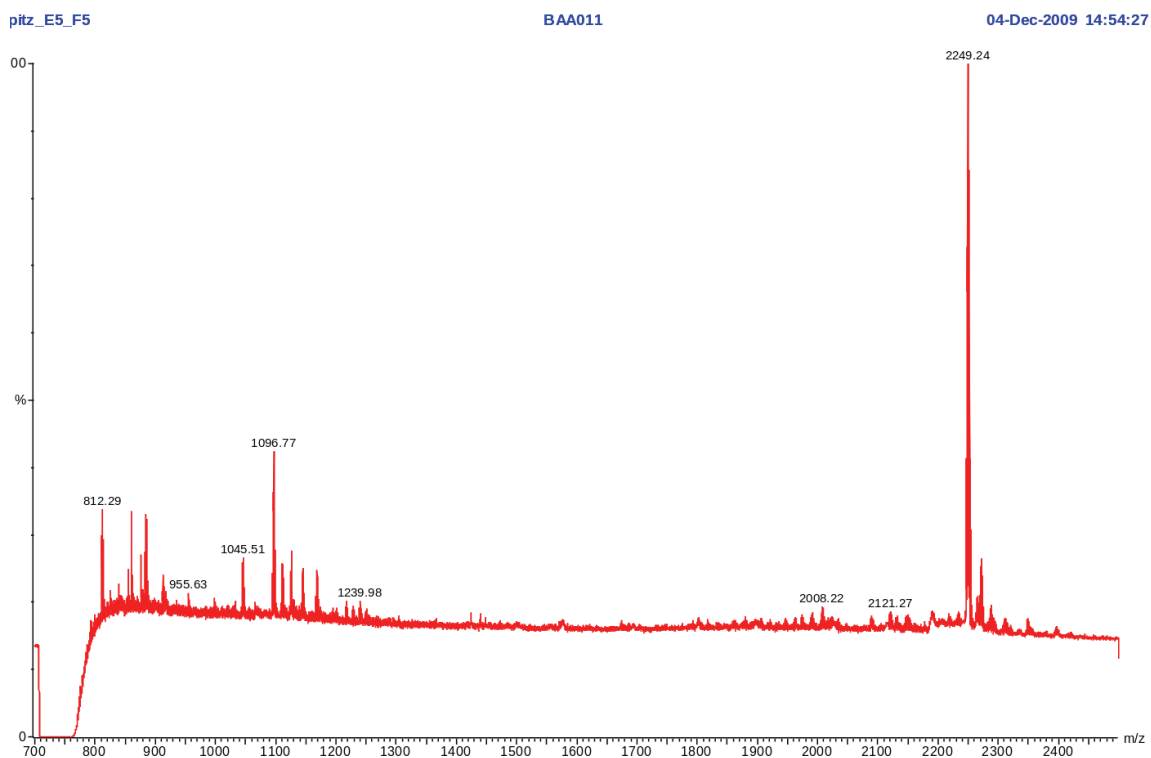


Figure B.12: Fraction #5 (25:16-25:31) from the TR-09-1 C18 HPLC run detailed in Figure 4.15 on page 86 was diluted two-fold with  $\alpha$ -CHC matrix and a MALDI spectrum was collected in reflectron mode. This is the highest purity TR-09-1 sample obtained to date, with the product peak at  $\sim$ 2249 dwarfing most impurities.





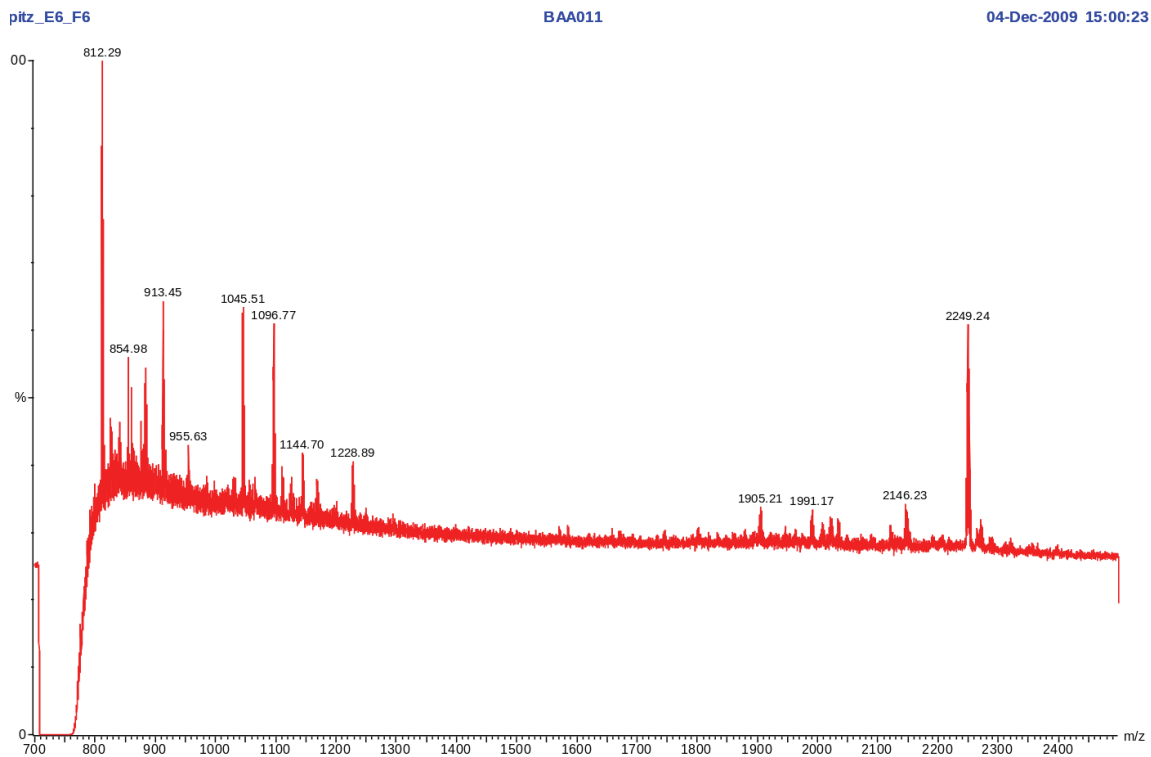


Figure B.14: Fraction #6 (25:31-25:46) from the TR-09-1 C18 HPLC run detailed in Figure 4.15 on page 86 was diluted two-fold with  $\alpha$ -CHC matrix and a MALDI spectrum was collected in reflectron mode.

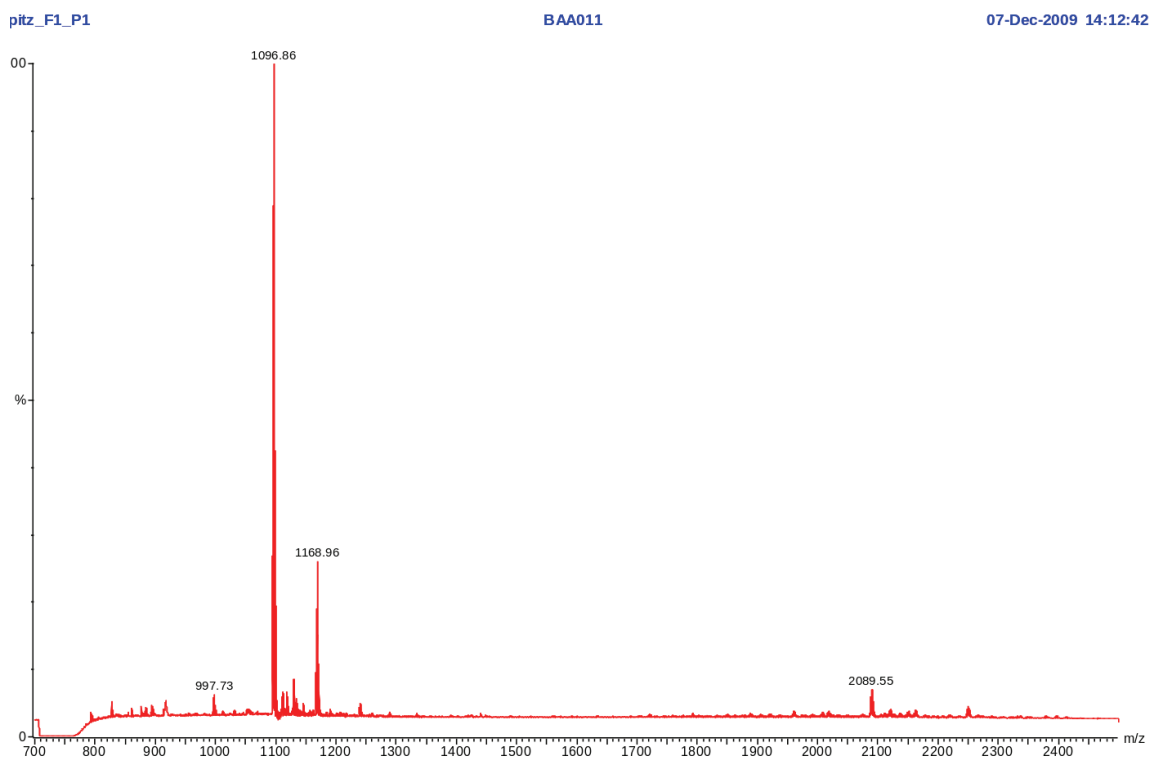


Figure B.15: Fraction #1 from the TR-09-1 C18 semi-preparative HPLC run detailed in Figure 4.17 (page 88) was diluted two-fold with  $\alpha$ -CHC matrix and the MALDI spectrum was collected in reflectron mode. The target peak  $\sim 2250$  m/z is dwarfed by a low molecular weight impurity  $\sim 1096$  m/z.

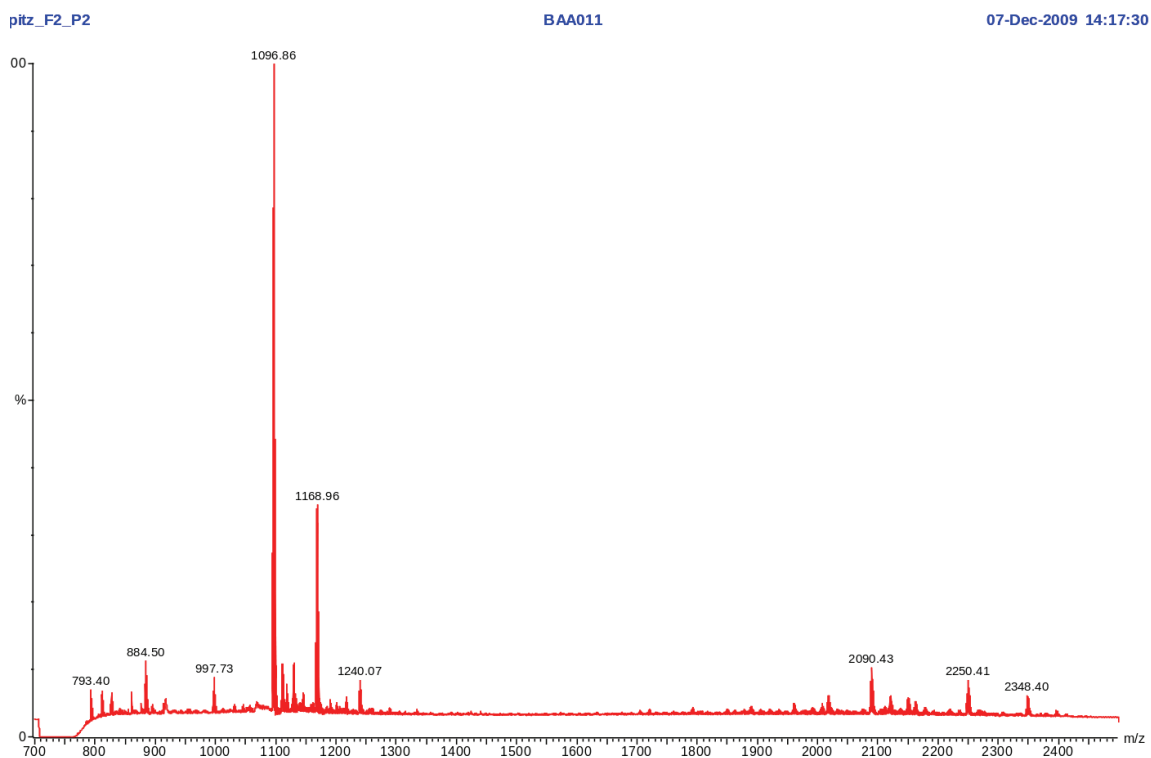


Figure B.16: Fraction #2 from the TR-09-1 C18 semi-preparative HPLC run detailed in Figure 4.17 (page 88) was diluted two-fold with  $\alpha$ -CHC matrix and the MALDI spectrum was collected in reflectron mode. The target peak  $\sim 2250$  m/z is dwarfed by a low molecular weight impurity  $\sim 1096$  m/z.

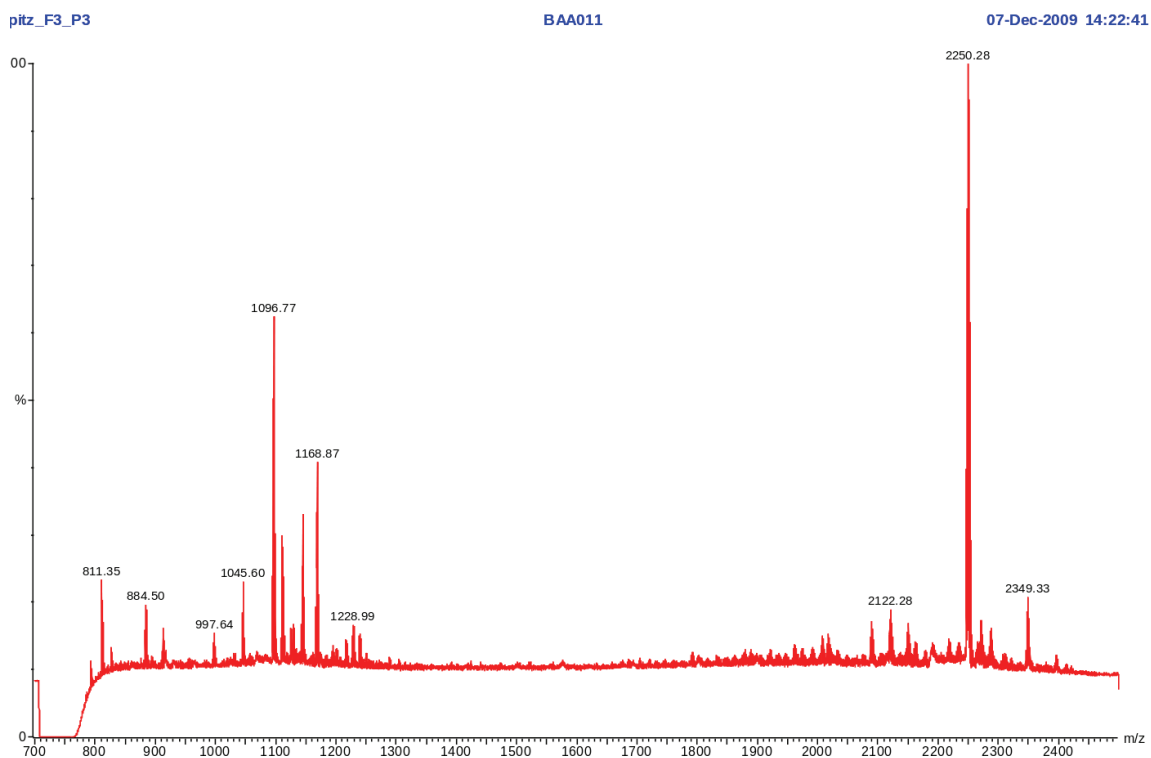


Figure B.17: Fraction #3 from the TR-09-1 C18 semi-preparative HPLC run detailed in Figure 4.17 (page 88) was diluted two-fold with  $\alpha$ -CHC matrix and the MALDI spectrum was collected in reflectron mode. The target peak  $\sim$ 2250 m/z is the major peak in the spectrum.

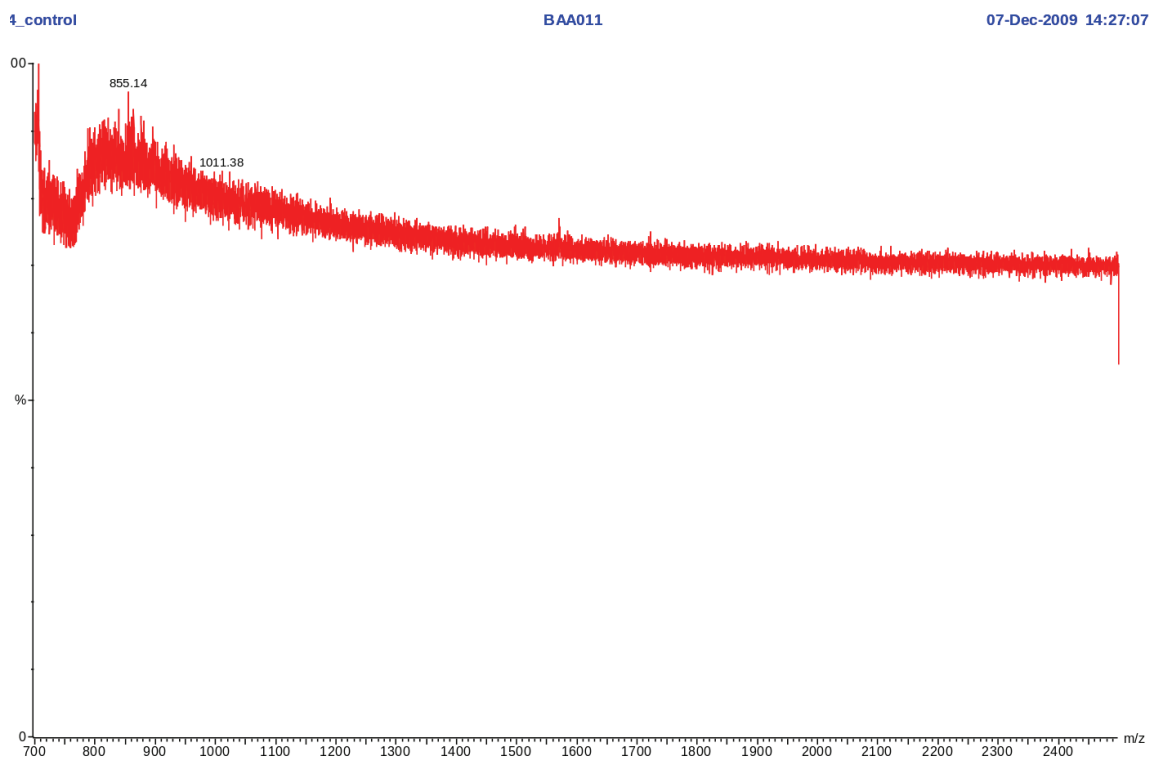


Figure B.18: A blank solution composed of 50% H<sub>2</sub>O/ACN (0.1% TFA) was combined with  $\alpha$ -CHC matrix to test for low molecular weight impurities which may reside in either the solvents or the matrix. For consistency with test results, the MALDI spectra were collected over the same m/z range in reflectron mode.

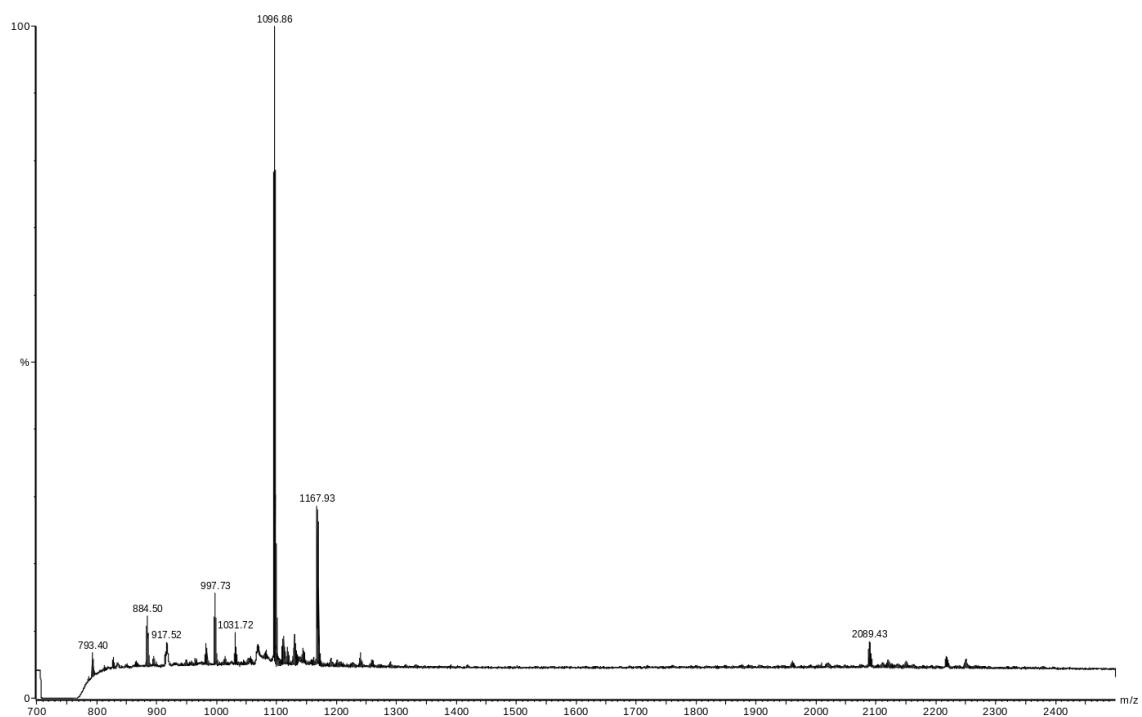


Figure B.19: This is the reflectron-mode MALDI spectrum of a representative fraction (26:40-26:55) from the first crude TR-09-1 bulk purification HPLC run detailed in Figure 4.21 on page 92. The fraction was diluted two-fold with  $\alpha$ -CHC matrix prior to collection of the spectrum. The ubiquitous impurity at  $\sim 1096$  m/z dwarfs the target product at  $\sim 2250$  m/z.

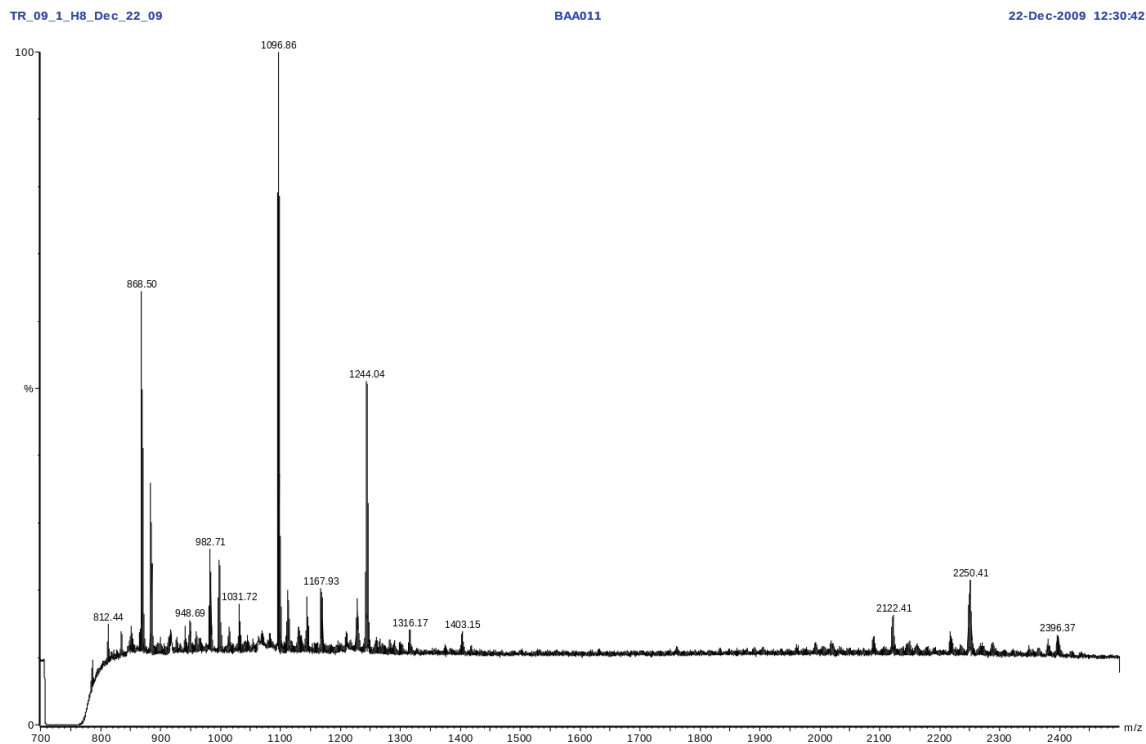


Figure B.20: The most promising MALDI-MS result (shown here) for the first TR-09-1 bulk purification run detailed in Figure 4.21 on page 92 actually corresponds to the recovery solution of eluent that flanks the collected fractions between 22-30 minutes. Given the apparent viscosity of the crude TR-09-1 preparation and the trace amount of target product in the collected fractions (see summary Table 4.5 on page 104 and a representative result in Figure B.19 on page 235) it appears likely that I need to collect fractions later in the HPLC run.



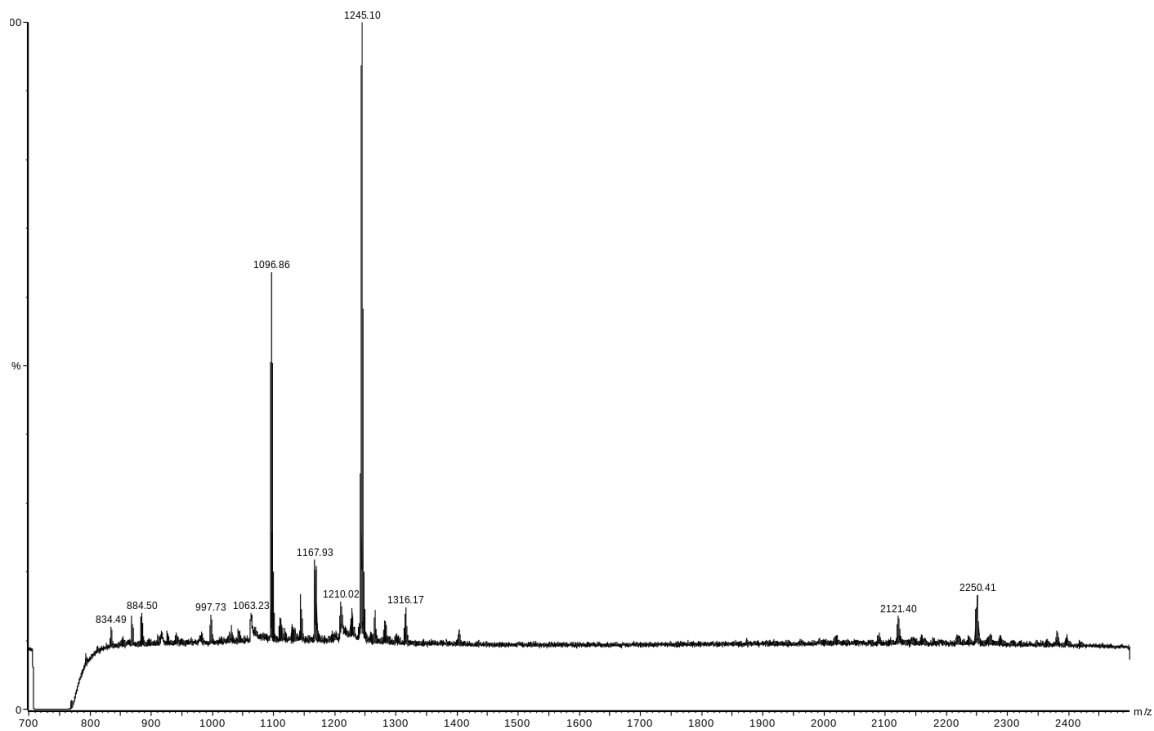


Figure B.21: MALDI reflectron-mode spectrum for the first fraction collected (26:55-27:25) from the *second* TR-09-1 bulk purification run detailed in Figure 4.21 on page 92. The target product  $\sim 2250$  m/z appears to be present in greater abundance than the truncated product at  $\sim 2121$  m/z, but both are dwarfed by lower molecular weight impurities.

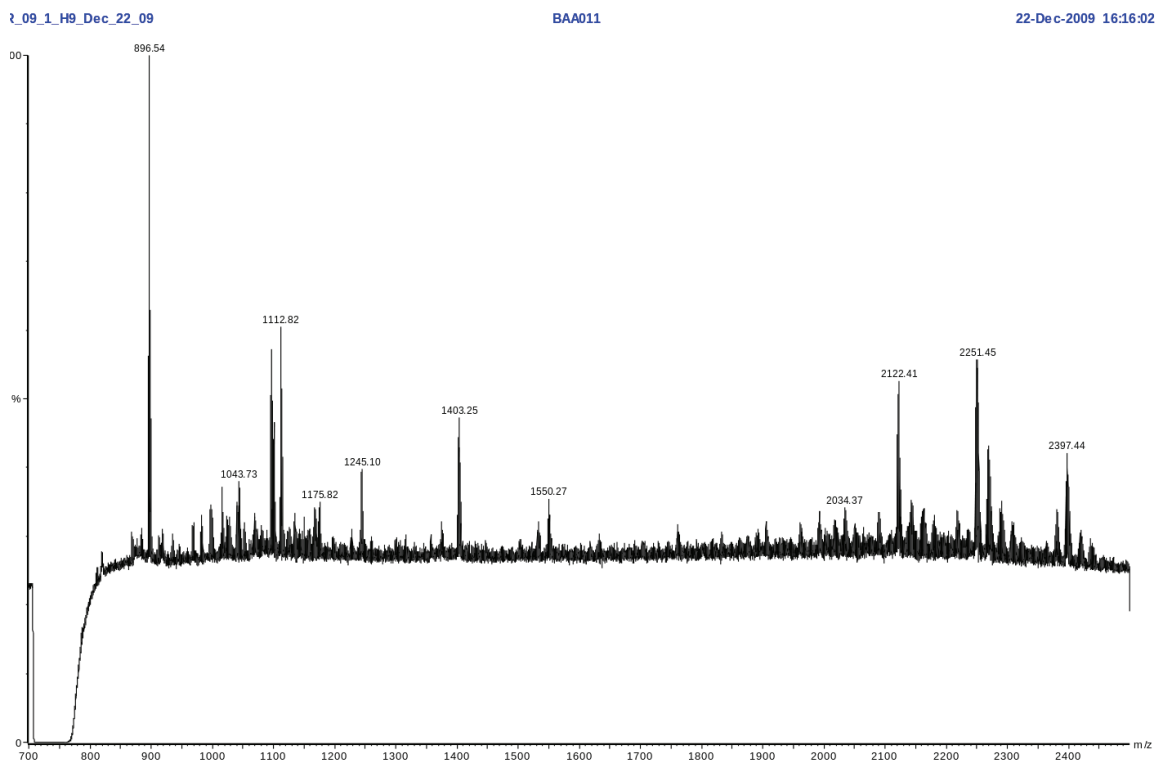


Figure B.22: MALDI reflectron-mode spectrum for the tenth fraction collected (31:25-31:55) from the *second* TR-09-1 bulk purification run detailed in Figure 4.21 on page 92. This is one of the better results (as summarized in Table 4.6 on page 105) with a fairly prominent  $\sim 2250$  m/z species. Still, low molecular weight impurities and the truncated product ( $\sim 2122$  m/z) persist.

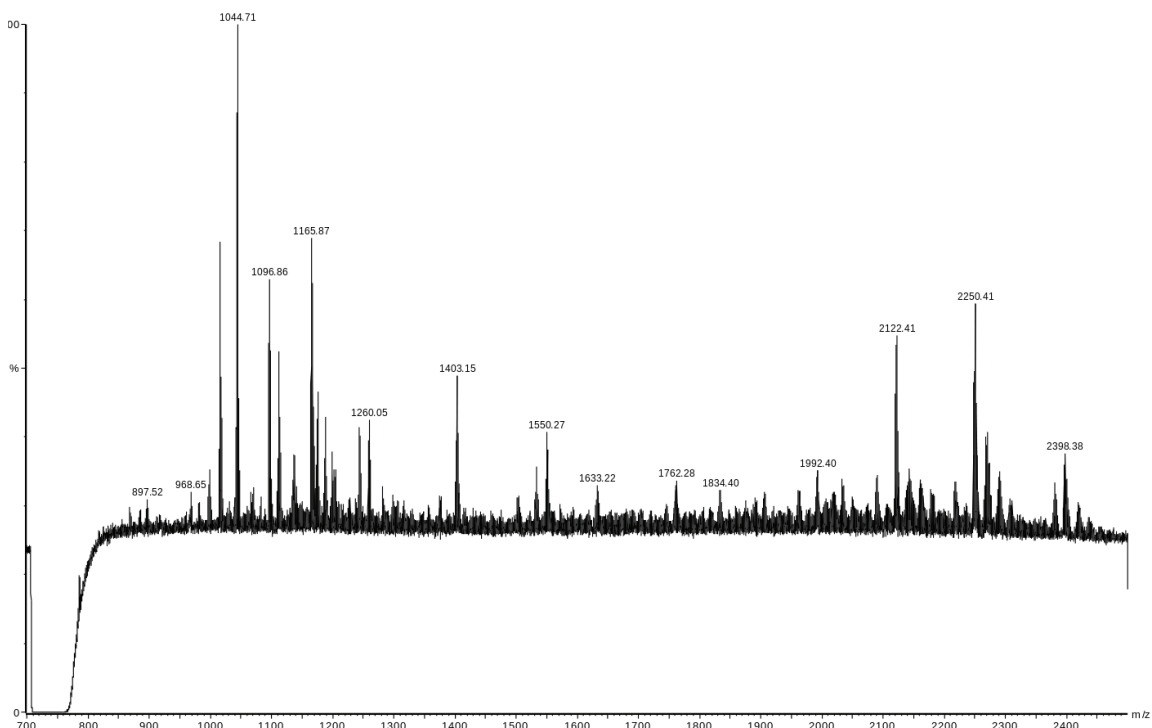


Figure B.23: MALDI reflectron-mode spectrum for the final fraction collected (33:25-33:55) from the *second* TR-09-1 bulk purification run detailed in Figure 4.21 on page 92. Accounting for the results from the first replicate run (Table 4.5 on page 104), the presence of the target product  $\sim 2250$  m/z here confirms that TR-09-1 elutes from the C18 semi-preparative column over an  $\sim 8$  minute window. This is outrageously poor separation.

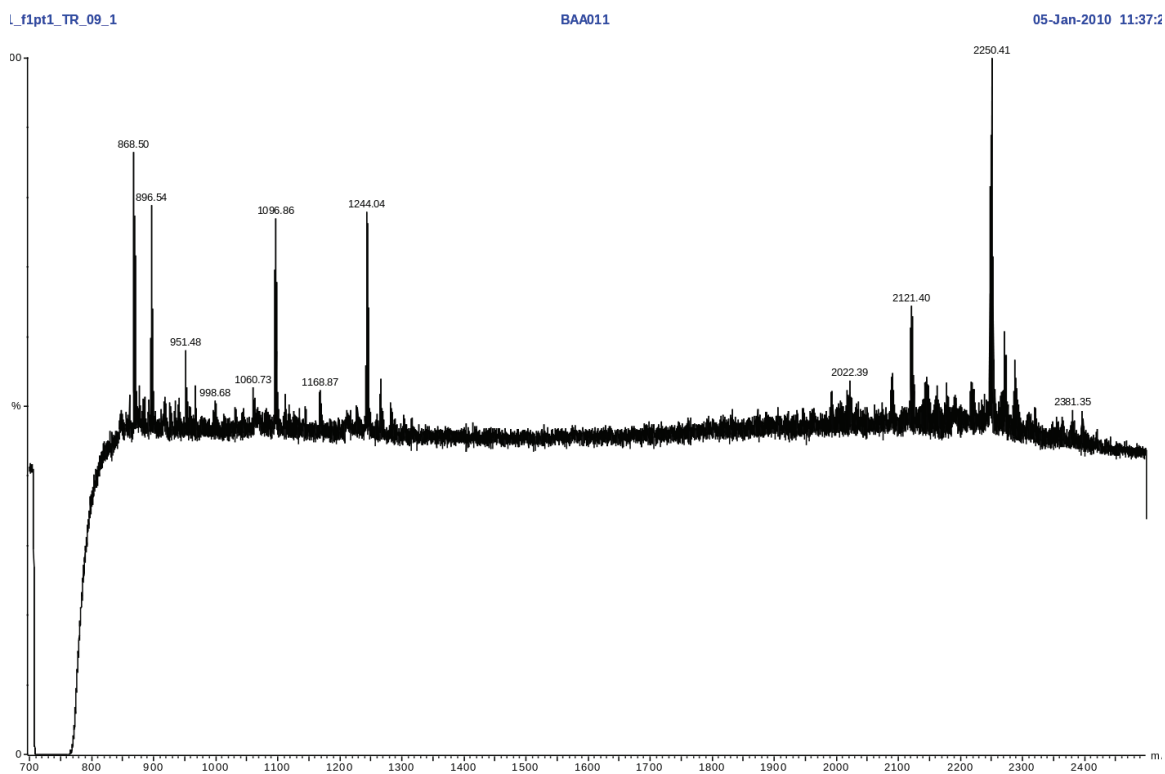


Figure B.24: MALDI reflectron-mode spectrum for the first fraction collected (16-18 minutes) from the TR-09-1 HPLC run detailed in Figure 4.22 on page 93. The target peak  $\sim 2250$  m/z is larger than the truncated product at  $\sim 2121$  m/z, but (ubiquitous) low molecular weight compounds at  $\sim 1096$  m/z and  $\sim 1244$  m/z are also co-eluting.

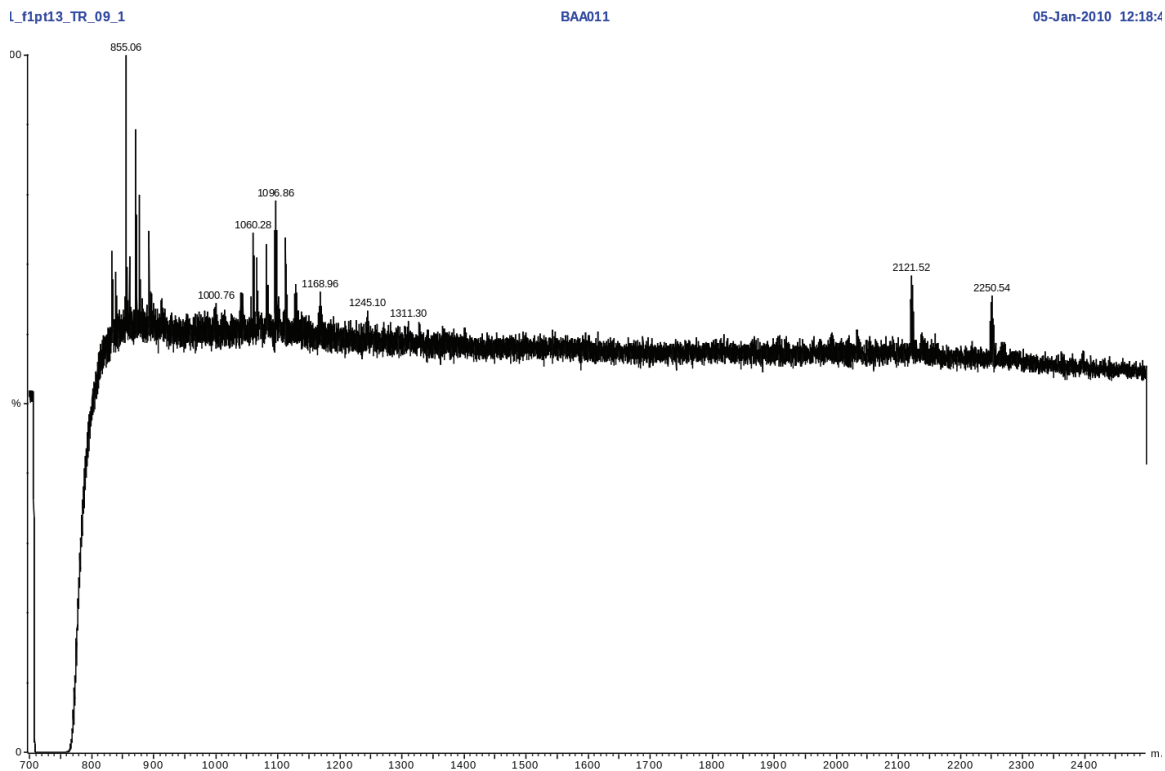


Figure B.25: MALDI reflectron-mode spectrum for the last fraction collected (40-42 minutes) from the TR-09-1 HPLC run detailed in Figure 4.22 on page 93. Despite the fact that this fraction elutes 26 minutes later than the fraction depicted in Figure B.24 (page 240), the product peak at  $\sim 2250$  m/z, the truncated product at  $\sim 2121$  m/z, and low molecular weight impurities at  $\sim 1096$  m/z and  $\sim 1244$  m/z, all persist in the sample.

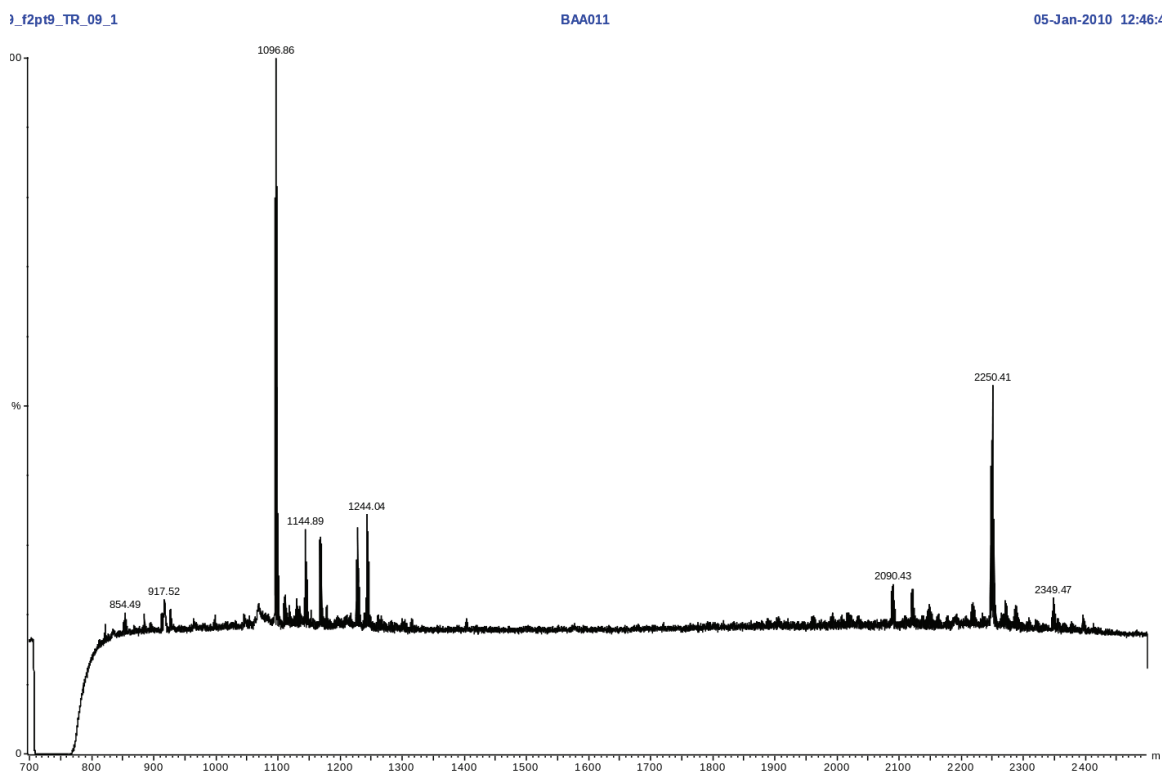


Figure B.26: MALDI reflectron-mode spectrum for the ninth fraction collected (22-24 minutes) from the TR-09-1 HPLC run detailed in Figure 4.23 on page 94. The product peak  $\sim 2250$  m/z is prominent, but the ubiquitous low molecular weight compounds at  $\sim 1096$  m/z and  $\sim 1244$  m/z have clearly co-eluted.

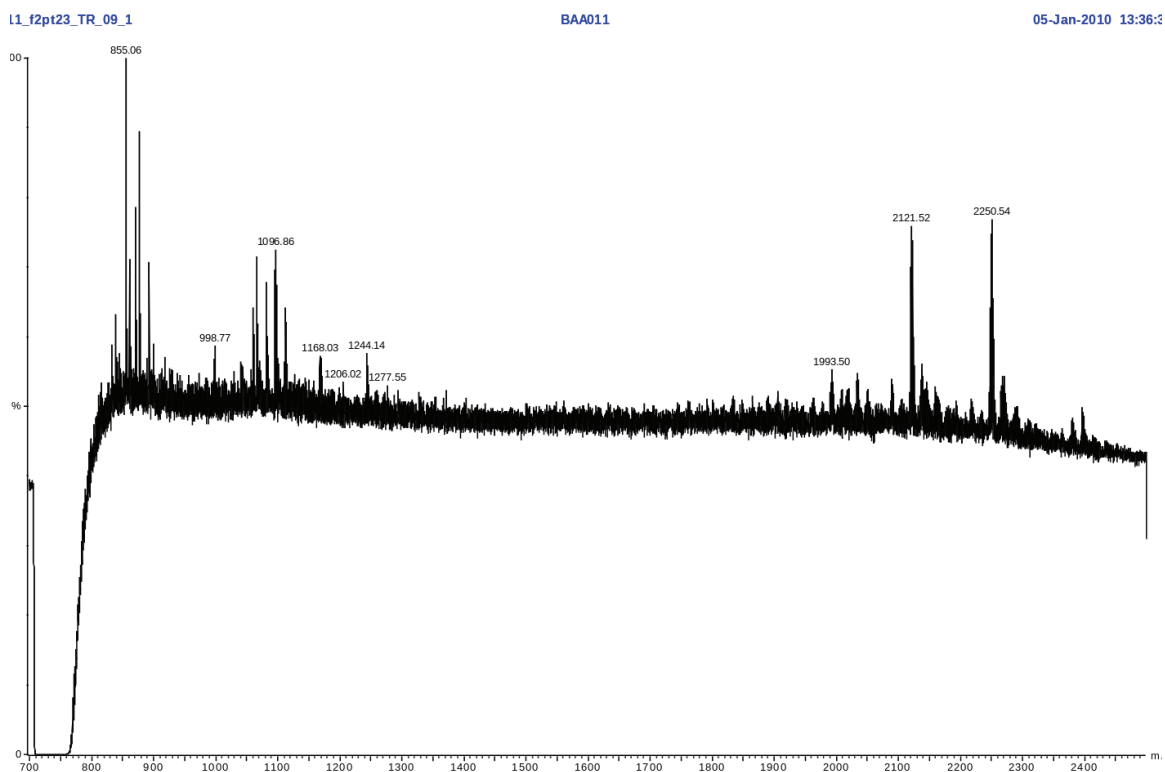


Figure B.27: MALDI reflectron-mode spectrum for the last fraction collected (50-52 minutes) from the TR-09-1 HPLC run detailed in Figure 4.23 on page 94. The product, truncated product, and low molecular weight impurities all continue to elute a full 42 minutes after the initial elution of product (see Table 4.8 on page 106).

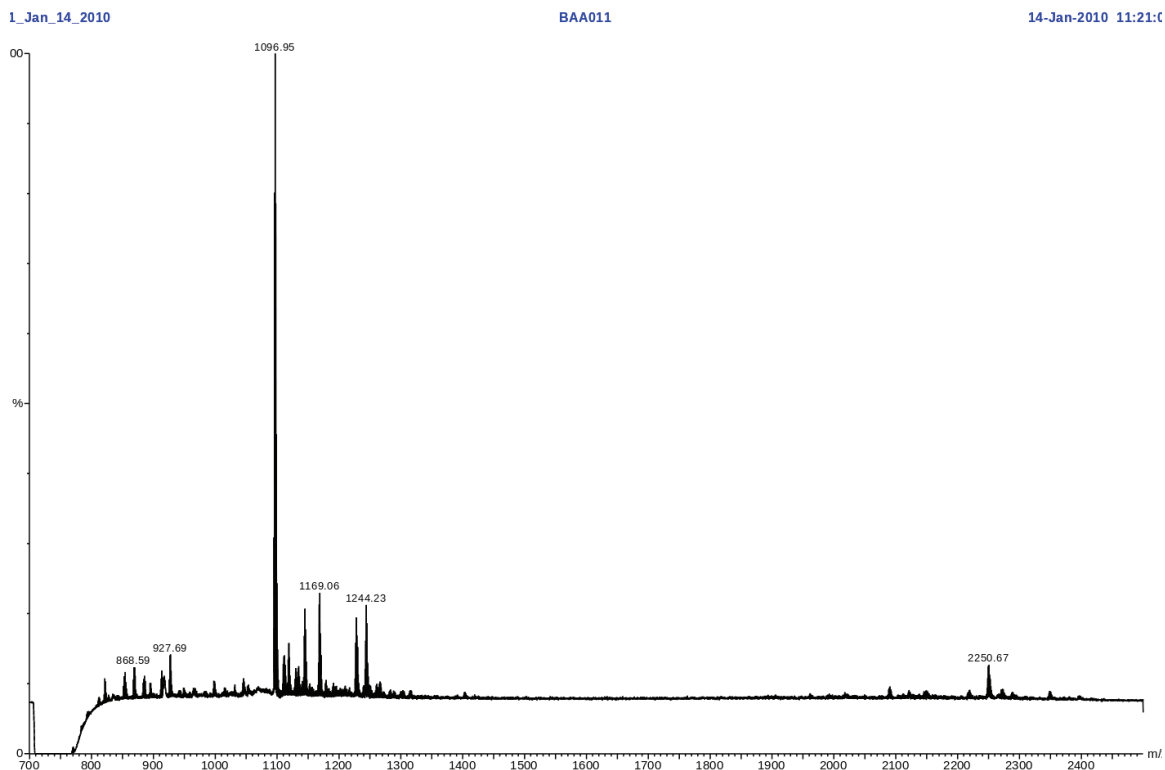


Figure B.28: TR-09-1 fractions collected in the target 22-24 minute window in replicate runs (matching the reference conditions detailed for HPLC purification in Figure 4.23 on page 94) were pooled, lyophilized, and reconstituted in a mixture of deionized water and  $\alpha$ -CHC matrix. This reflectron-mode MALDI spectrum is for the first (5.4 mg) of two pooled yields which correctly employed a 2 mL sample loop to match the reference HPLC run. Low molecular weight impurities (most notably the ubiquitous  $\sim 1096$  m/z species) are present alongside the target product  $\sim 2250$  m/z.



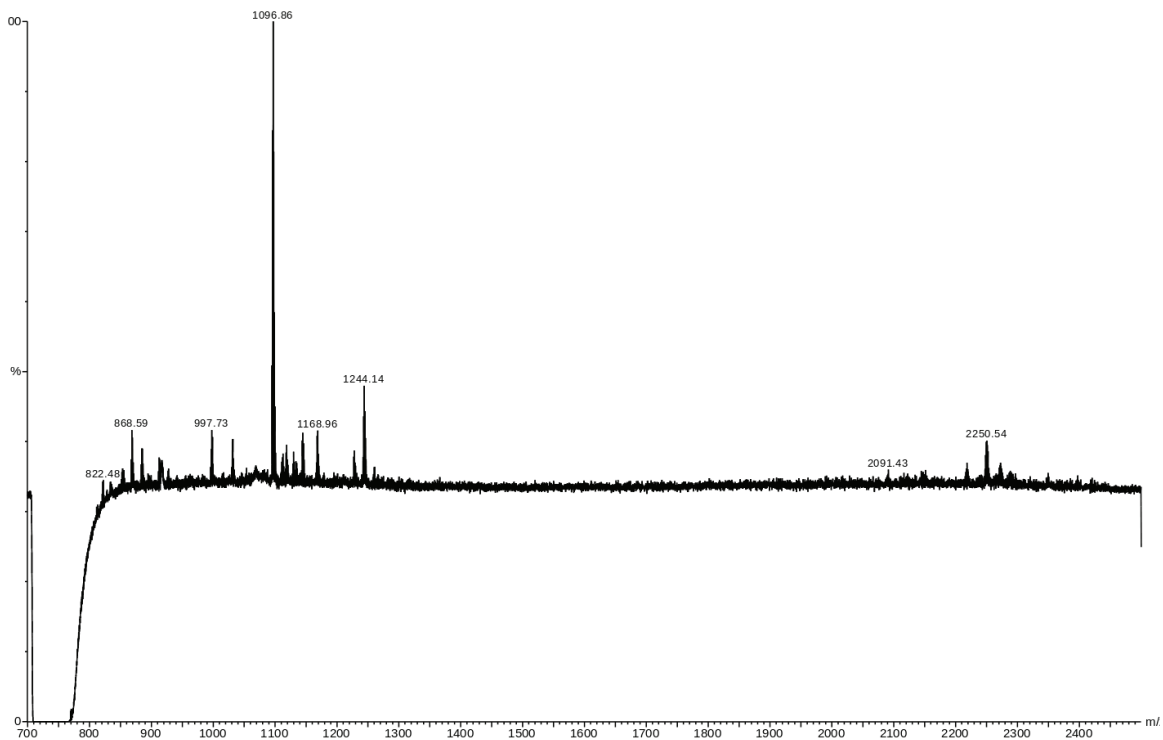


Figure B.29: This is a reflectron-mode MALDI spectrum for pooled TR-09-1 fractions (7.5 mg yield) similar to those described in Figure B.28 (page 244), but collected improperly using a 1 mL sample loop while the reference run employed a 2 mL sample loop. The resulting mass spectrum is effectively the same despite the error, and this is not surprising given the broad elution profile exhibited by TR-09-1.

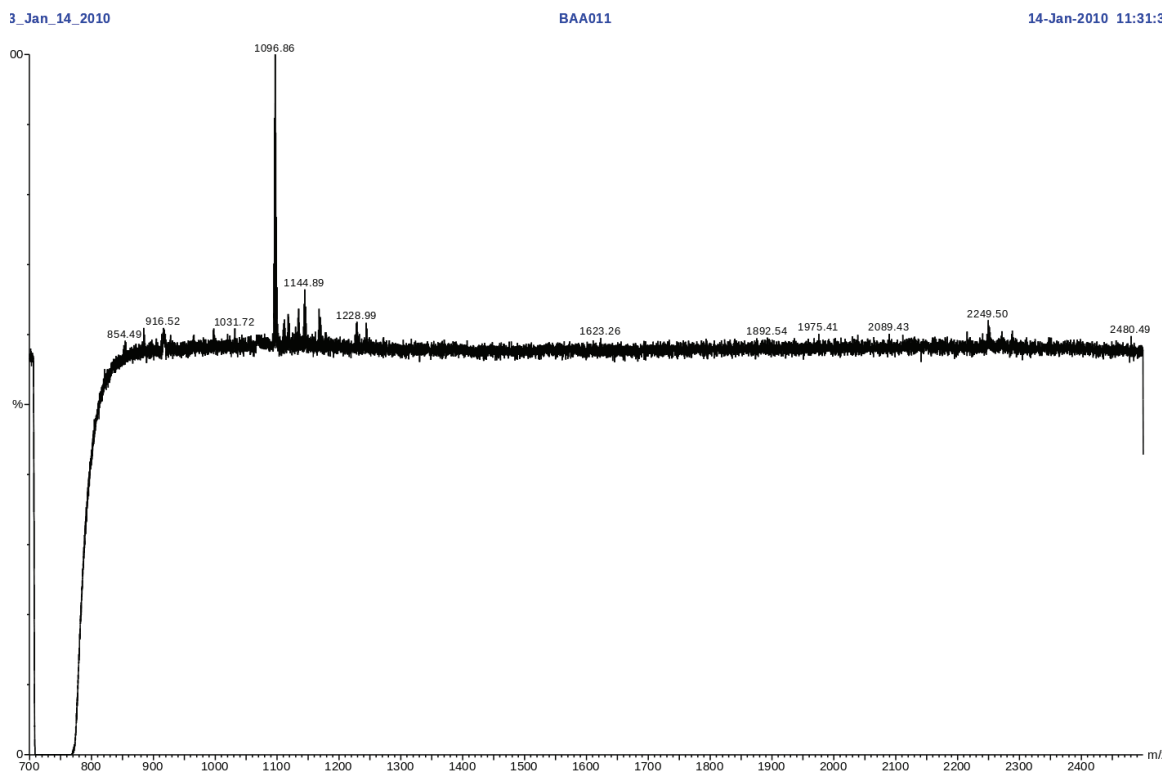


Figure B.30: This reflectron-mode MALDI spectrum was collected for the second pool (11.2 mg yield) of purified TR-09-1 fractions matching the description in Figure B.28 on page 244. Curiously, however, this matching pool of fractions exhibits less target product  $\sim 2250$  m/z relative to the impurity  $\sim 1096$  m/z.

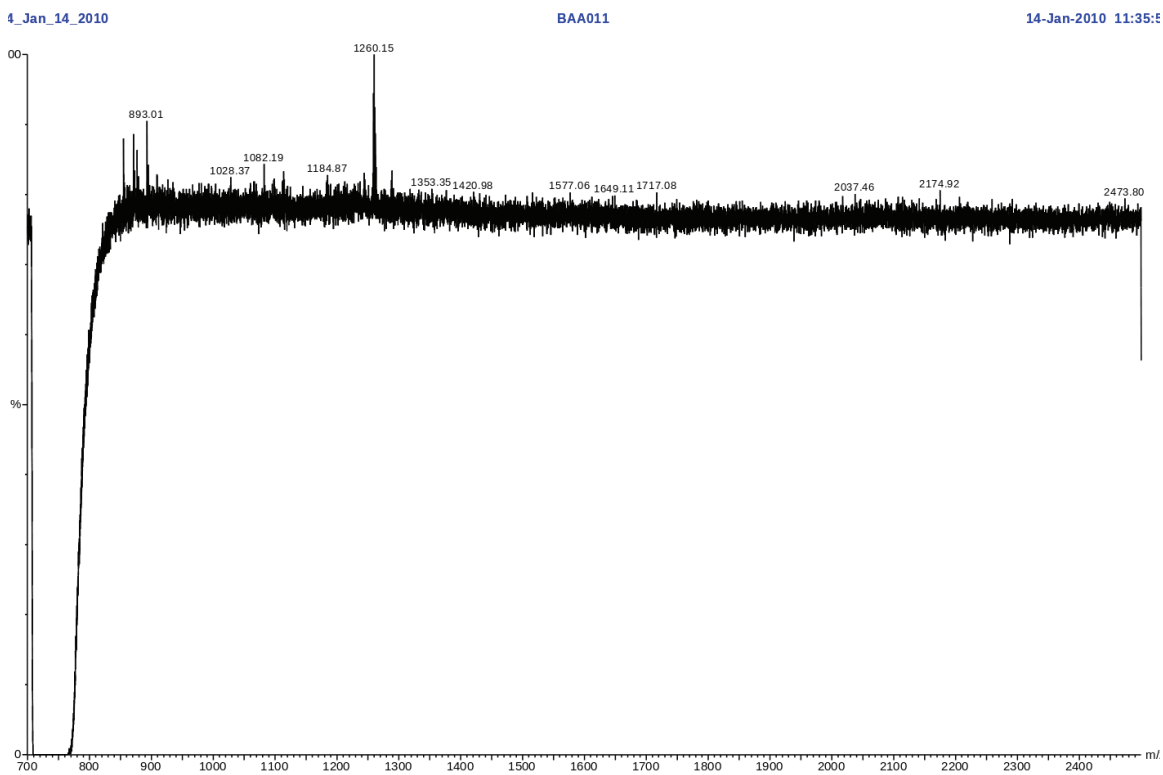


Figure B.31: A 50% DI-H<sub>2</sub>O/ $\alpha$ -CHC matrix mixture was used as a blank solution for testing the three pooled (partially purified) TR-09-1 fractions above. This MALDI spectrum does not show any substantial indication of the common low molecular weight impurities that permeate the majority of TR-09-1 HPLC fractions.

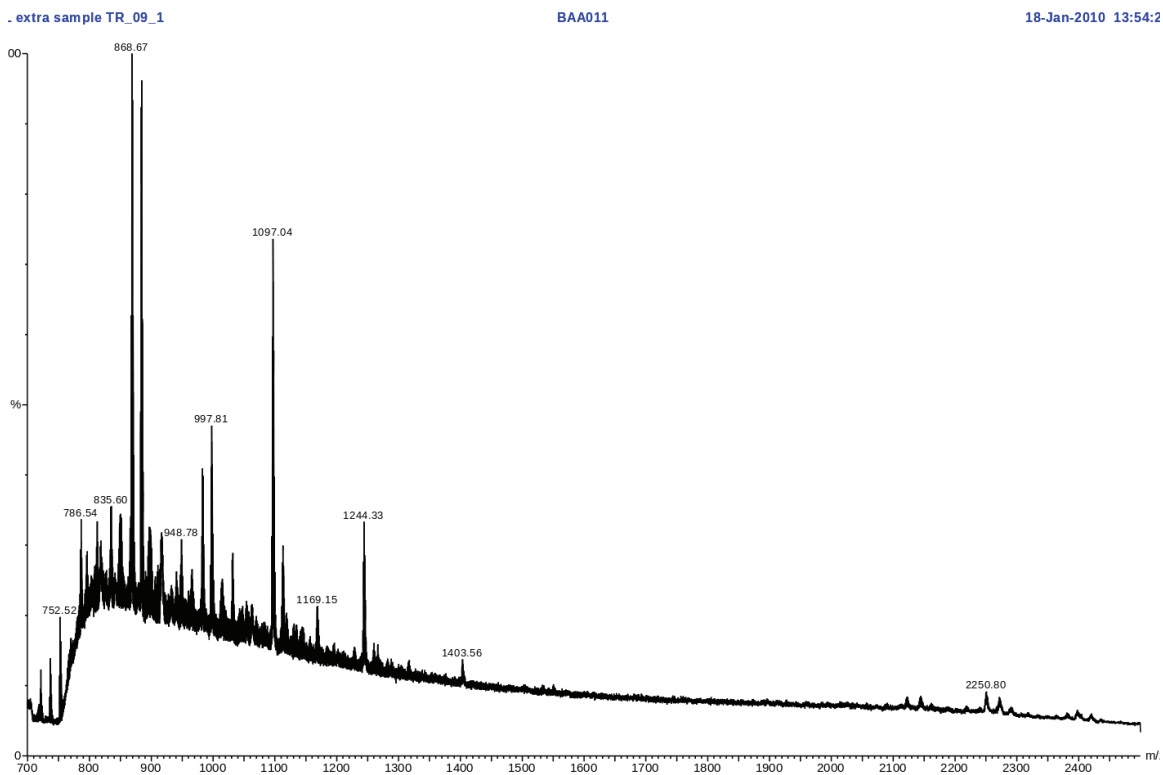


Figure B.32: Eluent collected between 24-52 minutes in TR-09-1 bulk purification runs (which match the reference run detailed in Figure 4.23 on page 94) was lyophilized and reconstituted with deionized water and  $\alpha$ -CHC matrix. This reflectron-mode MALDI spectrum demonstrates that the target product  $\sim 2250$  m/z is still present in the recovered fractions along with the ubiquitous impurities  $\sim 1096$  m/z and  $\sim 1244$  m/z.

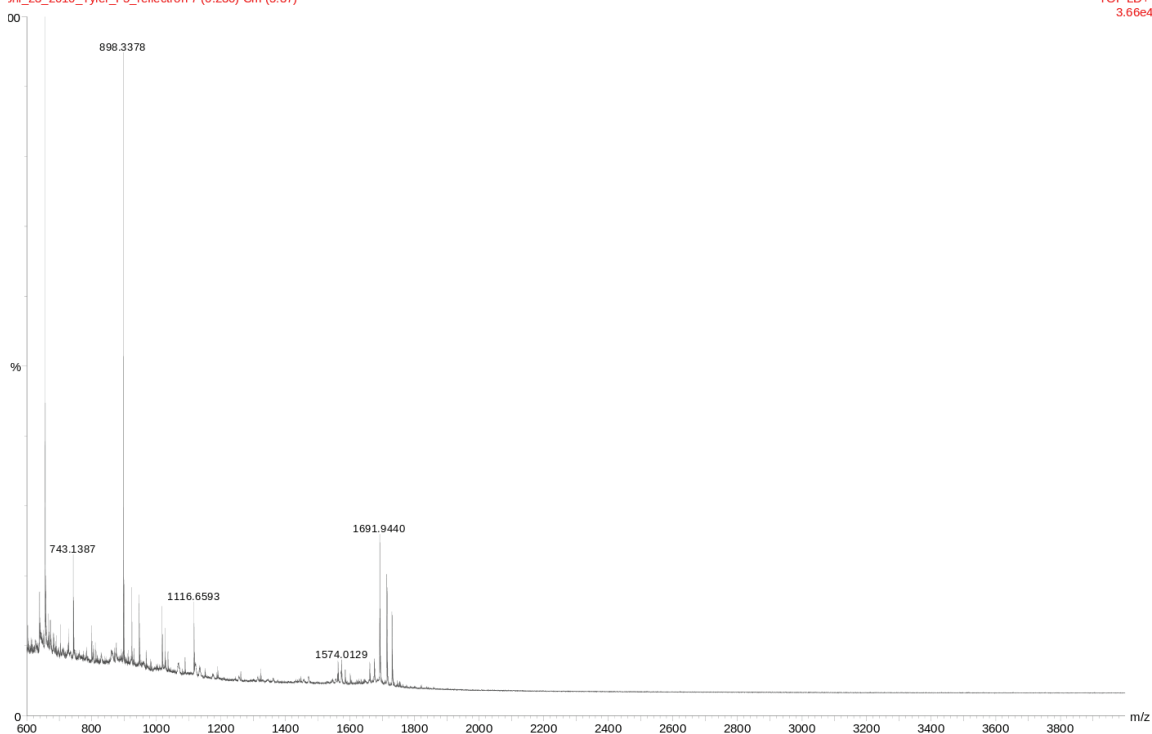


Figure B.33: Fraction 5 (30-36 minutes) from the TR-10-2 HPLC run detailed in Figure 4.27 (page 98) was diluted two-fold with  $\alpha$ -CHC matrix and this reflectron-mode MALDI spectrum was collected. The target product is clearly visible  $\sim 1691$  m/z, and its  $\text{Na}^+$  and  $\text{K}^+$  adducts are also present. However, there is also a low molecular weight impurity  $\sim 898$  m/z.

### B.1.2 TR-10-2 Construct

### B.1.3 TR-08-2 Construct

1-Apr-2010, 11:22:45, April\_23\_2010\_Tyler\_F6\_reflectron  
xrl\_23\_2010\_Tyler\_F6\_reflectron 7 (0.230) Cm (4:24)

BAA011

23-Apr-2010 11:22:45  
TOF LD+  
4.81e3

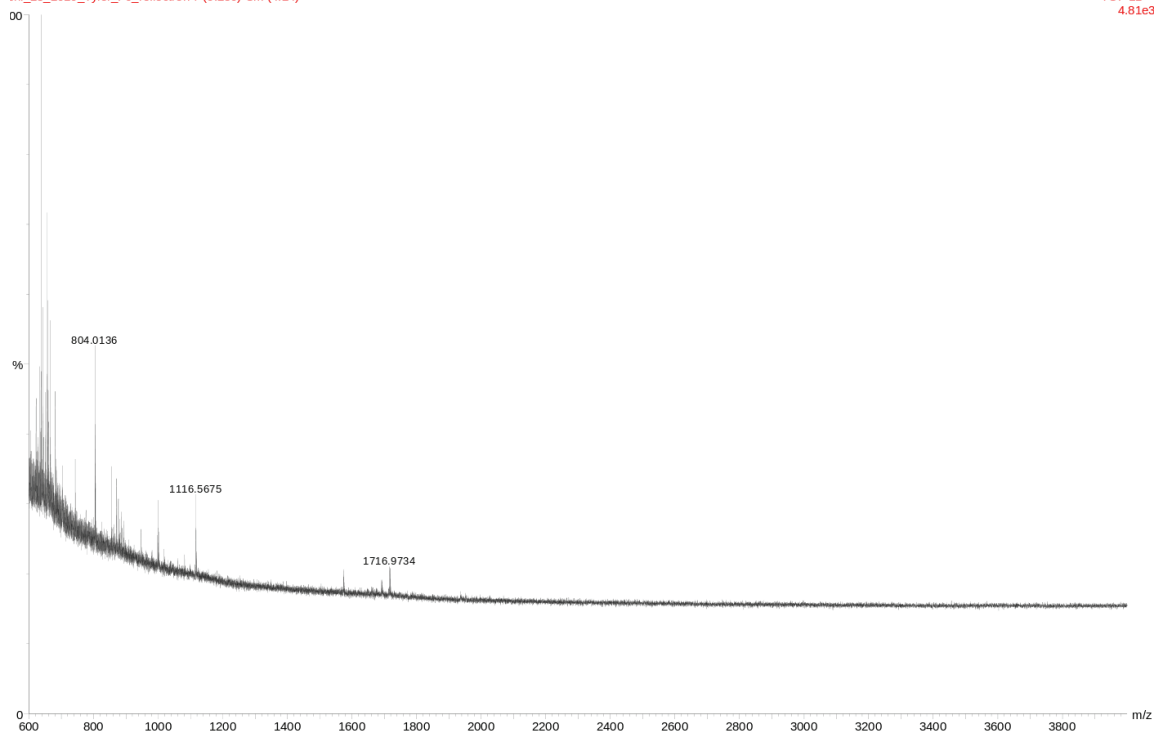


Figure B.34: Fraction 6 (36-42 minutes) from the TR-10-2 HPLC run detailed in Figure 4.27 (page 98) was diluted two-fold with  $\alpha$ -CHC matrix and this reflectron-mode MALDI spectrum was collected. The peak  $\sim 1716$  m/z most likely corresponds to the  $\text{Na}^+$  adduct of the target product ( $\sim 1691$  m/z).

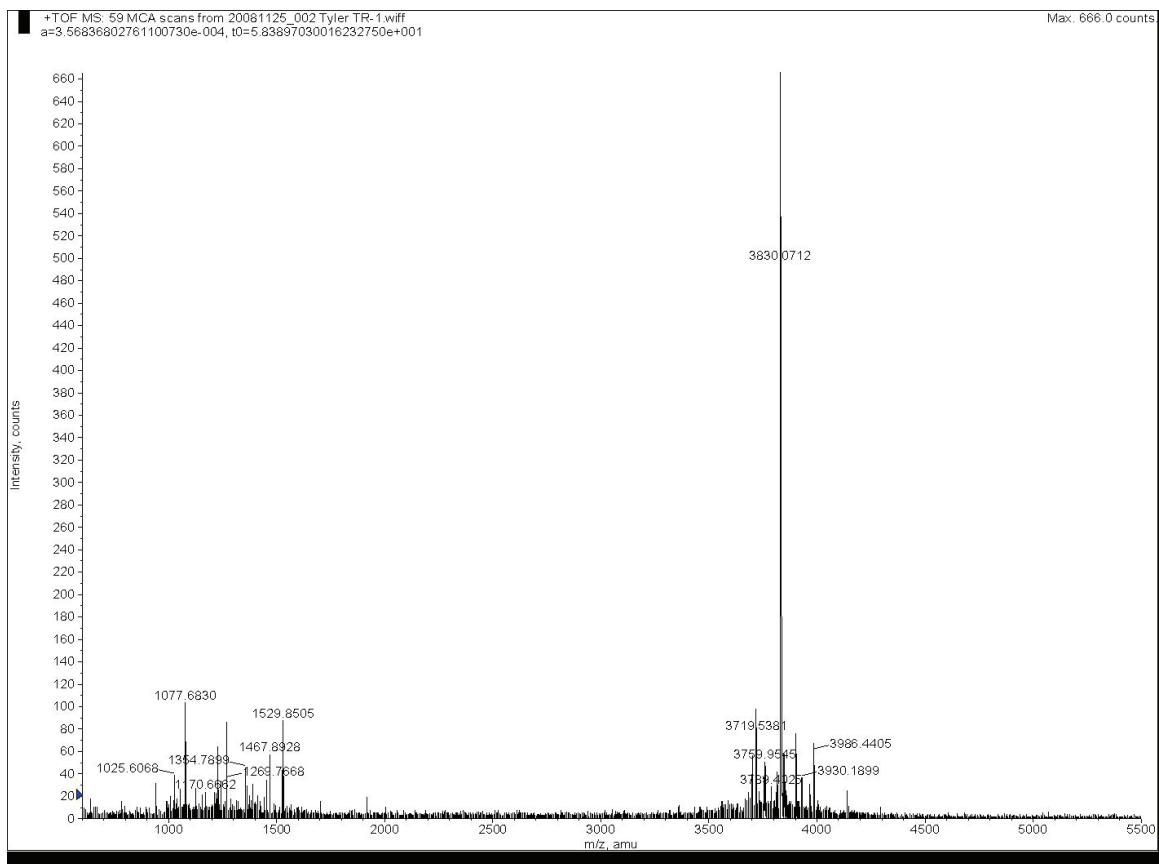


Figure B.35: MALDI spectrum for TR-08-2 fraction #1 from the HPLC run detailed in Figure 4.5 on page 77 demonstrating a relatively pure sample with major peak near the target mass  $\sim 3830$  g/mol.

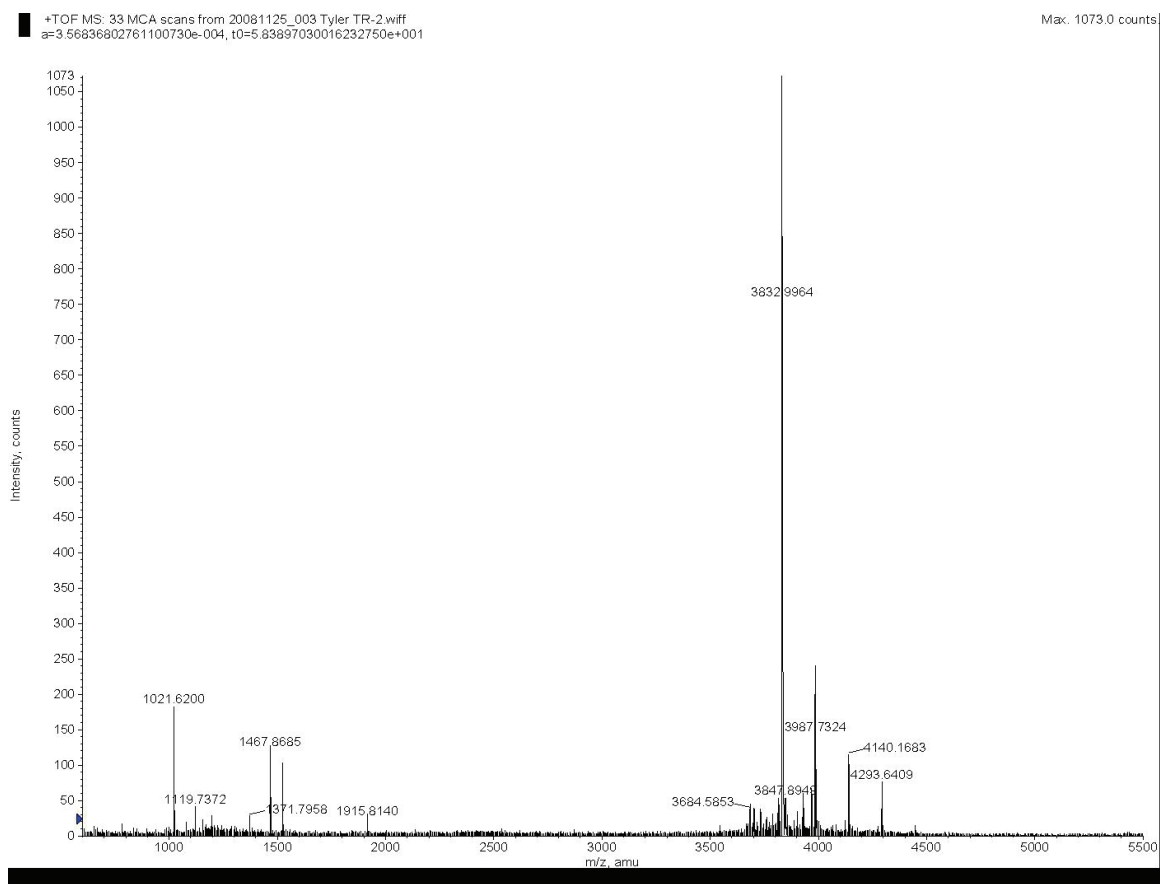


Figure B.36: MALDI spectrum for TR-08-2 fraction #2 from the HPLC run detailed in Figure 4.5 on page 77 demonstrating a relatively pure sample with major peak near the target mass ~3830 g/mol. Curiously, a fraction collected earlier in the same HPLC run produced a nearly matching MALDI profile (Figure B.35 on page 251).



# Appendix C: Source Code For Website Which Predicts NMR Spin Relaxation Parameter Trends With Magnetic Field Strength

## C.1 Introduction

The source code for the online relaxation-plotting program ([http://structbio.biochem.dal.ca/jrainey/Tyler\\_relaxation/](http://structbio.biochem.dal.ca/jrainey/Tyler_relaxation/)) is written in the Python programming language and uses the cgi module to interact with the web interface user input. I have not included all of the front-end html and css code for the website, but the source for the mathematical back-end and some of the user forms are detailed over the following pages.

## C.2 Source Code Proper

Listing C.2: Module containing basic spin relaxation constants and equations

```

1 import math
2
3 gyroH=2.6752*(10**8) #gyromagnetic ratio of 1H
4 gyroN=-2.712*(10**7) #gyromagnetic ratio of 15N
5 rNH=1.02*(10**-10) #length of N-H bond vector often used in literature
6 CSA=-160*(10**-6) #chemical shift anisotropy
7
8 #function to compute Larmor frequency in s-1 given gyromagnetic ratio (g) and magnetic
9 field strength (B)
10 def Larmor(g,B):
11     return (g*B)/(2*math.pi)
12
13 #define dipolar and CSA constants using Farrow et al., 1994 conventions:
14 d=math.sqrt((0.1*(gyroH**2)*(gyroN**2)*((6.62606896*(10**-34))**2))/(4*(math.pi**2))*(
15 rNH**6))*(10**-7)
16
17 #the constant associated with CSA is a function of external field strength(B)
18 def CSA_constant(B):
19     return math.sqrt((2.0/15)*(gyroN**2)*(B**2)*(CSA**2))
20
21 #spectral density functions:
22 def J(tau_m,Larmor_value,S_squared,tau_e,model="classical",S_squared_fast=1,
23 S_squared_slow=1,tau_slow=1*(10**-9),tau_fast=1*(10**-12)):
24     if model=="classical":
25         return (S_squared*tau_m)/(1+((Larmor_value**2)*(tau_m**2)))+(1.0-
26 S_squared)*(1.0/(1.0/tau_m+1.0/tau_e))/(1+(Larmor_value**2)
27 *((1.0/(1.0/tau_m+1.0/tau_e))**2))
28     elif model=="extended":

```

```

25     return ((S_squared_fast**2)*(S_squared_slow**2)*(tau_m))/(1+((
Larmor_value**2)*(tau_m**2))+((S_squared_fast**2)*(1-S_squared_slow
**2))*(1.0/(1.0/tau_m+1.0/tau_slow))/(1+(Larmor_value**2))*((1.0/(1.0/
tau_m+1.0/tau_slow)**2))+(1-S_squared_fast**2)*(1.0/(1.0/tau_m+1.0/
tau_fast))/(1+(Larmor_value**2))*((1.0/(1.0/tau_m+1.0/tau_fast)**2))

#Spin-lattice relaxation equation from Farrow et al., 1994:
27 def R1(tau_m,B,S_squared,tau_e,model="classical",S_squared_fast=1,S_squared_slow=1,
tau_slow=1*(10**-9),tau_fast=1*(10**-12)):
    return (d**2)*(J(tau_m,Larmor(gyroH,B)-Larmor(gyroN,B),S_squared,tau_e,model,
S_squared_fast,S_squared_slow,tau_slow,tau_fast)+3*J(tau_m,Larmor(gyroN,B),
S_squared,tau_e,model,S_squared_fast,S_squared_slow,tau_slow,tau_fast)+6*J(
tau_m,Larmor(gyroH,B)+Larmor(gyroN,B),S_squared,tau_e,model,S_squared_fast,
S_squared_slow,tau_slow,tau_fast))+((CSA_constant(B)**2)*J(tau_m,Larmor(gyroN
,B),S_squared,tau_e,model,S_squared_fast,S_squared_slow,tau_slow,tau_fast))
#quality check with Maple: #print R1(10**-8,11.7,0.85,50*(10**-12))

29

31 #Spin-spin relaxation equation from Farrow et al., 1994:
def R2(tau_m,B,S_squared,tau_e,model="classical",S_squared_fast=1,S_squared_slow=1,
tau_slow=1*(10**-9),tau_fast=1*(10**-12)):
    return ((d**2)/2.0)*(4.0*J(tau_m,0.0,S_squared,tau_e,model,S_squared_fast,
S_squared_slow,tau_slow,tau_fast)+J(tau_m,Larmor(gyroH,B)-Larmor(gyroN,B),
S_squared,tau_e,model,S_squared_fast,S_squared_slow,tau_slow,tau_fast)+3.0*J(
tau_m,Larmor(gyroN,B),S_squared,tau_e,model,S_squared_fast,S_squared_slow,
tau_slow,tau_fast)+6*J(tau_m,Larmor(gyroH,B),S_squared,tau_e,model,
S_squared_fast,S_squared_slow,tau_slow,tau_fast)+6*J(tau_m,Larmor(gyroH,B)+
Larmor(gyroN,B),S_squared,tau_e,model,S_squared_fast,S_squared_slow,tau_slow,
tau_fast))+((CSA_constant(B)**2)/6.0)*(3*J(tau_m,Larmor(gyroN,B),S_squared,
tau_e,model,S_squared_fast,S_squared_slow,tau_slow,tau_fast)+4*J(tau_m,0.0,
S_squared,tau_e,model,S_squared_fast,S_squared_slow,tau_slow,tau_fast))

```

```

#quality check with Maple: #print R2(10** -8, 11.7, 0.35, 50*(10** -12))

#Steady-state NOE equation from Farrow et al., 1994:
def NOE(tau_m, B, S_squared, tau_e, model="classical", S_squared_fast=1, S_squared_slow=1,
tau_slow=1*(10** -9), tau_fast=1*(10** -12)):
    return 1+(gyroH/gyroN)*(d**2)*(6*J(tau_m, Larmor(gyroH, B)+Larmor(gyroN, B),
S_squared, tau_e, model, S_squared_fast, S_squared_slow, tau_slow, tau_fast))-J(
tau_m, Larmor(gyroH, B)-Larmor(gyroN, B), S_squared, tau_e, model, S_squared_fast,
S_squared_slow, tau_slow, tau_fast))*(1/R1(tau_m, B, S_squared, tau_e, model,
S_squared_fast, S_squared_slow, tau_slow, tau_fast))
#quality check with Maple: #print NOE(10** -8, 11.7, 0.85, 50*(10** -12))

```

35

37

39

Listing C.3: Object-oriented backbone module of the spin relaxation plotting program

```

1  import sys
   import math
3  import Gnuplot, Gnuplot.funcutils
   from constants_equations import *
5  import os
   import decimal
7  import random
   import cgitb; cgitb.enable()
9  import time
   import csv

11 decimal.getcontext().prec=2
   #os.putenv('DISPLAY', ':0')
   #os.putenv('GNUTERM', 'postscript')

13
15
17 class multi_plotter:
   def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
       =10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
       S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
       #self.varied_parameter=varied_parameter
21      self.npar=npar+1 #number of gradations
       #self.unique_name=unique_name
       #self.varied_parameter=varied_parameter
23      #self.relax_parameter=relax_parameter
       self.parameter_min=parameter_min
       self.parameter_max=parameter_max

```

```

27 self.field_values=field_values
28 self.parameter_value_list=[]
29 self.L_plot=[]
30 self.plot_list=[]
31 self.plot_object_list=[]
32 self.relax_plot_list=['T_1','T_2','steady-state NOE','R_1','R_2']
33 self.y_label_list=['T_1 (s)','T_2 (s)','Steady-state NOE','R_1 (s^{-1})',
34                   ', 'R_2 (s^{-1})' ]
35 self.file_namer=['T1','T2','NOE','R1','R2']
36 self.plot_units_list=['(s)','(s)','(s^{-1})','(s^{-1})']
37 self.file_label_list=['<em>T</em>1</sub></em>','<em>T</em>2</sub></em>',
38                       '<em>NOE</em>','<em>R</em>1</sub></em>','<em>R</em>2</sub></em>']
39 #self.S_squared=S_squared
40 #self.tau_e=tau_e
41 #self.tau_m=tau_m
42 #self.model=model
43 #self.S_squared_fast=S_squared_fast
44 #self.S_squared_slow=S_squared_slow
45 #self.tau_fast=tau_fast
46 #self.tau_slow=tau_slow
47
48 for i in range(self.npar): #generate list of param values
49     self.parameter_value_list.append(self.parameter_min+i*((self.
50         parameter_max-self.parameter_min)/self.npar))
51     self.parameter_value_list.append(self.parameter_max)
52
53 #abstract superclass--subclasses fill in the work for the intervening
54 blocks here.

```

```

53 def plot_2(self):
54     #Splice the list in blocks by the varied parameter in preparation for
55     plotting:
56     for i in range(len(self.parameter_value_list)):
57         self.plot_list.append(self.L_plot[0+i*self.field_values:self.
58             field_values+i*self.field_values])
59     #Generate the physical gnuplot objects that can be plotted by gnuplot:
60     for i in range(len(self.parameter_value_list)):
61         #print decimal.Decimal(str(self.parameter_value_list[i]))
62         title_iteration=str(decimal.Decimal(str(self.
63             parameter_value_list[i]))*decimal.Decimal('1.00'))+self.
64             parameter_unit_label
65         self.plot_object=Gnuplot.PlotItems.Data(self.plot_list[i],
66             with_='lines', title=title_iteration)
67         self.plot_object_list.append(self.plot_object)
68     #control appearance of the plot generated by gnuplot:
69     self.graph_title='set title "Influence of %s on trend of %s with
70         magnetic field strength\\n\\n(%s %s)"' % self.g
71         self.gp=Gnuplot.Gnuplot(persist=0)
72         self.gp('set term postscript enhanced color')
73         self.gp('set data style lines')
74         self.gp(self.graph_title)
75         self.gp('set xlabel "Magnetic field strength (Tesla)"')
76         self.gp(self.y_label)
77         self.gp('set xrange [0:25]')
78         self.gp('set key outside bottom box') #position of legend
79         self.file_name='%stmp_%.d.ps' % (self.name_prefix,os.getpid())
80         self.input_to_gp='set output "../relaxation_tmp/%.s"' % self.file_name

```

```

77 self.gp(self.input_to_gp)
79 #curiously gnuplot doesn't seem to tolerate a python list of PlotItems.
    Data, but will tolerate explicit indexed values separated by commas
    as below... very odd
81 #would definitely be nice to be able to condense this redundant code...
    if self.npar==15:
        self.gp.plot(self.plot_object_list[0],self.plot_object_list[1],
            self.plot_object_list[2],self.plot_object_list[3],self.
            plot_object_list[4],self.plot_object_list[5],
83 self.plot_object_list[6],self.plot_object_list[7],self.plot_object_list[8],self.
            plot_object_list[9],self.plot_object_list[10],self.plot_object_list[11],
            self.plot_object_list[12],self.plot_object_list[13],self.plot_object_list[14],self.
            plot_object_list[15])
        elif self.npar==14:
            self.gp.plot(self.plot_object_list[0],self.plot_object_list[1],
                self.plot_object_list[2],self.plot_object_list[3],self.
                plot_object_list[4],self.plot_object_list[5],
87 self.plot_object_list[6],self.plot_object_list[7],self.plot_object_list[8],self.
                plot_object_list[9],self.plot_object_list[10],self.plot_object_list[11],
                self.plot_object_list[12],self.plot_object_list[13],self.plot_object_list[14])
            elif self.npar==13:
                self.gp.plot(self.plot_object_list[0],self.plot_object_list[1],
                    self.plot_object_list[2],self.plot_object_list[3],self.
                    plot_object_list[4],self.plot_object_list[5],
91 self.plot_object_list[6],self.plot_object_list[7],self.plot_object_list[8],self.
                    plot_object_list[9],self.plot_object_list[10],self.plot_object_list[11],
                    self.plot_object_list[12],self.plot_object_list[13])
                elif self.npar==12:

```



```

self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
self.plot_object_list[2], self.plot_object_list[3], self.
plot_object_list[4], self.plot_object_list[5],
self.plot_object_list[6], self.plot_object_list[7], self.plot_object_list[8], self.
plot_object_list[9], self.plot_object_list[10], self.plot_object_list[11], self.
plot_object_list[12])
elif self.npar==11:
self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
self.plot_object_list[2], self.plot_object_list[3], self.
plot_object_list[4], self.plot_object_list[5],
self.plot_object_list[6], self.plot_object_list[7], self.plot_object_list[8], self.
plot_object_list[9], self.plot_object_list[10], self.plot_object_list[11])
elif self.npar==10:
self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
self.plot_object_list[2], self.plot_object_list[3], self.
plot_object_list[4], self.plot_object_list[5],
self.plot_object_list[6], self.plot_object_list[7], self.plot_object_list[8], self.
plot_object_list[9], self.plot_object_list[10])
elif self.npar==9:
self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
self.plot_object_list[2], self.plot_object_list[3], self.
plot_object_list[4], self.plot_object_list[5],
self.plot_object_list[6], self.plot_object_list[7], self.plot_object_list[8], self.
plot_object_list[9])
elif self.npar==8:
self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
self.plot_object_list[2], self.plot_object_list[3], self.
plot_object_list[4], self.plot_object_list[5],
self.plot_object_list[6], self.plot_object_list[7], self.plot_object_list[8])
elif self.npar==7:

```

95

97

99

101

103

105

107

```

109         self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
110             self.plot_object_list[2], self.plot_object_list[3], self.
111             plot_object_list[4], self.plot_object_list[5],
112             self.plot_object_list[6], self.plot_object_list[7])
113         elif self.npar==6:
114             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
115                 self.plot_object_list[2], self.plot_object_list[3], self.
116                 plot_object_list[4], self.plot_object_list[5], self.
117                 plot_object_list[6])
118         elif self.npar==5:
119             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
120                 self.plot_object_list[2], self.plot_object_list[3], self.
121                 plot_object_list[4], self.plot_object_list[5])
122         elif self.npar==4:
123             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
124                 self.plot_object_list[2], self.plot_object_list[3], self.
125                 plot_object_list[4])
126         elif self.npar==3:
127             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
128                 self.plot_object_list[2], self.plot_object_list[3])
129         elif self.npar==2:
130             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1],
131                 self.plot_object_list[2])
132         elif self.npar==1:
133             self.gp.plot(self.plot_object_list[0], self.plot_object_list[1])
134         else:
135             print 'Number of gradations must be between 0 and 14'
136             sys.exit()
137
138 self.input_file='../relaxation_tmp/'+self.file_name

```

```

129 self.output_file='../relaxation_tmp/'+self.file_name.replace('ps', 'png'
130 )
131 convert_input='/usr/bin/convert -rotate 90 %s %s' % (self.input_file,
132 self.output_file)
133 time.sleep(1) # the sleep delay (1 second) is needed because on the Red
134 Hat server it seems that .png files sometimes aren't being produced
135 from the .ps-->speculating this is because gnuplot isn't closing fast
136 enough before ImageMagick tries to find the .ps file.
137 #also, shorter than 1 second seems to be too short, and missing image
138 links show up occasionally
139 os.system(convert_input)
140
141 print '<center></center>' % self.
142 output_file.replace('../relaxation_tmp', '')
143
144 #CSV output:
145 self.csv_out()
146 #reset lists
147 self.L_plot=[]
148 self.plot_list=[]
149 self.plot_object_list=[]
150 self.set_label=[]
151 self.plot_list_labelled=[]
152 self.position+=520
153 print '<hr />'
154
155 def csv_out(self):
156     self.plot_list_labelled=self.plot_list

```

```

151 file_string='.../relaxation_tmp/%stext_out%d.csv' % (self.
    varied_parameter, os.getpid()+random.randint(0,10**12))
152 f=open(file_string, 'w')
153 csv_f=csv.writer(f, dialect='excel')
    first_line='Field strength (Tesla),%s %s\n' % (self.relax_parameter,
    self.plot_units)
    # f.write(first_line)
    #self.set_label=[]
154 #self.set_label.append(self.varied_parameter)+'+str(self.
    parameter_value_list[0])+self.parameter_unit_label)
155 #for i in range(1,len(self.parameter_value_list)):
    # string_sub=self.varied_parameter+''+str(self.
    parameter_value_list[i])+self.parameter_unit_label
156 # self.set_label.append(string_sub)
157 #self.plot_list_labelled[0][0:0]=self.set_label[0],
    #for i in range(self.npar):
    # self.plot_list_labelled[i].append(self.set_label[i+1])
158
159 #f.write(str(self.plot_list_labelled)+'\n')
160
161 #for value in range(self.npar):
    #first_line_list=str(self.parameter_value_list[value])+self.
    parameter_unit_label
162
163 for i in range(len(self.parameter_value_list)):
    f.write(self.varied_parameter+''+'+'+str(self.
    parameter_value_list[i])+'+self.parameter_unit_label+'')
164
165 f.write('\n')
166
167
168
169
170
171
172
173

```

```

175 f.write(((Field strength (Tesla)),%s %s, ' % (self.relax_parameter, self.
176 plot_units))*(self.npar+1))+'\n')
177
178
179 for field_value in range(26): #field strength always varies b/w 0 and
25 Tesla
    row=[] #reset row list each iteration
    for curve in self.plot_list_labelled: #three is one curve for
        each set of parameter values
            row.append(curve[field_value]) #the row variable thus
            contains the x,y coordinates for a given x (field
            strength) for all curves
            f.write(str(row).replace('[', '').replace(',', '').replace('\n', '\n'))
181 f.close()
182
183 print '''<div class="extra_data" style="top:%dpx"><h4>Additional data</
184 h4><a href="%s">%s: CSV plot data file</a></div>''' % (self.position,
185 file_string.replace('..', './'), self.file_label)
186
187 class tau_m_varied_plotter(multi_plotter):
188     def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
189 =10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
190 S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
191     multi_plotter.__init__(self, parameter_min, parameter_max, npar,
192 field_values=26, tau_m=10**-8, S_squared=0.85, tau_e=50*(10**-12), model=
193 "classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**-9),
194 tau_fast=1*(10**-12))
195     self.i=5 #arbitrary
196     self.value=5 #arbitrary
197     self.symbol='{Symbol t}_m'
198     self.parameter_unit_label=' s'

```

```

self.constant_1='S^2'+'+'+str(S_squared)
self.constant_2=' {/Symbol t}_e'+'+'+str(tau_e)+' s'
self.varied_parameter='tau_m'

def plot_1(self,parameter_min,parameter_max,npar,field_values=26,tau_m=10**-8,
S_squared=0.85,tau_e=50*(10**-12),model="classical",S_squared_fast=1,
S_squared_slow=1,tau_slow=1*(10**-9),tau_fast=1*(10**-12)): #plot 1 is the
handler so call it to run plot 1 and plot 2 for a given instance
    self.position=10
    for j in range(5):
        self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
            self.constant_2)
        self.y_label="set ylabel '%s'" % self.y_label_list[j]
        for self.tau_m in self.parameter_value_list:
            for i in range(self.field_values):
                self.L_plot.append([i,[1.0/R1(self.tau_m,i,
                    S_squared,tau_e,model,S_squared_fast,
                    S_squared_slow,tau_slow,tau_fast),1.0/R2(self
                        .tau_m,i,S_squared,tau_e,model,S_squared_fast
                            ,S_squared_slow,tau_slow,tau_fast),NOE(self.
                                tau_m,i,S_squared,tau_e,model,S_squared_fast,
                                    S_squared_slow,tau_slow,tau_fast),R1(self.
                                        tau_m,i,S_squared,tau_e,model,S_squared_fast,
                                            S_squared_slow,tau_slow,tau_fast),R2(self.
                                                tau_m,i,S_squared,tau_e,model,S_squared_fast,
                                                    S_squared_slow,tau_slow,tau_fast)]]])

self.name_prefix=self.file_namer[j]+str(random.randint
(0,10**12))
self.relax_parameter=self.relax_plot_list[j]

```

193

195

197

199

201

266

203

205

```

207     self.plot_units=self.plot_units_list[j]
208     self.file_label=self.file_label_list[j]
209     self.plot_2()
210
211 class tau_e_varied_plotter(multi_plotter):
212     def __init__(self,parameter_min,parameter_max,npar,field_values=26,tau_m
213     =10**-8,S_squared=0.85,tau_e=50*(10**-12),model="classical",S_squared_fast=1,
214     S_squared_slow=1,tau_slow=1*(10**-9),tau_fast=1*(10**-12)):
215         multi_plotter.__init__(self,parameter_min,parameter_max,npar,
216         field_values=26,tau_m=10**-8,S_squared=0.85,tau_e=50*(10**-12),model=
217         "classical",S_squared_fast=1,S_squared_slow=1,tau_slow=1*(10**-9),
218         tau_fast=1*(10**-12))
219         self.i=5 #arbitrary
220         self.value=5 #arbitrary
221         self.symbol='{/Symbol t}_e'
222         self.parameter_unit_label=' s'
223         self.constant_1='S^2'+'+'+str(S_squared)
224         self.constant_2=' {/Symbol t}_m'+'+'+str(tau_m)+' s'
225         self.varied_parameter='tau_e'
226
227     def plot_1(self,parameter_min,parameter_max,npar,field_values=26,tau_m=10**-8,
228     S_squared=0.85,tau_e=50*(10**-12),model="classical",S_squared_fast=1,
229     S_squared_slow=1,tau_slow=1*(10**-9),tau_fast=1*(10**-12)): #plot 1 is the
230     handler so call it to run plot 1 and plot 2 for a given instance
231         self.position=10
232         for j in range(5):
233             self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
234             self.constant_2)
235             self.y_label="set ylabel '%s'" % self.y_label_list[j]
236             for self.tau_e in self.parameter_value_list:

```

```

231         for i in range(self.field_values):
232             self.L_plot.append([i, [1.0/R1(tau_m, i, S_squared
233                 , self.tau_e, model, S_squared_fast,
234                 S_squared_slow, tau_slow, tau_fast), 1.0/R2(
235                 tau_m, i, S_squared, self.tau_e, model,
236                 S_squared_fast, S_squared_slow, tau_slow,
237                 tau_fast), NOE(tau_m, i, S_squared, self.tau_e,
238                 model, S_squared_fast, S_squared_slow, tau_slow,
239                 tau_fast), R1(tau_m, i, S_squared, self.tau_e,
240                 model, S_squared_fast, S_squared_slow, tau_slow,
241                 tau_fast), R2(tau_m, i, S_squared, self.tau_e,
242                 model, S_squared_fast, S_squared_slow, tau_slow,
243                 tau_fast)]]])
244
245     self.name_prefix=self.file_namer[j]+str(random.randint
246         (0, 10**12))
247     self.relax_parameter=self.relax_plot_list[j]
248     self.plot_units=self.plot_units_list[j]
249     self.file_label=self.file_label_list[j]
250     self.plot_2()
251
252     class S_squared_varied_plotter(multi_plotter):
253         def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
254             =10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
255             S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
256             multi_plotter.__init__(self, parameter_min, parameter_max, npar,
257                 field_values=26, tau_m=10**-8, S_squared=0.85, tau_e=50*(10**-12), model=
258                 "classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**-9),
259                 tau_fast=1*(10**-12))
260             self.i=5 #arbitrary

```



```

241 self.value=5 #arbitrary
242 self.symbol='S^2'
243 self.parameter_unit_label=''
244 self.constant_1=' {/Symbol t}_e'+'+str(tau_e)+' s'
245 self.constant_2=' {/Symbol t}_m'+'+str(tau_m)+' s'
246 self.varied_parameter='S_squared'

247
248
249 def plot_1(self, parameter_min, parameter_max, npar, field_values=26, tau_m=10**-8,
250 S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
251 S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)): #plot 1 is the
252 handler so call it to run plot 1 and plot 2 for a given instance
253 self.position=10
254 for j in range(5):
255     self.g=(self.symbol, self.relax_plot_list[j], self.constant_1,
256             self.constant_2)
257     self.y_label="set ylabel '%s'" % self.y_label_list[j]
258     for self.S_squared in self.parameter_value_list:
259         for i in range(self.field_values):
260             self.L_plot.append([i, [1.0/R1(tau_m, i, self.
261 S_squared, tau_e, model, S_squared_fast,
262 S_squared_slow, tau_slow, tau_fast), 1.0/R2(
263 tau_m, i, self.S_squared, tau_e, model,
264 S_squared_fast, S_squared_slow, tau_slow,
265 tau_fast), NOE(tau_m, i, self.S_squared, tau_e,
266 model, S_squared_fast, S_squared_slow, tau_slow,
267 tau_fast), R1(tau_m, i, self.S_squared, tau_e,
268 model, S_squared_fast, S_squared_slow, tau_slow,
269 tau_fast), R2(tau_m, i, self.S_squared, tau_e,
270 model, S_squared_fast, S_squared_slow, tau_slow,
271 tau_fast)]]])

```

```

257     self.name_prefix=self.file_namer[j]+str(random.randint
258         (0,10**12))
259     self.relax_parameter=self.relax_plot_list[j]
260     self.plot_units=self.plot_units_list[j]
261     self.file_label=self.file_label_list[j]
262     self.plot_2()
263
264 class tau_fast_varied_plotter(multi_plotter):
265     def __init__(self,parameter_min,parameter_max,npar,field_values=26,tau_m
266         =10**8,S_squared=0.85,tau_e=50*(10**12),model="classical",S_squared_fast=1,
267         S_squared_slow=1,tau_slow=1*(10**9),tau_fast=1*(10**12)):
268         multi_plotter.__init__(self,parameter_min,parameter_max,npar,
269             field_values=26,tau_m=10**8,S_squared=0.85,tau_e=50*(10**12),model=
270             "classical",S_squared_fast=1,S_squared_slow=1,tau_slow=1*(10**9),
271             tau_fast=1*(10**12))
272         self.i=5 #arbitrary
273         self.value=5 #arbitrary
274         self.symbol='{/Symbol t}_{fast}'
275         self.parameter_unit_label=' s'
276         self.constant_1='S^2_{slow}'+'+'+str(S_squared_slow)+' ' + 'S^2_{
277             fast}'+'+'+str(S_squared_fast)
278         self.constant_2=' {/Symbol t}_{slow}'+'+'+str(tau_slow)+' s' + ' '+
279             '{/Symbol t}_{m}'+'+'+str(tau_m)+' s'
280         self.varied_parameter='tau_fast'
281         self.model='extended' # Extended Lipari-Szabo eq'n presented by Clore
282             et al.
283
284     def plot_1(self,parameter_min,parameter_max,npar,field_values=26,tau_m=10**8,
285         S_squared=0.85,tau_e=50*(10**12),model="classical",S_squared_fast=1,

```

```

S_squared_slow=1,tau_slow=1*(10**9),tau_fast=1*(10**-12)): #plot 1 is the
handler so call it to run plot 1 and plot 2 for a given instance
self.position=10
for j in range(5):
    self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
            self.constant_2)
    self.y_label="set ylabel '%s'" % self.y_label_list[j]
    for self.tau_fast in self.parameter_value_list:
        for i in range(self.field_values):
            self.L_plot.append([i,[1.0/R1(tau_m,i,S_squared
                ,tau_e,model,S_squared_fast,S_squared_slow,
                tau_slow,self.tau_fast),1.0/R2(tau_m,i,
                S_squared,tau_e,model,S_squared_fast,
                S_squared_slow,tau_slow,self.tau_fast),NOE(
                tau_m,i,S_squared,tau_e,model,S_squared_fast,
                S_squared_slow,tau_slow,self.tau_fast),R1(
                tau_m,i,S_squared,tau_e,model,S_squared_fast,
                S_squared_slow,tau_slow,self.tau_fast),R2(
                tau_m,i,S_squared,tau_e,model,S_squared_fast,
                S_squared_slow,tau_slow,self.tau_fast)]]])

self.name_prefix=self.file_namer[j]+str(random.randint
(0,10**12))
self.relax_parameter=self.relax_plot_list[j]
self.plot_units=self.plot_units_list[j]
self.file_label=self.file_label_list[j]
self.plot_2()

class tau_slow_varied_plotter(multi_plotter):

```

277

279

281

271

283

285

287

289

```

291 def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
=10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
292     multi_plotter.__init__(self, parameter_min, parameter_max, npar,
field_values=26, tau_m=10**-8, S_squared=0.85, tau_e=50*(10**-12), model=
"classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**-9),
tau_fast=1*(10**-12))
293     self.i=5 #arbitrary
self.value=5 #arbitrary
294     self.symbol='{/Symbol t}_{slow}'
self.parameter_unit_label=' s'
295     self.constant_1='S^2_{slow}'+'+'+str(S_squared_slow)+' ' + 'S^2_{
fast}'+'+'+str(S_squared_fast)
self.constant_2=' {/Symbol t}_{fast}'+'+'+str(tau_fast)+' s' + ' '+
296     '{/Symbol t}_{m}'+'+'+str(tau_m)+' s'
self.varied_parameter='tau_slow'
297     self.model='extended' # Extended Lipari-Szabo eq'n presented by Clore
et al.
298
299
300
301 def plot_1(self, parameter_min, parameter_max, npar, field_values=26, tau_m=10**-8,
S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)): #plot 1 is the
handler so call it to run plot 1 and plot 2 for a given instance
302     self.position=10
for j in range(5):
303         self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
self.constant_2)
304         self.y_label="set ylabel '%s'" % self.y_label_list[j]
for self.tau_slow in self.parameter_value_list:
305             for i in range(self.field_values):

```

```

309 self.L_plot.append([i, [1.0/R1(tau_m, i, S_squared
    , tau_e, model, S_squared_fast, S_squared_slow,
    self.tau_slow, tau_fast), 1.0/R2(tau_m, i,
    S_squared, tau_e, model, S_squared_fast,
    S_squared_slow, self.tau_slow, tau_fast), NOE(
    tau_m, i, S_squared, tau_e, model, S_squared_fast,
    S_squared_slow, self.tau_slow, tau_fast), R1(
    tau_m, i, S_squared, tau_e, model, S_squared_fast,
    S_squared_slow, self.tau_slow, tau_fast), R2(
    tau_m, i, S_squared, tau_e, model, S_squared_fast,
    S_squared_slow, self.tau_slow, tau_fast)]]])

311 self.name_prefix=self.file_namer[j]+str(random.randint
    (0, 10**12))
312
313 self.relax_parameter=self.relax_plot_list[j]
314 self.plot_units=self.plot_units_list[j]
315 self.file_label=self.file_label_list[j]
316 self.plot_2()

317 class S_squared_fast_varied_plotter(multi_plotter):
    def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
    =10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
    S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
    multi_plotter.__init__(self, parameter_min, parameter_max, npar,
    field_values=26, tau_m=10**-8, S_squared=0.85, tau_e=50*(10**-12), model=
    "classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**-9),
    tau_fast=1*(10**-12))
    self.i=5 #arbitrary
    self.value=5 #arbitrary
    self.symbol='S^2_{fast}'

321

```

```

323 self.parameter_unit_label=''
324 self.constant_1='S^2_{slow}'+'+'+str(S_squared_slow)+' '+'+'+' {/
325 Symbol t}_{slow}'+'+'+str(tau_slow)+' 's'
326 self.constant_2=' {/Symbol t}_{fast}'+'+'+str(tau_fast)+' 's' + ' '+'
327 {/Symbol t}_{m}'+'+'+str(tau_m)+' 's'
328 self.varied_parameter='S_squared_fast'
329 self.model='extended' # Extended Lipari-Szabo eq'n presented by Clore
330 et al.
331
332 def plot_1(self, parameter_min, parameter_max, npar, field_values=26, tau_m=10**-8,
333 S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
334 S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)): #plot 1 is the
335 handler so call it to run plot 1 and plot 2 for a given instance
336 self.position=10
337 for j in range(5):
338     self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
339             self.constant_2)
340     self.y_label="set ylabel '%s'" % self.y_label_list[j]
341     for self.S_squared_fast in self.parameter_value_list:
342         for i in range(self.field_values):
343             self.L_plot.append([i,[1.0/R1(tau_m,i,S_squared
344             ,tau_e,model,self.S_squared_fast,
345             S_squared_slow,tau_slow,tau_fast),1.0/R2(
346             tau_m,i,S_squared,tau_e,model,self.
347             S_squared_fast,S_squared_slow,tau_slow,
348             tau_fast),NOE(tau_m,i,S_squared,tau_e,model,
349             self.S_squared_fast,S_squared_slow,tau_slow,
350             tau_fast),R1(tau_m,i,S_squared,tau_e,model,
351             self.S_squared_fast,S_squared_slow,tau_slow,
352             tau_fast),R2(tau_m,i,S_squared,tau_e,model,

```

```

337         self.S_squared_fast, S_squared_slow, tau_slow,
338         tau_fast)][j])
339
340     self.name_prefix=self.file_namer[j]+str(random.randint
341         (0,10**12))
342
343     self.relax_parameter=self.relax_plot_list[j]
344     self.plot_units=self.plot_units_list[j]
345     self.file_label=self.file_label_list[j]
346     self.plot_2()
347
348 class S_squared_slow_varied_plotter(multi_plotter):
349     def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
350 =10**8, S_squared=0.85, tau_e=50*(10**12), model="classical", S_squared_fast=1,
351 S_squared_slow=1, tau_slow=1*(10**9), tau_fast=1*(10**12)):
352     multi_plotter.__init__(self, parameter_min, parameter_max, npar,
353         field_values=26, tau_m=10**8, S_squared=0.85, tau_e=50*(10**12), model=
354         "classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**9),
355         tau_fast=1*(10**12))
356     self.i=5 #arbitrary
357     self.value=5 #arbitrary
358     self.symbol='S^2_{slow}'
359     self.parameter_unit_label=''
360     self.constant_1='S^2_{fast}'+'+'+str(S_squared_fast)+' '+'+' {/
361     Symbol t}_{slow}'+'+'+str(tau_slow)+' s'
362     self.constant_2=' {/Symbol t}_{fast}'+'+'+str(tau_fast)+' s' + '+'
363     {/Symbol t}_{m}'+'+'+str(tau_m)+' s'
364     self.varied_parameter='S_squared_slow'
365     self.model='extended' # Extended Lipari-Szabo eq'n presented by Clore
366     et al.

```

```

def plot_1(self, parameter_min, parameter_max, npar, field_values=26, tau_m=10**-8,
    S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
    S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)): #plot 1 is the
    handler so call it to run plot 1 and plot 2 for a given instance
    self.position=10
    for j in range(5):
        self.g=(self.symbol,self.relax_plot_list[j],self.constant_1,
            self.constant_2)
        self.y_label="set ylabel '%s'" % self.y_label_list[j]
        for self.S_squared_slow in self.parameter_value_list:
            for i in range(self.field_values):
                self.L_plot.append([i,[1.0/R1(tau_m,i,S_squared
                    ,tau_e,model,S_squared_fast,self.
                    S_squared_slow,tau_slow,tau_fast),1.0/R2(
                    tau_m,i,S_squared,tau_e,model,S_squared_fast,
                    self.S_squared_slow,tau_slow,tau_fast),NOE(
                    tau_m,i,S_squared,tau_e,model,S_squared_fast,
                    self.S_squared_slow,tau_slow,tau_fast),R1(
                    tau_m,i,S_squared,tau_e,model,S_squared_fast,
                    self.S_squared_slow,tau_slow,tau_fast),R2(
                    tau_m,i,S_squared,tau_e,model,S_squared_fast,
                    self.S_squared_slow,tau_slow,tau_fast)][j]])

    self.name_prefix=self.file_namer[j]+str(random.randint
        (0,10**12))
    self.relax_parameter=self.relax_plot_list[j]
    self.plot_units=self.plot_units_list[j]
    self.file_label=self.file_label_list[j]
    self.plot_2()

```

357

359

361

363

365

367

369



```

371 class tau_m_extended_varied_plotter(multi_plotter):
    def __init__(self, parameter_min, parameter_max, npar, field_values=26, tau_m
=10**-8, S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)):
373     multi_plotter.__init__(self, parameter_min, parameter_max, npar,
        field_values=26, tau_m=10**-8, S_squared=0.85, tau_e=50*(10**-12), model=
        "classical", S_squared_fast=1, S_squared_slow=1, tau_slow=1*(10**-9),
        tau_fast=1*(10**-12))
375     self.i=5 #arbitrary
376     self.value=5 #arbitrary
377     self.symbol='{}/Symbol t}_{m}'
378     self.parameter_unit_label=' s'
379     self.constant_1='S^2_{fast}'+'+'+str(S_squared_fast)+ ' '+ 'S^2_{slow}
'+'+'+str(S_squared_slow)
380     self.constant_2=' {}/Symbol t}_{fast}'+'+'+str(tau_fast)+' s' + ' '+
        ' {}/Symbol t}_{slow}'+'+'+str(tau_slow)+' s'
381     self.varied_parameter='tau_m'
382     self.model='extended' # Extended Lipari-Szabo eq'n presented by Clore
        et al.
383
384     def plot_1(self, parameter_min, parameter_max, npar, field_values=26, tau_m=10**-8,
        S_squared=0.85, tau_e=50*(10**-12), model="classical", S_squared_fast=1,
        S_squared_slow=1, tau_slow=1*(10**-9), tau_fast=1*(10**-12)): #plot 1 is the
        handler so call it to run plot 1 and plot 2 for a given instance
385         self.position=10
386         for j in range(5):
387             self.g=(self.symbol, self.relax_plot_list[j], self.constant_1,
                self.constant_2)
388             self.y_label="set ylabel '%s'" % self.y_label_list[j]
389             for self.tau_m in self.parameter_value_list:

```

```

389         for i in range(self.field_values):
390             self.L_plot.append([i, [1.0/R1(self.tau_m, i,
391                                     S_squared, tau_e, model, S_squared_fast,
392                                     S_squared_slow, tau_slow, tau_fast), 1.0/R2(self
393                                     .tau_m, i, S_squared, tau_e, model, S_squared_fast
394                                     , S_squared_slow, tau_slow, tau_fast), NOE(self.
395                                     tau_m, i, S_squared, tau_e, model, S_squared_fast,
396                                     S_squared_slow, tau_slow, tau_fast), R1(self.
397                                     tau_m, i, S_squared, tau_e, model, S_squared_fast,
398                                     S_squared_slow, tau_slow, tau_fast), R2(self.
399                                     tau_m, i, S_squared, tau_e, model, S_squared_fast,
400                                     S_squared_slow, tau_slow, tau_fast)]])]
401
402         self.name_prefix=self.file_namer[j]+str(random.randint
403         (0, 10**12))
404         self.relax_parameter=self.relax_plot_list[j]
405         self.plot_units=self.plot_units_list[j]
406         self.file_label=self.file_label_list[j]
407         self.plot_2()
408
409     class single_plotter: #much better use of class structuring than 'plotter'
410         implementation for the multiple curves above
411         def __init__(self, relaxation_type, y_label, tau_m, tau_e, S_squared, field_values
412                     =26, model="classical", S_squared_fast=1, S_squared_slow=1, tau_fast=1, tau_slow
413                     =1):
414             #plotting component
415             self.relaxation_type=relaxation_type #'T_1', 'T_2' or 'NOE'; 'R_1', 'R_2
416             ,
417             self.gp=Gnuplot.Gnuplot(persist=1)

```

```

405 self.gp('set term postscript enhanced color')
406 self.gp('set data style lines')
407 self.gp('set xlabel "Magnetic field strength (Tesla)"')
408 self.gp('set xrange [0:25]')
409 self.gp('set nokey')
410 self.file_name='%s_tmp%d.ps' % (relaxation_type, os.getpid())
411 self.input_to_gp='set output "../relaxation_tmp/%s" % self.file_name
self.gp(self.input_to_gp)
self.input_file='../relaxation_tmp/'+self.file_name
self.output_file='../relaxation_tmp/'+self.file_name.replace('ps', 'png'
)
self.convert_input='/usr/bin/convert -rotate 90 %s %s' % (self.
input_file, self.output_file)
self.model=model
if self.model=="classical":
    self.graph_title='set title "Influence of magnetic field
strength on %s \\n\\n(%s %s %s)"' % (self.relaxation_type,
'{/Symbol t}_m='+str(tau_m)+' s,', ' {/Symbol t}_e='+str(
tau_e)+' s,', ' S^2='+str(S_squared))
else:
    self.graph_title='set title "Influence of magnetic field
strength on %s \\n\\n(%s %s %s)"' % (self.relaxation_type,
'{/Symbol t}_fast}'+str(tau_fast)+' s,', ' {/Symbol t}_slo
slow}'+str(tau_slow)+' s,', ' S^2_{fast}='+str(S_squared_fast
)+'', ' S^2_{slow}='+str(S_squared_slow)+'', ' + ' {/Symbol t}
_m='+str(tau_m)+' s,')
self.gp(self.graph_title)
self.y_label='set ylabel "%s" % y_label
self.gp(self.y_label)

```

405

407

409

411

413

415

279

417

419

421

423

```

self.tau_m=tau_m
self.S_squared=S_squared
self.tau_e=tau_e
self.S_squared_fast=S_squared_fast
self.S_squared_slow=S_squared_slow
self.tau_fast=tau_fast
self.tau_slow=tau_slow
self.T1_list=[]
self.T2_list=[]
self.NOE_list=[]
self.R1_list=[]
self.R2_list=[]
for i in range(field_values):
    self.T1_list.append([i,1.0/R1(self.tau_m,i,self.S_squared,self.
        tau_e,self.model,self.S_squared_fast,self.S_squared_slow,self
        .tau_fast,self.tau_slow)])
    self.T2_list.append([i,1.0/R2(self.tau_m,i,self.S_squared,self.
        tau_e,self.model,self.S_squared_fast,self.S_squared_slow,self
        .tau_fast,self.tau_slow)])
    self.NOE_list.append([i,NOE(self.tau_m,i,self.S_squared,self.
        tau_e,self.model,self.S_squared_fast,self.S_squared_slow,self
        .tau_fast,self.tau_slow)])
    self.R1_list.append([i,R1(self.tau_m,i,self.S_squared,self.
        tau_e,self.model,self.S_squared_fast,self.S_squared_slow,self
        .tau_fast,self.tau_slow)])
    self.R2_list.append([i,R2(self.tau_m,i,self.S_squared,self.
        tau_e,self.model,self.S_squared_fast,self.S_squared_slow,self
        .tau_fast,self.tau_slow)])

```

#CSV file component

```

445 def csv_printer(self):
446     if self.relaxation_type=='T_1':
447         self.plot_units='(s)'
448     elif self.relaxation_type=='T_2':
449         self.plot_units='(s)'
450     elif self.relaxation_type=='NOE':
451         self.plot_units=''
452     elif self.relaxation_type=='R_1':
453         self.plot_units='(s^{-1})'
454     elif self.relaxation_type=='R_2':
455         self.plot_units='(s^{-1})'
456     else:
457         print 'relaxation_type must be "T_1","T_2", "R_1","R_2" or "NOE"
458         ""
459     self.file_string='../relaxation_tmp/%stext_out%d.csv' % (self.
460         relaxation_type,os.getpid())
461     f=open(self.file_string,'w')
462     self.first_line='Field strength (Tesla),%s %s\n' % (self.
463         relaxation_type,self.plot_units)
464     f.write(self.first_line)
465     for coordinate in self.print_list:
466         f.write(str(coordinate).replace('[','').replace(']',')+'\n')
467     f.close()
468     print '''<div %s><h4>Additional data</h4><a href="%s">%s: CSV plot data
469         file</a></div>''' % (self.formatting,self.file_string.replace('../',
470         '../../' ), self.file_label)
471
472 class T1_single_plotter(single_plotter):
473     def plot(self):

```

```
469         self.gp.plot(self.T1_list)
470         time.sleep(1)
471         os.system(self.convert_input)
472
473     def csv_printer(self):
474         self.print_list=self.T1_list
475         self.file_label='<em>T<sub>1</sub></em>'
476         self.formatting='class="extra_data"'
477         single_plotter.csv_printer(self)
478
479     class T2_single_plotter(single_plotter):
480     def plot(self):
481         self.gp.plot(self.T2_list)
482         time.sleep(1)
483         os.system(self.convert_input)
484         self.csv_printer(self):
485         self.print_list=self.T2_list
486         self.file_label='<em>T<sub>2</sub></em>'
487         self.formatting='class="extra_data" style="top:526px"'
488         single_plotter.csv_printer(self)
489
490     class NOE_single_plotter(single_plotter):
491     def plot(self):
492         self.gp.plot(self.NOE_list)
493         time.sleep(1)
494         os.system(self.convert_input)
495         self.csv_printer(self):
496         self.print_list=self.NOE_list
497         self.file_label='<em>NOE</em>'
498         self.formatting='class="extra_data" style="top:1050px"'
499         single_plotter.csv_printer(self)
```

---

```
499 class R1_single_plotter(single_plotter):
500     def plot(self):
501         self.gp.plot(self.R1_list)
502         time.sleep(1)
503         os.system(self.convert_input)
504     def csv_printer(self):
505         self.print_list=self.R1_list
506         self.file_label='<em>R<sub>1</sub></em>'
507         self.formatting='class="extra_data" style="top:1581px"'
508         single_plotter.csv_printer(self)
509
510
511 class R2_single_plotter(single_plotter):
512     def plot(self):
513         self.gp.plot(self.R2_list)
514         time.sleep(1)
515         os.system(self.convert_input)
516     def csv_printer(self):
517         self.print_list=self.R2_list
518         self.file_label='<em>R<sub>2</sub></em>'
519         self.formatting='class="extra_data" style="top:2102px"'
520         single_plotter.csv_printer(self)
521
522
523 class frame2_generator:
524     def __init__(self, cgi_p_varied):
525         #self.a=cgi_user_param
526         self.b=cgi_p_varied
527
```

```

self.relax_book={'T1': '<em>T<sub>1</sub></em>', 'T2': '<em>T<sub>2</sub></em></sub></em>', 'NOE': '<em>NOE</em>'} #dictionary corresponding to first
radio selection of relax type

#self.r=self.relax_book[self.a]
if self.b not in ['tau_m', 'tau_e', 'S_squared']:
    self.constant_dictionary={'tau_m': '10', 'tau_fast': '10',
        'tau_slow': '3', 'S_squared_fast': '0.85', 'S_squared_slow': '0.77',
        'tau_m_extended': '10'}
    self.descriptor_book={'tau_fast': 'fast correlation time',
        'tau_slow': 'slow correlation time', 'S_squared_fast': 'fast
        partial order parameter', 'S_squared_slow': 'slow partial order
        parameter', 'tau_m_extended': 'overall rotational correlation
        time'}
    self.low_name_book={'tau_fast': 'tau_fast_low', 'tau_slow': '
        tau_slow_low', 'S_squared_fast': 'SS_fast_low', 'S_squared_slow'
        : 'SS_slow_low', 'tau_m_extended': 'tau_m_extended_low'}
    self.parameter_book={'tau_fast': '&tau<sub>fast</sub>', 'tau_slow
        ': '&tau<sub>slow</sub>', 'S_squared_fast': 'S<sup>2</sup><sub></sub></sup><sub></sub></sup>', 'S_squared_slow': 'S<sup>2</sup><sub></sub></sup><sub></sub></sup>',
        'tau_m_extended': '&tau<sub>m</sub>'}
    self.p=self.parameter_book[self.b]
else:
    self.constant_dictionary={'tau_m': '10', 'tau_e': '50', 'S_squared'
        : '0.85'}
    self.descriptor_book={'tau_m': 'overall rotational correlation
        times', 'tau_e': 'effective internal correlation time',
        'S_squared': 'generalized order parameter'}

```

529

531

533

535

537

539



```

541 self.low_name_book={'tau_m':'taum_low','tau_e':'taue_low',
542                   'S_squared':'SS_low'}
543 self.parameter_book={'tau_m':'&tau<sub>m</sub>','tau_e':'&tau<
544 sub>e</sub>','S_squared':'S<sup>2</sup>'} #dictionary
545     corresponding to second radio selection of params
546     self.p=self.parameter_book[self.b]
547
548 #html frame 2 generation for parameter input:
549 self.html_part1=''<html><head><link rel="stylesheet" type="text/css"
550 href="main.css" /></head>
551 <body class="fixed_width"><strong> Step 2 -
552 Set the parameter values</strong> (Effect of varying %s on relaxation trends with field
553 strength)<br />
554 <form method="POST" action="plot_%s.py" target="plot_frame">    ''' % (self.p,self.b)
555
556     def html_part2(self):
557         #self.low_value='' #1 or 10 or 100; the default low value in the input
558         box
559         #self.radio_low=''
560         #sample of radio_low:
561         #ps<input type="radio" name="tau_m_input_units1" value="ps"/>
562         #ns<input type="radio" name="tau_m_input_units1" value="ns" checked="
563         yes"/>
564         #s<input type="radio" name="tau_m_input_units1" value="s"/>
565         #self.high_value=''
566         #self.radio_high=''
567         #sample of radio_high:
568         #ps<input type="radio" name="tau_m_input_units2" value="ps"/>

```



```

583 if self.b not in ['tau_m', 'tau_e', 'S_squared']:
584     return '<table class="right_table"><tr><th>Values for
585         constant parameters</th></tr>
586         <tr><td>%s, %s -->
587         ''' % (self.descriptor_book[self.parameter_book.items()[0][0]],
588             self.parameter_book.items()[0][1]) + '''<input type="text"
589             name="%s_input" value="%s"> ''' % (self.parameter_book.items()
590             [0][0], self.constant_dictionary[self.parameter_book.items()
591             [0][0]]) + self.constant_radio_1 + '''</td></tr>
592         <tr><td>%s, %s --> ''' % (self.descriptor_book[self.
593             parameter_book.items()[1][0]], self.parameter_book.items()
594             [1][1]) + '''<input type="text"
595             name="%s_input" value="%s"> ''' % (self.parameter_book.items()
596             [1][0], self.constant_dictionary[self.parameter_book.items()
597             [1][0]]) + self.constant_radio_2 + '''</td>
598         </tr>''' + '''<tr><td>%s, %s --> ''' % (self.descriptor_book[
599             self.parameter_book.items()[2][0]], self.parameter_book.items
600             () [2][1]) + '''<input type="text"
601             name="%s_input" value="%s"> ''' % (self.parameter_book.items()
602             [2][0], self.constant_dictionary[self.parameter_book.items()
603             [2][0]]) + self.constant_radio_3 + '''</td></tr>
604         <tr><td>''' + '''%s, %s --> ''' % (self.descriptor_book[self.
605             parameter_book.items()[3][0]], self.parameter_book.items()
606             [3][1]) + '''<input type="text"
607             name="%s_input" value="%s"> ''' % (self.parameter_book.items()
608             [3][0], self.constant_dictionary[self.parameter_book.items()
609             [3][0]]) + self.constant_radio_4 + '''</td></tr>
610         </table><br>
611         <br>

```

597

599

```
<input class="input_right" style="top:220px" type="submit"
    value="plot">
</form>
</body>
</html>
'''
```

601

603

else:

605

```
return '''<table class="right_table"><tr><th>Values for
constant parameters</th></tr>
<tr><td>%s, %s -->
''' % (self.descriptor_book[self.parameter_book.items()[0][0]],
self.parameter_book.items()[0][1]) + '''<input type="text"
name="%s_input" value="%s"> ''' % (self.parameter_book.items()
[0][0], self.constant_dictionary[self.parameter_book.items()
[0][0]]) + self.constant_radio_1 + '''</td></tr>
<tr><td>%s, %s --> ''' % (self.descriptor_book[self.
parameter_book.items()[1][0]], self.parameter_book.items()
[1][1]) + '''<input type="text"
name="%s_input" value="%s"> ''' % (self.parameter_book.items()
[1][0], self.constant_dictionary[self.parameter_book.items()
[1][0]]) + self.constant_radio_2 + '''</td>
</tr></table><br>
<br>
```

607

288

609

611

613

615

617

```

619 <input class="input_right" style="top:165px" type="submit"
        value="plot">
620 </form>
621 </body>
622 </html>
623 '''
624
625 def print_html(self):
626     print self.html_part1+self.html_part2()+self.html_part3()
627
628 class SS_varied_input(frame2_generator):
629     def __init__(self, cgi_p_varied):
630         frame2_generator.__init__(self, cgi_p_varied)
631         #low and high input and defaults:
632         self.low_value='0.0'
633         self.radio_low='' # no unit selection for S^2
634         self.high_value='1.0'
635         self.radio_high='' # no unit selection for S^2
636         #constants input and defaults:
637         self.constant_radio_1=''ps<input type="radio" name="tau_e_input_units"
            value="ps" checked="yes"/>
638         ns<input type="radio" name="tau_e_input_units" value="ns"/>
639         s<input type="radio" name="tau_e_input_units" value="s"/>'''
640     self.constant_radio_2=''ps<input type="radio" name="tau_m_input_units"
            value="ps"/>
641     ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes"
            "/>
642     s<input type="radio" name="tau_m_input_units" value="s"/>'''

```

```

645 class tau_e_varied_input(frame2_generator):
646     def __init__(self, cgi_p_varied):
647         frame2_generator.__init__(self, cgi_p_varied)
648         #low and high input and defaults:
649         self.low_value='10.0'
650         self.radio_low=''ps<input type="radio" name="tau_e_input_units1" value
651             ="ps" checked="yes"/>
652         ns<input type="radio" name="tau_e_input_units1" value="ns"/>
653         s<input type="radio" name="tau_e_input_units1" value="s"/>'''
654         self.high_value='50.0'
655         self.radio_high=''ps<input type="radio" name="tau_e_input_units2"
656             value="ps"/>
657         ns<input type="radio" name="tau_e_input_units2" value="ns" checked="yes
658             "/>
659         s<input type="radio" name="tau_e_input_units2" value="s"/>'''
660         #constants input and defaults:
661         self.constant_radio_1=''ps<input type="radio" name="tau_m_input_units"
662             value="ps"/>
663         ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes
664             "/>
665         s<input type="radio" name="tau_m_input_units" value="s"/>'''
666         self.constant_radio_2='' # no unit selection for S^2
667
668 class tau_m_varied_input(frame2_generator):
669     def __init__(self, cgi_p_varied):
670         frame2_generator.__init__(self, cgi_p_varied)
671         #low and high input and defaults:
672         self.low_value='1'
673         self.radio_low=''ps<input type="radio" name="tau_m_input_units1" value
674             ="ps" checked="yes"/>

```

```

669 ns<input type="radio" name="tau_m_input_units1" value="ns"/>
670 s<input type="radio" name="tau_m_input_units1" value="s"/>'''
671 self.high_value='10.0'
672 self.radio_high=''ps<input type="radio" name="tau_m_input_units2"
        value="ps"/>
673 ns<input type="radio" name="tau_m_input_units2" value="ns" checked="yes"
        "/>
674 s<input type="radio" name="tau_m_input_units2" value="s"/>'''
675 #constants input and defaults:
676 self.constant_radio_1=''ps<input type="radio" name="tau_e_input_units"
        value="ps" checked="yes"/>
677 ns<input type="radio" name="tau_e_input_units" value="ns"/>
678 s<input type="radio" name="tau_e_input_units" value="s"/>'''
679 self.constant_radio_2='' # no unit selection for S^2
680
681 class tau_fast_varied_input(frame2_generator):
682     def __init__(self, cgi_p_varied):
683         frame2_generator.__init__(self, cgi_p_varied)
684         #low and high input and defaults:
685         self.low_value='0.25'
686         self.radio_low=''ps<input type="radio" name="tau_fast_input_units1"
            value="ps" checked="yes"/>
687 ns<input type="radio" name="tau_fast_input_units1" value="ns"/>
688 s<input type="radio" name="tau_fast_input_units1" value="s"/>'''
689 self.high_value='5'
690 self.radio_high=''ps<input type="radio" name="tau_fast_input_units2"
            value="ps" checked="yes"/>
691 ns<input type="radio" name="tau_fast_input_units2" value="ns"/>
692 s<input type="radio" name="tau_fast_input_units2" value="s"/>'''
693 #constants input and defaults:

```

```

693 self.constant_radio_1=''ps<input type="radio" name="tau_m_input_units"
        value="ps"/>
        ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes
        "/>
695 s<input type="radio" name="tau_m_input_units" value="s"/>'''
        self.constant_radio_2=''' # no unit selection for S slow
697 self.constant_radio_3=''' # no unit selection for S fast
        self.constant_radio_4=''ps<input type="radio" name="
            tau_slow_input_units" value="ps"/>
699 ns<input type="radio" name="tau_slow_input_units" value="ns" checked="
            yes"/>
            s<input type="radio" name="tau_slow_input_units" value="s"/>'''

701
703 class tau_slow_varied_input(frame2_generator):
705     def __init__(self, cgi_p_varied):
707         frame2_generator.__init__(self, cgi_p_varied)
            #low and high input and defaults:
            self.low_value='3'
            self.radio_low=''ps<input type="radio" name="tau_slow_input_units1"
                value="ps"/>
            ns<input type="radio" name="tau_slow_input_units1" value="ns" checked="
                yes"/>
            s<input type="radio" name="tau_slow_input_units1" value="s"/>'''
            self.high_value='5'
            self.radio_high=''ps<input type="radio" name="tau_slow_input_units2"
                value="ps"/>
            ns<input type="radio" name="tau_slow_input_units2" value="ns" checked="
                yes"/>
            s<input type="radio" name="tau_slow_input_units2" value="s"/>'''
713 #constants input and defaults:

```



```

715 self.constant_radio_1=''ps<input type="radio" name="tau_m_input_units"
      value="ps"/>
716 ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes
      "/>
717 s<input type="radio" name="tau_m_input_units" value="s"/>'''
718 self.constant_radio_2=''' # no unit selection for S slow
719 self.constant_radio_3=''' # no unit selection for S fast
720 self.constant_radio_4=''ps<input type="radio" name="
      tau_fast_input_units" value="ps" checked="yes"/>
721 ns<input type="radio" name="tau_fast_input_units" value="ns"/>
722 s<input type="radio" name="tau_fast_input_units" value="s"/>'''
723
724 class S_squared_fast_varied_input(frame2_generator):
725     def __init__(self, cgi_p_varied):
726         frame2_generator.__init__(self, cgi_p_varied)
727         #low and high input and defaults:
728         self.low_value='0.0'
729         self.radio_low='' # S fast has no units
730         self.high_value='1.0'
731         self.radio_high='' # S fast has no units
732         #constants input and defaults:
733         self.constant_radio_1=''ps<input type="radio" name="tau_m_input_units"
              value="ps"/>
              ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes
              "/>
              s<input type="radio" name="tau_m_input_units" value="s"/>'''
734         self.constant_radio_2=''' # no unit selection for S slow
735         self.constant_radio_3='''ps<input type="radio" name="
              tau_slow_input_units" value="ps"/>

```

```

739 ns<input type="radio" name="tau_slow_input_units" value="ns" checked="
      yes"/>
741 s<input type="radio" name="tau_slow_input_units" value="s"/>'''
      self.constant_radio_4='''ps<input type="radio" name="
          tau_fast_input_units" value="ps" checked="yes"/>
ns<input type="radio" name="tau_fast_input_units" value="ns"/>
s<input type="radio" name="tau_fast_input_units" value="s"/>'''

class S_squared_slow_varied_input(frame2_generator):
743     def __init__(self, cgi_p_varied):
745         frame2_generator.__init__(self, cgi_p_varied)
747         #low and high input and defaults:
            self.low_value='0.0'
            self.radio_low='' # S slow has no units
            self.high_value='1.0'
            self.radio_high='' # S slow has no units
            #constants input and defaults:
            self.constant_radio_1='''ps<input type="radio" name="tau_m_input_units"
                value="ps"/>
ns<input type="radio" name="tau_m_input_units" value="ns" checked="yes
    "/>
755 s<input type="radio" name="tau_m_input_units" value="s"/>'''
            self.constant_radio_2='' # no unit selection for S fast
            self.constant_radio_3='''ps<input type="radio" name="
                tau_slow_input_units" value="ps"/>
ns<input type="radio" name="tau_slow_input_units" value="ns" checked="
    yes"/>
757 s<input type="radio" name="tau_slow_input_units" value="s"/>'''
            self.constant_radio_4='''ps<input type="radio" name="
                tau_fast_input_units" value="ps" checked="yes"/>

```

```

761 ns<input type="radio" name="tau_fast_input_units" value="ns"/>
762 s<input type="radio" name="tau_fast_input_units" value="s"/>'''
763
764
765 class tau_m_extended_varied_input(frame2_generator):
766     def __init__(self, cgi_p_varied):
767         frame2_generator.__init__(self, cgi_p_varied)
768         #low and high input and defaults:
769         self.low_value='1.0'
770         self.radio_low=''ps<input type="radio" name="
771         tau_m_extended_input_units1" value="ps"/>
772         ns<input type="radio" name="tau_m_extended_input_units1" value="ns"
773         checked="yes"/>
774         s<input type="radio" name="tau_m_extended_input_units1" value="s"/>'''
775         self.high_value='40.0'
776         self.radio_high=''ps<input type="radio" name="
777         tau_m_extended_input_units2" value="ps"/>
778         ns<input type="radio" name="tau_m_extended_input_units2" value="ns"
779         checked="yes"/>
780         s<input type="radio" name="tau_m_extended_input_units2" value="s"/>'''
781         #constants input and defaults:
782         self.constant_radio_1='' # no unit selection for S slow
783         self.constant_radio_2='' # no unit selection for S fast
784         self.constant_radio_3=''ps<input type="radio" name="
785         tau_slow_input_units" value="ps"/>
786         ns<input type="radio" name="tau_slow_input_units" value="ns" checked="
787         yes"/>
788         s<input type="radio" name="tau_slow_input_units" value="s"/>'''
789         self.constant_radio_4=''ps<input type="radio" name="
790         tau_fast_input_units" value="ps" checked="yes"/>

```

```
ns<input type="radio" name="tau_fast_input_units" value="ns"/>
s<input type="radio" name="tau_fast_input_units" value="s"/>
'''
```

```
1  #!/home/structbi/bin/python2.5
2  import sys
3  import cgi
4  import cgitb; cgitb.enable()
5  import math
6  import Gnuplot, Gnuplot.functutils
7  from constants_equations import *
8  import class_storage
9  import os
10
11 #os.putenv('DISPLAY', ':10.0')
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<head><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></head>'
17
18 os.environ['PATH']=os.pathsep.join\
19     ([os.environ['PATH'], '/usr/local/bin'])
20
21 try:
22     form=cgi.FieldStorage() #retrieves a dictionary with form info
23
24     b=form['p_varied'].value
25
26     if b=='S_squared':
27         instance=class_storage.SS_varied_input(b)
28         instance.print_html()
```

```

29 elif b=='tau_e':
30     instance=class_storage.tau_e_varied_input(b)
31     instance.print_html()
32     #print '<meta http-equiv="refresh" content="0; url=%s_varied_%s.html
33         "/>' % (b,a)
34
35 elif b=='tau_m':
36     instance=class_storage.tau_m_varied_input(b)
37     instance.print_html()
38
39 elif b=='tau_fast':
40     instance=class_storage.tau_fast_varied_input(b)
41     instance.print_html()
42
43 elif b=='tau_slow':
44     instance=class_storage.tau_slow_varied_input(b)
45     instance.print_html()
46
47 elif b=='S_squared_fast':
48     instance=class_storage.S_squared_fast_varied_input(b)
49     instance.print_html()
50
51 elif b=='S_squared_slow':
52     instance=class_storage.S_squared_slow_varied_input(b)
53     instance.print_html()
54
55 elif b=='tau_m_extended':
56     instance=class_storage.tau_m_extended_varied_input(b)
57     instance.print_html()
58
59 except:
60     print '<p id="center" target="param_selection_frame"> Please fill in all fields
61         for step 1 </p>'
62     sys.exit()

```

Listing C.5: Module (plot\_S\_squared\_fast.py) to plot input data by calling appropriate class instance

```
2 #!/home/structbi/bin/python2.5
3
4 import sys
5 import cgi
6 import gitb; gitb.enable()
7 import math
8 import Gnuplot, Gnuplot.funcutils
9 from constants_equations import *
10 import class_storage
11
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     tau_fast=float(form['tau_fast_input'].value)
24     if form['tau_fast_input_units'].value=='ns':
25         tau_fast=tau_fast*(10**-9)
26     elif form['tau_fast_input_units'].value=='ps':
27         tau_fast=tau_fast*(10**-12)
```

---

```

28         else:
29             pass
30         c=tau_fast
31
32         tau_slow=float(form['tau_slow_input'].value)
33         if form['tau_slow_input_units'].value=='ns':
34             tau_slow=tau_slow*(10**-9)
35         elif form['tau_slow_input_units'].value=='ps':
36             tau_slow=tau_slow*(10**-12)
37         else:
38             pass
39         a=tau_slow
40         b=float(form['S_squared_slow_input'].value)
41
42         tau_m=float(form['tau_m_extended_input'].value)
43         if form['tau_m_input_units'].value=='ns':
44             tau_m=tau_m*(10**-9)
45         elif form['tau_m_input_units'].value=='ps':
46             tau_m=tau_m*(10**-12)
47         else:
48             pass
49         w=tau_m
50         #for key in form.keys():
51             #    print form[key].value
52         #print form.keys()
53     instance_1=class_storage.S_squared_fast_varied_plotter(tau_m=w,parameter_min=
54     float(form['SS_fast_low'].value),parameter_max=float(form['SS_fast_high']).
55     value),npar=int(form['npar'].value),field_values=26,model="extended",tau_slow
56     =a,S_squared_slow=b,tau_fast=c)

```



```
54 instance_1.plot_1(tau_m=w, parameter_min=float(form['SS_fast_low'].value),
55                  parameter_max=float(form['SS_fast_high'].value), npar=int(form['npar'].value),
56                  field_values=26, model="extended", tau_slow=a, S_squared_slow=b, tau_fast=c)
57
58 except:
59     print '<center>Please ensure all input values are number characters</center>'
60     sys.exit()
```

Listing C.6: Module (plot\_S\_squared.py) to plot input data by calling appropriate class instance

```
2 #!/home/structbi/bin/python2.5
3
4 import sys
5 import cgi
6 import gitb; gitb.enable()
7 import math
8 import Gnuplot, Gnuplot.functils
9 from constants_equations import *
10 import class_storage
11 import os
12 #os.putenv('DISPLAY', ':10.0')
13
14 print 'Content-Type: text/html\n\n'
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16 main.css" /></header>'
17
18 os.environ['PATH']=os.pathsep.join\
19 ([os.environ['PATH'], '/usr/local/bin'])
20
21 try:
22     form=cgi.FieldStorage() #retrieves a dictionary with form info
23
24     tau_m=float(form['tau_m_input'].value)
25     if form['tau_m_input_units'].value=='ns':
26         tau_m=tau_m*(10**-9)
27     elif form['tau_m_input_units'].value=='ps':
28         tau_m=tau_m*(10**-12)
```

```

28         else:
29             pass
30         a=tau_m
31
32         tau_e=float(form['tau_e_input'].value)
33         if form['tau_e_input_units'].value=='ns':
34             tau_e=tau_e*(10**-9)
35         elif form['tau_e_input_units'].value=='ps':
36             tau_e=tau_e*(10**-12)
37         else:
38             pass
39         b=tau_e
40
41     instance_1=class_storage.S_squared_varied_plotter(parameter_min=float(form['
42         SS_low'].value), parameter_max=float(form['SS_high'].value), npar=int(form['
43         npar'].value), field_values=26, tau_m=a, tau_e=b)
44     instance_1.plot_1(parameter_min=float(form['SS_low'].value), parameter_max=float
45         (form['SS_high'].value), npar=int(form['npar'].value), field_values=26, tau_m=a,
46         tau_e=b)
47
48     except:
49         print '<center>Please ensure all input values are number characters</center>'
50         sys.exit()

```

Listing C.7: Module (plot\_S\_squared\_slow.py) to plot input data by calling appropriate class instance

```
2 #!/home/structbi/bin/python2.5
3
4 import sys
5 import cgi
6 import gitb; gitb.enable()
7 import math
8 import Gnuplot, Gnuplot.funcutils
9 from constants_equations import *
10 import class_storage
11
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16 main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18 ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     tau_fast=float(form['tau_fast_input'].value)
24     if form['tau_fast_input_units'].value=='ns':
25         tau_fast=tau_fast*(10**-9)
26     elif form['tau_fast_input_units'].value=='ps':
27         tau_fast=tau_fast*(10**-12)
```

```

28         else:
29             pass
30         c=tau_fast
31
32         tau_slow=float(form['tau_slow_input'].value)
33         if form['tau_slow_input_units'].value=='ns':
34             tau_slow=tau_slow*(10**-9)
35         elif form['tau_slow_input_units'].value=='ps':
36             tau_slow=tau_slow*(10**-12)
37         else:
38             pass
39         a=tau_slow
40         b=float(form['S_squared_fast_input'].value)
41
42         tau_m=float(form['tau_m_extended_input'].value)
43         if form['tau_m_input_units'].value=='ns':
44             tau_m=tau_m*(10**-9)
45         elif form['tau_m_input_units'].value=='ps':
46             tau_m=tau_m*(10**-12)
47         else:
48             pass
49         w=tau_m
50         #for key in form.keys():
51             #    print form[key].value
52         #print form.keys()
53         instance_1=class_storage.S_squared_slow_varied_plotter(tau_m=w,parameter_min=
54             float(form['SS_slow_low'].value),parameter_max=float(form['SS_slow_high']).
55             value),npar=int(form['npar'].value),field_values=26,model="extended",tau_slow
56             =a,S_squared_fast=b,tau_fast=c)

```

```
54 instance_1.plot_1(tau_m=w, parameter_min=float('SS_slow_low').value),
    parameter_max=float('SS_slow_high').value), npar=int(form['npar'].value),
    field_values=26, model="extended", tau_slow=a, S_squared_fast=b, tau_fast=c)

56 except:
    print '<center>Please ensure all input values are number characters</center>'
58     sys.exit()
```

Listing C.8: Module (plot\_tau\_e.py) to plot input data by calling appropriate class instance

```
2 #!/home/structbi/bin/python2.5
3
4 import sys
5 import cgi
6 import gitb; gitb.enable()
7 import math
8 import Gnuplot, Gnuplot.functils
9 from constants_equations import *
10 import class_storage
11 import os
12
13 #os.system("alias gnuplot='/home/structbi/Tyler_programs/gnuplot-4.2.5/src/gnuplot'")
14
15 print 'Content-Type: text/html\n\n'
16
17 #os.putenv('DISPLAY', ':10.0')
18
19 #print os.popen('which gnuplot').readlines()
20
21 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
22 main.css" /></header>'
23
24 os.environ['PATH']=os.pathsep.join\
25 ([os.environ['PATH'], '/home/structbi/Tyler_programs/gnuplot-4.2.5/src/gnuplot'
26 ])
```

```
26 try:
28
29     form=cgi.FieldStorage() #retrieves a dictionary with form info
30
31     if form['tau_e_input_units1'].value=='ns':
32         k=(10**-9)
33     elif form['tau_e_input_units1'].value=='ps':
34         k=(10**-12)
35     else:
36         k=1
37
38     if form['tau_e_input_units2'].value=='ns':
39         j=(10**-9)
40     elif form['tau_e_input_units2'].value=='ps':
41         j=(10**-12)
42     else:
43         j=1
44
45     tau_m=float(form['tau_m_input'].value)
46     if form['tau_m_input_units'].value=='ns':
47         tau_m=tau_m*(10**-9)
48     elif form['tau_m_input_units'].value=='ps':
49         tau_m=tau_m*(10**-12)
50     else:
51         pass
52     a=tau_m
53     b=float(form['S_squared_input'].value)
```



```
56 instance_1=class_storage.tau_e_varied_plotter(parameter_min=float(form['
    taue_low'].value)*k,parameter_max=float(form['taue_high'].value)*j,npar=int(
    form['npar'].value),field_values=26,tau_m=a,S_squared=b)
58 instance_1.plot_1(parameter_min=float(form['taue_low'].value)*k,parameter_max=
    float(form['taue_high'].value)*j,npar=int(form['npar'].value),field_values
    =26,tau_m=a,S_squared=b)
60
except:
    print '<center>Please ensure all input values are number characters</center>'
    sys.exit()
```

Listing C.9: Module (plot\_tau\_fast.py) to plot input data by calling appropriate class instance

```
1 #!/home/structbi/bin/python2.5
2
3 import sys
4 import cgi
5 import gitb; gitb.enable()
6 import math
7 import Gnuplot, Gnuplot.functils
8 from constants_equations import *
9 import class_storage
10 import os
11
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     if form['tau_fast_input_units1'].value=='ns':
24         k=(10**-9)
25     elif form['tau_fast_input_units1'].value=='ps':
26         k=(10**-12)
27     else:
```

```

27         k=1
29         if form['tau_fast_input_units2'].value=='ns':
31             j=(10**-9)
33             elif form['tau_fast_input_units2'].value=='ps':
35                 j=(10**-12)
37             else:
39                 j=1
41
43         tau_slow=float(form['tau_slow_input'].value)
45         if form['tau_slow_input_units'].value=='ns':
47             tau_slow=tau_slow*(10**-9)
49             elif form['tau_slow_input_units'].value=='ps':
51                 tau_slow=tau_slow*(10**-12)
53             else:
55                 pass
57         a=tau_slow
59         b=float(form['S_squared_slow_input'].value)
61         c=float(form['S_squared_fast_input'].value)
63
65         tau_m=float(form['tau_m_extended_input'].value)
67         if form['tau_m_input_units'].value=='ns':
69             tau_m=tau_m*(10**-9)
71             elif form['tau_m_input_units'].value=='ps':
73                 tau_m=tau_m*(10**-12)
75             else:
77                 pass
79         w=tau_m
81         #for key in form.keys():
83             #
85             print form[key].value

```

57

```
instance_1=class_storage.tau_fast_varied_plotter(tau_m=w,parameter_min=float(
form['tau_fast_low'].value)*k,parameter_max=float(form['tau_fast_high'].value
)*j,npar=int(form['npar'].value),field_values=26,model="extended",tau_slow=a,
S_squared_slow=b,S_squared_fast=c)
instance_1.plot_1(tau_m=w,parameter_min=float(form['tau_fast_low'].value)*k,
parameter_max=float(form['tau_fast_high'].value)*j,npar=int(form['npar'].
value),field_values=26,model="extended",tau_slow=a,S_squared_slow=b,
S_squared_fast=c)
```

59

61

```
except:
```

```
print '<center>Please ensure all input values are number characters</center>'
sys.exit()
```

63

Listing C.10: Module (plot\_tau\_m\_extended.py) to plot input data by calling appropriate class instance

```
1 #!/home/structbi/bin/python2.5
2
3 import sys
4 import cgi
5 import gitb; gitb.enable()
6 import math
7 import Gnuplot, Gnuplot.functils
8 from constants_equations import *
9 import class_storage
10 import os
11
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     if form['tau_m_extended_input_units1'].value=='ns':
24         k=(10**-9)
25     elif form['tau_m_extended_input_units1'].value=='ps':
26         k=(10**-12)
27     else:
```

```
27         k=1
29         if form['tau_m_extended_input_units2'].value=='ns':
31             j=(10**-9)
33             elif form['tau_m_extended_input_units2'].value=='ps':
35                 j=(10**-12)
37             else:
39                 j=1
41
43         tau_slow=float(form['tau_slow_input'].value)
45         if form['tau_slow_input_units'].value=='ns':
47             tau_slow=tau_slow*(10**-9)
49         elif form['tau_slow_input_units'].value=='ps':
51             tau_slow=tau_slow*(10**-12)
53         else:
55             pass
57         a=tau_slow
59         b=float(form['S_squared_slow_input'].value)
61         c=float(form['S_squared_fast_input'].value)
63
65         tau_fast=float(form['tau_fast_input'].value)
67         if form['tau_fast_input_units'].value=='ns':
69             tau_fast=tau_fast*(10**-9)
71         elif form['tau_fast_input_units'].value=='ps':
73             tau_fast=tau_fast*(10**-12)
75         else:
77             pass
79         w=tau_fast
81         #for key in form.keys():
83         #    print form[key].value
```

57

```
instance_1=class_storage.tau_m_extended_varied_plotter(tau_fast=w,parameter_min
=float(form['tau_m_extended_low'].value)*k,parameter_max=float(form['
tau_m_extended_high'].value)*j,npar=int(form['npar'].value),field_values=26,
model="extended",tau_slow=a,S_squared_slow=b,S_squared_fast=c)
instance_1.plot_1(tau_fast=w,parameter_min=float(form['tau_m_extended_low'].
value)*k,parameter_max=float(form['tau_m_extended_high'].value)*j,npar=int(
form['npar'].value),field_values=26,model="extended",tau_slow=a,
S_squared_slow=b,S_squared_fast=c)
```

59

61

```
except:
```

```
    print '<center>Please ensure all input values are number characters</center>'
    sys.exit()
```

63

Listing C.11: Module (plot\_tau\_m.py) to plot input data by calling appropriate class instance

```
1 #!/home/structbi/bin/python2.5
2
3 import sys
4 import cgi
5 import gitb; gitb.enable()
6 import math
7 import Gnuplot, Gnuplot.functils
8 from constants_equations import *
9 import class_storage
10 import os
11 #os.putenv('DISPLAY', ':10.0')
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     if form['tau_m_input_units1'].value=='ns':
24         k=(10**-9)
25     elif form['tau_m_input_units1'].value=='ps':
26         k=(10**-12)
27     else:
28         k=1
```



27

```
if form['tau_m_input_units2'].value=='ns':
    j=(10**-9)
elif form['tau_m_input_units2'].value=='ps':
    j=(10**-12)
else:
    j=1
```

33

```
tau_e=float(form['tau_e_input'].value)
if form['tau_e_input_units'].value=='ns':
    tau_e=tau_e*(10**-9)
elif form['tau_e_input_units'].value=='ps':
    tau_e=tau_e*(10**-12)
```

39

```
else:
    pass
a=tau_e
b=float(form['S_squared_input'].value)
```

41

43

45

47

```
instance_1=class_storage.tau_m_varied_plotter(parameter_min=float(form['
    taum_low'].value)*k,parameter_max=float(form['taum_high'].value)*j,npar=int(
    form['npar'].value),field_values=26,tau_e=a,S_squared=b)
instance_1.plot_1(parameter_min=float(form['taum_low'].value)*k,parameter_max=
    float(form['taum_high'].value)*j,npar=int(form['npar'].value),field_values
    =26,tau_e=a,S_squared=b)
```

49

```
except:
```

51

```
print '<center>Please ensure all input values are number characters</center>'
```

`sys.exit()`

Listing C.12: Module (plot\_tau\_slow.py) to plot input data by calling appropriate class instance

```
1 #!/home/structbi/bin/python2.5
2
3 import sys
4 import cgi
5 import gitb; gitb.enable()
6 import math
7 import Gnuplot, Gnuplot.funcutils
8 from constants_equations import *
9 import class_storage
10 import os
11
12
13 print 'Content-Type: text/html\n\n'
14
15 print '<header><link rel="stylesheet" type="text/css" href="/jrainey/Tyler_relaxation/
16     main.css" /></header>'
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 try:
21     form=cgi.FieldStorage() #retrieves a dictionary with form info
22
23     if form['tau_slow_input_units1'].value=='ns':
24         k=(10**-9)
25     elif form['tau_slow_input_units1'].value=='ps':
26         k=(10**-12)
27     else:
```

---

```

27     k=1
29
29     if form['tau_slow_input_units2'].value=='ns':
30         j=(10**-9)
31     elif form['tau_slow_input_units2'].value=='ps':
32         j=(10**-12)
33     else:
34         j=1
35
36
37     tau_fast=float(form['tau_fast_input'].value)
38     if form['tau_fast_input_units'].value=='ns':
39         tau_fast=tau_fast*(10**-9)
40     elif form['tau_fast_input_units'].value=='ps':
41         tau_fast=tau_fast*(10**-12)
42     else:
43         pass
44     a=tau_fast
45     b=float(form['S_squared_slow_input'].value)
46     c=float(form['S_squared_fast_input'].value)
47     #print a
48     #print form.keys()
49     #print form['tau_fast_input_units'].value
50
51     tau_m=float(form['tau_m_extended_input'].value)
52     if form['tau_m_input_units'].value=='ns':
53         tau_m=tau_m*(10**-9)
54     elif form['tau_m_input_units'].value=='ps':
55         tau_m=tau_m*(10**-12)
56     else:
57         pass

```

---

```

57 w=tau_m
instance_1=class_storage.tau_slow_varied_plotter(tau_m=w, parameter_min=float(
form['tau_slow_low'].value)*k, parameter_max=float(form['tau_slow_high'].value
)*j, npar=int(form['npar'].value), field_values=26, model="extended", tau_fast=a,
S_squared_slow=b, S_squared_fast=c)
59 instance_1.plot_1(tau_m=w, parameter_min=float(form['tau_slow_low'].value)*k,
parameter_max=float(form['tau_slow_high'].value)*j, npar=int(form['npar'].
value), field_values=26, model="extended", tau_fast=a, S_squared_slow=b,
S_squared_fast=c)

61 except:
print '<center>Please ensure all input values are number characters</center>'
63 sys.exit()

```

Listing C.13: A plot, csv data file, and html output printer module (single\_curve\_extended.py)

```
1 #!/home/structbi/bin/python2.5
2
3 import sys
4 import cgi
5 import gitb; gitb.enable()
6 import math
7 import Gnuplot, Gnuplot.functils
8 from constants_equations import *
9 import class_storage
10 import os
11
12 print 'Content-Type: text/html\n\n'
13
14
15
16
17 os.environ['PATH']=os.pathsep.join\
18     ([os.environ['PATH'], '/usr/local/bin'])
19
20 form=cgi.FieldStorage() #retrieves a dictionary with form info
21 try:
22     if form:
23
24         tau_m=float(form['tau_m_input'].value)
25         if form['tau_m_input_units'].value=='ns':
26             tau_m=tau_m*(10**-9)
27         elif form['tau_m_input_units'].value=='ps':
```

```
29         tau_m=tau_m*(10**-12)
30     else:
31         pass
32     a=tau_m
33     tau_fast=float(form['tau_fast_input'].value)
34     if form['tau_fast_input_units'].value=='ns':
35         tau_fast=tau_fast*(10**-9)
36     elif form['tau_fast_input_units'].value=='ps':
37         tau_fast=tau_fast*(10**-12)
38     else:
39         pass
40     b=tau_fast
41
42     tau_slow=float(form['tau_slow_input'].value)
43     if form['tau_slow_input_units'].value=='ns':
44         tau_slow=tau_slow*(10**-9)
45     elif form['tau_slow_input_units'].value=='ps':
46         tau_slow=tau_slow*(10**-12)
47     else:
48         pass
49     c=tau_slow
50
51     d=float(form['S_squared_fast_input'].value)
52     e=float(form['S_squared_slow_input'].value)
53
54     print '<link rel="stylesheet" type="text/css" href="/jrainey/
55         Tyler_relaxation/main.css" />'
```

```

instance_1=class_storage.T1_single_plotter('T_1', 'T_1 (s)', tau_m=a,
tau_e=1, S_squared=1, model="extended", S_squared_fast=d, S_squared_slow=
e, tau_fast=b, tau_slow=c)
instance_1.plot()
instance_1.csv_printer()
print '<center></center>' % instance_1.
output_file.replace('../Documents', '')
print '<hr />'
instance_2=class_storage.T2_single_plotter('T_2', 'T_2 (s)', tau_m=a,
tau_e=1, S_squared=1, model="extended", S_squared_fast=d, S_squared_slow=
e, tau_fast=b, tau_slow=c)
instance_2.plot()
instance_2.csv_printer()
print '<center></center>' % instance_2.
output_file.replace('../Documents', '')
print '<hr />'
instance_3=class_storage.NOE_single_plotter('NOE', 'NOE', tau_m=a, tau_e
=1, S_squared=1, model="extended", S_squared_fast=d, S_squared_slow=e,
tau_fast=b, tau_slow=c)
instance_3.plot()
instance_3.csv_printer()
print '<center></center>' % instance_3.
output_file.replace('../Documents', '')
print '<hr />'
instance_4=class_storage.R1_single_plotter('R_1', 'R_1 (s^{-1})', tau_m=a
, tau_e=1, S_squared=1, model="extended", S_squared_fast=d, S_squared_slow
=e, tau_fast=b, tau_slow=c)
instance_4.plot()
instance_4.csv_printer()

```

57

59

61

63

65

67

69

71

73



```

75 print '<center></center>' % instance_4.
    output_file.replace('../Documents', '')
76 print '<hr />'
instance_5=class_storage.R2_single_plotter('R_2', 'R_2 (s^{-1})', tau_m=a
77 , tau_e=1, S_squared=1, model="extended", S_squared_fast=d, S_squared_slow
    =e, tau_fast=b, tau_slow=c)
instance_5.plot()
78 instance_5.csv_printer()
79 print '<center></center>' % instance_5.
    output_file.replace('../Documents', '')
80 print '<hr />'
81
82 else:
83     print ''
84     <html>
85
86     <style type="text/css" media="all">
87
88
89
90
91     #paged #a {background-color:white}
92     #paged #b {background-color:white}
93     #paged #c {background-color:white}
94     #paged #d {background-color:white}
95
96     #current {text-decoration:none;
97     background-color:#2E8B57;
98     color:black;
99

```

---

```
font-weight:bold;

}

#other {text-decoration:none;
color:#8B8682
}

#paged {
list-style: none;
padding:0;
margin:0;
}

#paged li {
float: left;
border: 1px solid;
border-bottom-width: 0;
margin: 0 0.5em 0 0;
}

#content1 {
border: 10px solid;
color:#2E8B57;
clear: both;
background: white;
}
```

101

103

105

107

109

111

113

326  
115

117

119

121

123

125

127

129

```
131         padding: 1em;
132     }
133     #paged #d {
134         position: relative;
135         top: 1px;
136         background: #2E8B57;
137     }
138
139
140
141 </style>
142
143
144
145
146
147
148
149 </head>
150 <body class="fixed_width">
151     <ul id="paged" class="tabs">
152     <li id="a"><a id="other" href="../../../jrainey/Tyler_relaxation/
153         initial_radio_classical_multi.html">Classical - Multiple Curves</a></li>
154     <li id="b"><a id="other" href="../../../jrainey/Tyler_relaxation/
155         initial_radio_extended_multi.html">Extended - Multiple Curves</a></li>
156     <li id="c"><a id="other" href="../../../structbi/cgi-bin/single_curve.py">
157         Classical - Single Curve</a></li>
158     <li id="d"><a id="current" href="../../../structbi/cgi-bin/single_curve_extended.
159         py">Extended - Single Curve</a></li>
```

```

157 </ul>
158 <center><div id="content1"> <strong> Set the constant parameter values </
159 strong>
160
161 <form method="POST" action="../../../structbi/cgi-bin/
162 single_curve_extended.py" target="plot_frame">
163 <table class="left_table">
164
165 <tr><td style="border:1px solid">
166
167 Overall rotational correlation time,  $\tau$ <sub>m</sub></td> --> <
168 input type="text" name="tau_m_input" value="10">
169 ps<input type="radio" name="tau_m_input_units" value="ps"/>
170 ns<input type="radio" name="tau_m_input_units" value="ns"
171 checked="yes"/>
172 s<input type="radio" name="tau_m_input_units" value="s"/>
173 </td>
174
175 <td style="border:1px solid">
176 Fast internal correlation time,  $\tau$ <sub>fast</sub></td> --> <input
177 type="text" name="tau_fast_input" value="5">
178 ps<input type="radio" name="tau_fast_input_units" value="ps"
179 checked="yes"/>
180 ns<input type="radio" name="tau_fast_input_units" value="ns"/>
181 s<input type="radio" name="tau_fast_input_units" value="s"/>
182 </td></tr>

```

```

181 <tr><td style="border:1px solid">
Slow internal correlation time,  $\tau$ <sub>slow</sub></td> --> <input
183   type="text" name="tau_slow_input" value="5">
ps<input type="radio" name="tau_slow_input_units" value="ps"/>
ns<input type="radio" name="tau_slow_input_units" value="ns"
185   checked="yes"/>
s<input type="radio" name="tau_slow_input_units" value="s"/>
187 </td>
188 <td style="border:1px solid">
Fast partial order parameter,  $S$ <sup>2</sup></td><sub>fast</sub></td> -->
190   <input type="text" name="S_squared_fast_input" value="0.85">
</td></tr>
192 <tr><td style="border:1px solid">
Slow partial order parameter,  $S$ <sup>2</sup></td><sub>slow</sub></td> -->
194   <input type="text" name="S_squared_slow_input" value="0.77">
</td></tr>
196 </table>
<input style="position:absolute;top:150px;right:540px" type="
198   submit" value="plot">
199
200 </form>
201 </div></center>
202 </body>
203 </html>'''

```

**except :**

```
print '<center>Please ensure all input values are number characters</center>'  
sys.exit()
```

Listing C.14: A plot, csv data file, and html output printer module (single-curve.py)

```
2 #!/home/structbi/bin/python2.5
3 import sys
4 import cgi
5 import cgitb; cgitb.enable()
6 import math
7 import Gnuplot, Gnuplot.functils
8 from constants_equations import *
9 import class_storage
10 import os
11
12 print 'Content-Type: text/html\n\n'
13
14 os.environ['PATH']=os.pathsep.join\
15     ([os.environ['PATH'], '/usr/local/bin'])
16
17 form=cgi.FieldStorage() #retrieves a dictionary with form info
18 try:
19     if form:
20
21         tau_m=float(form['tau_m_input'].value)
22         if form['tau_m_input_units'].value=='ns':
23             tau_m=tau_m*(10**-9)
24         elif form['tau_m_input_units'].value=='ps':
25             tau_m=tau_m*(10**-12)
26         else:
27             pass
28         a=tau_m
```

```

30 tau_e=float(form['tau_e_input'].value)
31 if form['tau_e_input_units'].value=='ns':
32     tau_e=tau_e*(10**-9)
33 elif form['tau_e_input_units'].value=='ps':
34     tau_e=tau_e*(10**-12)
35 else:
36     pass
37 b=tau_e
38
39 c=float(form['S_squared_input'].value)
40
41 print '<link rel="stylesheet" type="text/css" href="/jrainey/
42 Tyler_relaxation/main.css" />'
43 instance_1=class_storage.T1_single_plotter('T_1', 'T_1 (s)', a,b,c)
44 instance_1.plot()
45 instance_1.csv_printer()
46 print '<center></center>' % instance_1.
47     output_file.replace('../Documents', '')
48 print '<hr />'
49 instance_2=class_storage.T2_single_plotter('T_2', 'T_2 (s)', a,b,c)
50 instance_2.plot()
51 instance_2.csv_printer()
52 print '<center></center>' % instance_2.
53     output_file.replace('../Documents', '')
54 print '<hr />'
55 instance_3=class_storage.NOE_single_plotter('NOE', 'NOE', a,b,c)
56 instance_3.plot()
57 instance_3.csv_printer()

```



```

56 print '<center></center>' % instance_3.
    output_file.replace('../Documents', '')
57 print '<hr />'
58 instance_4=class_storage.R1_single_plotter('R_1', 'R_1 (s^{-1})', a,b,c)
    instance_4.plot()
59 instance_4.csv_printer()
60 print '<center></center>' % instance_4.
    output_file.replace('../Documents', '')
61 print '<hr />'
62 instance_5=class_storage.R2_single_plotter('R_2', 'R_2 (s^{-1})', a,b,c)
    instance_5.plot()
63 instance_5.csv_printer()
64 print '<center></center>' % instance_5.
    output_file.replace('../Documents', '')
65 print '<hr />'

66
67
68
69
70 else:
71     print '''
72     <html>
73     <head>
74
75     <style type="text/css" media="all">
76
77     #pageC #a {background-color:white}
78     #pageC #b {background-color:white}
79     #pageC #c {background-color:white}
80     #pageC #d {background-color:white}

```

---

```
82 #current {text-decoration:none;
84 background-color:#2E8B57;
86 color:black;
88 font-weight:bold;
90 }
92 #other {text-decoration:none;
94 color:#8B8682
96 }
98 #pageC {
100 list-style: none;
padding:0;
margin:0;
}
102 #pageC li {
104 float: left;
border: 1px solid;
border-bottom-width: 0;
margin: 0 0.5em 0 0;
106 }
108 #content1 {
border: 10px solid;
```

---

```
112 color:#2E8B57;
113 clear: both;
114
115 background: white;
116 padding: 1em;
117
118 #pageC #c {
119     position: relative;
120     top: 1px;
121     background: #2E8B57;
122
123 }
124
125
126
127
128
129
130
131
132 </style>
133
134 </head>
135 <body class="fixed_width">
136
137 <ul id="pageC" class="tabs">
138 <li id="a"><a id="other" href="../../jrainey/Tyler_relaxation/
139     initial_radio_classical_multi.html">Classical - Multiple Curves</a></li>
140 <li id="b"><a id="other" href="../../jrainey/Tyler_relaxation/
141     initial_radio_extended_multi.html">Extended - Multiple Curves</a></li>
142 <li id="c"><a id="current" href="../../structbi/cgi-bin/single_curve.py">
143     Classical - Single Curve</a></li>
```

```

138 <li id="d"><a id="other" href="../../../structbi/cgi-bin/single_curve_extended.py
139     ">Extended - Single Curve</a></li>
140 </ul>
141 <center><div id="content1">
142     <strong> Set the constant parameter values </strong><br />
143     <form method="POST" action="../../../structbi/cgi-bin/single_curve.py"
144         target="plot_frame">
145         <table class="left_table">
146             <tr><td style="border:1px solid">
147
148                 Overall rotational correlation time,  $\tau$  </td><td style="border:1px solid">
149                     <input type="text" name="tau_m_input" value="10">
150                     ps<input type="radio" name="tau_m_input_units" value="ps"/>
151                     ns<input type="radio" name="tau_m_input_units" value="ns"
152                         checked="yes"/>
153                     s<input type="radio" name="tau_m_input_units" value="s"/>
154                 </td></tr>
155
156                 <tr><td style="border:1px solid">
157                     Effective internal correlation time,  $\tau_e$  </td><td style="border:1px solid">
158                         <input type="text" name="tau_e_input" value="50">
159                         ps<input type="radio" name="tau_e_input_units" value="ps"
160                             checked="yes"/>
161                         ns<input type="radio" name="tau_e_input_units" value="ns"/>
162                         s<input type="radio" name="tau_e_input_units" value="s"/>
163                     </td></tr>

```

```
162 <tr><td style="border:1px solid">
Order parameter,  $S^2$ </sup> --> <input type="text" name="
S_squared_input" value="0.85">
164 </td></tr>
</table>
166 <input style="position:absolute;top:170px;right:600px" type="
submit" value="plot">
168
</form>
170 </div></center>
</body>
172 </html>'''
except:
print '<center>Please ensure all input values are number characters</center>'
sys.exit()
```

# Appendix D: Source Code For FGFR3 MD Simulation Analysis

## D.1 Introduction

This appendix contains the Python (MDAnalysis-based) source code used for parsing the production-length GROMACS trajectories for the FGFR3 CG-MD replicate simulations.

## D.2 Python Source Code For Trajectory Parsing With MDAnalysis

```
Listing D.15: This module (analyze_FGFR3_dimer_simulations.py) recursively moves through replicate simulation directories and parses the dimer trajectories using functions stored in the dimer_geometric_tools.py module D.17 on page 356
```

```
'''For specific analysis of the 30 FGFR3 DIMER trajectories. Analysis is carried out by
```

```

3 parsing files organized in a directory with symlinks to the original production-length
4 data folders.'''
5
6 import MDAnalysis, dimer_geometric_tools, os
7
8 dimer_symlink_directory = '/sansom/sc2/bioc1009/Documents/FGFR3_work/
9 dimer_batch_analysis_symlink/'
10
11 #folders for primary and secondary dimer interface testing (split trajectory over
12 transition in mutant homodimer replicate 4):
13 #folder_name_list = ['primary_interface_mut4','secondary_interface_mut4']
14
15 #small test set:
16 #folder_name_list = ['mutant_dimer_replicate_4']
17
18 #GpA control folders (first 4):
19 #folder_name_list = ['GpA_dimer_replicate_1', 'GpA_dimer_replicate_2', '
20 GpA_dimer_replicate_3', 'GpA_dimer_replicate_4']
21
22 #FGFR3 WT folders:
23 #folder_name_list = ['wildtype_dimer_replicate_1','wildtype_dimer_replicate_2','
24 wildtype_dimer_replicate_3', 'wildtype_dimer_replicate_4', '
25 wildtype_dimer_replicate_5', 'wildtype_dimer_replicate_6', '
26 wildtype_dimer_replicate_7', 'wildtype_dimer_replicate_8', '
27 wildtype_dimer_replicate_9', 'wildtype_dimer_replicate_10']
28
29 #FGFR3 heterodimer and mutant folders:
30 '''folder_name_list = ['heterodimer_replicate_1','heterodimer_replicate_2','
31 heterodimer_replicate_3','heterodimer_replicate_4','heterodimer_replicate_5',
32 heterodimer_replicate_6','heterodimer_replicate_7','heterodimer_replicate_8',

```

```

heterodimer_replicate_9', 'heterodimer_replicate_10', 'mutant_dimer_replicate_1',
'mutant_dimer_replicate_2', 'mutant_dimer_replicate_3', 'mutant_dimer_replicate_4',
mutant_dimer_replicate_5', 'mutant_dimer_replicate_6', 'mutant_dimer_replicate_7',
'mutant_dimer_replicate_8', 'mutant_dimer_replicate_9', 'mutant_dimer_replicate_10']'''

#all FGFR3 folders:
folder_name_list = ['heterodimer_replicate_1', 'heterodimer_replicate_2', '
heterodimer_replicate_3', 'heterodimer_replicate_4', 'heterodimer_replicate_5', '
heterodimer_replicate_6', 'heterodimer_replicate_7', 'heterodimer_replicate_8', '
heterodimer_replicate_9', 'heterodimer_replicate_10', 'mutant_dimer_replicate_1',
'mutant_dimer_replicate_2', 'mutant_dimer_replicate_3', 'mutant_dimer_replicate_4', '
mutant_dimer_replicate_5', 'mutant_dimer_replicate_6', 'mutant_dimer_replicate_7',
'mutant_dimer_replicate_8', 'mutant_dimer_replicate_9', 'mutant_dimer_replicate_10', '
wildtype_dimer_replicate_1', 'wildtype_dimer_replicate_2', 'wildtype_dimer_replicate_3'
,
'wildtype_dimer_replicate_4', 'wildtype_dimer_replicate_5', 'wildtype_dimer_replicate_6',
'wildtype_dimer_replicate_7', 'wildtype_dimer_replicate_8', 'wildtype_dimer_replicate_9
',
'wildtype_dimer_replicate_10']

#all folders without lipid flip-flop (FGFR3 WT dimer replicates 1 and 4 removed):
'''folder_name_list = ['heterodimer_replicate_1', 'heterodimer_replicate_2', '
heterodimer_replicate_3', 'heterodimer_replicate_4', 'heterodimer_replicate_5', '
heterodimer_replicate_6', 'heterodimer_replicate_7', 'heterodimer_replicate_8', '
heterodimer_replicate_9', 'heterodimer_replicate_10', 'mutant_dimer_replicate_1',
'mutant_dimer_replicate_2', 'mutant_dimer_replicate_3', 'mutant_dimer_replicate_4', '
mutant_dimer_replicate_5', 'mutant_dimer_replicate_6', 'mutant_dimer_replicate_7',
'mutant_dimer_replicate_8', 'mutant_dimer_replicate_9', 'mutant_dimer_replicate_10', '
wildtype_dimer_replicate_2', 'wildtype_dimer_replicate_3',

```

23

25

27

340

29

31

33

35



```

41 'wildtype_dimer_replicate_5','wildtype_dimer_replicate_6','wildtype_dimer_replicate_7
42 ', 'wildtype_dimer_replicate_8','wildtype_dimer_replicate_9',
43 'wildtype_dimer_replicate_10', 'GpA_dimer_replicate_1', 'GpA_dimer_replicate_2', '
44   GpA_dimer_replicate_3', 'GpA_dimer_replicate_4', 'GpA_dimer_replicate_5']'''
45
46 def list_data_paths(dimer_symlink_directory, folder_name_list):
47     '''Returns a list of tuples with pairs of pdb (index 0) and xtc (index 1) paths
48     for a given simulation folder. The name of the folder is (index 2) in the tuple
49     .'''
50     list_of_pdb_file_paths=[]
51     list_of_xtc_file_paths=[]
52     for folder in folder_name_list: #files names can be changed for differently
53         processed_trajectories (i.e., simplified and centered or not)
54             list_of_pdb_file_paths.append(dimer_symlink_directory + folder + '/' +
55             'alpha_carbon_popc_final_snapshot.gro') #,centered_CA_POPC.pdb ; gro
56             files should be accepted now too; 'original_final_snapshot.gro'
57             list_of_xtc_file_paths.append(dimer_symlink_directory + folder + '/' +
58             'alpha_carbon_popc_and_centered_trajectory.xtc') #no_jump_trajectory.
59             xtc , 'original_trajectory.xtc', , 'interface.xtc'
60
61     combined_path_pair_list = zip(list_of_pdb_file_paths,list_of_xtc_file_paths,
62     folder_name_list)
63     return combined_path_pair_list
64
65 def create_universe_selections(dimer_symlink_directory, folder_name_list):
66     '''Creates a nested list of [universe (all atoms in a system selected)
67     MDAnalysis objects, folder_name]'''
68     universe_list=[]
69     for pdb_file, xtc_file, folder_name in list_data_paths(dimer_symlink_directory,
70     folder_name_list):

```

```

55     universe_list.append([MDAnalysis.Universe(pdb_file,xtc_file),
56         folder_name])
57     return universe_list
58
59 #lists for dumping overall FGFR3 'top five closest' contact data for overall plot:
60 #mutant_overall_list = []
61 #wildtype_overall_list = []
62 #hetero_overall_list = []
63 primary_count_list=[]
64 secondary_count_list=[]
65 other_count_list=[]
66 #initialize lists for dumping overall FGFR3 'thermal plot' data:
67 heterodimer_helix_2_x_coord_list = []
68 heterodimer_helix_2_y_coord_list = []
69 mutant_helix_2_x_coord_list = []
70 mutant_helix_2_y_coord_list = []
71 wildtype_helix_2_x_coord_list = []
72 wildtype_helix_2_y_coord_list = []
73
74 def main(primary_count_list,secondary_count_list, other_count_list,
75     dimer_symlink_directory,folder_name_list, heterodimer_helix_2_x_coord_list,
76     heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
77     mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
78     wildtype_helix_2_y_coord_list, WT_FGFR3_distal_thickness_list,
79     WT_FGFR3_local_thickness_pre_dimer_list, WT_FGFR3_local_thickness_post_dimer_list,
80     hetero_FGFR3_distal_thickness_list, hetero_FGFR3_local_thickness_pre_dimer_list,
81     hetero_FGFR3_local_thickness_post_dimer_list, mutant_FGFR3_distal_thickness_list,
82     mutant_FGFR3_local_thickness_pre_dimer_list,
83     mutant_FGFR3_local_thickness_post_dimer_list, GpA_distal_thickness_list,
84     GpA_local_thickness_pre_dimer_list, GpA_local_thickness_post_dimer_list, skip_frames,

```

```

helix_tilt_output_file, geoZ_output_file, closest_approach_output_file, GpA_top_five_output_file
, FGFR3_top_five_output_file, relative_helical_motion_output_file, fixed_thermal_output_file,
bilayer_output_file, relative_Z_output_file, local_thickness_output_file, frame_position_output_file,
efficient_contacts_output_file, bilayer_tracking_file, lipid_shell_output_file,
flip_flop_output_file):
    '''prints data to files in symlink directories by calling various functions'''
    for universe_object, folder_name in create_universe_selections(
        dimer_symlink_directory, folder_name_list):
        os.chdir(dimer_symlink_directory + folder_name) #move to the
            appropriate directory before executing data processing functions
        #call data processing functions; output should be in gnuplot format:
        #just comment out functions when you are not needing them to be re-run:

#dimer_geometric_tools.helix_tilt_vs_bilayer_normal(folder_name,
    universe_object, skip_frames, output_file=helix_tilt_output_file)
#dimer_geometric_tools.GpA_helix_tilt_vs_bilayer_normal(folder_name,
    universe_object, skip_frames, output_file=helix_tilt_output_file)
#dimer_geometric_tools.geo_Z_tracking(folder_name, universe_object,
    skip_frames, system_name='FGFR3', output_file = geoZ_output_file)
#dimer_geometric_tools.closest_approach_GpA(folder_name,
    universe_object, skip_frames, output_file = closest_approach_output_file)
#dimer_geometric_tools.top_five_closest_residues_GpA(folder_name,
    universe_object, skip_frames, output_file=GpA_top_five_output_file)
#dimer_geometric_tools.top_five_closest_residues_FGFR3(folder_name,
    universe_object, skip_frames, output_file=FGFR3_top_five_output_file)
#dimer_geometric_tools.merged_top_five_FGFR3(mutant_overall_list,
    wildtype_overall_list, hetero_overall_list, folder_name,
    universe_object, skip_frames)
#dimer_geometric_tools.relative_helical_motion(folder_name,
    universe_object, skip_frames, output_file=

```

73

75

77

79

81

83

85

```

relative_helical_motion_outfile, system_name='GpA')
#dimer_geometric_tools.relative_helical_motion(folder_name,
universe_object, skip_frames, output_file=
relative_helical_motion_outfile, system_name='FGFR3')
#dimer_geometric_tools.fixed_helix_thermal(folder_name, universe_object
, skip_frames, dimer_symlink_directory, create_universe_selections,
system_name='FGFR3', output_file=fixed_thermal_outfile)
#dimer_geometric_tools.fixed_helix_thermal_merged(folder_name,
universe_object, skip_frames, dimer_symlink_directory,
create_universe_selections, heterodimer_helix_2_x_coord_list,
heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
wildtype_helix_2_y_coord_list)
#dimer_geometric_tools.track_bilayer_thickness(folder_name,
universe_object, skip_frames, dimer_symlink_directory, outfile=
bilayer_outfile)
#dimer_geometric_tools.geo_Z_tracking_relative_to_bilayer(folder_name,
universe_object, skip_frames, system_name='FGFR3', output_file=
relative_Z_outfile)
#dimer_geometric_tools.local_bilayer_thickness(folder_name,
universe_object, skip_frames, system_name = 'FGFR3', output_file =
local_thickness_outfile)
#dimer_geometric_tools.frame_abstracted_relative_position(folder_name,
universe_object, skip_frames, dimer_symlink_directory,
create_universe_selections, system_name='GpA', output_file=
frame_position_outfile)
#dimer_geometric_tools.closest_contacts_efficient(folder_name,
universe_object, skip_frames, system_name = 'FGFR3', output_file=
efficient_contacts_outfile)
#dimer_geometric_tools.distance_versus_crossing_angle()

```

```
#dimer_geometric_tools.cartesian_to_polar_theta()
#dimer_geometric_tools.correlate_helixcrossing_polar_theta()
#dimer_geometric_tools.closest_approach_representative(folder_name,
    universe_object, skip_frames, system_name='FGFR3')
#dimer_geometric_tools.absolute_value_Z_tracking()
#dimer_geometric_tools.absolute_delta_Z_and_closest_approach()
#dimer_geometric_tools.split_Z_file(folder_name)
#dimer_geometric_tools.box_size_assessment(folder_name, universe_object
    , skip_frames)
#dimer_geometric_tools.optimize_leaflet_selection_cutoff(folder_name,
    universe_object)
#dimer_geometric_tools.analyze_leaflets(folder_name, universe_object,
    skip_frames, outfile_name = bilayer_tracking_file, local_definition =
    16, cutoff = 15.5, system_name = 'GpA')
#dimer_geometric_tools.count_lipids_in_local_shell(folder_name,
    universe_object, skip_frames, outfile_name = lipid_shell_outfile,
    local_definition = 16, cutoff = 15.5, system_name = 'GpA')
#dimer_geometric_tools.flip_flop_tracker(folder_name, universe_object,
    skip_frames, outfile_name = flip_flop_outfile, cutoff = 15.5)
#dimer_geometric_tools.bilayer_thickness_average_results(
    WT_FGFR3_distal_thickness_list,
    WT_FGFR3_local_thickness_pre_dimer_list,
    WT_FGFR3_local_thickness_post_dimer_list,
    hetero_FGFR3_distal_thickness_list,
    hetero_FGFR3_local_thickness_pre_dimer_list,
    hetero_FGFR3_local_thickness_post_dimer_list,
    mutant_FGFR3_distal_thickness_list,
    mutant_FGFR3_local_thickness_pre_dimer_list,
    mutant_FGFR3_local_thickness_post_dimer_list,
    GpA_distal_thickness_list, GpA_local_thickness_pre_dimer_list,
```

97

99

101

103

105  
345

107

```

109     GpA_local_thickness_post_dimer_list, folder_name,
        dimerization_criterion = 6.0)
        #dimer_geometric_tools.simple_moving_average_polar_theta(folder_name,
            window_size = 10)
        dimer_geometric_tools.interface_filtered_merged_top_five_FGFR3(
            primary_count_list, secondary_count_list, other_count_list,
            folder_name, universe_object, skip_frames)

111     if __name__ == "__main__":
        #initialize some global lists for the overall bilayer thickness averaging
            function:
113         WT_FGFR3_distal_thickness_list = []
115         WT_FGFR3_local_thickness_pre_dimer_list = []
            WT_FGFR3_local_thickness_post_dimer_list = []
            hetero_FGFR3_distal_thickness_list = []
            hetero_FGFR3_local_thickness_pre_dimer_list = []
            hetero_FGFR3_local_thickness_post_dimer_list = []
            mutant_FGFR3_distal_thickness_list = []
            mutant_FGFR3_local_thickness_pre_dimer_list = []
            mutant_FGFR3_local_thickness_post_dimer_list = []
            GpA_distal_thickness_list = []
            GpA_local_thickness_pre_dimer_list = []
            GpA_local_thickness_post_dimer_list = []

125     main(primary_count_list, secondary_count_list, other_count_list,
            dimer_symlink_directory, folder_name_list, heterodimer_helix_2_x_coord_list,
            heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
            mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
            wildtype_helix_2_y_coord_list, WT_FGFR3_distal_thickness_list,
            WT_FGFR3_local_thickness_pre_dimer_list,

```

```

WT_FGFR3_local_thickness_post_dimer_list, hetero_FGFR3_distal_thickness_list,
hetero_FGFR3_local_thickness_pre_dimer_list,
hetero_FGFR3_local_thickness_post_dimer_list,
mutant_FGFR3_distal_thickness_list,
mutant_FGFR3_local_thickness_pre_dimer_list,
mutant_FGFR3_local_thickness_post_dimer_list, GpA_distal_thickness_list,
GpA_local_thickness_pre_dimer_list, GpA_local_thickness_post_dimer_list,
skip_frames=10, helix_tilt_output_file='dimer_tilt_test.out', geoZ_outfile='
DualZ_tracking_nocenter.out', closest_approach_outfile = '
testing_MDA_contacts.out', GpA_top_five_outfile='top_five_testing.out',
FGFR3_top_five_outfile='interface_top_five.out',
relative_helical_motion_outfile='relative_helical_nojump.out',
fixed_thermal_outfile='fixed_thermal.out', bilayer_outfile='
bilayer_thickness_tracking.out', relative_Z_outfile = 'relative_Z.out',
local_thickness_outfile='local_bilayer_thickness.out', frame_position_outfile
='frame_correlated_position_noskip.out', efficient_contacts_outfile='
closest_contacts_full.out', bilayer_tracking_file = 'bilayer_thickness.out',
lipid_shell_outfile = 'lipid_shell_count.out', flip_flop_outfile = 'flip_flop
.out')

#print bilayer thickness results to a file:
#dimer_geometric_tools.bilayer_stats(WT_FGFR3_distal_thickness_list,
WT_FGFR3_local_thickness_pre_dimer_list,
WT_FGFR3_local_thickness_post_dimer_list, hetero_FGFR3_distal_thickness_list,
hetero_FGFR3_local_thickness_pre_dimer_list,
hetero_FGFR3_local_thickness_post_dimer_list,
mutant_FGFR3_local_thickness_pre_dimer_list,
mutant_FGFR3_distal_thickness_list,
mutant_FGFR3_local_thickness_pre_dimer_list,
mutant_FGFR3_local_thickness_post_dimer_list, GpA_distal_thickness_list,
GpA_local_thickness_pre_dimer_list, GpA_local_thickness_post_dimer_list)

```

131

```
#dimer_geometric_tools.thermal_bins(helix_2_x_coord_list=  
heterodimer_helix_2_x_coord_list, helix_2_y_coord_list=  
heterodimer_helix_2_y_coord_list, output_file='/sansom/sc2/bioc1009/Documents  
/FGFR3_work/dimer_batch_analysis_symlink/overall_thermal_plot_data/  
heterodimer_thermal_merged.out')
```

133

```
#dimer_geometric_tools.thermal_bins(helix_2_x_coord_list=  
mutant_helix_2_x_coord_list, helix_2_y_coord_list=mutant_helix_2_y_coord_list  
, output_file='/sansom/sc2/bioc1009/Documents/FGFR3_work/  
dimer_batch_analysis_symlink/overall_thermal_plot_data/  
mutant_dimer_thermal_merged.out')
```

135

```
#dimer_geometric_tools.thermal_bins(helix_2_x_coord_list=  
wildtype_helix_2_x_coord_list, helix_2_y_coord_list=  
wildtype_helix_2_y_coord_list, output_file='/sansom/sc2/bioc1009/Documents/  
FGFR3_work/dimer_batch_analysis_symlink/overall_thermal_plot_data/  
wildtype_dimer_thermal_merged.out')
```

348

137

```
#for overall parsing of FGFR3 'top five contacts' need to do the analysis  
outside of the trajectory looping:  
dimer_geometric_tools.interface_filtered_parse_overall_FGFR3_top_five_data(  
primary_count_list, secondary_count_list, other_count_list)  
#dimer_geometric_tools.parse_overall_FGFR3_top_five_data(mutant_overall_list,  
wildtype_overall_list,hetero_overall_list)
```

139



Listing D.16: This module (`analyze_sidekick_FGFR3_dimer_simulations.py`) serves a similar purpose to the (`analyze_FGFR3_dimer_simulations.py`) module D.15 on page 338, but has been redesigned to parse the enormous amount of FGFR3 *dimer* data spread over many directories produced by the high-throughput SIDEKICK simulation program (created by Benjamin Hall, University of Oxford)

```
2 '''Analyzing the SIDEKICK FGFR3 data for dimer simulations in DPPC (only lipid option
3 for SIDEKICK dimers at the moment).
4
5 The high-throughput data (far too much space to copy locally) is located at: /sbc/sn2/
6 hall/nfsmount/Data/FGFR3_dimers_Tyler/Bond
7
8 Will have to adjust my current local parsing script to deal with this different
9 directory structure, but I am using the previous stuff as a template (this file
10 started as a copy of the local analysis module for non-SIDEKICK results).
11
12 Looks like there are 90+ 500 ns simulations complete for each FGFR3 condition.'''
13
14 import MDAnalysis, dimer_geometric_tools, os
15
16 #this data directory has three subdivisions of high throughput results: wild-type FGFR3,
17 heterodimer, and mutant homodimer simulation data:
18 sidekick_data_directory = '/sbc/sn2/hall/nfsmount/Data/FGFR3_dimers_Tyler/Bond/'
19
20 #the subdirectory names reflect the sequences for the above three mentioned FGFR3
21 constructs:
22 folder_name_list = ['RRAGSVYAGILSYGVGFFLFILVVAAVTLCRLR_----- -
23 RRAGSVYAGILSYGVGFFLFILVVAAVTLCRLR_-----/DPPC/1/',
24 RRAGSVYAGILSYGVGFFLFILVVAAVTLCRLR_----- -RRAGSVYAGILSYRVGFFLFILVVAAVTLCRLR_-----/
25 DPPC/1/', 'RRAGSVYAGILSYRVGFFLFILVVAAVTLCRLR_----- -
```

```
RRAGSVYAGILSYRVGFFLFILVVAAVTLCRLR_-----/DPPC/1/']
```

#Each of the above paths contains another (large -- 90+ folders) set of data subdirectories, and the files of interest in those directories are one of center.xtc, t\_0.xtc, or xy\_fit.xtc, depending on which trajectory processing you want (see email from Ben), and em.gro (the starting structure for the sim). Try to set things up so you can read these in to MDAnalysis. These folders have long numerical names corresponding in part to their seed numbers.

```
def full_paths(sidekick_data_directory, folder_name_list):
    '''Go through the three folders corresponding to each of the FGFR3 conditions
    and produce a list for the full path of each replicate subfolder. Each of
    those subfolders contain the relevant .xtc and .gro files so we want direct
    access to their full paths for parsing and in some cases dumping the results
    as they are parsed.'''
    list_of_full_paths = []
    for folder_name in folder_name_list:
        parent_path = sidekick_data_directory + folder_name
        for replicate_folder in os.listdir(parent_path):
            list_of_full_paths.append(parent_path + replicate_folder)
    return list_of_full_paths
#list_of_full_paths now contains 286 full paths, each corresponding to one of
the replicates from any of the FGFR3 conditions (98 WT replicates, 96 hetero
replicates, and 92 double mutant replicates)
#for element in list_of_full_paths:
#    print element
#print len(list_of_full_paths)

def list_data_paths(list_of_full_paths):
    '''Returns a list of tuples with pairs of gro (index 0) and xtc (index 1) paths
```

16

18

20

350

22

24

26

28

30

32

```

34 for a given simulation folder. The full path of the folder is (index 2) in the
    tuple. '''
35 list_of_gro_file_paths=[]
36 list_of_xtc_file_paths=[]
37 for path in list_of_full_paths:
38     list_of_gro_file_paths.append(path + '/' + 'em.gro') #the starting
        structure
39     list_of_xtc_file_paths.append(path + '/' + 'center.xtc') #can use other
        options if needed: t_0.xtc is the unfiltered trajectory
40 combined_path_pair_list = zip(list_of_gro_file_paths,list_of_xtc_file_paths,
        list_of_full_paths)
41 return combined_path_pair_list

42
43 def create_universe_selections(combined_path_pair_list):
44     '''Creates a nested list of [universe (all atoms in a system selected)
        MDAnalysis objects, full_path string]'''
45     universe_list=[]
46     for gro_file, xtc_file, full_path in combined_path_pair_list:#slice this for
        testing small portions of data set (less replicates)
47         universe_list.append([MDAnalysis.Universe(gro_file,xtc_file), full_path
        ])
48     return universe_list

49
50 #lists for dumping overall FGFR3 'top five closest' contact data for overall plot:
51 #mutant_overall_list = []
52 #wildtype_overall_list = []
53 #hetero_overall_list = []
54
55 #initialize lists for dumping overall FGFR3 'thermal plot' data:
56 heterodimer_helix_2_x_coord_list = []

```

```

heterodimer_helix_2_y_coord_list = []
mutant_helix_2_x_coord_list = []
mutant_helix_2_y_coord_list = []
wildtype_helix_2_x_coord_list = []
wildtype_helix_2_y_coord_list = []

def main(universe_list, heterodimer_helix_2_x_coord_list,
heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
wildtype_helix_2_y_coord_list, skip_frames, efficient_contacts_outfile):
    '''Prints analyzed data to files in replicate subfolders or other directories
    by calling various functions'''
    for universe_object, path_name in universe_list:
        os.chdir(path_name) #move to the appropriate directory before executing
        data processing functions
        #call data processing functions; output should be in gnuplot format:
        #just comment out functions when you are not needing them to be re-run:

        #dimer_geometric_tools.helix_tilt_vs_bilayer_normal(folder_name,
        universe_object, skip_frames, output_file=helix_tilt_output_file)
        #dimer_geometric_tools.GpA_helix_tilt_vs_bilayer_normal(folder_name,
        universe_object, skip_frames, output_file=helix_tilt_output_file)
        #dimer_geometric_tools.geo_Z_tracking(folder_name, universe_object,
        skip_frames, system_name='FGFR3', output_file = geoZ_outputfile)
        #dimer_geometric_tools.closest_approach_GpA(folder_name,
        universe_object, skip_frames, output_file = closest_approach_outputfile)
        #dimer_geometric_tools.top_five_closest_residues_GpA(folder_name,
        universe_object, skip_frames, output_file=GpA_top_five_outputfile)
        #dimer_geometric_tools.top_five_closest_residues_FGFR3(folder_name,
        universe_object, skip_frames, output_file=FGFR3_top_five_outputfile)

```

```

76 #dimer_geometric_tools.merged_top_five_FGFR3(mutant_overall_list,
    wildtype_overall_list, hetero_overall_list, folder_name,
    universe_object, skip_frames)
78 #dimer_geometric_tools.relative_helical_motion(folder_name,
    universe_object, skip_frames, output_file=
    relative_helical_motion_outfile, system_name='GpA')
    #dimer_geometric_tools.relative_helical_motion(folder_name,
    universe_object, skip_frames, output_file=
    relative_helical_motion_outfile, system_name='FGFR3')
    #dimer_geometric_tools.fixed_helix_thermal(folder_name, universe_object
    , skip_frames, sidekick_data_directory, create_universe_selections,
    system_name='FGFR3', output_file=fixed_thermal_outfile)
80 #dimer_geometric_tools.fixed_helix_thermal_merged(folder_name,
    universe_object, skip_frames, sidekick_data_directory,
    create_universe_selections, heterodimer_helix_2_x_coord_list,
    heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
    mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
    wildtype_helix_2_y_coord_list)
    #dimer_geometric_tools.track_bilayer_thickness(folder_name,
    universe_object, skip_frames, sidekick_data_directory, outfile=
    bilayer_outfile)
    #dimer_geometric_tools.geo_Z_tracking_relative_to_bilayer(folder_name,
    universe_object, skip_frames, system_name='FGFR3', output_file=
    relative_Z_outfile)
    #dimer_geometric_tools.local_bilayer_thickness(folder_name,
    universe_object, skip_frames, system_name = 'FGFR3', output_file =
    local_thickness_outfile)
84 #dimer_geometric_tools.frame_abstracted_relative_position(folder_name,
    universe_object, skip_frames, sidekick_data_directory,
    create_universe_selections, system_name='GpA', output_file=

```

```

86     frame_position_outfile)
87     #dimer_geometric_tools.distance_versus_crossing_angle()
88     #dimer_geometric_tools.cartesian_to_polar_theta()
89     #dimer_geometric_tools.correlate_helixcrossing_polar_theta()
90     #dimer_geometric_tools.closest_approach_representative(folder_name,
91     universe_object, skip_frames, system_name='FGFR3')
92     #dimer_geometric_tools.absolute_value_Z_tracking()
93     #dimer_geometric_tools.absolute_delta_Z_and_closest_approach()
94     #dimer_geometric_tools.split_Z_file(folder_name)
95     #dimer_geometric_tools.box_size_assessment(folder_name, universe_object
96     , skip_frames)
97
98     #dimer_geometric_tools.closest_contacts_efficient_SIDEKICK(path_name,
99     universe_object, skip_frames, system_name = 'FGFR3', output_file=
100     efficient_contacts_outfile)
101     #dimer_geometric_tools.fixed_helix_thermal_merged_SIDEKICK(path_name,
102     universe_object, skip_frames, heterodimer_helix_2_x_coord_list,
103     heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
104     mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
105     wildtype_helix_2_y_coord_list)
106
107     if __name__ == "__main__":
108         list_of_full_paths = full_paths(sidekick_data_directory, folder_name_list)
109         combined_path_pair_list = list_data_paths(list_of_full_paths)
110         universe_list = create_universe_selections(combined_path_pair_list)
111         main(universe_list, heterodimer_helix_2_x_coord_list,
112             heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
113             mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
114             wildtype_helix_2_y_coord_list, skip_frames=10, efficient_contacts_outfile='

```

```
closest_contacts.out')
#call thermal binning function for helix 2 relative position in each of the
three FGFR3 conditions
#dimer_geometric_tools.thermal_bins_SIDEKICK(wildtype_helix_2_x_coord_list,
wildtype_helix_2_y_coord_list, outfile_path='/sansom/sc2/bioc1009/Documents/
FGFR3_work/sidekick_dimer_batch_analysis/thermal_interface_results/
WT_thermal_fixed.out')
#dimer_geometric_tools.thermal_bins_SIDEKICK(heterodimer_helix_2_x_coord_list,
heterodimer_helix_2_y_coord_list, outfile_path='/sansom/sc2/bioc1009/
Documents/FGFR3_work/sidekick_dimer_batch_analysis/thermal_interface_results/
hetero_thermal_fixed.out')
#dimer_geometric_tools.thermal_bins_SIDEKICK(mutant_helix_2_x_coord_list,
mutant_helix_2_y_coord_list, outfile_path='/sansom/sc2/bioc1009/Documents/
FGFR3_work/sidekick_dimer_batch_analysis/thermal_interface_results/
mutant_thermal_fixed.out')
```

Listing D.17: This module (`dimer_geometric_tools.py`) serves as a library of functions that can be called by the head script (`analyze_FGFR3_dimer_simulations.py`) D.15 on page 338 or (`analyze_sidekick_FGFR3_dimer_simulations.py`) D.16 on page 349 to parse the FGFR3 *dimer* replicate MD trajectories

```
1 '''MDAnalysis tools for extracting information from FGFR3 DIMER trajectories.'''
3 import MDAnalysis
4 import numpy
5 import math
7 def geometric_center_z_coordinate_CA(selection):
8     '''Returns Z coordinate of the geometric center of mass for CA particles in a
9     selection.'''
10    ca = selection.selectAtoms("name CA")
11    geometric_center_xyz = ca.centerOfGeometry()
12    return geometric_center_xyz[2]
13 def geo_Z_tracking(folder_name, universe_object, skip_frames, system_name, output_file)
14 :
15     '''Prints the Z coordinate of the geometric center for each of the GpA or FGFR3
16     helices (based on 'system_name' flag) to a specified
17     gnuplot-ready output file for a specified frame interval. This function should
18     only be used on trajectories that have not been
19     centered by GROMACS trjconv because centering will reduce the amplitude of Z
20     coordinate motion for helix 1 relative to helix 2 (i.e., since I always
21     center helix 1).'''
22     if system_name == 'FGFR3':
23         helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
24             monomer
```



```

21     helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
22         FGFR3 monomer
23     elif system_name == 'GpA':
24         helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
25             monomer
26         helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
27             monomer
28     Z_outfile = open(output_file, 'w')
29     Z_outfile.write('#column 1: frame # || column 2: helix 1 geometric center Z
30         coordinate || column 3: helix 2 geometric center Z coordinate\n')
31     for ts in universe_object.trajectory[::skip_frames]:
32         helix_1_CA_center_Z = universe_object.selectAtoms(helix_1_CA_selection)
33             .centerOfGeometry()[2]
34         helix_2_CA_center_Z = universe_object.selectAtoms(helix_2_CA_selection)
35             .centerOfGeometry()[2]
36         Z_outfile.write(str(ts.frame) + ' ' + str(helix_1_CA_center_Z) + ' ' +
37             str(helix_2_CA_center_Z) + '\n')
38         print str(folder_name) + ' -- frame: ' + str(ts.frame)
39     Z_outfile.close()
40
41 def helix_tilt_vs_bilayer_normal(folder_name, universe_object, skip_frames, output_file
42 ):
43     '''Calculates the tilt of helix (CA backbone) relative to bilayer normal using
44     linear algebra
45     SVD technique to define helix axis (first eigenvector) and bilayer normal (
46     third eigenvector). This
47     particular function has been extended to deal with FGFR3 dimer trajectories.'''
48     helix_tilt_angle_file = open(output_file, 'w')
49     helix_tilt_angle_file.write('#column 1: frame # || column 2: helix 1 tilt angle
50         (degrees) relative to bilayer normal (eigenvector approach)\n#column 3:

```

```

helix 2 tilt angle (degrees) relative to bilayer normal (eigenvector approach
)\n')
for ts in universe_object.trajectory[::skip_frames]:
#removed many of the comments used in monomer version, and made a few
simple adjustments to select and deal with two monomers instead of
one
helix_1_CA_selection = universe_object.selectAtoms("name CA and resid
1:33")
helix_1_CA_coordinates = helix_1_CA_selection.coordinates()
helix_1_CA_geometric_center = helix_1_CA_selection.centerOfGeometry()

helix_2_CA_selection = universe_object.selectAtoms("name CA and resid
34:66")
helix_2_CA_coordinates = helix_2_CA_selection.coordinates()
helix_2_CA_geometric_center = helix_2_CA_selection.centerOfGeometry()

bilayer_phosphate_selection = universe_object.selectAtoms("name P04")
phosphate_coordinates = bilayer_phosphate_selection.coordinates()
phosphate_geometric_center = bilayer_phosphate_selection
centerOfGeometry()

#the idea is to do a SVD on the centered coordinates; so subtract the
geometric center coordinate value from the coordinates of the system
#when you call the numpy SVD function:
uu, dd, vv = numpy.linalg.svd(helix_1_CA_coordinates -
helix_1_CA_geometric_center)
helix_1_vector = vv[0] #this is the first eigenvector; checking in VMD,
it looks right along the helix axis
#repeat for second helix in dimer sims:

```

39

41

43

45

47  
58

49

51

53

55

57

```

59 uu, dd, vv = numpy.linalg.svd(helix_2_CA_coordinates -
60     helix_2_CA_geometric_center)
61 helix_2_vector = vv[0]
62 #repeat the eigenvector calculation process for the bilayer (phosphates
63     )
64 aa, bb, cc = numpy.linalg.svd(bilayer_phosphate_selection.coordinates()
65     -bilayer_phosphate_selection.centerOfGeometry())
66 bilayer_normal = cc[2] #check in VMD: yes, the third eigenvector seems
67     to produce a reasonable vector up +Z axis for bilayer normal
68
69 #angle between helix axis and bilayer normal vectors depends on the dot
70     product:
71 helix_1_angle = math.acos(numpy.dot(helix_1_vector, bilayer_normal))
72 #convert to degrees:
73 theta_1 = math.degrees(helix_1_angle)
74 if theta_1 > 90:
75     theta_1 = 180-theta_1
76
77 helix_2_angle = math.acos(numpy.dot(helix_2_vector, bilayer_normal))
78 #convert to degrees:
79 theta_2 = math.degrees(helix_2_angle)
80 if theta_2 > 90:
81     theta_2 = 180-theta_2
82
83 helix_tilt_angle_file.write(str(ts.frame) + ' ' + str(theta_1) + ' ' +
84     str(theta_2) + '\n') #formatting for gnuplot-ready columns
85 print str(folder_name) + ' -- frame: ' + str(ts.frame)
86
87 def GpA_helix_tilt_vs_bilayer_normal(folder_name, universe_object, skip_frames,
88     output_file):

```

```

81 '''Slightly altered version of helix tilt function to deal with 23 residue GpA
      system.'''
82 helix_tilt_angle_file = open(output_file, 'w')
83 helix_tilt_angle_file.write('#column 1: frame # || column 2: helix 1 tilt angle
      (degrees) relative to bilayer normal (eigenvector approach)||\n#column 3:
      helix 2 tilt angle (degrees) relative to bilayer normal (eigenvector approach
      )\n')
      for ts in universe_object.trajectory[::skip_frames]:
84 #removed many of the comments used in monomer version, and made a few
      simple adjustments to select and deal with two monomers instead of
      one
85 helix_1_CA_selection = universe_object.selectAtoms("name CA and resid
      1:23")
86 helix_1_CA_coordinates = helix_1_CA_selection.coordinates()
87 helix_1_CA_geometric_center = helix_1_CA_selection.centerOfGeometry()
88
89 helix_2_CA_selection = universe_object.selectAtoms("name CA and resid
      24:46")
90 helix_2_CA_coordinates = helix_2_CA_selection.coordinates()
91 helix_2_CA_geometric_center = helix_2_CA_selection.centerOfGeometry()
92
93 bilayer_phosphate_selection = universe_object.selectAtoms("name P04")
94 phosphate_coordinates = bilayer_phosphate_selection.coordinates()
95 phosphate_geometric_center = bilayer_phosphate_selection.coordinates()
96 centerOfGeometry()
97
98 #the idea is to do a SVD on the centered coordinates; so subtract the
99 geometric center coordinate value from the coordinates of the system
      #when you call the numpy SVD function:

```

```

uu, dd, vv = numpy.linalg.svd(helix_1_CA_coordinates -
    helix_1_CA_geometric_center)
helix_1_vector = vv[0] #this is the first eigenvector; checking in VMD,
    it looks right along the helix axis
#repeat for second helix in dimer sims:
uu, dd, vv = numpy.linalg.svd(helix_2_CA_coordinates -
    helix_2_CA_geometric_center)
helix_2_vector = vv[0]
#repeat the eigenvector calculation process for the bilayer (phosphates
)
aa, bb, cc = numpy.linalg.svd(bilayer_phosphate_selection.coordinates()
    -bilayer_phosphate_selection.centerOfGeometry())
bilayer_normal = cc[2] #check in VMD: yes, the third eigenvector seems
    to produce a reasonable vector up +Z axis for bilayer normal

#angle between helix axis and bilayer normal vectors depends on the dot
    product:
helix_1_angle = math.acos(numpy.dot(helix_1_vector, bilayer_normal))
#convert to degrees:
theta_1 = math.degrees(helix_1_angle)
if theta_1 > 90:
    theta_1 = 180-theta_1

helix_2_angle = math.acos(numpy.dot(helix_2_vector, bilayer_normal))
#convert to degrees:
theta_2 = math.degrees(helix_2_angle)
if theta_2 > 90:
    theta_2 = 180-theta_2

```

101

103

105

107

361

109

111

113

115

117

119

121

```

123 helix_tilt_angle_file.write(str(ts.frame) + ' ' + str(theta_1) + ' ' +
124     str(theta_2) + '\n') #formatting for gnuplot-ready columns
125     print str(folder_name) + ' -- frame: ' + str(ts.frame)
126
127 def closest_approach_GpA(folder_name, universe_object, skip_frames, output_file):
128     '''Calculates the closest approach distance between the two helices in the (
129     control) GpA dimer simulations for
130     a given frame range and interval. This function has been confirmed to work
131     properly against my VMD tcl script
132     that was designed for the same purpose.'''
133     closest_approach_file = open(output_file, 'w')
134     closest_approach_file.write('# column 1: frame number || column 2: closest
135     helix-helix contact (Angstroms)\n')
136     for ts in universe_object.trajectory[::skip_frames]:
137         boundary = 1 #start with contacts within 1 Angstrom in any direction
138         helix_1_CA_selection = "( name CA and resid 1:23 )"
139         helix_2_CA_selection = "( name CA and resid 24:46 )"
140         distance_list = []
141         while 1: #until contacts are found within a given cutoff distance
142             between_helices
143                 helix_2_selection_string = "( around %s %s ) and %s" % (str(
144                     boundary), helix_1_CA_selection, helix_2_CA_selection)
145                 helix_1_selection_string = "( around %s %s ) and %s" % (str(
146                     boundary), helix_2_CA_selection, helix_1_CA_selection)
147                 try: #assigning an empty atom selection raises an exception, so
148                     using a try block
149                         helix_2_residues_within_cutoff = universe_object.
150                             selectAtoms(helix_2_selection_string)
151                         helix_1_residues_within_cutoff = universe_object.
152                             selectAtoms(helix_1_selection_string)

```

```

143         break #if there are contacts you can exit the while
144             loop to assess the distance
145     except Exception: #increase the boundary by 2 Angstroms until a
146         contact on helix 2 is found
147             boundary += 2
148
149     #I don't think the atomselections will be directly 'zippable' for
150     pairwise distance measures and will have to loop through everything:
151     for helix_1_CA in helix_1_residues_within_cutoff:
152         for helix_2_CA in helix_2_residues_within_cutoff:
153             current_distance = numpy.linalg.norm(numpy.subtract(
154                 helix_1_CA.pos, helix_2_CA.pos)) #calculate magnitude
155                 (length) of distance vectors for contacts
156                 distance_list.append(current_distance)
157
158     distance_list.sort()
159     closest_contact_distance = distance_list[0]
160     closest_approach_file.write(str(ts.frame) + ' ' + str(
161         closest_contact_distance) + '\n')
162     print str(folder_name) + ' -- frame: ' + str(ts.frame)
163     closest_approach_file.close()
164
165 def top_five_closest_residues_GpA(folder_name, universe_object, skip_frames,
166     output_file):
167     '''The idea for this function is to take the top five closest GpA residues in
168     each frame after the helices are within
169     some cutoff distance from one another. The reason to require some degree of
170     proximity is the likelihood
171     that the helices have stochastic facial presentations until they are relatively
172     close together. Will probably
173     want this function to bin the data categorically by residue name and number so
174     a single bar plot can be generated with

```

```

161 N->C residue name on x-axis and normalized frequency of occurrence in top five
162     contacts on the y-axis.'''
163 helix_proximity_requirement = 7 # helices must be within this number of
164     Angstroms for contacts to be counted
165 top_five_file = open(output_file, 'w')
166 compound_distance_list = [] #the top five closest contacts from each acceptable
167     frame will be appended here; resets at the start of a new trajectory/
168     simulation
169 for ts in universe_object.trajectory[::skip_frames]:
170     helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
171         monomer
172     helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
173         monomer
174     helix_2_selection_string = "( around %s %s ) and %s" % (str(
175         helix_proximity_requirement), helix_1_CA_selection,
176         helix_2_CA_selection)
177     try:
178         helix_2_residues_within_cutoff = universe_object.selectAtoms(
179             helix_2_selection_string) #raises Exception if there are no
180             helix 2 contacts within the helix_proximity_requirement
181             distance from helix 1 set above. If there are contacts, no
182             exception will be raised and the function can move on to
183             ranking the closest contacts by residue name and number.
184         #Now that the helices are close enough select all CA atoms in
185             each:
186             helix_1_CA_residues = universe_object.selectAtoms(
187                 helix_1_CA_selection)
188             helix_2_CA_residues = universe_object.selectAtoms(
189                 helix_2_CA_selection)

```



```
#Use the atomselections to generate lists of ordered
coordinates for each helix:
helix_1_CA_coordinates = helix_1_CA_residues.coordinates()
helix_2_CA_coordinates = helix_2_CA_residues.coordinates()

#Use the atomselections to generate a list of ordered residue
identifiers (MDAnalysis format) for each helix:
helix_1_identifiers = list(helix_1_CA_residues)
helix_2_identifiers = list(helix_2_CA_residues)

#test output format at terminal:
#print helix_1_CA_coordinates
#print helix_1_identifiers[0].resname, helix_1_identifiers[0].
resid

#make a list of ordered residue names and numbers (in tuples at
the moment)
helix_1_ordered_residue_names_numbers = [(residue.resname,
residue.resid) for residue in helix_1_identifiers]
helix_2_ordered_residue_names_numbers = [(residue.resname,
residue.resid) for residue in helix_2_identifiers]

#now zip the residue names and numbers with their respective CA
coordinates
helix_1_merged_list = zip(helix_1_ordered_residue_names_numbers
, helix_1_CA_coordinates)
helix_2_merged_list = zip(helix_2_ordered_residue_names_numbers
, helix_2_CA_coordinates)
```

```
197 #this would give the [x y z] coordinate of the 23rd CA in helix
198     1: print helix_1_merged_list[22][1]
199 #the zeroth element would be the corresponding residue name and
200     number
201
202 #go through all possible combinations of residue interactions
203     between the two helices and make a list of the the distance (
204     magnitude)
205     #between the CA particles which retains the residue names that
206     correspond to the interaction
207     labelled_interhelix_distances = []
208     for (helix_1_CA, coordinate_1) in helix_1_merged_list:
209         for (helix_2_CA, coordinate_2) in helix_2_merged_list:
210             measured_distance = numpy.linalg.norm(numpy.
211                 subtract(coordinate_1, coordinate_2))
212             labelled_interhelix_distances.append([
213                 helix_1_CA, helix_2_CA, measured_distance])
214
215 #sort the list of labelled distances by the 'third element,'
216     which is the distance between CA particles
217     labelled_interhelix_distances.sort(key = lambda element:
218         element [2])
219
220 #so, now there is an ascending list of closest contacts between
221     helices for this given frame (if they are close enough)
222
223 #append the five closest contacts from this given frame to an
224     overall list for each frame:
225     for element in labelled_interhelix_distances[0:5]:
```

```
215     compound_distance_list.append(element) #still contains
216         residue 1 and 2 names along with distance
217
218     except Exception: #the helices aren't close enough for a meaningful
219         interface residue assessment defined by helix_proximity_requirement
220         above
221         pass #so just leave the try block and jump to the next frame
222         iteration in the for loop
223
224     print str(folder_name) + ' -- frame: ' + str(ts.frame)
225
226 #the compound_distance_list should now contain all the top five contacts
227     concatenated from all eligible frames for the current trajectory
228 #want to go through this list and the number of times each pair shows up and
229     divide by the total number of pairs
230
231 #determine number of top five contacts in total:
232 num_pairs = len(compound_distance_list)
233
234 helix_1_partner_list = []
235 helix_2_partner_list = []
236 #make separate lists with each of the residue names (including duplicates)
237     found in close contacts from the respective helices
238 for (helix_1_partner, helix_2_partner, separation) in compound_distance_list:
239     helix_1_partner_list.append(helix_1_partner) #the involved residue from
240         helix 1 gets appended
241     helix_2_partner_list.append(helix_2_partner) #same for cognate helix 2
242         residue in a separate list
```

```

235 #Now, each list basically has an entry for every time a given residue showed up
      in the top five for an acceptable frame from one of the helices
236 #This includes duplicates so can use the 'set' data structure to remove
      duplicates for iteration purposes without actually removing the important
237 #multiple occurrences within the lists proper
      helix_1_no_duplicates = set(helix_1_partner_list)
      helix_2_no_duplicates = set(helix_2_partner_list)
238
239 #test print compare the two to verify duplicate removal:
      #top_five_file.write(str(helix_1_partner_list)+'\n')
      #top_five_file.write(str(helix_1_no_duplicates)+'\n')
240
241 #results will be printed in the following format, specified at top of file in
      gnuplot-friendly form:
242 top_five_file.write('#first three columns: helix 1 rename || resnum ||
      frequency in top 5 closest contacts \n')
243 top_five_file.write('#last three columns: helix 2 rename || resnum ||
      frequency in top 5 closest contacts \n')
244
245 #go through the list of all residues counted by frame and count the unique
      occurrence of a given residue from a given helix, and print result to file
246 for (residue1_name, residue2_name) in zip(helix_1_no_duplicates,
      helix_2_no_duplicates): #limited by the shortest list though; this could be a
      problem if only a really small number of unique residues were picked from
      one helix
247     residue1_count = helix_1_partner_list.count(residue1_name) #raw residue
      counts for helix1
      residue2_count = helix_2_partner_list.count(residue2_name) #raw residue
      counts for helix2
248
249
250
251
252
253

```

```

255 residue1_frequency = float(residue1_count)/float(num_pairs) #because we
      want the 'normalized frequency'
256 residue2_frequency = float(residue2_count)/float(num_pairs)
257
258 #print out results that are simultaneously corrected for residue
      numbers to match those use in Emi's work (Biochemistry 2008 vol. 47
      paper)
259 top_five_file.write(str(residue1_name[0]).strip("")) + ' ' +str(
      residue1_name[1]+72) + ' ' + str(residue1_frequency) + ' ' + str(
      residue2_name[0]).strip("")) + ' ' +str(residue2_name[1]+(72-23)) + '
      ' + str(residue2_frequency) + '\n')
260
261 top_five_file.close()
262
263 def top_five_closest_residues_FGFR3(folder_name, universe_object, skip_frames,
      output_file):
264     '''Modified version of the similarly named function adjusted to work for FGFR3.
      '''
265     helix_proximity_requirement = 7 # helices must be within this number of
      Angstroms for contacts to be counted
266     top_five_file = open(output_file, 'w')
267     compound_distance_list = [] #the top five closest contacts from each acceptable
      frame will be appended here; resets at the start of a new trajectory/
      simulation
268     for ts in universe_object.trajectory[::skip_frames]:
269         helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
            monomer
            helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
            FGFR3 monomer

```

```
271 helix_2_selection_string = "( around %s %s ) and %s" % (str(
    helix_proximity_requirement), helix_1_CA_selection,
    helix_2_CA_selection)
272
273 try:
    helix_2_residues_within_cutoff = universe_object.selectAtoms(
        helix_2_selection_string) #raises Exception if there are no
        helix_2 contacts within the helix_proximity_requirement
        distance from helix 1 set above. If there are contacts, no
        exception will be raised and the function can move on to
        ranking the closest contacts by residue name and number.
274
275 #Now that the helices are close enough select all CA atoms in
    each:
    helix_1_CA_residues = universe_object.selectAtoms(
        helix_1_CA_selection)
    helix_2_CA_residues = universe_object.selectAtoms(
        helix_2_CA_selection)
276
277 #Use the atomselections to generate lists of ordered
    coordinates for each helix:
    helix_1_CA_coordinates = helix_1_CA_residues.coordinates()
    helix_2_CA_coordinates = helix_2_CA_residues.coordinates()
278
279 #Use the atomselections to generate a list of ordered residue
    identifiers (MDAnalysis format) for each helix:
    helix_1_identifiers = list(helix_1_CA_residues)
    helix_2_identifiers = list(helix_2_CA_residues)
280
281 #test output format at terminal:
    #print helix_1_CA_coordinates
282
283
284
285
286
287
```

```

289 #print helix_1_identifiers[0].resname, helix_1_identifiers[0].
    resid
291 #make a list of ordered residue names and numbers (in tuples at
    the moment)
    helix_1_ordered_residue_names_numbers = [(residue.resname,
    residue.resid) for residue in helix_1_identifiers]
293 helix_2_ordered_residue_names_numbers = [(residue.resname,
    residue.resid) for residue in helix_2_identifiers]
295 #now zip the residue names and numbers with their respective CA
    coordinates
    helix_1_merged_list = zip(helix_1_ordered_residue_names_numbers
    , helix_1_CA_coordinates)
    helix_2_merged_list = zip(helix_2_ordered_residue_names_numbers
    , helix_2_CA_coordinates)
299 #this would give the [x y z] coordinate of the 23rd CA in helix
    1: print helix_1_merged_list[22][1]
    #the zeroth element would be the corresponding residue name and
    number
301 #go through all possible combinations of residue interactions
    between the two helices and make a list of the the distance (
    magnitude)
303 #between the CA particles which retains the residue names that
    correspond to the interaction
    labelled_interhelix_distances = []
305 for (helix_1_CA, coordinate_1) in helix_1_merged_list:
    for (helix_2_CA, coordinate_2) in helix_2_merged_list:

```

```

307 measured_distance = numpy.linalg.norm(numpy.
308     subtract(coordinate_1, coordinate_2))
309     labelled_interhelix_distances.append([
310         helix_1_CA, helix_2_CA, measured_distance])
311
312 #sort the list of labelled distances by the 'third element,'
313     which is the distance between CA particles
314     labelled_interhelix_distances.sort(key = lambda element:
315         element [2])
316
317 #so, now there is an ascending list of closest contacts between
318     helices for this given frame (if they are close enough)
319
320 #append the five closest contacts from this given frame to an
321     overall list for each frame:
322     for element in labelled_interhelix_distances[0:5]:
323         compound_distance_list.append(element) #still contains
324             residue 1 and 2 names along with distance
325
326 except Exception: #the helices aren't close enough for a meaningful
327     interface residue assessment defined by helix_proximity_requirement
328     above
329     pass #so just leave the try block and jump to the next frame
330     iteration in the for loop
331
332 print str(folder_name) + ' -- frame: ' + str(ts.frame)
333
334 #the compound_distance_list should now contain all the top five contacts
335     concatenated from all eligible frames for the current trajectory

```



```

327 #want to go through this list and the number of times each pair shows up and
    divide by the total number of pairs
329
330 #determine number of top five contacts in total:
    num_pairs = len(compound_distance_list)
331
332 helix_1_partner_list = []
    helix_2_partner_list = []
333 #make separate lists with each of the residue names (including duplicates)
    found in close contacts from the respective helices
334 for (helix_1_partner, helix_2_partner, separation) in compound_distance_list:
    helix_1_partner_list.append(helix_1_partner) #the involved residue from
        helix 1 gets appended
    helix_2_partner_list.append(helix_2_partner) #same for cognate helix 2
        residue in a separate list
335
336 #Now, each list basically has an entry for every time a given residue showed up
    in the top five for an acceptable frame from one of the helices
337 #This includes duplicates so can use the 'set' data structure to remove
    duplicates for iteration purposes without actually removing the important
    #multiple occurrences within the lists proper
338 helix_1_no_duplicates = set(helix_1_partner_list)
    helix_2_no_duplicates = set(helix_2_partner_list)
339
340 #test print compare the two to verify duplicate removal:
    #top_five_file.write(str(helix_1_partner_list)+'\n')
    #top_five_file.write(str(helix_1_no_duplicates)+'\n')
341
342 #results will be printed in the following format, specified at top of file in
    gnuplot-friendly form:
343
344
345
346
347

```

```

349 top_five_file.write('#first three columns: helix 1 resname || resnum ||
frequency in top 5 closest contacts \n')
351 top_five_file.write('#last three columns: helix 2 resname || resnum ||
frequency in top 5 closest contacts \n')

#go through the list of all residues counted by frame and count the unique
occurrence of a given residue from a given helix, and print result to file
353 for (residue1_name,residue2_name) in zip(helix_1_no_duplicates,
helix_2_no_duplicates): #limited by the shortest list though; this could be a
problem if only a really small number of unique residues were picked from
one helix
    residue1_count = helix_1_partner_list.count(residue1_name) #raw residue
counts for helix1
    residue2_count = helix_2_partner_list.count(residue2_name) #raw residue
counts for helix2

355 residue1_frequency = float(residue1_count)/float(num_pairs) #because we
want the 'normalized frequency'
357 residue2_frequency = float(residue2_count)/float(num_pairs)

#print out results that are simultaneously corrected for residue
numbers to match those used in FGFR3 literature
359 top_five_file.write(str(residue1_name[0]).strip("'") + ' ' +str(
residue1_name[1]+366) + ' ' + str(residue1_frequency) + ' ' + str(
residue2_name[0]).strip("'") + ' ' +str(residue2_name[1]+(366-33)) +
' ' + str(residue2_frequency) + '\n')

361
363
365 top_five_file.close()

```

```

367 def merged_top_five_FGFR3(mutant_overall_list, wildtype_overall_list, hetero_overall_list
, folder_name, universe_object, skip_frames):
    '''Parses and merges the 'top five' contact data for wild-type, mutant, and
hetero- dimers into separate lists
which are parsed and compared in a similar fashion to that used for individual
trajectories.'''
369 helix_proximity_requirement = 7 # helices must be within this number of
Angstroms for contacts to be counted

371 compound_distance_list = [] #the top five closest contacts from each acceptable
frame will be appended here; resets at the start of a new trajectory/
simulation
373 for ts in universe_object.trajectory[::skip_frames]:
    helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
monomer
    helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
FGFR3 monomer
375 helix_2_selection_string = "( around %s %s ) and %s" % (str(
helix_proximity_requirement), helix_1_CA_selection,
helix_2_CA_selection)
    try:
377         helix_2_residues_within_cutoff = universe_object.selectAtoms(
helix_2_selection_string) #raises Exception if there are no
helix 2 contacts within the helix_proximity_requirement
distance from helix 1 set above. If there are contacts, no
exception will be raised and the function can move on to
ranking the closest contacts by residue name and number.

#Now that the helices are close enough select all CA atoms in
each:

```

```
381 helix_1_CA_residues = universe_object.selectAtoms(  
    helix_1_CA_selection)  
382 helix_2_CA_residues = universe_object.selectAtoms(  
    helix_2_CA_selection)  
383  
384 #Use the atomselections to generate lists of ordered  
    coordinates for each helix:  
385 helix_1_CA_coordinates = helix_1_CA_residues.coordinates()  
    helix_2_CA_coordinates = helix_2_CA_residues.coordinates()  
386  
387 #Use the atomselections to generate a list of ordered residue  
    identifiers (MDAnalysis format) for each helix:  
388 helix_1_identifiers = list(helix_1_CA_residues)  
    helix_2_identifiers = list(helix_2_CA_residues)  
389  
390 #test output format at terminal:  
391 #print helix_1_CA_coordinates  
392 #print helix_1_identifiers[0].resname, helix_1_identifiers[0].  
    resid  
393  
394 #make a list of ordered residue names and numbers (in tuples at  
    the moment)  
395 helix_1_ordered_residue_names_numbers = [(residue.resname,  
    residue.resid) for residue in helix_1_identifiers]  
396 helix_2_ordered_residue_names_numbers = [(residue.resname,  
    residue.resid) for residue in helix_2_identifiers]  
397  
398 #now zip the residue names and numbers with their respective CA  
    coordinates
```

```

401 helix_1_merged_list = zip(helix_1_ordered_residue_names_numbers
    , helix_1_CA_coordinates)
402 helix_2_merged_list = zip(helix_2_ordered_residue_names_numbers
    , helix_2_CA_coordinates)
403
404 #this would give the [x y z] coordinate of the 23rd CA in helix
    1: print helix_1_merged_list[22][1]
405 #the zeroth element would be the corresponding residue name and
    number
406
407 #go through all possible combinations of residue interactions
    between the two helices and make a list of the the distance (
    magnitude)
408 #between the CA particles which retains the residue names that
    correspond to the interaction
409 labelled_interhelix_distances = []
    for (helix_1_CA, coordinate_1) in helix_1_merged_list:
410         for (helix_2_CA, coordinate_2) in helix_2_merged_list:
411             measured_distance = numpy.linalg.norm(numpy.
                subtract(coordinate_1, coordinate_2))
412             labelled_interhelix_distances.append([
                helix_1_CA, helix_2_CA, measured_distance])
413
414 #sort the list of labelled distances by the 'third element,'
    which is the distance between CA particles
415 labelled_interhelix_distances.sort(key = lambda element:
    element [2])
416
417 #so, now there is an ascending list of closest contacts between
    helices for this given frame (if they are close enough)

```

```

419 #append the five closest contacts from this given frame to an
      overall list for each frame:
421 for element in labelled_interhelix_distances[0:5]:
      compound_distance_list.append(element) #still contains
      residue 1 and 2 names along with distance

      except Exception: #the helices aren't close enough for a meaningful
      interface residue assessment defined by helix_proximity_requirement
      above
      pass #so just leave the try block and jump to the next frame
      iteration in the for loop

      print str(folder_name) + ' -- frame: ' + str(ts.frame)

#the compound_distance_list should now contain all the top five contacts
  concatenated from all eligible frames for the current trajectory
**the part that is different with this merged approach is that I will want to
  simply dump the compound_distance_list to the appropriate
#list defined in merged_distance_lists() function above (which is called in
  main()), and then use another function to parse the trajectory
#after each of the three conditions/lists are ready
  if 'heterodimer' in folder_name:
      for element in compound_distance_list:
          hetero_overall_list.append(element)
  elif 'mutant_dimer' in folder_name:
      for element in compound_distance_list:
          mutant_overall_list.append(element)
  elif 'wildtype_dimer' in folder_name:

```

```

441     for element in compound_distance_list:
442         wildtype_overall_list.append(element)
443
444     def parse_overall_FGFR3_top_five_data(mutant_overall_list, wildtype_overall_list,
445     hetero_overall_list):
446         '''Perform final analysis on merged 'top five contacts' data for each of the
447         three FGFR3 conditions and print to file.'''
448
449         #determine number of top five contacts in total:
450         num_hetero_pairs = len(hetero_overall_list)
451         num_mutant_pairs = len(mutant_overall_list)
452         num_wildtype_pairs = len(wildtype_overall_list)
453
454     def parser_printer(input_list, condition, num_pairs): #does most of the work
455     for a given list of contact pairs
456         print input_list
457         file_path = '/sansom/sc2/bioc1009/Documents/FGFR3_work/
458         dimer_batch_analysis_symlink/merged_top_five_data/%s_merged_topfive.
459         out' % condition
460         merged_top_five_outfile = open(str(file_path), 'w')
461         helix_1_partner_list = []
462         helix_2_partner_list = []
463         #make separate lists with each of the residue names (including
464         duplicates) found in close contacts from the respective helices
465         for (helix_1_partner, helix_2_partner, separation) in input_list:
466             helix_1_partner_list.append(helix_1_partner) #the involved
467                 residue from helix 1 gets appended
468             helix_2_partner_list.append(helix_2_partner) #same for cognate
469                 helix 2 residue in a separate list

```

```

463 #Now, each list basically has an entry for every time a given residue
      showed up in the top five for an acceptable frame from one of the
      helices
465 #This includes duplicates so can use the 'set' data structure to remove
      duplicates for iteration purposes without actually removing the
      important
467 #multiple occurrences within the lists proper
      helix_1_no_duplicates = set(helix_1_partner_list)
      helix_2_no_duplicates = set(helix_2_partner_list)
469 #results will be printed in the following format, specified at top of
      file in gnuplot-friendly form:
      string_1 = '#columns 1-3 (%s): helix 1 resname || resnum || frequency
      in top 5 closest contacts \n' % condition
      string_2 = '#columns 4-6 (%s): helix 2 resname || resnum || frequency
      in top 5 closest contacts \n' % condition
471 merged_top_five_outfile.write(string_1)
      merged_top_five_outfile.write(string_2)
473
475 #go through the list of all residues counted by frame and count the
      unique occurrence of a given residue from a given helix, and print
      result to file
      for (residue1_name,residue2_name) in zip(helix_1_no_duplicates,
      helix_2_no_duplicates): #limited by the shortest list though; this
      could be a problem if only a really small number of unique residues
      were picked from one helix
          residue1_count = helix_1_partner_list.count(residue1_name) #raw
              residue counts for helix1
          residue2_count = helix_2_partner_list.count(residue2_name) #raw
              residue counts for helix2
477

```



```

479 residue1_frequency = float(residue1_count)/float(num_pairs) #
      because we want the 'normalized frequency'
481 residue2_frequency = float(residue2_count)/float(num_pairs)

      #print out results that are simultaneously corrected for
      residue numbers to match those used in FGFR3 literature
483 merged_top_five_outfile.write(str(residue1_name[0]).strip("")
      + ' ' +str(residue1_name[1]+366) + ' ' + str(
      residue1_frequency) + ' ' + str(residue2_name[0]).strip("")
      + ' ' +str(residue2_name[1]+(366-33)) + ' ' + str(
      residue2_frequency) + '\n')

      merged_top_five_outfile.close()

485
487
      #now call the function to parse and print three files of merged data for each
      FGFR3 dimer condition simulated
489 parser_printer(hetero_overall_list, 'heterodimer', num_hetero_pairs)
      parser_printer(mutant_overall_list, 'mutant', num_mutant_pairs)
491 parser_printer(wildtype_overall_list, 'wildtype', num_wildtype_pairs)

493 def relative_helical_motion(folder_name, universe_object, skip_frames, output_file,
      system_name):
      '''Tracks the motion of one helix (its geometric center) relative to a helix
      with geometric center coordinates adjusted to (0, 0, 0) in each frame.
      Basically subtract the coordinates of the geometric center of the reference
      helix from both of the geometric centers--its own and that of the other helix
      . The idea is to produce something similar to Figure 4 in Biochemistry, Vol.
      47, No.40, 2008, 10507. Accepts system name of 'GpA' to parse glycoporphin A

```

```

trajectories or 'FGFR3' to parse those trajectories. Important note: ** this
should be performed on xtc files with -pbc nojump and -center used in trjconv
because of the periodic boundary behaviour (avoid jumping from one side of
box to the other).'''
if system_name == 'FGFR3':
    helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
    monomer
    helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
    FGFR3 monomer
elif system_name == 'GpA':
    helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
    monomer
    helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
    monomer

helix_tracking_file = open(output_file, 'w')
helix_tracking_file.write('#coordinate of geo center, format: ref helix (x) |
ref helix (y) | helix_2 (x) | helix_2 (y)\n')
for ts in universe_object.trajectory[::skip_frames]:
    #select all the CA particles in each helix:
    helix_1_CA_residues = universe_object.selectAtoms(
        helix_1_CA_selection)
    helix_2_CA_residues = universe_object.selectAtoms(
        helix_2_CA_selection)

    #find the center of geometry for each helix:
    helix_1_center = helix_1_CA_residues.centerOfGeometry()
    helix_2_center = helix_2_CA_residues.centerOfGeometry()

#for the purpose of the plot we want to produce, helix 1 will
    be the reference helix with geometric center at 0,0,0 each

```

495

497

499

501

503

505

507

509

511

```

513         frame
514     #so, subtract its center coordinates from each helix in each
515         frame
516     helix_1_plot_coordinate = helix_1_center - helix_1_center
517     helix_2_plot_coordinate = helix_2_center - helix_1_center
518
519     #write the x,y coordinates to file (helix 1 should always be
520     #0,0 so might be able to stop writing that after testing)
521     helix_tracking_file.write(str(helix_1_plot_coordinate[0]) + ' '
522     + str(helix_1_plot_coordinate[1]) + ' ' + str(
523     helix_2_plot_coordinate[0]) + ' ' + str(
524     helix_2_plot_coordinate[1]) + '\n')
525     print str(folder_name) + ' -- frame: ' + str(ts.frame)
526
527     helix_tracking_file.close()
528
529     def compute_rmsd(a,b):
530     '''Returns RMSD between two coordinate sets a and b. Copied from rmsfit.py module
531     distributed with MDAnalysis.'''
532     return numpy.sqrt(numpy.sum(numpy.power(a-b,2))/a.shape[0])
533
534     def fixed_helix_thermal(folder_name, universe_object, skip_frames,
535     dimer_symlink_directory, create_universe_selections, system_name, output_file):
536     '''Similar to the 'relative_helical_motion' function but designed so that the
537     reference helix always faces the same direction to allow study of the
538     preferred dimerization interface (if there is one). Idea is to produce a '
539     thermal' plot similar to that used for Fig. 5 in Biochemistry (2008), 47,
540     10507. Will probably bin the data in a format appropriate for the gnuplot '
541     splot' style. **Important: use with trajectories processed with trjconv -pbc
542     mol and NOT -pbc nojump'''

```

```

#challenge: can I use the same reference structure for all three FGFR3
conditions (i.e., use WT configuration as the rmsd reference for mutant as
well)? Basically, I want each of the three FGFR3 conditions to be directly
comparable by visual inspection of the figure.
outfile = open(output_file, 'w')
outfile.write("#Format: ref. helix x || ref. helix y || helix 2 x || helix 2 y
\n") #not really what gets printed
list_helix_1_centers = []
list_helix_2_centers = []
#for the reference structure I'm currently planning to use the first frame (and
first helix) of the first WT dimer simulation for either FGFR3 or GpA. To
allow creation of an MDAnalysis Universe and atom selection from either of
these systems the function needs to know which folder the reference
trajectories reside in and which residues to select:
if system_name == 'FGFR3':
    ref_folder_name_list = ['wildtype_dimer_replicate_1']
    helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
    monomer
    helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
    FGFR3 monomer
elif system_name == 'GpA':
    ref_folder_name_list = ['GpA_dimer_replicate_1']
    helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
    monomer
    helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
    monomer

#now create the (universe object, folder_name) list (of length 1) for the
system using a function defined in the head script:

```

539

531

533

535  
84

537

539

541

543

```

545 universe_data = create_universe_selections(dimer_symlink_directory,
      ref_folder_name_list)
547 #universe object is the first element in the sublist:
      ref_universe_object = universe_data[0][0] #careful here! you had incorrectly
      reassigned universe_object which is very bad
549 #select CA particles corresponding to the reference structure:
      reference_CA_selection = ref_universe_object.selectAtoms(helix_1_CA_selection)
551 #get the reference (center of geometry zero'd) coordinates:
      ref_coordinates = reference_CA_selection.coordinates() - reference_CA_selection
      .centerOfGeometry()
553 #need masses for input to rms/transformation function:
      masses = reference_CA_selection.masses()

555 #So, we now have the reference structure with center of geometry at (0,0,0). We
      are ready to start looping through frames in the trajectory and doing work
      there:
557 for ts in universe_object.trajectory[::skip_frames]:
      #Because helix 1 may move a bit we need to select it again in each
      frame. We will also want to select helix 2 because we are going to
      track its relative position:
559 current_helix_1_CA_selection = universe_object.selectAtoms(
      helix_1_CA_selection)
      current_helix_2_CA_selection = universe_object.selectAtoms(
      helix_2_CA_selection)
561 #Find the (center of geometry zero'd) coordinates for helix 1 in each
      frame. To keep things consistent in relative terms, perform the same
      operation on helix 2 (i.e., translate it by the same amount):
      current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
      () - current_helix_1_CA_selection.centerOfGeometry()

```

```

563 current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
564     () - current_helix_1_CA_selection.centerOfGeometry()

565 #We need to figure out what kind of transformation is needed for the
566 best (rmsd) fit of current helix 1 back to the reference structure.
567 This is based on the MDAnalysis/examples/rmsfit/rmsfit.py example code
568 distributed with the most recent release of MDAnalysis:
569 transformation = numpy.matrix(MDAnalysis.core.rms_fitting.
570     rms_rotation_matrix(current_helix_1_coordinates,ref_coordinates,
571     masses))

572 #Apply the transformation that puts helix 1 in the best (rmsd fit)
573 match to the reference:
574 helix_1_best_fit_coordinates = current_helix_1_coordinates *
575     transformation
576 #We want to perform the same transformation on helix 2 so that we
577 maintain consistency relative to the reference structure in each
578 frame (both helices must always experience the same transformation):
579 helix_2_updated_coordinates = current_helix_2_coordinates *
580     transformation

581 #After the transformations, both helices are represented as numpy
582 matrices of coordinates. Since the CA particles all have the same
583 mass, we can treat the mass as 1 and simply find the average position
584 of all particles to represent the center of mass for each helix:
585 #calc com helix 1:
586 x_sum=0; y_sum=0; z_sum=0
587 for x,y,z in helix_1_best_fit_coordinates.tolist(): #convert numpy
588     matrix object to list for std indexing methods
589         x_sum += x; y_sum += y; z_sum += z

```

```

center_of_mass_helix_1 = [x_sum/len(helix_1_best_fit_coordinates), y_sum
/len(helix_1_best_fit_coordinates), z_sum/len(
helix_1_best_fit_coordinates)]
#repeat for helix 2:
x_sum=0; y_sum=0; z_sum=0
for x,y,z in helix_2_updated_coordinates.tolist():
    x_sum += x; y_sum += y; z_sum += z
center_of_mass_helix_2 = [x_sum/len(helix_2_updated_coordinates), y_sum/
len(helix_2_updated_coordinates), z_sum/len(
helix_2_updated_coordinates)]
#put the centers in a list so they can be accessed by the code that
will bin the data after the frame-stepping loop exits
list_helix_1_centers.append(center_of_mass_helix_1)
list_helix_2_centers.append(center_of_mass_helix_2)
print str(folder_name) + ' -- frame: ' + str(ts.frame)

```

```

#now the function has finished looping through the current simulation and
grabbing the appropriate centers of mass for each helix
#helix 1 is more or less fixed near the origin because its geometric center is
zeroed for referencing purposes
#for x and y coordinate bin ranges we'll want to generated the bounds using the
min() and max() values for x and y coordinates in the helix 2 list:
helix_2_x_coord_list = [x for (x,y,z) in list_helix_2_centers]
helix_2_y_coord_list = [y for (x,y,z) in list_helix_2_centers]
#set a bin separation and build bins around min and max values
bin_separation = 0.4
list_of_bin_counts = []
#populate the bin value lists based on the bounds of data
x = min(helix_2_x_coord_list)
x_bin_value_list = []

```

579

581

583

585

587

589

591

593

595

597

599

```

while x <= max(helix_2_x_coord_list):
    x_bin_value_list.append(x)
    x += bin_separation
#
y = min(helix_2_y_coord_list)
y_bin_value_list = []
while y <= max(helix_2_y_coord_list):
    y_bin_value_list.append(y)
    y += bin_separation

#for every x coordinate of helix 2 find a matching x-bin where it belongs:
for helix_2_x, helix_2_y, helix_2_z in list_helix_2_centers:
    for x_bin in x_bin_value_list:
        if helix_2_x > (x_bin - (bin_separation/2.0)) and helix_2_x <=
            (x_bin + (bin_separation/2.0)):
            #if the above condition is satisfied then we have found the
            appropriate x-bin for the x coordinate and we need to iterate
            through the y bins to see where the y coordinate fits:
                for y_bin in y_bin_value_list:
                    if helix_2_y > (y_bin - (bin_separation/2.0))
                        and helix_2_y <= (y_bin + (bin_separation
                            /2.0)):
                        #every time the helix 2 (x,y)
                        coordinate falls in a 2D bin, print
                        the x,y bin values in a tuple and
                        append to a list:
                            list_of_bin_counts.append((x_bin,y_bin)
                                )

```

601

603

605

607

609

611

613

388

615

617

619



```

#the x and y bin lists should be sorted by default so it should be sensible to
loop through each and produce a gnuplot-ready probability for each 2d bin
for x_bin in x_bin_value_list:
    for y_bin in y_bin_value_list:
        outfile.write(str(x_bin) + ' ' + str(y_bin) + ' ' + str(float(
            list_of_bin_counts.count((x_bin,y_bin)))/float(len(
                list_helix_2_centers))) + '\n')
        #we want an empty line between the blocks of (x-bin) data for gnuplot:
        outfile.write('\n')

outfile.close()
#gnuplot splot wants space-separated blocks of x, y, z data with the same
number of points in each block (i.e., a block for each x bin value, iterate
through all y bin values giving the corresponding z value in third column)

def fixed_helix_thermal_merged(folder_name, universe_object, skip_frames,
dimer_symlink_directory, create_universe_selections, heterodimer_helix_2_x_coord_list
, heterodimer_helix_2_y_coord_list, mutant_helix_2_x_coord_list,
mutant_helix_2_y_coord_list, wildtype_helix_2_x_coord_list,
wildtype_helix_2_y_coord_list):
    '''Modified version of the similarly named function in this module. This
function is designed to consider the data from all 10 FGFR3 replicates for
each of the three simulation conditions and produce data intended for a
single summary plot. I will probably remove some extraneous things in the
code that weren't really needed in the original either.'''

list_helix_2_centers = []
#the reference structure is simply the first helix in the first frame of the
first WT simulation
ref_folder_name_list = ['wildtype_dimer_replicate_1']

```

621

623

625

627

629

631

633

635

```

637 helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3 monomer
638 helix_2_CA_selection = "( name CA and resid 34:66 )" #select second FGFR3
639 monomer
640
641 #now create the (universe object, folder_name) list (of length 1) for the
642 system using a function defined in the head script:
643 universe_data = create_universe_selections(dimer_symlink_directory,
644 ref_folder_name_list)
645 #universe object is the first element in the sublist:
646 ref_universe_object = universe_data[0][0]
647 #select CA particles corresponding to the reference structure:
648 reference_CA_selection = ref_universe_object.selectAtoms(helix_1_CA_selection)
649 #get the reference (center of geometry zero'd) coordinates:
650 ref_coordinates = reference_CA_selection.coordinates() - reference_CA_selection
651 .centerOfGeometry()
652 #need masses for input to rms/transformation function:
653 masses = reference_CA_selection.masses()
654
655 #So, we now have the reference structure with center of geometry at (0,0,0). We
656 are ready to start looping through frames in the trajectory and doing work
657 there:
658 for ts in universe_object.trajectory[::skip_frames]:
659     #Because helix 1 may move a bit we need to select it again in each
660     frame. We will also want to select helix 2 because we are going to
661     track its relative position:
662     current_helix_1_CA_selection = universe_object.selectAtoms(
663         helix_1_CA_selection)
664     current_helix_2_CA_selection = universe_object.selectAtoms(
665         helix_2_CA_selection)

```

```
657 #Find the (center of geometry zero'd) coordinates for helix 1 in each
    frame. To keep things consistent in relative terms, perform the same
    operation on helix 2 (i.e., translate it by the same amount):
    current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
    () - current_helix_1_CA_selection.centerOfGeometry()
    current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
    () - current_helix_1_CA_selection.centerOfGeometry()
```

```
659 #We need to figure out what kind of transformation is needed for the
    best (rmsd) fit of current helix 1 back to the reference structure.
    This is based on the MDAnalysis/examples/rmsfit.py example code
    distributed with the most recent release of MDAnalysis:
    transformation = numpy.matrix(MDAnalysis.core.rms_fitting.
    rms_rotation_matrix(current_helix_1_coordinates,ref_coordinates,
    masses))
```

```
661 #We don't actually need to apply the transformation to helix 1--just
    want to know where helix 2 is relative to 1 now that 1 is fixed:
    helix_2_updated_coordinates = current_helix_2_coordinates *
    transformation
```

```
663 #Helix 2 is represented as a numpy matrix of coordinates. Since the CA
    particles all have the same mass, we can treat the mass as 1 and
    simply find the average position of all particles to represent the
    center of mass.
    x_sum=0; y_sum=0; z_sum=0
    for x,y,z in helix_2_updated_coordinates.tolist():
        x_sum += x; y_sum += y; z_sum += z
    center_of_mass_helix_2 = [x_sum/len(helix_2_updated_coordinates),y_sum/
    len(helix_2_updated_coordinates),z_sum/len(
```

```

671     helix_2_updated_coordinates)]
        #put the centers in a list so they can be accessed by the code that
        will bin the data after the frame-stepping loop exits
        list_helix_2_centers.append(center_of_mass_helix_2)
673     print str(folder_name) + ' -- frame: ' + str(ts.frame)

675     #we want to extract the x and y coordinates for helix 2 such that they are
        filtered by simulation type (i.e., folder name) and append them to an
        appropriately named list (these lists are initialized in the head script so
        they can be appended as the function loops through)
        if 'heterodimer' in folder_name:
677             for (x,y,z) in list_helix_2_centers:
                    heterodimer_helix_2_x_coord_list.append(x)
                    heterodimer_helix_2_y_coord_list.append(y)

        elif 'mutant' in folder_name:
679             for (x,y,z) in list_helix_2_centers:
                    mutant_helix_2_x_coord_list.append(x)
                    mutant_helix_2_y_coord_list.append(y)

        elif 'wildtype' in folder_name:
681             for (x,y,z) in list_helix_2_centers:
                    wildtype_helix_2_x_coord_list.append(x)
                    wildtype_helix_2_y_coord_list.append(y)

683     #so, after going through all 30 FGFR3 dimer simulations, each of the lists will
        contain the appropriate set of coordinates from 10 simulations. For binning
        and probability calculation it will be worthwhile to generalize the portion
        of the code that deals with that in a function and then call it after this
        function from the head script

685     def thermal_bins(helix_2_x_coord_list, helix_2_y_coord_list, output_file):
687
689

```

```

'''Parses the FGFR3 helix 2 coordinate list data from the
fixed_helix_thermal_merged() function and writes the binned probability data
to the appropriate file. Meant to work with the *overall* FGFR3 'thermal'
position data, so the lists contain coordinates from all 10 replicates in
each of the three simulation conditions. Call this function after
fixed_helix_thermal_merged() (and outside of main) in the head script. Note
that the non-merged function is different in that it contains its own binning
routines so you don't have to call this.'''

outfile = open(output_file, 'w')
#set a bin separation and build bins around min and max values
bin_separation = 0.4
list_of_bin_counts = []
#populate the bin value lists based on the bounds of data
x = min(helix_2_x_coord_list)
x_bin_value_list = []
while x <= max(helix_2_x_coord_list):
    x_bin_value_list.append(x)
    x += bin_separation
#
y = min(helix_2_y_coord_list)
y_bin_value_list = []
while y <= max(helix_2_y_coord_list):
    y_bin_value_list.append(y)
    y += bin_separation

#for every x coordinate of helix 2 find a matching x-bin where it belongs:
for helix_2_x, helix_2_y in zip(helix_2_x_coord_list, helix_2_y_coord_list):
    for x_bin in x_bin_value_list:

```

693

695

697

699  
393

701

703

705

707

709

711

713

```

715 if helix_2_x > (x_bin - (bin_separation/2.0)) and helix_2_x <=
      (x_bin + (bin_separation/2.0)):
716     #if the above condition is satisfied then we have found the
717     appropriate x-bin for the x coordinate and we need to iterate
718     through the y bins to see where the y coordinate fits:
719     for y_bin in y_bin_value_list: #the x,y coordinates
720         should match because lists are ordered
721         if helix_2_y > (y_bin - (bin_separation/2.0))
722             and helix_2_y <= (y_bin + (bin_separation
723                 /2.0)):
724             #every time the helix 2 (x,y)
725             coordinate falls in a 2D bin, print
726             the x,y bin values in a tuple and
727             append to a list:
728             list_of_bin_counts.append((x_bin,y_bin)
729                 )
730
731 #the x and y bin lists should be sorted by default so it should be sensible to
732 loop through each and produce a gnuplot-ready probability for each 2d bin
733
734 for x_bin in x_bin_value_list:
735     for y_bin in y_bin_value_list:
736         outfile.write(str(x_bin) + ' ' + str(y_bin) + ' ' + str(float(
737             list_of_bin_counts.count((x_bin,y_bin)))/float(len(
738                 helix_2_x_coord_list))) + '\n')
739         #we want an empty line between the blocks of (x-bin) data for gnuplot:
740         outfile.write('\n')
741     outfile.close()

```

```

731 def track_bilayer_thickness(folder_name, universe_object, skip_frames,
dimer_symlink_directory, outfile):
    '''This function should track the thickness of the POPC bilayer over the
simulation by measuring the difference in the Z coordinate for the two
geometric centers of phosphate group populations in each of the leaflets.
This avoids one problem with measuring the 'distance scalar' if there are
distortions in one of the leaflets causing an angled (and therefore arguably
spuriously long) distance line between centers. For now delta Z should do. If
we come up with a more rigorous metric or decide that we want a measure that
is 'local' to the peptides we can make those adjustments later.'''
#because we are dealing exclusively with POPC phosphate groups here, the system
(FGFR3 or GpA) won't matter and both will have output files with the same
name

733 import MDAnalysis.analysis.leaflet #because this is not imported by default at
the top level; basically it has dependencies that some users might not have,
but I want to use this module to simplify the leaflet selection process

outfile = open(outfile, 'w')
outfile.write('#column format: frame # || bilayer thickness (Angstroms)\n')

737 #loop through the trajectory frames:
for ts in universe_object.trajectory[::skip_frames]:
    #the LeafletFinder() method will divide the selection into leaflets
741 leaflets = MDAnalysis.analysis.leaflet.LeadletFinder(universe_object,
    rename POP* and name P*)

743 leaflet_1_phosphate_selection = leaflets.atoms(0) #atom selection
object for first leaflet

```

```

747 leaflet_2_phosphate_selection = leaflets.atoms(1) #atom selection
      object for second leaflet

749 #Subtract the Z coordinates of the respective centers of geometry for
      the phosphate populations in each of the bilayers. The absolute value
      of this difference is an estimate of bilayer thickness:
      bilayer_thickness = abs(leaflet_1_phosphate_selection.centerOfGeometry
      ()[2] - leaflet_2_phosphate_selection.centerOfGeometry()[2])

751 outfile.write(str(ts.frame) + ' ' + str(bilayer_thickness) + '\n')
      #track progress of iteration:
      print str(folder_name) + ' -- frame: ' + str(ts.frame)

753
755 outfile.close()

757 def geo_Z_tracking_relative_to_bilayer(folder_name, universe_object, skip_frames,
      system_name, output_file):
      '''Prints the Z coordinate of the geometric center *relative to the center of
      the bilayer (defined as the average of the center of mass of the two leaflet
      phosphate populations)* for each of the GpA or FGFR3 helices (based on '
      system_name' flag) to a specified gnuplot-ready output file for a specified
      frame interval. This function should only be used on trajectories that have
      not been centered by GROMACS trjconv because centering will reduce the
      amplitude of Z coordinate motion for helix 1 relative to helix 2 (i.e., since
      I always center helix 1).'''
      import MDAnalysis.analysis.leaflet
      if system_name == 'FGFR3':
          helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
          monomer

```



```

763 helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
      FGFR3 monomer
765 elif system_name == 'GpA':
      helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
        monomer
      helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
        monomer
      Z_outfile = open(output_file, 'w')
      Z_outfile.write('#column 1: frame # || column 2: helix 1 relative Z coordinate
767 || column 3: helix 2 relative Z coordinate\n')
      for ts in universe_object.trajectory[::skip_frames]:
769 helix_1_CA_center_Z = universe_object.selectAtoms(helix_1_CA_selection)
        .centerOfGeometry()[2]
      helix_2_CA_center_Z = universe_object.selectAtoms(helix_2_CA_selection)
        .centerOfGeometry()[2]
      leaflets = MDAnalysis.analysis.leaflet.LeadletFinder(universe_object, '
771 resname POP* and name P*') #behaves better with phosphate only for
        now; but you could probably get it to work with the full POPC
        molecules by reducing the cutoff from 15 Angstrom to something
        smaller
      leaflet_1_selection = leaflets.atoms(0) #atom selection object for
        first leaflet
      leaflet_2_selection = leaflets.atoms(1) #atom selection object for
        second leaflet
      middle_Z_position_of_bilayer = (leaflet_1_selection.centerOfGeometry()
773 [2] + leaflet_2_selection.centerOfGeometry()[2]) / 2.0
      #We average the Z coordinate of the center of geometry of each leaflet,
        and take the result to be an estimate of the bilayer center from
        which we subtract the Z positions of the helical geometric centers to
        get a relative position.

```

```

777     relative_Zcoordinate_helix_1 = middle_Z_position_of_bilayer -
        helix_1_CA_center_Z
779     relative_Zcoordinate_helix_2 = middle_Z_position_of_bilayer -
        helix_2_CA_center_Z
        #print relative_Zcoordinate_helix_1, relative_Zcoordinate_helix_2
        Z_outfile.write(str(ts.frame) + ' ' + str(relative_Zcoordinate_helix_1)
            + ' ' + str(relative_Zcoordinate_helix_2) + '\n')
        print str(folder_name) + ' -- frame: ' + str(ts.frame)
        Z_outfile.close()

781
783 def local_bilayer_thickness(folder_name, universe_object, skip_frames, system_name,
output_file):
    '''Should track the (interphosphate) thickness of the bilayer in the immediate
    vicinity of the peptide dimer. Still buggy as of August 2/ 2010 because
    bilayer thickness spikes down really low for certain simulations. Algorithm
    problem?'''
    import MDAnalysis.analysis.leaflet
    if system_name == 'FGFR3':
        helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
            monomer
        helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
            FGFR3 monomer
787     elif system_name == 'GpA':
        helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
            monomer
        helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
            monomer
791
        outfile = open(output_file, 'w')
        outfile.write('#column 1: frame # || column 2: local leaflet interphosphate Z
            separation (Angstroms) || column 3: global Z sep. (A) \n')
793

```

```

795 for ts in universe_object.trajectory[:, :skip_frames]:
796     #find the average of the x coordinates of the geometric centers of the
797     two helices (assumes TM peptides and bilayer in xy plane):
798     helix_1_CA_center_X = universe_object.selectAtoms(helix_1_CA_selection)
799     .centerOfGeometry()[0]
800     helix_2_CA_center_X = universe_object.selectAtoms(helix_2_CA_selection)
801     .centerOfGeometry()[0]
802     helix_1_CA_center_Y = universe_object.selectAtoms(helix_1_CA_selection)
803     .centerOfGeometry()[1]
804     helix_2_CA_center_Y = universe_object.selectAtoms(helix_2_CA_selection)
805     .centerOfGeometry()[1]
806     average_helical_center_X = (helix_1_CA_center_X + helix_2_CA_center_X)
807     / 2.0
808     average_helical_center_Y = (helix_1_CA_center_Y + helix_2_CA_center_Y)
809     / 2.0
810     #use MDAnalysis leaflet selection feature to pick phosphates from each
811     of the leaflets, but limit to selecting atoms with x or y coordinates
812     within 7 Angstroms of the average x or y coordinates of the helical
813     geometric centers (i.e., local phosphates):
814     local_selection_string = '( ( prop x <= %s and prop x >= %s ) or ( prop
815     y <= %s and prop y >= %s ) ) and rename POP* and name P*' % (7.0 +
816     average_helical_center_X, average_helical_center_X - 7.0, 7.0 +
817     average_helical_center_Y, average_helical_center_Y - 7.0)
818     local_leaflet_phosphates = MDAnalysis.analysis.leaflet.Leadfinder(
819     universe_object, local_selection_string)
820     leaflet_1_selection = local_leaflet_phosphates.atoms(0) #atom selection
821     object for first leaflet
822     leaflet_2_selection = local_leaflet_phosphates.atoms(1) #atom selection
823     object for second leaflet

```

```

807 #the magnitude of the difference in Z geometric centers of each leaflet
      phosphate population can be used to estimate the local bilayer
      thickness:
      local_interphosphate_Z_separation = abs(leaflet_1_selection.
centerOfGeometry()[2] - leaflet_2_selection.centerOfGeometry()[2])
809 #for comparison print the global bilayer thickness in the third column
      (i.e., to see if the local differs from overall):
      global_leaflet_phosphates = MDAnalysis.analysis.leaflet.leafletFinder(
      universe_object, 'rename POP* and name P*')
811 global_leaflet_1_selection = global_leaflet_phosphates.atoms(0)
      global_leaflet_2_selection = global_leaflet_phosphates.atoms(1)
813 global_interphosphate_Z_separation = abs(global_leaflet_1_selection.
centerOfGeometry()[2] - global_leaflet_2_selection.centerOfGeometry(
[2])

      outfile.write(str(ts.frame) + ' ' + str(
      local_interphosphate_Z_separation) + ' ' + str(
      global_interphosphate_Z_separation) + '\n')
      print str(folder_name) + ' -- frame:' + str(ts.frame)

      outfile.close()

815
817
819 def frame_abstracted_relative_position(folder_name, universe_object, skip_frames,
dimer_symlink_directory, create_universe_selections, system_name, output_file):
      '''Tracks the x and y (center of geometry) position of the second helix in the
      rmsd-fixed frame of the first helix to produce gnuplot-ready data with frame
      # on the Z-axis so you can follow the x and y positions as the simulation
      proceeds from bottom to top of 3D plot. Since the RMSD fixing seems to be
      okay with alpha carbon atom selections, should be able to read in the
      centered CA trajectories (-center and -pbc mol trjconv flags). This function
      is largely for the purpose of obtaining representative structures for the

```

```

different interaction faces based on the thermal plots. For example, with
this new set of 3D plots it should be much easier to track down the frame # (
from Z axis of plot) corresponding to a data (dimer conformation) point of
interest. Will definitely borrow code from other functions in this module.'''
outfile = open(output_file, 'w')
outfile.write('#Format: helix 2 COM x position (A) || helix 2 COM y position (A
) || frame # (starts at 1 for MDA)\n')
list_helix_1_centers = []
list_helix_2_centers = []
#for the reference structure I'm currently planning to use the first frame (and
first helix) of the first WT dimer simulation for either FGFR3 or GpA. To
allow creation of an MDAnalysis Universe and atom selection from either of
these systems the function needs to know which folder the reference
trajectories reside in and which residues to select:
if system_name == 'FGFR3':
    ref_folder_name_list = ['wildtype_dimer_replicate_1']
    helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
    monomer
    helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
    FGFR3 monomer
elif system_name == 'GpA':
    ref_folder_name_list = ['GpA_dimer_replicate_1']
    helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
    monomer
    helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
    monomer

#now create the (universe object, folder_name) list (of length 1) for the
system using a function defined in the head script:

```

821

823

825

401

827

829

831

833

835

---

```

universe_data = create_universe_selections(dimer_symlink_directory,
ref_folder_name_list)
#universe object is the first element in the sublist:
ref_universe_object = universe_data[0][0] #careful here! you had incorrectly
reassigned universe_object which is very bad
#select CA particles corresponding to the reference structure:
reference_CA_selection = ref_universe_object.selectAtoms(helix_1_CA_selection)
#get the reference (center of geometry zero'd) coordinates:
ref_coordinates = reference_CA_selection.coordinates() - reference_CA_selection
.centerOfGeometry()
#need masses for input to rms/transformation function:
masses = reference_CA_selection.masses()

#So, we now have the reference structure with center of geometry at (0,0,0). We
are ready to start looping through frames in the trajectory and doing work
there:
for ts in universe_object.trajectory[::skip_frames]:
#Because helix 1 may move a bit we need to select it again in each
frame. We will also want to select helix 2 because we are going to
track its relative position:
current_helix_1_CA_selection = universe_object.selectAtoms(
helix_1_CA_selection)
current_helix_2_CA_selection = universe_object.selectAtoms(
helix_2_CA_selection)
#Find the (center of geometry zero'd) coordinates for helix 1 in each
frame. To keep things consistent in relative terms, perform the same
operation on helix 2 (i.e., translate it by the same amount):
current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
() - current_helix_1_CA_selection.centerOfGeometry()

```

---

837

839

841

843

845

847

849

851

```

current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
() - current_helix_1_CA_selection.centerOfGeometry()

#We need to figure out what kind of transformation is needed for the
best (rmsd) fit of current helix 1 back to the reference structure.
This is based on the MDAnalysis/examples/rmsfit/rmsfit.py example code
distributed with the most recent release of MDAnalysis:
transformation = numpy.matrix(MDAnalysis.core.rms_fitting.
rms_rotation_matrix(current_helix_1_coordinates,ref_coordinates,
masses))

#Apply the transformation that puts helix 1 in the best (rmsd fit)
match to the reference:
helix_1_best_fit_coordinates = current_helix_1_coordinates *
transformation

#We want to perform the same transformation on helix 2 so that we
maintain consistency relative to the reference structure in each
frame (both helices must always experience the same transformation):
helix_2_updated_coordinates = current_helix_2_coordinates *
transformation

#After the transformations, both helices are represented as numpy
matrices of coordinates. Since the CA particles all have the same
mass, we can treat the mass as 1 and simply find the average position
of all particles to represent the center of mass for each helix:
#calc com helix 1: [We can probably ignore helix 1 when we're sure
things are working properly]
x_sum=0; y_sum=0; z_sum=0
for x,y,z in helix_1_best_fit_coordinates.tolist(): #convert numpy
matrix object to list for std indexing methods

```

853  
855  
857  
859  
861  
863  
865

```

867         x_sum += x; y_sum += y; z_sum += z
869         center_of_mass_helix_1 = [x_sum/len(helix_1_best_fit_coordinates), y_sum
871         /len(helix_1_best_fit_coordinates), z_sum/len(
873         helix_1_best_fit_coordinates)]
875         #repeat for helix 2:
877         x_sum=0; y_sum=0; z_sum=0
879         for x,y,z in helix_2_updated_coordinates.tolist():
881             x_sum += x; y_sum += y; z_sum += z
883             center_of_mass_helix_2 = [x_sum/len(helix_2_updated_coordinates), y_sum/
885             len(helix_2_updated_coordinates), z_sum/len(
887             helix_2_updated_coordinates)]
889             outfile.write(str(center_of_mass_helix_2[0]) + ' ' + str(
891             center_of_mass_helix_2[1]) + ' ' + str(ts.frame) + '\n')
893             print str(folder_name) + ' -- frame: ' + str(ts.frame)
895             print center_of_mass_helix_1[0], center_of_mass_helix_1[1] # to make
897             sure helix 1 properly centered at 0,0 for plotting purposes
899
901     def closest_contacts_efficient(folder_name, universe_object, skip_frames, system_name,
902     output_file):
903         '''A generalized closest-contacts between helices script for CA-CA distances
904         per frame. Uses the fast algorithm built-in to MDAnalysis which is coded in C
905         . Using this to generate closest contacts data that doesn't skip frames so I
906         can more easily use the resulting data files to plot a scatter along with
907         helix crossing angle (for which I have every 25th frame starting at some
908         specific frame depending on simulation). Originally I only had every 10th
909         frame for closest contacts, but that would be a problem for retrieving data
910         on an every 25th frame basis to match with helix crossing. (August 7/2010)'''
911         outfile = open(output_file, 'w')
912         outfile.write('#format: column 1: frame number || column 2: closest approach
913         between CA particles of each helix (Angstroms)\n')

```



```

883 if system_name == 'FGFR3':
884     helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
885         monomer
886     helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
887         FGFR3 monomer
888     elif system_name == 'GpA':
889         helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
890             monomer
891         helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
892             monomer
893     for ts in universe_object.trajectory[::skip_frames]:
894         current_helix_1_CA_selection = universe_object.selectAtoms(
895             helix_1_CA_selection) #select CA particles in first helix
896         current_helix_2_CA_selection = universe_object.selectAtoms(
897             helix_2_CA_selection) #select CA particles in second helix
898         current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
899             () #get a numpy array with helix 1 CA coordinates
900         current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
901             () #get a numpy array with helix 2 CA coordinates
902         #now use the built-in MDAnalysis C-implemented function to quickly
903             generate a list of all (CA-CA) distance between helices
904         contact_distances = MDAnalysis.distances.distance_array(
905             current_helix_1_coordinates, current_helix_2_coordinates)
906         #convert the result from a numpy array to a python list:
907         contact_distances = contact_distances.tolist()
908         #the resulting list is a nested list of the n contact distances for any
909             given residue on the first selection with any element of the second
910             selection
911         #so, find the closest contact for each of the residues from the first
912             selection:

```

```

899 closest_contacts_by_residue = map(min, contact_distances)
900 #now find the overall closest contact between helices in a given frame
901     by finding the minimum of the above list:
902 closest_contact_overall = min(closest_contacts_by_residue)
903
904     print str(folder_name) + ' -- frame: ' + str(ts.frame)
905     outfile.write(str(ts.frame) + ' ' + str(closest_contact_overall) + '\n'
906                 )
907     outfile.close()
908
909 def distance_versus_crossing_angle():
910     '''This function doesn't actually call MDAnalysis directly, it simply grabs the
911     appropriate matching data from the 'closest_contacts_full.out' and 'done.xvg
912     ' files to merge into a single file that can be plotted with closest CA-CA
913     approach between helices on y axis and the corresponding helix crossing angle
914     for that frame on the x axis. Based on the plot seen in Khairul's seminar. (
915     August 7/ 2010)'''
916     #note that the parent/controlling script deals with changing into and out of
917     folders as the function finishes so I can just open/write files directly
918     without a path
919
920     #The helix crossing script is a bit of a black box because it is in binary
921     format and there is no available source that I know of. The script clearly
922     only starts producing crossing angles when certain interhelix proximity
923     conditions are met, but I don't know what those conditions are. This,
924     combined with the fact that the data for crossing angle corresponds to every
925     25th frame (once it starts calculating) means that I will have to match this
926     back intelligently to the closest contact data for them to correspond
927     correctly.

```

```

913 #Open the helix crossing file for reading:
914 helix_crossing_file = open('done.svg', 'r')
915 list_of_frame_angle_pairs = []
916 for line in helix_crossing_file:
917     frame_number = (int(line.split()[0])) * 25 #because every 25th frame
           parsed
918     helix_crossing_angle = float(line.split()[1])
919     list_of_frame_angle_pairs.append([frame_number, helix_crossing_angle])
920     helix_crossing_file.close()
921 #so now we have a nested list of [ [frame #, helix crossing angle], [...],
           [...], etc. ]
922
923 #Open the closest contacts file (has data for every frame):
924 closest_contacts_file = open('closest_contacts_full.out', 'r')
925 list_of_frame_closest_contact_pairs = []
926 for line in closest_contacts_file:
927     if line.split()[0] == '#format:': continue #skip the first line which
           is a comment that starts with this string
928     frame_number = int(line.split()[0])
929     closest_contact = float(line.split()[1])
930     list_of_frame_closest_contact_pairs.append([frame_number,
           closest_contact])
931 #so now we have a nested list of [ [frame #, closest contact distance], [...],
           [...], etc. ]
932
933 outfile = open('distance_vs_crossing_angle.out', 'w')
934 outfile.write('#format: column 1: helix crossing angle (degrees) || column 2:
           closest CA-CA interhelical contact (A) \n')
935 #the 'limiting' information is the crossing angle stuff which starts at some
           strict frame number that depends on interhelix distance and only runs every

```

```

25th frame, so use the helix crossing list as the 'searcher':

for helix_crossing_frame_number, helix_crossing_angle in
list_of_frame_angle_pairs:
    for closest_contact_frame_number, closest_contact_distance in
list_of_frame_closest_contact_pairs:
        if helix_crossing_frame_number == closest_contact_frame_number:
            outfile.write(str(helix_crossing_angle) + ' ' + str(
                closest_contact_distance) + '\n')
        print helix_crossing_frame_number,
            closest_contact_frame_number

def cartesian_to_polar_theta():
    '''Scrape through the files in the data directory that contain x and y
coordinates for the position of helix 2 in the rmsd-fixed frame of helix 1
and convert to a gnuplot-ready file with polar theta and frame # in data
columns. The input file in each subfolder is named 'frame_correlated_position
.out' and the overhead with changing directories is dealt with by the parent
script, so just have to parse the data, do the work, and output it in the
correct format in a file for each dimer simulation directory.'''
    #open the file that contains relevant helix 2 position data
    #the first line of this file reads: #Format: helix 2 COM x position (A) ||
    helix 2 COM y position (A) || frame # (starts at 1 for MDA)

with open('frame_correlated_position.out', 'r') as cartesian_file: #this wasy
you won't need an explicit file.close() at the end
    cartesian_coordinate_frame_list = []
    for line in cartesian_file:

```

937

939

941

943

945

947

949

951

```

953     if line.split()[0] == '#Format:': continue #skip the first
954     line which starts with the string '#Format:' as the first
955     word
956     x_cartesian = float(line.split()[0])
957     y_cartesian = float(line.split()[1])
958     frame = int(line.split()[2])
959     cartesian_coordinate_frame_list.append([x_cartesian, y_cartesian
960     ,frame])
961
962 #so, now we a nested list with format: [ [x_cartesian, y_cartesian, frame],
963     [...], [...] ]
964
965 the corresponding polar coordinate angle theta:
966 polar_list = []
967
968 for x,y,frame in cartesian_coordinate_frame_list:
969     polar_theta = math.atan2(y,x)
970     polar_list.append([frame, polar_theta])
971
972 #write the polar data to a gnuplot-ready output file:
973 with open('polar_interface_theta.out', 'w') as polar_file:
974     polar_file.write('#Format: column 1: frame # || column 2: polar theta
975     for helix 2 in rmsd-fixed frame of helix 1 (radians) \n')
976     for element in polar_list:
977         print element[0], element[1] #just so I can see it running at
978         the terminal
979         polar_file.write(str(element[0]) + ' ' + str(element[1]) + '\n'
980         )
981
982 def correlate_helixcrossing_polar_theta():

```

```

'''This function should take the helix crossing data (every 25th frame) in the
'done.xvg' files and the polar theta data calculated for every frame in the '
frame_correlated_position_noskip.out' files, determine which data points
match by frame, and output helix crossing angle and the corresponding polar
theta value (calculated from the certesians) to a new gnuplot-ready file.
Directory switching overhead is dealt with in parent script so just parse the
files and write the output file in this function.'''
#put the helix crossing data in a list:
helix_cross_list = []
with open('done.xvg','r') as helix_crossing_file:
    for line in helix_crossing_file:
        frame_number = (int(line.split()[0])) * 25 #because every 25th
            frame parsed by helix crossing script
        helix_crossing_angle = float(line.split()[1])
        helix_cross_list.append([frame_number, helix_crossing_angle])
#helix_cross_list is now a nested list: [ [frame_number, crossing angle],
    [...], [...], etc. ]

#there is a comment that starts with a hash on the first line of the frame
correlated position file so adjust for that when parsing
theta_list = []
with open('frame_correlated_position_noskip.out','r') as position_file:
    for line in position_file:
        if line.split()[0][0] == '#': continue #skip the first line of
            the file which contains a comment
        x_coordinate = float(line.split()[0])
        y_coordinate = float(line.split()[1])
        frame_number = int(line.split()[2])
        polar_theta = math.atan2(y_coordinate, x_coordinate)
        theta_list.append([frame_number, polar_theta])

```

973

975

977

979

410

981

983

985

987

989

991

```

993 #theta_list is now a nested list: [ [frame_number, polar theta], [...],
994 [...], etc. ]
995
996 #iterate through the lists and pick out frame-matched data
997 matched_data_list = []
998 #for helix_frame_number, helix_crossing_angle in helix_cross_list:
999     #for polar_frame_number, polar_theta in theta_list:
1000         #if helix_frame_number == polar_frame_number:
1001             matched_data_list.append([helix_frame_number,
1002                                     helix_crossing_angle, polar_theta])
1003
1004 #make an output file and write the data there
1005 with open('theta_helix_crossing_correlate.out','w') as outfile:
1006     #for element in matched_data_list:
1007         #print element[0], element[1], element[2]
1008         outfile.write(str(element[0]) + ' ' + str(element[1]) + ' ' +
1009                       str(element[2]) + '\n')
1009
1010 def closest_approach_representative(folder_name, universe_object, skip_frames,
1011 system_name):
1012     '''I have a set of pdb structures (frames) from a few simulations that I'd like
1013     to compile a list of CA-CA contact distances for. These frames are
1014     candidates for the starting points of atomistic simulations and are intended
1015     to represent different GpA or FGFR3 helix dimer interfaces. Very minor
1016     modification to a previous function that deals with finding the single
1017     closest approach distance.'''
1018     with open('FGFR3_heterodimer_replicate_7_frame_3271.out','w') as outfile:
1019         outfile.write('#format: residue # || closest contact by residue helix 1
1020                       (Angstroms) || closest contact by residue helix 2 (Angstroms)\n')

```

```

1013 if system_name == 'FGFR3':
1015     helix_1_CA_selection = "( name CA and resid 1:33 )" #select
        first FGFR3 monomer
1017     helix_2_CA_selection = "( name CA and resid 34:66 )" #select
        second FGFR3 monomer
        native_adjustment = 366 #this number is used to generate the
        native primary sequence numbering in the output
        elif system_name == 'GpA':
            helix_1_CA_selection = "( name CA and resid 1:23 )" #select
                first GpA monomer
            helix_2_CA_selection = "( name CA and resid 24:46 )" #select
                second GpA monomer
            native_adjustment = 72 #this number is used to generate the
            native primary sequence numbering in the output
        for ts in universe_object.trajectory[3270:3271:skip_frames]: #adjust
            this for the specific frame/pdb file (MDA index is -1 relative to
            true frame)
            current_helix_1_CA_selection = universe_object.selectAtoms(
                helix_1_CA_selection) #select CA particles in first helix
            current_helix_2_CA_selection = universe_object.selectAtoms(
                helix_2_CA_selection) #select CA particles in second helix
            current_helix_1_coordinates = current_helix_1_CA_selection.
                coordinates() #get a numpy array with helix 1 CA coordinates
            current_helix_2_coordinates = current_helix_2_CA_selection.
                coordinates() #get a numpy array with helix 2 CA coordinates
            #now use the built-in MDAnalysis C-implemented function to
                quickly generate a list of all (CA-CA) distance between
                helices
            contact_distances_helix_1 = MDAnalysis.distances.distance_array
                (current_helix_1_coordinates, current_helix_2_coordinates)

```



```

1027 contact_distances_helix_2 = MDAnalysis.distances.distance_array
      (current_helix_2_coordinates, current_helix_1_coordinates)
1029 #convert the results from a numpy array to a python list:
      contact_distances_helix_1 = contact_distances_helix_1.tolist()
      contact_distances_helix_2 = contact_distances_helix_2.tolist()
1031 #the resulting list is a nested list of the n contact distances
      for any given residue on the first selection with any
      element of the second selection
      #so, find the closest contact for each of the residues from the
      first selection (in each case):
1033 closest_contacts_by_residue_helix_1 = map(min,
      contact_distances_helix_1)
      closest_contacts_by_residue_helix_2 = map(min,
      contact_distances_helix_2)
1035 #in this case that's all I want--they should be ordered from
      first to last residue for *each* of the helices in the dimer
1037 for i in range (0,len(closest_contacts_by_residue_helix_1)):
      outfile.write(str(i+1+native_adjustment) + ' ' + str(
          closest_contacts_by_residue_helix_1[i]) + ' ' + str(
          closest_contacts_by_residue_helix_2[i]) + '\n')
1039 print str(folder_name) + ' -- frame: ' + str(ts.frame)

def absolute_value_Z_tracking():
    '''Read in the 'relative_Z.out' file in each directory and write out a new file
    that contains the absolute value of the difference in the Z coordinate of
    the geometric center of each helix.'''
    with open('absolute_relative_Z.out','w') as output_file:
        with open('relative_Z.out','r') as input_file:
            for line in input_file:

```

```

1045 if line.split()[0][0] == '#': continue #skip line if
      it is a comment
1047 frame_number = line.split()[0]
      helix_1_Z_coord = float(line.split()[1])
      helix_2_Z_coord = float(line.split()[2])
1049 delta_Z_absolute = abs(helix_1_Z_coord -
      helix_2_Z_coord)
      output_file.write(str(frame_number) + ' ' + str(
      delta_Z_absolute) + '\n')
1051 print frame_number,delta_Z_absolute

1053 def absolute_delta_Z_and_closest_approach():
      '''Take the contents of the 'closest_contacts.out' and 'absolute_relative_Z.out
      ' files and put frame #, closest contacts, and absolute difference between
      helical geometric center Z coordinates, into a single gnuplot-ready file.'''

      with open('correlate_Z_closest_approach.out','w') as output_file:
1057         output_file.write('#format: frame # || closest contact (A) || absolute
            difference in helical geo center Z coordinate (A)\n')
            with open('closest_contacts.out','r') as contacts_file:
1059                 frame_contact_list = []
                    for line in contacts_file:
1061                         if line.split()[0][0] == '#': continue #skip line if
                                it is a comment
                                    frame = int(line.split()[0])
1063                                     closest_contact = float(line.split()[1])
                                        frame_contact_list.append([frame, closest_contact])

1065 with open('absolute_relative_Z.out','r') as Z_file:
        Z_list = []
        for line in Z_file:
1067

```

```

1069         if line.split()[0][0] == '#': continue #skip line if
1070             it is a comment
1071             absolute_delta_Z = float(line.split()[1])
1072             Z_list.append(absolute_delta_Z)
1073
1074     for frame_and_contact_list, Z_value in zip(frame_contact_list, Z_list):
1075         output_file.write(str(frame_and_contact_list[0]) + ' ' + str(
1076             frame_and_contact_list[1]) + ' ' + str(Z_value) + '\n')
1077         print frame_and_contact_list[0], frame_and_contact_list[1],
1078             Z_value
1079
1080 def split_Z_file(folder_name):
1081     '''This function should read in the 'relative_Z.out' file and split it into two
1082         new files, one pre- and the other post- dimerization, based on the frame at
1083         which dimerization starts (defined as a closest-contact distance less than 6
1084         Angstroms in the file 'closest_contacts.out').'''
1085
1086     #determine the point at which the dimer forms (defined as closest-contact
1087         distance under 6 Angstroms)
1088     with open('closest_contacts.out', 'r') as contacts_file:
1089         for line in contacts_file:
1090             if line.split()[0][0] == '#': continue #skip line if it is a
1091                 comment
1092             if float(line.split()[1]) < 6.0:
1093                 dimer_start_frame = int(line.split()[0]) #assign the
1094                     dimer start frame when the distance is correct
1095                 break
1096
1097     print 'folder:', folder_name, '; dimer start frame #:', dimer_start_frame
1098         #print a progress check

```

```
1087 #make some lists for storing the helix 1 and helix 2 values for pre- and post-
1089 dimerization so I can get a correlation coefficient for each simulation
1091 before and after dimerization:
pre_dimer_helix_1_Z_list = []
pre_dimer_helix_2_Z_list = []
post_dimer_helix_1_Z_list = []
post_dimer_helix_2_Z_list = []

1093 with open('relative_Z.out', 'r') as Z_file:
1095     with open('post_dimer_Z_correlation.out', 'w') as post_dimer_Z_file:
        with open('pre_dimer_Z_correlation.out', 'w') as
            pre_dimer_Z_file:

1097         #write the new files based on frame # relative to when
1099         dimerization starts:
        for line in Z_file:
            if line.split()[0][0] == '#': continue #skip
                line if it is a comment

1101         if int(line.split()[0]) >= dimer_start_frame: #
            if the system has reached or exceeded the
                frame where dimerization occurs
                    post_dimer_Z_file.write(line)
            #also split the line and dump the helix
                1 and helix 2 Z values to lists for
                    usage later in correlation
            post_dimer_helix_1_Z_list.append(line.
                split()[1])

1103
1105
```

```

1107     post_dimer_helix_2_Z_list.append(line.
1108         split()[2])
1109     else: #if the system hasn't dimerized yet (
1110         farther than 6 A apart)
1111         pre_dimer_Z_file.write(line)
1112         pre_dimer_helix_1_Z_list.append(line.
1113             split()[1])
1114         pre_dimer_helix_2_Z_list.append(line.
1115             split()[2])
1116
1117     #calculate correlation coefficients using numpy:
1118     pre_dimer_R_value = numpy.corrcoef(numpy.array(
1119         pre_dimer_helix_1_Z_list), numpy.array(
1120         pre_dimer_helix_2_Z_list))
1121     post_dimer_R_value = numpy.corrcoef(numpy.array(
1122         post_dimer_helix_1_Z_list), numpy.array(
1123         post_dimer_helix_2_Z_list))
1124
1125     #convert numpy arrays back to lists and grab the
1126     appropriate R values by indexing:
1127     print 'Pre-dimer R:', pre_dimer_R_value.tolist()[1][0]
1128     print 'Post-dimer R:', post_dimer_R_value.tolist()[1][0]
1129     #write the values to the end of the appropriate files:
1130     post_dimer_Z_file.write('R_value ' + str(
1131         post_dimer_R_value.tolist()[1][0]))
1132     pre_dimer_Z_file.write('R_value ' + str(
1133         pre_dimer_R_value.tolist()[1][0]))
1134
1135     def box_size_assessment(folder_name, universe_object, skip_frames):
1136         '''For getting an idea of simulation box size as the simulation proceeds. At
1137         the moment, simply iterate through the trajectory and print out unit cell

```

```

1125         dimensions.'''
1126     with open('box_sizes.out', 'w') as outfile:
1127         for ts in universe_object.trajectory[::25]:
1128             outfile.write(str(ts.frame) + ' ' + str(universe_object.
1129                 trajectory.ts) + '\n')
1130             print folder_name, ts.frame
1131
1132     def closest_contacts_efficient_SIDEKICK(folder_name, universe_object, skip_frames,
1133         system_name, output_file):
1134         '''August 29/ 2010: Modified version of the similarly named function. Basically
1135             the only difference here is that I have designed this function to write the
1136             data in the separate local folder for FGFR3 SIDEKICK results because I can't
1137             write files to the mount where the large number of replicate data folders are
1138             stored (at least not remotely when executing this from home).'''
1139         output_file_path = '/sansom/sc2/bioc1009/Documents/FGFR3_work/
1140             sidekick_dimer_batch_analysis/closest_contact_results/' + folder_name.replace
1141             ('/', '') + output_file #this should make the file names unique because the
1142             '/'-stripped string of the full path on the mount (folder name) precedes the
1143             generic output_file name
1144         outfile = open(output_file_path, 'w')
1145         outfile.write('#format: column 1: frame number || column 2: closest approach
1146             between CA particles of each helix (Angstroms)\n')
1147         if system_name == 'FGFR3':
1148             helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
1149                 monomer
1150             helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
1151                 FGFR3 monomer
1152         elif system_name == 'GpA':
1153             helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
1154                 monomer

```

```

1139 helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
1140     monomer
1141 for ts in universe_object.trajectory[::skip_frames]:
1142     current_helix_1_CA_selection = universe_object.selectAtoms(
1143         helix_1_CA_selection) #select CA particles in first helix
1144     current_helix_2_CA_selection = universe_object.selectAtoms(
1145         helix_2_CA_selection) #select CA particles in second helix
1146     current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
1147     () #get a numpy array with helix 1 CA coordinates
1148     current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
1149     () #get a numpy array with helix 2 CA coordinates
1150 #now use the built-in MDAnalysis C-implemented function to quickly
1151     generate a list of all (CA-CA) distance between helices
1152     contact_distances = MDAnalysis.distances.distance_array(
1153         current_helix_1_coordinates, current_helix_2_coordinates)
1154 #convert the result from a numpy array to a python list:
1155     contact_distances = contact_distances.tolist()
1156 #the resulting list is a nested list of the n contact distances for any
1157     given residue on the first selection with any element of the second
1158     selection
1159 #so, find the closest contact for each of the residues from the first
1160     selection:
1161     closest_contacts_by_residue = map(min, contact_distances)
1162 #now find the overall closest contact between helices in a given frame
1163     by finding the minimum of the above list:
1164     closest_contact_overall = min(closest_contacts_by_residue)
1165     print str(folder_name) + ' -- frame: ' + str(ts.frame)
1166     outfile.write(str(ts.frame) + ' ' + str(closest_contact_overall) + '\n'
1167 )

```

```

1157         outfile.close()
1159     def fixed_helix_thermal_merged_SIDEKICK(path_name, universe_object, skip_frames,
heterodimer_helix_2_x_coord_list, heterodimer_helix_2_y_coord_list,
mutant_helix_2_x_coord_list, mutant_helix_2_y_coord_list,
wildtype_helix_2_x_coord_list, wildtype_helix_2_y_coord_list):
'''Modifying this function again to deal with the SIDEKICK results for FGFR3 in
DPPC.'''

1161     list_helix_2_centers = []
1163     helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3 monomer
helix_2_CA_selection = "( name CA and resid 34:66 )" #select second FGFR3
monomer

1165     #The reference structure is the first helix in the first frame of the first WT
simulation. This is actually from a manual POPC simulation and not from the
SIDEKICK DPPC data directory on the mount, so provide an absolute path when
creating the reference universe object:
ref_universe_object = MDAnalysis.Universe('/sansom/sc2/bioc1009/Documents/
FGFR3_work/dimer_batch_analysis_symlink/wildtype_dimer_replicate_1/
centered_CA_POPC.pdb', '/sansom/sc2/bioc1009/Documents/FGFR3_work/
dimer_batch_analysis_symlink/wildtype_dimer_replicate_1/
alpha_carbon_popc_and_centered_trajectory.xtc')
1167     #select CA particles corresponding to the reference structure:
reference_CA_selection = ref_universe_object.selectAtoms(helix_1_CA_selection)
#get the reference (center of geometry zero'd) coordinates:
ref_coordinates = reference_CA_selection.coordinates() - reference_CA_selection
.centerOfGeometry()
1169     #need masses for input to rms/transformation function:
masses = reference_CA_selection.masses()
1171
1173

```



#So, we now have the reference structure with center of geometry at (0,0,0). We are ready to start looping through frames in the trajectory and doing work there:

```

for ts in universe_object.trajectory[:, :skip_frames]:
    #Because helix 1 may move a bit we need to select it again in each
    frame. We will also want to select helix 2 because we are going to
    track its relative position:
    current_helix_1_CA_selection = universe_object.selectAtoms(
        helix_1_CA_selection)
    current_helix_2_CA_selection = universe_object.selectAtoms(
        helix_2_CA_selection)
    #Find the (center of geometry zero'd) coordinates for helix 1 in each
    frame. To keep things consistent in relative terms, perform the same
    operation on helix 2 (i.e., translate it by the same amount):
    current_helix_1_coordinates = current_helix_1_CA_selection.coordinates
    () - current_helix_1_CA_selection.centerOfGeometry()
    current_helix_2_coordinates = current_helix_2_CA_selection.coordinates
    () - current_helix_1_CA_selection.centerOfGeometry()

```

#We need to figure out what kind of transformation is needed for the best (rmsd) fit of current helix 1 back to the reference structure. This is based on the MDAnalysis/examples/rmsfit/rmsfit.py example code distributed with the most recent release of MDAnalysis:
  
transformation = numpy.matrix(MDAnalysis.core.rms\_fitting.
rms\_rotation\_matrix(current\_helix\_1\_coordinates, ref\_coordinates,
masses))

```

1189 #We don't actually need to apply the transformation to helix 1--just
1190 want to know where helix 2 is relative to 1 now that 1 is fixed:
1191 helix_2_updated_coordinates = current_helix_2_coordinates *
1192 transformation

1193 #Helix 2 is represented as a numpy matrix of coordinates. Since the CA
1194 particles all have the same mass, we can treat the mass as 1 and
1195 simply find the average position of all particles to represent the
1196 center of mass.
1197 x_sum=0; y_sum=0; z_sum=0
1198 for x,y,z in helix_2_updated_coordinates.tolist():
1199     x_sum += x; y_sum += y; z_sum += z
1200 center_of_mass_helix_2 = [x_sum/len(helix_2_updated_coordinates),y_sum/
1201 len(helix_2_updated_coordinates),z_sum/len(
1202 helix_2_updated_coordinates)]
1203 #put the centers in a list so they can be accessed by the code that
1204 will bin the data after the frame-stepping loop exits
1205 list_helix_2_centers.append(center_of_mass_helix_2)
1206 print str(path_name) + ' -- frame: ' + str(ts.frame)

1207 #we want to extract the x and y coordinates for helix 2 such that they are
1208 filtered by simulation type (i.e., folder name) and append them to an
1209 appropriately named list (these lists are initialized in the head script so
1210 they can be appended as the function loops through)
1211 if 'RRAGSVYAGILSYGVGFFLFILVVAAVTLCRLR' in path_name:
1212     RRAGSVYAGILSYRVGFFLFILVVAAVTLCRLR_ in list_helix_2_centers:
1213         for (x,y,z) in list_helix_2_centers:
1214             heterodimer_helix_2_x_coord_list.append(x)
1215             heterodimer_helix_2_y_coord_list.append(y)

```

```

1205 elif 'RRAGSVYAGILSYRVGFFLFIILVVAAVTLCRLR' in path_name:
1206     RRAGSVYAGILSYRVGFFLFIILVVAAVTLCRLR
1207     for (x,y,z) in list_helix_2-centers:
1208         mutant_helix_2_x_coord_list.append(x)
1209         mutant_helix_2_y_coord_list.append(y)
1210     elif 'RRAGSVYAGILSYGVGFFLFIILVVAAVTLCRLR' in path_name:
1211         RRAGSVYAGILSYGVGFFLFIILVVAAVTLCRLR
1212         for (x,y,z) in list_helix_2-centers:
1213             wildtype_helix_2_x_coord_list.append(x)
1214             wildtype_helix_2_y_coord_list.append(y)

```

#so, after going through all 286 FGFR3 SIDEKICK dimer simulations, each of the lists will contain the appropriate set of coordinates from 10 simulations. For binning and probability calculation it will be worthwhile to generalize the portion of the code that deals with that in a function and then call it after this function from the head script

```

1215 def thermal_bins_SIDEKICK(helix_2_x_coord_list, helix_2_y_coord_list, outfile_path):
1216     '''Parses the FGFR3 helix 2 coordinate list data from the
1217     fixed_helix_thermal_merged() function and writes the binned probability data
1218     to the appropriate file. Meant to work with the *overall* FGFR3 'thermal'
1219     position data, so the lists contain coordinates from all replicates in each
1220     of the three simulation conditions. Call this function after
1221     fixed_helix_thermal_merged() (and outside of main) in the head script. Note
1222     that the non-merged function is different in that it contains its own binning
1223     routines so you don't have to call this.'''
1224
1225     outfile = open(outfile_path, 'w')
1226     #set a bin separation and build bins around min and max values
1227     bin_separation = 0.4

```

---

```

1223 list_of_bin_counts = []
1224 #populate the bin value lists based on the bounds of data
1225 x = min(helix_2_x_coord_list)
1226 x_bin_value_list = []
1227 while x <= max(helix_2_x_coord_list):
1228     x_bin_value_list.append(x)
1229     x += bin_separation
1230 #
1231 y = min(helix_2_y_coord_list)
1232 y_bin_value_list = []
1233 while y <= max(helix_2_y_coord_list):
1234     y_bin_value_list.append(y)
1235     y += bin_separation
1236
1237 #for every x coordinate of helix 2 find a matching x-bin where it belongs:
1238 for helix_2_x, helix_2_y in zip(helix_2_x_coord_list, helix_2_y_coord_list):
1239     for x_bin in x_bin_value_list:
1240         if helix_2_x > (x_bin - (bin_separation/2.0)) and helix_2_x <=
1241             (x_bin + (bin_separation/2.0)):
1242             #if the above condition is satisfied then we have found the
1243             appropriate x-bin for the x coordinate and we need to iterate
1244             through the y bins to see where the y coordinate fits:
1245             for y_bin in y_bin_value_list: #the x,y coordinates
1246                 should match because lists are ordered
1247                 if helix_2_y > (y_bin - (bin_separation/2.0))
1248                     and helix_2_y <= (y_bin + (bin_separation
1249                         /2.0)):
1250                     #every time the helix 2 (x,y)
1251                     coordinate falls in a 2D bin, print
1252                     the x,y bin values in a tuple and

```

---

```

1245         append to a list:
1246         list_of_bin_counts.append((x_bin, y_bin)
1247     )
1248
1249 #the x and y bin lists should be sorted by default so it should be sensible to
1250 loop through each and produce a gnuplot-ready probability for each 2d bin
1251
1252 for x_bin in x_bin_value_list:
1253     for y_bin in y_bin_value_list:
1254
1255         outfile.write(str(x_bin) + ' ' + str(y_bin) + ' ' + str(float(
1256             list_of_bin_counts.count((x_bin, y_bin)))/float(len(
1257                 helix_2_x_coord_list))) + '\n')
1258
1259         #we want an empty line between the blocks of (x-bin) data for gnuplot:
1260         outfile.write('\n')
1261
1262     outfile.close()
1263
1264 def optimize_leaflet_selection_cutoff(folder_name, universe_object):
1265     '''Oct. 7/ 2010: Adapting for use with FGFR3 simulations. A distance cutoff is
1266     used by MDanalysis.analysis.leaflet.LeafletFinder() to establish networks of
1267     molecules belonging to a given leaflet. The purpose of this function is to
1268     call MDanalysis.analysis.leaflet.optimize_cutoff(), which is a built-in
1269     MDAnalysis tool for optimizing this cutoff for the selection of two leaflets,
1270     rather than establishing extra networks or having cutoff problems when going
1271     straight to the LeafletFinder() routine.'''
1272     import MDAnalysis.analysis.leaflet
1273     optimize_cutoff = MDAnalysis.analysis.leaflet.leaflet.optimize_cutoff
1274     #the head script that calls this function will automatically switch directories
1275     to pick up new universe objects, so just feed them in to the optimize_cutoff
1276     function along with the necessary parameters for testing leaflet selection

```

```
1261 on a per-replicate basis
1262
1263 #the selection string should pick out all the lipid phosphates in the system:
1264 selection = 'resname POP* and name P*'
1265 #the range of cutoffs to test for establishing leaflet networks between
1266 headgroup phosphates (in Angstroms):
1267 dmin = 8.0
1268 dmax = 20.0
1269 #the step size (in Angstroms) for testing cutoffs:
1270 step = 0.5
1271 print folder_name #so I can track which replicate is being parsed
1272 #call the MDAnalysis function which should indicate the number of leaflets
1273 selected (ideally two obviously) for a given cutoff:
1274 result = optimize_cutoff(universe_object, selection, dmin, dmax, step)
1275 print result #returns tuple: (optimum cutoff, number of leaflets established
1276 with this cutoff)
1277
1278 def analyze_leaflets(folder_name, universe_object, skip_frames, outfile_name,
1279 local_definition, cutoff, system_name):
1280     '''Oct. 7/ 2010: Adapting for use with FGFR3 simulations. Having established a
1281     cutoff for leaflet selection from the MDAnalysis.analysis.leaflet.
1282     optimize_cutoff function I could try to feed this into the LeafletFinder()
1283     class for every frame of a simulation replicate to refresh the leaflet
1284     selections. However, feeding the changing coordinates to LeafletFinder()
1285     every frame may result in unnatural leaflet definitions and separation
1286     distances as the bilayer distorts (especially if the internetwork cutoff <=
1287     intra-network cutoff at some positions). Instead, I think it is perhaps
1288     better to simply select two phosphate particle groups (one for each leaflet
1289     selected presumably from the first frame of each simulation by LeafletFinder
1290     ()) and use these as absolute leaflet selections which can be tracked for
```

```

1277 various changes without worrying about which particles get selected as the
1278 bilayer distorts. Oct 5/2010: put in various print checks for testing; looks
1279 like I fixed the problem at check 3a and now commenting out the print checks.

1280 local_definition is the allowable distance (in Angstroms) between phosphate
1281 particles and protein CA residues for inclusion in the 'local lipid shell'

1282 the cutoff parameter is for the MDAnalysis network selection of leaflets and
1283 the optimal value varies for WT and hetero/mutant FGFR3 conditions
1284
1285 '''
1286
1287 if system_name == 'FGFR3':
1288     helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
1289     monomer
1290     helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
1291     FGFR3 monomer
1292 elif system_name == 'GpA':
1293     helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
1294     monomer
1295     helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
1296     monomer
1297
1298 with open(outfile_name, 'w') as outfile:
1299     outfile.write('#format: frame | thickness near first monomer |
1300                 thickness near second monomer | distal thickness | (Angstroms)\n')
1301     import MDAnalysis.analysis.leaflet
1302     #use the same selection string used for optimizing the cutoff:
1303     phosphate_selection = 'resname POP* and name P*'
1304     #generate the leaflet selections from the first frame using the built-
1305     in MDAnalysis class LeafletFinder():

```

```

leaflets = MDAnalysis.analysis.leaflet.LleafletFinder(universe_object,
    phosphate_selection, cutoff)
#pull the two leaflet selections out from the object:
bottom_leaflet_phosphates = leaflets.atoms(0) #clearly the 'bottom'
    leaflet in my system using the selection conversion to VMD shown
    below
top_leaflet_phosphates = leaflets.atoms(1) #therefore, this is the 'top
    ' (+Z) set of phosphates (top leaflet)

#Now I want python lists of phosphate residues for these leaflets (.
resids() method) to be converted so they can be parsed by the
MDAnalysis u.selectAtoms() parser. Basically adding in spaces and 'or
' etc.:
bottom_phosphate_list = []
top_phosphate_list = []

#for bottom leaflet:
for phosphate_residue_number in bottom_leaflet_phosphates.resids():
    if bottom_leaflet_phosphates.resids().index(
        phosphate_residue_number) < (len(bottom_leaflet_phosphates.
resids())-1): #for all but the last resid element
        bottom_phosphate_string = 'resid ' + str(
            phosphate_residue_number) + ' or '
    else: #we'll want a right ')' after the last resid element
        bottom_phosphate_string = 'resid ' + str(
            phosphate_residue_number) + ')'
    bottom_phosphate_list.append(bottom_phosphate_string)

#repeat for top leaflet:
for phosphate_residue_number in top_leaflet_phosphates.resids():

```

1295

1297

1299

1301

1303

1305

1307

1309

1311



```

1313 if top_leaflet_phosphates.resids().index(
1314     phosphate_residue_number) < (len(top_leaflet_phosphates.
1315     resids())-1): #for all but the last resid element
1316     top_phosphate_string = 'resid ' + str(
1317         phosphate_residue_number) + ' or '
1318 else: #we'll want a right ')' after the last resid element
1319     top_phosphate_string = 'resid ' + str(
1320         phosphate_residue_number) + ' )'
1321 top_phosphate_list.append(top_phosphate_string)
1322
1323 #assign the formatted selection strings:
1324 bottom_leaflet_phosphate_selection_string = '(' + ''.join(
1325     bottom_phosphate_list) + ' and ( rename POP* and name P* )'
1326 top_leaflet_phosphate_selection_string = '(' + ''.join(
1327     top_phosphate_list) + ' and ( rename POP* and name P* )'
1328
1329 #Now that the 'absolute' leaflet selection strings are ready I will
1330 start the trajectory looping:
1331 for ts in universe_object.trajectory[:,skip_frames]:
1332
1333     #I want to use MDAnalysis to parse for phosphates of each
1334     leaflet within local_definition Angstroms of either protein
1335     molecule.
1336
1337     #selection strings for FGFR3 monomers for TOP leaflet
1338     phosphates within local_definition:
1339     phosph_around_first_monomer_top_leaflet_string = "( around %s %
1340     s )" % (str(local_definition), helix_1_CA_selection) + ' and
1341     ' + top_leaflet_phosphate_selection_string

```

```

phosph_around_second_monomer_top_leaflet_string = "( around %s
%s )" % (str(local_definition), helix_2_CA_selection) + '
and ' + top_leaflet_phosphate_selection_string

#same thing for bottom leaflet phosphates within
local_definition of FGFR3 monomer 1 or 2:
phosph_around_first_monomer_bottom_leaflet_string = "( around %
s %s )" % (str(local_definition), helix_1_CA_selection) + '
and ' + bottom_leaflet_phosphate_selection_string
phosph_around_second_monomer_bottom_leaflet_string = "( around
%s %s )" % (str(local_definition), helix_2_CA_selection) + '
and ' + bottom_leaflet_phosphate_selection_string

#will need separate selection strings for leaflet phosphates
that are NOT within local_definition of either protein:
top_distal_phosphate_string =
top_leaflet_phosphate_selection_string + ' and not ' + "(
around %s ( name CA )" % str(local_definition)
bottom_distal_phosphate_string =
bottom_leaflet_phosphate_selection_string + ' and not ' + "(
around %s ( name CA )" % str(local_definition)

#produce the actual MDAnalysis AtomGroup selections:
phosph_around_first_monomer_top_leaflet = universe_object.
selectAtoms(phosph_around_first_monomer_top_leaflet_string)
phosph_around_second_monomer_top_leaflet = universe_object.
selectAtoms(phosph_around_second_monomer_top_leaflet_string)
phosph_around_first_monomer_bottom_leaflet = universe_object.
selectAtoms(phosph_around_first_monomer_bottom_leaflet_string
)

```

1331

1333

1335

430

1337

1339

1341

1343

```

1345 phosph_around_second_monomer_bottom_leaflet = universe_object.
      selectAtoms(
1346     phosph_around_second_monomer_bottom_leaflet_string)
      top_distal_phosphate = universe_object.selectAtoms(
1347     top_distal_phosphate_string)
      bottom_distal_phosphate = universe_object.selectAtoms(
1348     bottom_distal_phosphate_string)
1349
1350 #Calculate the difference in the Z coordinate (i.e., a measure
      of bilayer thickness) of the leaflet phosphate centers of
1351     geometry for a few different scenarios:
1352
1353 #near first FGFR3 monomer:
      bilayer_Z_thickness_near_first_monomer = abs(
1354     phosph_around_first_monomer_top_leaflet.centerOfGeometry()[2]
1355     - phosph_around_first_monomer_bottom_leaflet.
      centerOfGeometry()[2])
1356 #near second FGFR3 monomer:
      bilayer_Z_thickness_near_second_monomer = abs(
1357     phosph_around_second_monomer_top_leaflet.centerOfGeometry()
1358     [2] - phosph_around_second_monomer_bottom_leaflet.
      centerOfGeometry()[2])
1359 #everywhere else (not within local_definition of either protein
      ):
1360     bilayer_Z_thickness_distal = abs(top_distal_phosphate.
1361     centerOfGeometry()[2] - bottom_distal_phosphate.
1362     centerOfGeometry()[2])
1363 #track progress on the prompt:
1364     print str(folder_name) + ' -- frame: ' + str(ts.frame)

```

```

1359         outfile.write(str(ts.frame) + ' ' + str(
1361             bilayer_Z_thickness_near_first_monomer) + ' ' + str(
                bilayer_Z_thickness_near_second_monomer) + ' ' + str(
                bilayer_Z_thickness_distal) + '\n')

def count_lipids_in_local_shell(folder_name, universe_object, skip_frames, outfile_name
, local_definition, cutoff, system_name):
    '''Oct. 9/2010: Adapting this function for use with GpA and FGFR3 simulations.
    Because White and co-workers (Structure 17, 395-405, 2009) define 'local'
    lipids as the first 2-3 shells (~90 lipids) around rhomboid I want to see how
    many lipids are being used to define 'local' based on the cutoff distance (
    parameter: local_definition) I've used in the analyze_leaflets() function
    above. Could be that my sample size is a bit small. Adjustments made on Oct.
    5/ 2010 to match the repair to the phosphate selection issues in
    analyze_leaflets() function above.

    local_definition is the allowable distance (in Angstroms) between
    phosphate particles and protein CA residues for inclusion in the '
    local lipid shell'
    cutoff argument is for the MDAnalysis network-leaflet generation

    '''
    if system_name == 'FGFR3':
        helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
            monomer
        helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
            FGFR3 monomer
    elif system_name == 'GpA':
        helix_1_CA_selection = "( name CA and resid 1:23 )" #select first GpA
            monomer

```

```

1371 helix_2_CA_selection = "( name CA and resid 24:46 )" #select second GpA
1372     monomer
1373 with open(outfile_name, 'w') as outfile:
1374     outfile.write('#format for lipid resid count: helix1_top |
1375     helix1_bottom | helix2_top | helix2_bottom | distal_top |
1376     distal_bottom\n')
1377 import MDAnalysis.analysis.leaflet
1378 selection = 'resname POP* and name P*'
1379 leaflets = MDAnalysis.analysis.leaflet.LeadletFinder(universe_object,
1380 selection, cutoff)
1381 bottom_leaflet_phosphates = leaflets.atoms(0)
1382 top_leaflet_phosphates = leaflets.atoms(1)
1383 bottom_phosphate_list = []
1384 top_phosphate_list = []
1385 #for bottom leaflet:
1386 for phosphate_residue_number in bottom_leaflet_phosphates.resids():
1387     if bottom_leaflet_phosphates.resids().index(
1388 phosphate_residue_number) < (len(bottom_leaflet_phosphates.
1389 resids())-1): #for all but the last resid element
1390         bottom_phosphate_string = 'resid ' + str(
1391 phosphate_residue_number) + ' or '
1392     else: #we'll want a right ')' after the last resid element
1393         bottom_phosphate_string = 'resid ' + str(
1394 phosphate_residue_number) + ')'
1395     bottom_phosphate_list.append(bottom_phosphate_string)
1396 #repeat for top leaflet:
1397 for phosphate_residue_number in top_leaflet_phosphates.resids():
1398     if top_leaflet_phosphates.resids().index(
1399 phosphate_residue_number) < (len(top_leaflet_phosphates.

```

---

```

resids()-1): #for all but the last resid element
    top_phosphate_string = 'resid ' + str(
        phosphate_residue_number) + ' or '
else: #we'll want a right ')' after the last resid element
    top_phosphate_string = 'resid ' + str(
        phosphate_residue_number) + ')'
top_phosphate_list.append(top_phosphate_string)

#assign the formatted selection strings (Oct. 5/ 2010: adjusted to
properly include phosphate specification):
bottom_leaflet_phosphate_selection_string = '(' + ''.join(
    bottom_phosphate_list) + ' and ( rename POP* and name P* )'
top_leaflet_phosphate_selection_string = '(' + ''.join(
    top_phosphate_list) + ' and ( rename POP* and name P* )'

#Now that the 'absolute' leaflet selection strings are ready I will
start the trajectory looping:
for ts in universe_object.trajectory[:,skip_frames]:
    #I want to find out how many lipids (although I'm parsing their
    phosphates) are selected per analysis frame

    phosph_around_helix1_top_leaflet_string = "( around %s %s )" %
        (str(local_definition), helix1_CA_selection) + ' and ' +
    top_leaflet_phosphate_selection_string
    phosph_around_helix2_top_leaflet_string = "( around %s %s )" %
        (str(local_definition), helix2_CA_selection) + ' and ' +
    top_leaflet_phosphate_selection_string

```

---

1393

1395

1397

1399

434<sub>1401</sub>

1403

1405

1407

```

1409 phosph_around_helix1_bottom_leaflet_string = "( around %s %s )"
      % (str(local_definition), helix1_CA_selection) + ' and ' +
      bottom_leaflet_phosphate_selection_string
1410 phosph_around_helix2_bottom_leaflet_string = "( around %s %s )"
      % (str(local_definition), helix2_CA_selection) + ' and ' +
      bottom_leaflet_phosphate_selection_string
1411
1412 top_distal_phosphate_string =
      top_leaflet_phosphate_selection_string + ' and not ' + "(
      around %s ( name CA ) )" % str(local_definition)
1413 bottom_distal_phosphate_string =
      bottom_leaflet_phosphate_selection_string + ' and not ' + "(
      around %s ( name CA ) )" % str(local_definition)
1414
1415 #produce the actual MDAnalysis AtomGroup selections:
      phosph_around_helix1_top_leaflet = universe_object.selectAtoms(
      phosph_around_helix1_top_leaflet_string)
1416 phosph_around_helix2_top_leaflet = universe_object.selectAtoms(
      phosph_around_helix2_top_leaflet_string)
      phosph_around_helix1_bottom_leaflet = universe_object.
      selectAtoms(phosph_around_helix1_bottom_leaflet_string)
      phosph_around_helix2_bottom_leaflet = universe_object.
      selectAtoms(phosph_around_helix2_bottom_leaflet_string)
1417 top_distal_phosphate = universe_object.selectAtoms(
      top_distal_phosphate_string)
      bottom_distal_phosphate = universe_object.selectAtoms(
      bottom_distal_phosphate_string)
1418
1419 #now I want a way to count the number of lipid 'residues'
      selected in each case--the .numberOfResidues() method should

```

```

do the trick:
num_phosph_around_helix1_top_leaflet =
    phosph_around_helix1_top_leaflet.numberOfResidues()
num_phosph_around_helix2_top_leaflet =
    phosph_around_helix2_top_leaflet.numberOfResidues()
num_phosph_around_helix1_bottom_leaflet =
    phosph_around_helix1_bottom_leaflet.numberOfResidues()
num_phosph_around_helix2_bottom_leaflet =
    phosph_around_helix2_bottom_leaflet.numberOfResidues()
num_top_distal_phosphate = top_distal_phosphate.
    numberOfResidues()
num_bottom_distal_phosphate = bottom_distal_phosphate.
    numberOfResidues()

#track progress on the prompt:
print str(folder_name) + ' -- frame: ' + str(ts.frame)
outfile.write(str(num_phosph_around_helix1_top_leaflet) + ' ' +
    str(num_phosph_around_helix1_bottom_leaflet) + ' ' + str(
    num_phosph_around_helix2_top_leaflet) + ' ' + str(
    num_phosph_around_helix2_bottom_leaflet) + ' ' + str(
    num_top_distal_phosphate) + ' ' + str(
    num_bottom_distal_phosphate) + '\n')

```

```

def flip_flop_tracker(folder_name, universe_object, skip_frames, outfile_name, cutoff):
    '''Oct. 11/ 2010: Adjusting for use with GpA/FGFR3 simulations; Oct. 6/ 2010:
    To ensure that my bilayer thickness tracking algorithm is valid with this
    system I have to be able to show that there is no lipid flip-flop between
    leaflets after the first frame because I use an absolute definition of what
    lipid residue is in which leaflet and simply track the lipids from there.

```



```

1439 cutoff is the networkx cutoff use by MDAnalysis for leaflet selection and
1440 varies for Gpa, FGFR3 WT, and FGFR3 hetero/mutant
1441 '''
1442 with open(outfile_name, 'w') as outfile:
1443     outfile.write('#format: largest_Z_top_phosphate |
1444                 smallest_Z_top_phosphate | largest_Z_bottom_phosphate |
1445                 smallest_Z_bottom_phosphate | system_Z_center\n')
1446 #use the built-in MDAnalysis leaflet module to select the leaflet
1447     residues from the first frame as usual:
1448     import MDAnalysis.analysis.leaflet
1449     selection = 'resname POP* and name P*'
1450     leaflets = MDAnalysis.analysis.leaflet.Leadfinder(universe_object,
1451                                                       selection, cutoff)
1452     bottom_leaflet_phosphates = leaflets.atoms(0)
1453     top_leaflet_phosphates = leaflets.atoms(1)
1454     #as I've done for the other two bilayer analysis functions, select the
1455     phosphate particles in each leaflet:
1456     bottom_phosphate_list = []
1457     top_phosphate_list = []
1458     #for bottom leaflet:
1459     for phosphate_residue_number in bottom_leaflet_phosphates.resids():
1460         if bottom_leaflet_phosphates.resids().index(
1461             phosphate_residue_number) < (len(bottom_leaflet_phosphates.
1462             resids())-1): #for all but the last resid element
1463             bottom_phosphate_string = 'resid ' + str(
1464                 phosphate_residue_number) + ' or '
1465             #we'll want a right ')' after the last resid element
1466             bottom_phosphate_string = 'resid ' + str(
1467                 phosphate_residue_number) + ' )'
1468             bottom_phosphate_list.append(bottom_phosphate_string)

```

```

1459 #repeat for top leaflet:
1461 for phosphate_residue_number in top_leaflet_phosphates.resids():
    if top_leaflet_phosphates.resids().index(
        phosphate_residue_number) < (len(top_leaflet_phosphates.
resids())-1): #for all but the last resid element
        top_phosphate_string = 'resid ' + str(
            phosphate_residue_number) + ' or '
    else: #we'll want a right ')' after the last resid element
        top_phosphate_string = 'resid ' + str(
            phosphate_residue_number) + ')'
    top_phosphate_list.append(top_phosphate_string)

#assign the formatted selection strings (Oct. 5/ 2010: adjusted to
properly include phosphate specification):
bottom_leaflet_phosphate_selection_string = '(' + ''.join(
    bottom_phosphate_list) + ' and ( rename POP* and name P* )'
top_leaflet_phosphate_selection_string = '(' + ''.join(
    top_phosphate_list) + ' and ( rename POP* and name P* )'

#Now that the 'absolute' leaflet selection strings are ready I will
start the trajectory looping:
for ts in universe_object.trajectory[:,skip_frames]:

    #produce a list of the Z coordinates for each phosphate in the
    top leaflet:
    top_phosphate_coordinates = universe_object.selectAtoms(
        top_leaflet_phosphate_selection_string).coordinates()
    #find the largest Z coordinate in the top leaflet phosphates:
    largest_Z_top_phosphate = top_phosphate_coordinates[0][2]

```

1459

1461

1463

1465

1467

438

1469

1471

1473

1475

1477

```

1479 for xyz_coordinate in top_phosphate_coordinates:
1481     if xyz_coordinate[2] > largest_Z_top_phosphate:
1483         largest_Z_top_phosphate = xyz_coordinate[2]
1485 #find the smallest Z coordinate in the top leaflet phosphates:
1487 smallest_Z_top_phosphate = top_phosphate_coordinates[0][2]
1489 for xyz_coordinate in top_phosphate_coordinates:
1491     if xyz_coordinate[2] < smallest_Z_top_phosphate:
1493         smallest_Z_top_phosphate = xyz_coordinate[2]

#repeat the process for the bottom leaflet:
#produce a list of the Z coordinates for each phosphate in the
bottom leaflet:
bottom_phosphate_coordinates = universe_object.selectAtoms(
    bottom_leaflet_phosphate_selection_string).coordinates()
#find the largest Z coordinate in the bottom leaflet phosphates
:
largest_Z_bottom_phosphate = bottom_phosphate_coordinates[0][2]
for xyz_coordinate in bottom_phosphate_coordinates:
    if xyz_coordinate[2] > largest_Z_bottom_phosphate:
        largest_Z_bottom_phosphate = xyz_coordinate[2]
#find the smallest Z coordinate in the bottom leaflet
phosphates:
smallest_Z_bottom_phosphate = bottom_phosphate_coordinates
[0][2]
for xyz_coordinate in bottom_phosphate_coordinates:
    if xyz_coordinate[2] < smallest_Z_bottom_phosphate:
        smallest_Z_bottom_phosphate = xyz_coordinate[2]

#find the Z coordinate of the center of geometry of the entire
(phosphate) lipid system in this frame:

```

```

system_Z_center = universe_object.selectAtoms('resname POP* and
name P*').centerOfGeometry()[2]

#print the various Z coordinates to file so I can check for
lipid flip-flop activity:
outfile.write(str(largest_Z_top_phosphate) + ' ' + str(
smallest_Z_top_phosphate) + ' ' + str(
largest_Z_bottom_phosphate) + ' ' + str(
smallest_Z_bottom_phosphate) + ' ' + str(system_Z_center) +
'\n')

#print out results in testing phase:
#print largest_Z_top_phosphate, smallest_Z_top_phosphate,
largest_Z_bottom_phosphate, smallest_Z_bottom_phosphate,
system_Z_center
#track progress on the prompt:
print str(folder_name) + ' -- frame: ' + str(ts.frame)

def bilayer_thickness_average_results(WT_FGFR3_distal_thickness_list,
WT_FGFR3_local_thickness_pre_dimer_list, WT_FGFR3_local_thickness_post_dimer_list,
hetero_FGFR3_distal_thickness_list, hetero_FGFR3_local_thickness_pre_dimer_list,
hetero_FGFR3_local_thickness_post_dimer_list, mutant_FGFR3_distal_thickness_list,
mutant_FGFR3_local_thickness_pre_dimer_list,
mutant_FGFR3_local_thickness_post_dimer_list, GpA_distal_thickness_list,
GpA_local_thickness_pre_dimer_list, GpA_local_thickness_post_dimer_list, folder_name,
dimerization_criterion):
'''Oct. 13/2010: Now I want averaged (with standard deviation) protein-local
and -distal bilayer thickness results for easy comparison of GpA, FGFR3 WT,
FGFR3 hetero, and FGFR3 mutant conditions. I would like to further subdivide
the protein-local bilayer thickness to an average of the two helices before

```

1503

1505

1507

440<sup>1509</sup>

1511

1513

```

1515         and after dimerization.

1516         dimerization_criterion : the CA-CA closest approach distance cutoff (in
1517         Angstroms) for defining the frame as a 'dimer frame' where the trajectory is
1518         then split for the purposes of averaging pre- and post- dimer local bilayer
1519         thicknesses (around 6 Angstroms is probably reasonable)

1520     I will use a separate function to calculate average and standard deviation of
1521     the global lists and print this to a file.

1522     '''
1523     #to track progress:
1524     print folder_name, 'starting'

1525     #check for condition based on folder name to decide which lists get appended to
1526     :
1527     if 'GpA' in folder_name:
1528         distal_list = GpA_distal_thickness_list
1529         local_pre_dimer_list = GpA_local_thickness_pre_dimer_list
1530         local_post_dimer_list = GpA_local_thickness_post_dimer_list
1531         elif 'heterodimer' in folder_name:
1532             distal_list = hetero_FGFR3_distal_thickness_list
1533             local_pre_dimer_list = hetero_FGFR3_local_thickness_pre_dimer_list
1534             local_post_dimer_list = hetero_FGFR3_local_thickness_post_dimer_list
1535         elif 'wildtype' in folder_name:
1536             distal_list = WT_FGFR3_distal_thickness_list
1537             local_pre_dimer_list = WT_FGFR3_local_thickness_pre_dimer_list
1538             local_post_dimer_list = WT_FGFR3_local_thickness_post_dimer_list
1539         elif 'mutant' in folder_name:

```

```

1539 distal_list = mutant_FGFR3_distal_thickness_list
1540 local_pre_dimer_list = mutant_FGFR3_local_thickness_pre_dimer_list
1541 local_post_dimer_list = mutant_FGFR3_local_thickness_post_dimer_list

#the head script will take care of changing directories so I simply need to
  open the closest approach output file to get access to the data that I need
  to assess the approximate frame at which dimerization occurs:
1543 with open('closest_contacts_full.out', 'r') as closest_contact_file:
1544     for line in closest_contact_file:
1545         if '#' in line: continue #ignore the line if it's a comment
1546             starting with a hash (i.e., the first line)
1547         frame_number = int(line.split()[0])
1548         closest_approach = float(line.split()[1])
1549         if closest_approach <= dimerization_criterion: #if a dimer has
1550             formed
1551             first_dimer_frame = frame_number
1552             print 'first dimer frame: ', first_dimer_frame #tracking
1553                 progress
1554             break #no need to continue parsing once the dimer has
1555                 formed because none of my POPC simulations have
1556                 protein dissociation events after dimerization so it'
1557                 s a clean split of the trajectory at this frame
1558                 number for averaging purposes

#now open the file that contains the protein-local and -distal bilayer
  thickness results:
1559 with open('bilayer_thickness.out', 'r') as bilayer_thickness_file:
1560     for line in bilayer_thickness_file:
1561         if '#' in line: continue #ignore comment line(s)

```

```

1559 frame_number = int(line.split()[0])
1560 thickness_near_monomer_1 = float(line.split()[1])
1561 thickness_near_monomer_2 = float(line.split()[2])
1562 distal_thickness = float(line.split()[3])
1563 #always append the distal thickness because I don't care about
1564 dimerization for that value:
1565 distal_list.append(distal_thickness)
1566
1567 if frame_number < first_dimer_frame: #if dimerization has not
1568 yet occurred:
1569     local_pre_dimer_list.append(thickness_near_monomer_1)
1570     local_pre_dimer_list.append(thickness_near_monomer_2)
1571     elif frame_number >= first_dimer_frame: #use the post-dimer
1572     local list if dimerization has occurred
1573     local_post_dimer_list.append(thickness_near_monomer_1)
1574     local_post_dimer_list.append(thickness_near_monomer_2)
1575
1576 #to track progress:
1577 print folder_name, 'done'
1578
1579 def bilayer_stats(WT_FGFR3_distal_thickness_list,
1580 WT_FGFR3_local_thickness_pre_dimer_list, WT_FGFR3_local_thickness_post_dimer_list,
1581 hetero_FGFR3_distal_thickness_list, hetero_FGFR3_local_thickness_pre_dimer_list,
1582 hetero_FGFR3_local_thickness_post_dimer_list, mutant_FGFR3_distal_thickness_list,
1583 mutant_FGFR3_local_thickness_pre_dimer_list,
1584 mutant_FGFR3_local_thickness_post_dimer_list, GpA_distal_thickness_list,
1585 GpA_local_thickness_pre_dimer_list, GpA_local_thickness_post_dimer_list):
1586     '''Take the data in the global lists populated by
1587     bilayer_thickness_average_results() and generate stats, then output to a file
1588     . This is for tracking overall average protein-local and -distal bilayer
1589     thicknesses for GpA and FGFR3 conditions.'''

```

```

1577 #use numpy to find the average value of the bilayer thickness lists:
1578 WT_distal_avg = numpy.average(WT_FGFR3_distal_thickness_list)
1579 WT_local_pre_avg = numpy.average(WT_FGFR3_local_thickness_pre_dimer_list)
1580 WT_local_post_avg = numpy.average(WT_FGFR3_local_thickness_post_dimer_list)
1581 hetero_distal_avg = numpy.average(hetero_FGFR3_distal_thickness_list)
1582 hetero_local_pre_avg = numpy.average(
1583     hetero_FGFR3_local_thickness_pre_dimer_list)
1584 hetero_local_post_avg = numpy.average(
1585     hetero_FGFR3_local_thickness_post_dimer_list)
1586 mutant_distal_avg = numpy.average(mutant_FGFR3_distal_thickness_list)
1587 mutant_local_pre_avg = numpy.average(
1588     mutant_FGFR3_local_thickness_pre_dimer_list)
1589 mutant_local_post_avg = numpy.average(
1590     mutant_FGFR3_local_thickness_post_dimer_list)
1591 GpA_distal_avg = numpy.average(GpA_distal_thickness_list)
1592 GpA_local_pre_avg = numpy.average(GpA_local_thickness_pre_dimer_list)
1593 GpA_local_post_avg = numpy.average(GpA_local_thickness_post_dimer_list)
1594 #use numpy to calculate the corresponding standard deviations:
1595 WT_distal_std = numpy.std(WT_FGFR3_distal_thickness_list)
1596 WT_local_pre_std = numpy.std(WT_FGFR3_local_thickness_pre_dimer_list)
1597 WT_local_post_std = numpy.std(WT_FGFR3_local_thickness_post_dimer_list)
1598 hetero_distal_std = numpy.std(hetero_FGFR3_distal_thickness_list)
1599 hetero_local_pre_std = numpy.std(hetero_FGFR3_local_thickness_pre_dimer_list)
1600 hetero_local_post_std = numpy.std(hetero_FGFR3_local_thickness_post_dimer_list)
1601 mutant_distal_std = numpy.std(mutant_FGFR3_distal_thickness_list)
1602 mutant_local_pre_std = numpy.std(mutant_FGFR3_local_thickness_pre_dimer_list)
1603 mutant_local_post_std = numpy.std(mutant_FGFR3_local_thickness_post_dimer_list)
1604 GpA_distal_std = numpy.std(GpA_distal_thickness_list)
1605 GpA_local_pre_std = numpy.std(GpA_local_thickness_pre_dimer_list)

```



```

1603 GpA_local_post_std = numpy.std(GpA_local_thickness_post_dimer_list)
1604
1605 #write the results to file for gnuplot histogram plotting:
1606 with open('/sansom/sc2/bioc1009/Documents/FGFR3_work/
1607     dimer_batch_analysis_symlink/bilayer_histogram.out', 'w') as outfile:
1608     #row for file format:
1609     outfile.write('#format: category | distal_avg | distal_std |
1610         local_pre_avg | local_pre_std | local_post_avg | local_post_std (
1611         bilayer thicknesses in A)\n')
1612     #next row is for GpA results:
1613     outfile.write('GpA ' + str(GpA_distal_avg) + ' ' + str(GpA_distal_std)
1614         + ' ' + str(GpA_local_pre_avg) + ' ' + str(GpA_local_pre_std) + ' ' +
1615         str(GpA_local_post_avg) + ' ' + str(GpA_local_post_std)+'\n')
1616     #next row for WT FGFR3 results:
1617     outfile.write('WT-FGFR3 ' + str(WT_distal_avg) + ' ' + str(
1618         WT_distal_std) + ' ' + str(WT_local_pre_avg) + ' ' + str(
1619         WT_local_pre_std) + ' ' + str(WT_local_post_avg) + ' ' + str(
1620         WT_local_post_std)+'\n')
1621     #next row for hetero FGFR3 results:
1622     outfile.write('hetero-FGFR3 ' + str(hetero_distal_avg) + ' ' + str(
1623         hetero_distal_std) + ' ' + str(hetero_local_pre_avg) + ' ' + str(
1624         hetero_local_pre_std) + ' ' + str(hetero_local_post_avg) + ' ' + str(
1625         hetero_local_post_std)+'\n')
1626     #next row for mutant FGFR3 results:
1627     outfile.write('mut-FGFR3 ' + str(mutant_distal_avg) + ' ' + str(
1628         mutant_distal_std) + ' ' + str(mutant_local_pre_avg) + ' ' + str(
1629         mutant_local_pre_std) + ' ' + str(mutant_local_post_avg) + ' ' + str(
1630         mutant_local_post_std))
1631
1632 def simple_moving_average_polar_theta(folder_name, window_size):

```

```
'''Oct. 28/2010: The function is actually doing a weighted average now and also
has a 'spike filter' to remove short-lived high-amplitude excursions from
interfaces. Oct. 27/ 2010: This function will calculate the simple moving
average (SMA) for the FGFR3 polar theta tracking data (already present in a
file) by taking advantage of the numpy.convolve method. My objective is to
smooth the data so that I can more easily define polar theta boundaries
around which primary and secondary dimer interface frames can be categorized
for an overall interface analysis across all replicate trajectories.
```

1619

```
window_size represents the size of the sliding window used to smooth the data
```

1621

```
'''
```

```
#Before grabbing polar theta values I need to figure out in which frame the
dimer actually forms (<7A separation between helices)--if I don't filter for
this then I could classify monomeric configurations based on polar theta and
I don't want that:
```

1623

```
with open('closest_contacts_full.out', 'r') as contact_file:
```

1625

```
    for line in contact_file:
```

```
        if '#' in line: continue #skip comment lines
```

1627

```
        frame_number = int(line.split()[0])
```

```
        closest_contact = float(line.split()[1])
```

1629

```
        if closest_contact < 6.0:
```

```
            dimer_start_frame = frame_number
```

1631

```
            break #leave the loop when the dimer formation is
            identified
```

1633

```
#I have already calculated the polar theta tracking values in a file in each
replicate simulation directory:
```

1635

```
polar_theta_file_name = 'polar_interface_theta.out'
```

```
1637 smooth_theta_file_name = 'smooth_polar_theta.out'
1639
1641 frame_list = [] #initialize a list for storing the frame numbers
1643 theta_list = [] #initialize a list for storing polar theta values
1645 with open(polar_theta_file_name, 'r') as input_file:
    for line in input_file:
        if '#' in line: continue #skip comment lines
        frame_number = int(line.split()[0])
        polar_theta = float(line.split()[1])
        if frame_number >= dimer_start_frame: #if the dimer has formed
            based on the closest contacts file then include the polar
            theta information:
                frame_list.append(frame_number)
                theta_list.append(polar_theta)

counter = 0
#this is tricky, but I'd like a way to ignore/attenuate large/short-lived polar
theta excursions:
for polar_theta in theta_list:
    index = theta_list.index(polar_theta)
    if index > 20.0:
        if abs(polar_theta - numpy.average(theta_list[index - 20 :
            index])) > 1.0 and abs(polar_theta - numpy.average(theta_list
            [index: index + 20])) > 1.0: #if the current polar theta
            value is different by spike_threshold from the preceding AND
            following averaged 20 data points, replace it by the average
            of the 10 preceding points; note that it has to satisfy both
```

1637

1639

1641

1643

1645

1647

447

1649

1651

1653

1655

1657

```
1659 conditions because I don't want to attenuate a real/long-
1660 lived transition between interfaces
1661 counter += 1.0
1662 theta_list[index] = numpy.average(theta_list[index -
1663 10: index])
1664
1665 print counter
1666
1667 #numpy convolve trick (see http://stackoverflow.com/questions/1335392/iteration-over-list-slices/1339575#1339575 and the numpy.convolve online documentation for explanation of how the second array argument is flipped and slid across the first for convolution with the default arguments):
1668
1669 #for a weighted moving average (see WMA wikipedia entry):
1670 denominator = (window_size * (window_size + 1.0)) / 2.0
1671
1672 sliding_array = numpy.ones(window_size)/denominator
1673 coefficient_list = [coefficient for coefficient in range(1, window_size + 1)] #
1674 list of coefficients for weighted average
1675
1676 adjusted_coefficients = [] #the list will get reversed before convolution so we
1677 can leave the list of coefficients as increasing because the last value
1678 becomes the first value:
1679 for unit_value, coefficient in zip(sliding_array, coefficient_list):
1680     adjusted_coefficients.append(unit_value*coefficient)
1681 print numpy.array(adjusted_coefficients)
1682 smooth_theta_array = numpy.convolve(numpy.array(adjusted_coefficients), numpy.
1683 array(theta_list))[window_size-1:-window_size+1]
1684
1685
```

```

1679 #for testing purposes:
1680 print 'check 1a: theta_list length: ', len(theta_list)
1681 print 'check 1b: smooth_theta_array length: ', len(smooth_theta_array)
1682
1683 with open(smooth_theta_file_name, 'w') as output_file:
1684     output_file.write('#format: frame number | smoothed polar theta value
1685     for helix 2 COM position in rmsd-fixed reference frame of helix 1\n'
1686     )
1687     for frame_number, smooth_data_point in zip(frame_list,
1688     smooth_theta_array): #the smoothed list will usually be shorter and
1689     limiting
1690         output_file.write(str(frame_number) + ' ' + str(
1691         smooth_data_point) + '\n')
1692     #monitor progress on prompt:
1693     #print str(folder_name), 'frame_number: ', str(frame_number), '
1694     windowed polar theta value: ', str(smooth_data_point)
1695
1696 def interface_filtered_merged_top_five_FGFR3(primary_count_list,secondary_count_list,
1697 other_count_list, folder_name, universe_object, skip_frames):
1698     '''Oct. 28/2010: Modifying the merged top 5 contacts function to additionally
1699     filter all data based on weighted-average and spike-filtered polar theta
1700     values available in the file 'smooth_polar_theta.out'. Parses and merges the
1701     'top five' contact data for wild-type, mutant, and hetero- dimers into one of
1702     three lists: primary, secondary, or 'other' polar theta interface. An
1703     additional function must be called outside of 'main' in the head script to
1704     parse the overall data in these lists.'''
1705     #intialize lists for storing the frame numbers that correspond to primary,
1706     secondary, and 'other' dimer interfaces:
1707     primary_frame_list = []
1708     secondary_frame_list = []

```

```

1695 other_frame_list = []
1696
1697 #the head script handles the directory switching so I can just parse the '
1698 smooth_polar_theta.out' file in each directory to pull out the values I want:
1699 frame_theta_list = [] #store (frame number, corrected theta)
1700 with open('smooth_polar_theta.out', 'r') as corrected_theta_input_file:
1701     for line in corrected_theta_input_file:
1702         if '#' in line: continue #skip comment lines
1703         frame_number = int(line.split()[0])
1704         corrected_theta = float(line.split()[1])
1705         frame_theta_list.append((frame_number, corrected_theta))
1706
1707 #now filter the data using the cutoffs I've determined empirically for each
1708 dimer interface:
1709
1710 #Oct. 29/2010: adjusting the filtering to only allow frame classification in a
1711 dimer interface if 50% of the subsequent points also fall into that interface
1712 :
1713 for (frame_number, corrected_theta) in frame_theta_list:
1714     current_index = frame_theta_list.index((frame_number, corrected_theta))
1715     current_window = frame_theta_list[current_index : current_index + 10]
1716     successful_test = 0
1717     if corrected_theta > -3.0 and corrected_theta < -1.5: #if conditions
1718         for primary interface are satisfied:
1719             for (frame_number, corrected_theta) in current_window: #check
1720                 the number of frames in this window that satisfy interface
1721                 requirement
1722                 if corrected_theta > -3.0 and corrected_theta < -1.5:
1723                     successful_test += 1
1724             if successful_test >= 5: #if 50% of the points in the current
1725                 window satisfy interface requirements then this frame can be
1726                 appended to interface list:

```

```

1715         primary_frame_list.append(frame_number)
1717     elif corrected_theta > -1.0 and corrected_theta < 1.0: #if conditions
        for secondary interface are satisfied:
            for (frame_number, corrected_theta) in current_window: #check
                the number of frames in this window that satisfy interface
                requirement
                    if corrected_theta > -1.0 and corrected_theta < 1.0:
                        successful_test += 1
1719         if successful_test >= 5: #if 50% of the points in the current
            window satisfy interface requirements then this frame can be
            appended to interface list:
                secondary_frame_list.append(frame_number)
1721     else: #for any frames where the polar theta of helix 2 does not fall
            into either category:
                other_frame_list.append(frame_number)

#so, now we know which frames correspond to which interfaces for the trajectory
we are about to parse with MDAnalysis, and these frames have already been
filtered to be within 6 Angstroms

1727 for ts in universe_object.trajectory[::skip_frames]:
    helix_1_CA_selection = "( name CA and resid 1:33 )" #select first FGFR3
    monomer
1729    helix_2_CA_selection = "( name CA and resid 34:66 )" #select second
    FGFR3 monomer

1731 #Select all CA atoms in each helix:
    helix_1_CA_residues = universe_object.selectAtoms(helix_1_CA_selection)
1733    helix_2_CA_residues = universe_object.selectAtoms(helix_2_CA_selection)

```

```

1735 #Use the atomselections to generate lists of ordered coordinates for
      each helix:
1737 helix_1_CA_coordinates = helix_1_CA_residues.coordinates()
      helix_2_CA_coordinates = helix_2_CA_residues.coordinates()

1739 #Use the atomselections to generate a list of ordered residue
      identifiers (MDAnalysis format) for each helix:
1741 helix_1_identifiers = list(helix_1_CA_residues)
      helix_2_identifiers = list(helix_2_CA_residues)

1743 #make a list of ordered residue names and numbers (in tuples at the
      moment)
      helix_1_ordered_residue_names_numbers = [(residue.resname, residue.
      resid) for residue in helix_1_identifiers]
      helix_2_ordered_residue_names_numbers = [(residue.resname, residue.
      resid) for residue in helix_2_identifiers]

1747 #now zip the residue names and numbers with their respective CA
      coordinates
      helix_1_merged_list = zip(helix_1_ordered_residue_names_numbers,
      helix_1_CA_coordinates)
      helix_2_merged_list = zip(helix_2_ordered_residue_names_numbers,
      helix_2_CA_coordinates)

1749 #this would give the [x y z] coordinate of the 23rd CA in helix 1:
      print helix_1_merged_list[22][1]
1751 #the zeroth element would be the corresponding residue name and number

      #go through all possible combinations of residue interactions between
      the two helices and make a list of the distance (magnitude)
1753

```



```

1755 #between the CA particles which retains the residue names that
1756     correspond to the interaction
1757 labelled_interhelix_distances = []
1758 for (helix_1_CA, coordinate_1) in helix_1_merged_list:
1759     for (helix_2_CA, coordinate_2) in helix_2_merged_list:
1760         measured_distance = numpy.linalg.norm(numpy.subtract(
1761             coordinate_1, coordinate_2))
1762         labelled_interhelix_distances.append([helix_1_CA,
1763             helix_2_CA, measured_distance])
1764
1765 #sort the list of labelled distances by the 'third element,' which is
1766     the distance between CA particles
1767 labelled_interhelix_distances.sort(key = lambda element: element[2])
1768
1769 #so, now there is an ascending list of closest contacts between helices
1770     for this given frame
1771
1772 #append the five closest contacts from this given frame to an overall
1773     list based on polar theta interface categories:
1774 for element in labelled_interhelix_distances[0:5]:
1775     if ts.frame in primary_frame_list:
1776         primary_count_list.append(element) #still contains
1777             residue 1 and 2 names along with distance
1778     elif ts.frame in secondary_frame_list:
1779         secondary_count_list.append(element)
1780     else:
1781         other_count_list.append(element)
1782
1783 print str(folder_name) + ' -- frame: ' + str(ts.frame)
1784
1785
1786
1787

```

```

#each of the contact count lists will be parsed by another function
  called outside of main to get the final data for plotting 'top 5
  contacts' for primary, secondary, and other dimer interfaces

def interface_filtered_parse_overall_FGFR3_top_five_data(primary_count_list,
  secondary_count_list, other_count_list):
  '''Oct. 28: Modifying this function to work for polar-theta-filtered data.
  Perform final analysis on merged 'top five contacts' data for each of the
  three FGFR3 *dimer interfaces* and print to file.'''

  #determine number of top five contacts in total:
  num_primary_pairs = len(primary_count_list)
  num_secondary_pairs = len(secondary_count_list)
  num_other_pairs = len(other_count_list)

def parser_printer(input_list, condition, num_pairs): #does most of the work
  for a given list of contact pairs
    print len(input_list)
    file_path = '/sansom/sc2/bioc1009/Documents/FGFR3_work/
    dimer_batch_analysis_symlink/test_case_top5/%s_merged_topfive.out' %
    condition
    merged_top_five_outfile = open(str(file_path), 'w')
    helix_1_partner_list = []
    helix_2_partner_list = []
    #make separate lists with each of the residue names (including
    duplicates) found in close contacts from the respective helices
    for (helix_1_partner, helix_2_partner, separation) in input_list:
      helix_1_partner_list.append(helix_1_partner) #the involved
      residue from helix 1 gets appended

```

1779

1781

1783

1785

4787  
54

1789

1791

1793

1795

1797

```
helix_2_partner_list.append(helix_2_partner) #same for cognate
    helix_2_residue in a separate list
```

1799

```
#Now, each list basically has an entry for every time a given residue
showed up in the top five for an acceptable frame from one of the
helices
```

1801

```
#This includes duplicates so can use the 'set' data structure to remove
duplicates for iteration purposes without actually removing the
important multiple occurrences within the lists proper
helix_1_no_duplicates = set(helix_1_partner_list)
helix_2_no_duplicates = set(helix_2_partner_list)
```

1803

```
#results will be printed in the following format, specified at top of
file in gnuplot-friendly form:
```

1805  
44  
51

```
string_1 = '#columns 1-3 (%s): helix 1 resname || resnum || frequency
in top 5 closest contacts \n' % condition
string_2 = '#columns 4-6 (%s): helix 2 resname || resnum || frequency
in top 5 closest contacts \n' % condition
merged_top_five_outfile.write(string_1)
merged_top_five_outfile.write(string_2)
```

1807

```
#go through the list of all residues counted by frame and count the
unique occurrence of a given residue from a given helix, and print
result to file
```

1809

```
for (residue1_name, residue2_name) in zip(helix_1_no_duplicates,
helix_2_no_duplicates): #limited by the shortest list though; this
could be a problem if only a really small number of unique residues
were picked from one helix
```

1811

```
    residue1_count = helix_1_partner_list.count(residue1_name) #raw
        residue counts for helix1
```

```

1813 residue2_count = helix_2_partner_list.count(residue2_name) #raw
      residue counts for helix2

1815 residue1_frequency = float(residue1_count)/float(num_pairs) #
      because we want the 'normalized frequency'
1817 residue2_frequency = float(residue2_count)/float(num_pairs)

      #print out results that are simultaneously corrected for
      residue numbers to match those used in FGFR3 literature
1819 merged_top_five_outfile.write(str(residue1_name[0]).strip("")
      + ' ' +str(residue1_name[1]+366) + ' ' + str(
      residue1_frequency) + ' ' + str(residue2_name[0]).strip("")
      + ' ' +str(residue2_name[1]+(366-33)) + ' ' + str(
      residue2_frequency) + '\n')

      merged_top_five_outfile.close()

1823

      #now call the function to parse and print three files of merged data for each
      FGFR3 dimer interface:
1825 parser_printer(primary_count_list, 'primary_interface', num_primary_pairs)
      parser_printer(secondary_count_list, 'secondary_interface', num_secondary_pairs
      )
1827 parser_printer(other_count_list, 'other_interface', num_other_pairs)

```

Listing D.18: This module (`analyze_FGFR3_monomer_simulations.py`) serves a similar purpose to the (`analyze_FGFR3_dimer_simulations.py`) module D.15 on page 338, but instead moves through `FGFR3 monomer` replicate simulation directories and parses the trajectories using functions from the (`monomer_geometric_tools.py`) library D.19 on page 460

```
1 '''For specific analysis of the ten FGFR3 monomer trajectories. Analysis is carried out
   by
   parsing files organized in a directory with symlinks to the original production-length
   data folders.'''
3
4
5 import MDAnalysis
6 import monomer_geometric_tools
7 import os
8
9 monomer_symlink_directory = '/sansom/sc2/bioc1009/Documents/FGFR3_work/
   monomer_batch_analysis_symlink/'
10
11 folder_name_list = ['mutant_monomer_replicate_1', 'mutant_monomer_replicate_2', '
   mutant_monomer_replicate_3', 'mutant_monomer_replicate_4',
   'mutant_monomer_replicate_5', 'wildtype_monomer_replicate_1', '
   wildtype_monomer_replicate_2', 'wildtype_monomer_replicate_3', '
   wildtype_monomer_replicate_4',
   'wildtype_monomer_replicate_5']
12
13 def list_data_paths(monomer_symlink_directory, folder_name_list):
14     '''Returns a list of tuples with pairs of pdb (index 0) and xtc (index 1) paths
   for a given simulation folder. The name of the folder is (index 2) in the tuple
   .'''
15     list_of_pdb_file_paths=[]
16
17
```

```

19 list_of_xtc_file_paths=[]
20 for folder in folder_name_list: #files names can be changed for differently
21     processed trajectories (i.e., simplified and centered or not)
22     list_of_pdb_file_paths.append(monomer_symlink_directory + folder + '/'
23     + 'prod.gro') #centered_CA_POPC.pdb
24     list_of_xtc_file_paths.append(monomer_symlink_directory + folder + '/'
25     + 'prod.xtc') #centered_CA_POPC.xtc
26 combined_path_pair_list = zip(list_of_pdb_file_paths,list_of_xtc_file_paths,
27     folder_name_list)
28 return combined_path_pair_list
29
30 def create_universe_selections(monomer_symlink_directory, folder_name_list):
31     '''Creates a nested list of [universe (all atoms in a system selected)
32     MDAnalysis objects, folder_name]'''
33     universe_list=[]
34     for pdb_file, xtc_file, folder_name in list_data_paths(
35     monomer_symlink_directory, folder_name_list):
36         universe_list.append([MDAnalysis.Universe(pdb_file, xtc_file),
37         folder_name])
38     return universe_list
39
40 def main(monomer_symlink_directory, folder_name_list, skip_frames, geo_z_output_file,
41     helix_tilt_output_file):
42     '''prints data to files in symlink directories by calling various functions'''
43     for universe_object, folder_name in create_universe_selections(
44     monomer_symlink_directory, folder_name_list):
45         os.chdir(monomer_symlink_directory + folder_name) #move the appropriate
46         directory before executing data processing functions
47         #call data processing functions; output should be in gnuplot format:
48         #just comment out functions when you are not needing them to be re-run:

```

```
39 monomer_geometric_tools.geo_z_tracker(folder_name, universe_object,
41 skip_frames, output_file=geo_z_output_file)
   #monomer_geometric_tools.helix_tilt_vs_bilayer_normal(folder_name,
   universe_object, skip_frames, output_file=helix_tilt_output_file)

43 if __name__ == "__main__":
   main(monomer_symlink_directory, folder_name_list, skip_frames=10,
   geo_z_output_file='Z_tracking-skip10_no_center.out', helix_tilt_output_file='
   tilt_test.out')
```

Listing D.19: This module (`monomer_geometric_tools.py`) serves as a library of functions that can be called by the head script (`analyze_FGFR3_monomer_simulations.py`) D.18 on page 457 to parse the FGFR3 *monomer* replicate MD trajectories

```
1 '''Start building up MDAnalysis tools here for analyzing the motion
2 of a monomer peptide in an MD trajectory.'''
3
4 import MDAnalysis
5 import numpy
6 import math
7
8
9 def geometric_center_z_coordinate_CA(selection):
10     '''Returns Z coordinate of the geometric center of mass for CA particles in a
11     selection.'''
12     ca = selection.selectAtoms("name CA")
13     geometric_center_xyz = ca.centerOfGeometry()
14     return geometric_center_xyz[2]
15
16 def geo_z_tracker(folder_name, universe_object, skip_frames, output_file):
17     '''Prints the Z-coordinate of the geometric center (only CA particles
18     considered) for a given universe_object
19     in a given frame, with frame intervals defined by skip_frames, to a
20     specified output_file'''
21     geo_z_tracking_file = open(output_file, 'w')
22     geo_z_tracking_file.write("#column 1: frame number || column 2: Z
23     position of geometric center for CA-monomer \n")
24     for ts in universe_object.trajectory[::skip_frames]: #xtc trajectory
25         object
```



```

21 #write Z coordinate of CA geometric center for each frame of
    each simulation (every nth frame):
22     geo_z_tracking_file.write(str(ts.frame) + ' ' + str(
23         geometric_center_z_coordinate_CA(universe_object))+'\n')
        print folder_name + ' : ' + str(ts.frame)
        geo_z_tracking_file.close()

24 def helix_tilt_vs_bilayer_normal(folder_name, universe_object, skip_frames, output_file
25 ):
    '''Calculates the tilt of helix (CA backbone) relative to bilayer normal using
26     linear algebra
        SVD technique to define helix axis (first eigenvector) and bilayer normal (
27         third eigenvector).'''
        helix_tilt_angle_file = open(output_file, 'w')
28     helix_tilt_angle_file.write('#column 1: frame # || column 2: helix tilt angle (
29         degrees) relative to bilayer normal (eigenvector approach)\n')
        for ts in universe_object.trajectory[::skip_frames]:
30
31         CA_selection = universe_object.selectAtoms("name CA")
32         CA_coordinates = CA_selection.coordinates()
33         CA_geometric_center = CA_selection.centerOfGeometry()
34
35         bilayer_phosphate_selection = universe_object.selectAtoms("name PO4")
36         phosphate_coordinates = bilayer_phosphate_selection.coordinates()
37         phosphate_geometric_center = bilayer_phosphate_selection.
            centerOfGeometry()
38
39 #the idea is to do a SVD on the centered coordinates; so subtract the
        geometric center coordinate value from the coordinates of the system
        #when you call the numpy SVD function:
40

```

```

43 uu, dd, vv = numpy.linalg.svd(CA_coordinates-CA_geometric_center)
44 helix_vector = vv[0] #this is the first eigenvector; checking in VMD,
45 it looks right along the helix axis
46 #vv[1] and vv[2] are the other two eigenvectors; note that for all
47 these eigenvectors the result you get
48 #back is an (x,y,z) coordinate that serves as a 'direction' from the
49 origin (0,0,0) for drawing the given vector
50
51 #import os
52 #os.system("vmd -dispdev text -eofexit < input.tcl > output.log") #
53 could eventually set up to render the vectors for certain frames in
54 pdb file
55 #automatically, but I think there are some challenges here
56
57 #for testing: multiply by -7 and +7 to make an extended vector, then
58 add the geometric center back for the given frame:
59 #lower_helix_vector_coordinate = [coordinate * -7 for coordinate in
60 helix_vector]
61 #upper_helix_vector_coordinate = [coordinate * 7 for coordinate in
62 helix_vector]
63 #lower_helix_vector_coordinate = map(sum, zip(
64 lower_helix_vector_coordinate, CA_geometric_center))
65 #upper_helix_vector_coordinate = map(sum, zip(
66 upper_helix_vector_coordinate, CA_geometric_center))
67
68 #repeat the eigenvector calculation process for the bilayer (phosphates
69 )
70 aa, bb, cc = numpy.linalg.svd(bilayer_phosphate_selection.coordinates()
71 -bilayer_phosphate_selection.centerOfGeometry())

```

```

61 bilayer_normal = cc[2] #check in VMD: yes, the third eigenvector seems
    to produce a reasonable vector up +Z axis for bilayer normal
62
63 #testing, as for the helix vector:
    #lower_bilayer_vector_coordinate = [coordinate * -7 for coordinate in
        bilayer_normal]
    #upper_bilayer_vector_coordinate = [coordinate * 7 for coordinate in
        bilayer_normal]
64 #lower_bilayer_vector_coordinate = map(sum, zip(
        lower_bilayer_vector_coordinate, phosphate_geometric_center))
    #upper_bilayer_vector_coordinate = map(sum, zip(
        upper_bilayer_vector_coordinate, phosphate_geometric_center))
65
66 #helix_tilt_angle_file.write("lower helix coord: " + str(
        lower_helix_vector_coordinate) + '\n' + "upper helix coord: " + str(
        upper_helix_vector_coordinate) + '\n' + "lower bilayer coord: " + str
        (lower_bilayer_vector_coordinate) + '\n' + "upper bilayer coord: " +
        str(upper_bilayer_vector_coordinate) + '\n')
67
68 #angle between helix axis and bilayer normal vectors depends on the dot
        product:
69 angle = math.acos(numpy.dot(helix_vector, bilayer_normal))
    #convert to degrees:
70 theta = math.degrees(angle)
    if theta > 90:
        theta = 180-theta
71
72 helix_tilt_angle_file.write(str(ts.frame) + ' ' + str(theta) + '\n') #
    formatting for gnuplot-ready columns
73 print str(folder_name) + ' -- frame: ' + str(ts.frame)
74
75
76
77

```

]

# Bibliography

- [1] Reddy, T., J. Ding, X. Li, B. D. Sykes, J. K. Rainey, and L. Fliegel. 2008. Structural and functional characterization of transmembrane segment IX of the NHE1 isoform of the Na<sup>+</sup>/H<sup>+</sup> exchanger. *J. Biol. Chem.* 283:22018–22030.
- [2] Reddy, T., X. Li, L. Fliegel, B. D. Sykes, and J. K. Rainey. 2010. Correlating structure, dynamics, and function in transmembrane segment VII of the Na<sup>+</sup>/H<sup>+</sup> exchanger isoform 1. *Biochim. Biophys. Acta-Biomembr.* 1798:94–104.
- [3] Reddy, T., and J. K. Rainey. 2010. Interpretation of biomolecular NMR spin relaxation parameters. *Biochem. Cell Biol.* 88:131–142.
- [4] Wang, Y., Y. Zhang, and Y. Ha. 2006. Crystal structure of a rhomboid family intramembrane protease. *Nature* 444:179–183.
- [5] Bordag, N., and S. Keller. 2010. alpha-Helical transmembrane peptides: A “Divide and Conquer” approach to membrane proteins. *Chem. Phys. Lipids* 163:1–26.
- [6] Lipari, G., and A. Szabo. 1982. Model-free approach to the interpretation of nuclear magnetic-resonance relaxation in macromolecules .1. theory and range of validity. *J. Am. Chem. Soc.* 104:4546–4559.
- [7] Lipari, G., and A. Szabo. 1982. Model-free approach to the interpretation of

- nuclear magnetic-resonance relaxation in macromolecules .2. analysis of experimental results. *J. Am. Chem. Soc.* 104:4559–4570.
- [8] Farrow, N., O. Zhang, A. Szabo, D. Torchia, and L. Kay. 1995. Spectral density-function mapping using N-15 relaxation data exclusively. *J. Biomol. NMR* 6:153–162.
- [9] Lefevre, J., K. Dayie, J. Peng, and G. Wagner. 1996. Internal mobility in the partially folded DNA binding and dimerization domains of GAL4: NMR analysis of the N-H spectral density functions. *Biochemistry* 35:2674–2686.
- [10] Freeman, M. 2008. Rhomboid Proteases and their Biological Functions. *Annu. Rev. Genet.* 42:191–210.
- [11] Han, X., M. Mihailescu, and K. Hristova. 2006. Neutron diffraction studies of fluid bilayers with transmembrane proteins: structural consequences of the achondroplasia mutation. *Biophysical journal* 91:3736–47.
- [12] Fliegel, L. 2005. The Na<sup>+</sup>/H<sup>+</sup> exchanger isoform 1. *Int. J. Biochem. Cell Biol.* 37:33–37.
- [13] Wakabayashi, S., T. Pang, X. Su, and M. Shigekawa. 2000. A novel topology model of the human Na<sup>+</sup>/H<sup>+</sup> exchanger isoform 1. *J. Biol. Chem.* 275:7942–7949.
- [14] Landau, M., K. Herz, E. Padan, and N. Ben-Tal. 2007. Model structure of the Na<sup>+</sup>/H<sup>+</sup> exchanger 1 (NHE1) - Functional and clinical implications. *J. Biol. Chem.* 282:37854–37863.
- [15] Orłowski, J., and R. Kandasamy. 1996. Delineation of transmembrane domains of the Na<sup>+</sup>/H<sup>+</sup> exchanger that confer sensitivity to pharmacological antagonists. *J. Biol. Chem.* 271:19922–19927.

- [16] Wang, D., D. Balkovetz, and D. Warnock. 1995. Mutational analysis of transmembrane histidines in the amiloride-sensitive Na<sup>+</sup>/H<sup>+</sup> exchanger. *Am. J. Physiol.-Cell Physiol.* 269:C392–C402.
- [17] Khadilkar, A., P. Iannuzzi, and J. Orłowski. 2001. Identification of sites in the second exomembrane loop and ninth transmembrane helix of the mammalian Na<sup>+</sup>/H<sup>+</sup> exchanger important for drug recognition and cation translocation. *J. Biol. Chem.* 276:43792–43800.
- [18] Noel, J., D. Germain, and J. Vadnais. 2003. Glutamate 346 of human Na<sup>+</sup>-H<sup>+</sup> exchanger NHE1 is crucial for modulating both the affinity for Na<sup>+</sup> and the interaction with amiloride derivatives. *Biochemistry* 42:15361–15368.
- [19] Delaglio, F., S. Grzesiek, G. Vuister, G. Zhu, J. Pfeifer, and A. Bax. 1995. Nmrpipe - a multidimensional spectral processing system based on unix pipes. *J. Biomol. NMR* 6:277–293.
- [20] Goddard, T.D., And Kneller, D. 2001. SPARKY 3.
- [21] Schwieters, C., J. Kuszewski, N. Tjandra, and G. Clore. 2003. The Xplor-NIH NMR molecular structure determination package. *J. Magn. Reson.* 160:65–73.
- [22] Ding, J., J. K. Rainey, C. Xu, B. D. Sykes, and L. Fliegel. 2006. Structural and functional characterization of transmembrane segment VII of the Na<sup>+</sup>/H<sup>+</sup> exchanger isoform 1. *J. Biol. Chem.* 281:29817–29829.
- [23] Andrade, M., P. Chacon, J. Merelo, and F. Moran. 1993. Evaluation of secondary structure of proteins from UV circular-dichroism spectra using an unsupervised learning neural-network. *Protein Eng.* 6:383–390.
- [24] Whitmore, L., and B. Wallace. 2004. DICHROWEB, an online server for pro-

- tein secondary structure analyses from circular dichroism spectroscopic data. *Nucleic Acids Res.* 32:W668–W673.
- [25] Lobley, A., L. Whitmore, and B. Wallace. 2002. DICHROWEB: an interactive website for the analysis of protein secondary structure from circular dichroism spectra. *Bioinformatics* 18:211–212.
- [26] Davis, J., D. Clare, R. Hodges, and M. Bloom. 1983. Interaction of a synthetic amphiphilic polypeptide and lipids in a bilayer structure. *Biochemistry* 22:5298–5305.
- [27] Slepkov, E., J. Rainey, X. Li, Y. Liu, F. Cheng, D. Lindhout, B. Sykes, and L. Fliegel. 2005. Structural and functional characterization of transmembrane segment IV of the NHE1 isoform of the Na<sup>+</sup>/H<sup>+</sup> exchanger. *J. Biol. Chem.* 280:17863–17872.
- [28] Krueger-Koplin, R., P. Sorgen, S. Krueger-Koplin, A. Rivera-Torres, S. Cahill, D. Hicks, L. Grinius, T. Krulwich, and M. Girvin. 2004. An evaluation of detergents for NMR structural studies of membrane proteins. *J. Biomol. NMR* 28:43–57.
- [29] Rainey, J. K., L. Fliegel, and B. D. Sykes. 2006. Strategies for dealing with conformational sampling in structural calculations of flexible or kinked transmembrane peptides. *Biochem. Cell Biol.* 84:918–929.
- [30] Mao, D., E. Wachter, and B. Wallace. 1982. Folding of the mitochondrial proton adenosine-triphosphatase proteolipid channel in phospholipid-vesicles. *Biochemistry* 21:4960–4968.
- [31] Kabsch, W. 1976. Solution for best rotation to relate 2 sets of vectors. *Acta Crystallogr. Sect. A* 32:922–923.



- [32] 4., C. C. P. N. 1994. The CCP4 Suite: Programs for protein crystallography. *Acta Crystallographica Section D Biological Crystallography* 50:760–763.
- [33] Hunt, J., T. Earnest, O. Bousche, K. Kalghatgi, K. Reilly, C. Horvath, K. Rothschild, and D. Engelman. 1997. A biophysical study of integral membrane protein folding. *Biochemistry* 36:15156–15176.
- [34] Katragadda, M., J. Alderfer, and P. Yeagle. 2001. Assembly of a polytopic membrane protein structure from the solution structures of overlapping peptide fragments of bacteriorhodopsin. *Biophys. J.* 81:1029–1036.
- [35] Katragadda, M., A. Chopra, M. Bennett, J. Alderfer, P. Yeagle, and A. Albert. 2001. Structures of the transmembrane helices of the G-protein coupled receptor, rhodopsin. *J. Pept. Res.* 58:79–89.
- [36] Oblattmontal, M., G. Reddy, T. Iwamoto, J. Tomich, and M. Montal. 1994. Identification of an ion channel-forming motif in the primary structure of CFTR, the cystic-fibrosis chloride channel. *Proc. Natl. Acad. Sci. U. S. A.* 91:1495–1499.
- [37] Wigley, W., S. Vijayakumar, J. Jones, C. Slaughter, and P. Thomas. 1998. Transmembrane domain of cystic fibrosis transmembrane conductance regulator: Design, characterization, and secondary structure of synthetic peptides m1-m6. *Biochemistry* 37:844–853.
- [38] Naider, F., S. Khare, B. Arshava, B. Severino, J. Russo, and J. Becker. 2005. Synthetic peptides as probes for conformational preferences of domains of membrane receptors. *Biopolymers* 80:199–213.
- [39] Damberg, P., J. Jarvet, and A. Graslund. 2001. Micellar systems as solvents in peptide and protein structure determination. *In* Nuclear magnetic resonance

- of biological macromolecules, PT B, volume 339 of *Methods in enzymology*. Academic press inc, 525 B street, suite 1900, San Diego, CA 92101-4495 USA, 271–285.
- [40] Henry, G., and B. Sykes. 1994. Methods to study membrane-protein structure in solution. *In* Nuclear magnetic resonance, PT C, volume 239 of *Methods in enzymology*. Academic press inc, 525 B street, suite 1900, San Diego, CA 92101-4495, 515–535.
- [41] Moncoq, K., G. Kemp, X. Li, L. Fliegel, and H. S. Young. 2008. Dimeric structure of human Na<sup>+</sup>/H<sup>+</sup> exchanger isoform 1 overproduced in *Saccharomyces cerevisiae*. *J. Biol. Chem.* 283:4145–4154.
- [42] Hunte, C., E. Screpanti, M. Venturi, A. Rimon, E. Padan, and H. Michel. 2005. Structure of a Na<sup>+</sup>/H<sup>+</sup> antiporter and insights into mechanism of action and regulation by pH. *Nature* 435:1197–1202.
- [43] Reithmeier, R.A., And Deber, C. 1992. No Title. CRC Press, Inc., Boca Raton, FL, 337–393.
- [44] Tang, X., R. Kovacs, D. Sterling, and J. Casey. 1999. Identification of residues lining the translocation pore of human AE1, plasma membrane anion exchange protein. *J. Biol. Chem.* 274:3557–3564.
- [45] Ding, J., R. W. P. Ng, and L. Fliegel. 2007. Functional characterization of the transmembrane segment VII of the NHE1 isoform of the Na<sup>+</sup>/H<sup>+</sup> exchanger. *Can. J. Physiol. Pharmacol.* 85:319–325.
- [46] Dibrov, P., and L. Fliegel. 1998. Comparative molecular analysis of Na<sup>+</sup>/H<sup>+</sup> exchangers: a unified model for Na<sup>+</sup>/H<sup>+</sup> antiport? *FEBS Lett.* 424:1–5.

- [47] Haug, T., D. Sigg, S. Ciani, L. Toro, E. Stefani, and R. Olcese. 2004. Regulation of K<sup>+</sup> flow by a ring of negative charges in the outer pore of BKCa channels. Part I: Aspartate 292 modulates K<sup>+</sup> conduction by external surface charge effect. *J. Gen. Physiol.* 124:173–184.
- [48] Poet, M., M. Tauc, E. Lingueglia, P. Cance, P. Poujeol, M. Lazdunski, and L. Counillon. 2001. Exploration of the pore structure of a peptide-gated Na<sup>+</sup> channel. *Embo J.* 20:5595–5602.
- [49] Wishart, D., C. Bigam, J. Yao, F. Abildgaard, H. Dyson, E. Oldfield, J. Markley, and B. Sykes. 1995. H-1, C-13 and N-15 chemical-shift referencing in biomolecular NMR. *J. Biomol. NMR* 6:135–140.
- [50] Wishart, D., B. Sykes, and F. Richards. 1992. The chemical-shift index - a fast and simple method for the assignment of protein secondary structure through NMR-spectroscopy. *Biochemistry* 31:1647–1651.
- [51] Hyberts, S., M. Goldberg, T. Havel, and G. Wagner. 1992. The solution structure of eglin-c based on measurements of many NOEs and coupling-constants and its comparison with X-ray structures. *Protein Sci.* 1:736–751.
- [52] Sreerama, N., S. Venyaminov, and R. Woody. 1999. Estimation of the number of alpha-helical and beta-strand segments in proteins using circular dichroism spectroscopy. *Protein Sci.* 8:370–380.
- [53] Lees, J. G., A. J. Miles, F. Wien, and B. A. Wallace. 2006. A reference database for circular dichroism spectroscopy covering fold and secondary structure space. *Bioinformatics* 22:1955–1962.
- [54] Morin, S., and S. M. Gagne. 2009. NMR Dynamics of PSE-4 beta-Lactamase: An Interplay of ps-ns Order and mu s-ms Motions in the Active Site. *Biophys. J.* 96:4681–4691.

- [55] Lescop, E., L. Briand, J.-C. Pernellet, and E. Guittet. 2009. Structural Basis of the Broad Specificity of a General Odorant-Binding Protein from Honeybee. *Biochemistry* 48:2431–2441.
- [56] Beierlein, J. M., L. Deshmukh, K. M. Frey, O. Vinogradova, and A. C. Anderson. 2009. The Solution Structure of Bacillus anthracis Dihydrofolate Reductase Yields Insight into the Analysis of Structure-Activity Relationships for Novel Inhibitors. *Biochemistry* 48:4100–4108.
- [57] Johnson, E., L. Bruschiweiler-Li, S. A. Showalter, G. W. Vuister, F. Zhang, and R. Bruschiweiler. 2008. Structure and dynamics of Ca<sup>2+</sup>-binding domain 1 of the Na<sup>+</sup>/Ca<sup>2+</sup> exchanger in the presence and in the absence of Ca<sup>2+</sup>. *J. Mol. Biol.* 377:945–955.
- [58] Lescop, E., Z. Lu, Q. Liu, H. Xu, G. Li, B. Xia, H. Yan, and C. Jin. 2009. Dynamics of the Conformational Transitions in the Assembling of the Michaelis Complex of a Bisubstrate Enzyme: A N-15 Relaxation Study of Escherichia coli 6-Hydroxymethyl-7,8-dihydropterin Pyrophosphokinase. *Biochemistry* 48:302–312.
- [59] Bruschiweiler, R. 2003. New approaches to the dynamic interpretation and prediction of NMR relaxation data from proteins. *Curr. Opin. Struct. Biol.* 13:175–183.
- [60] Daragan, V., and K. Mayo. 1997. Motional model analyses of protein and peptide dynamics using C-13 and N-15 NMR relaxation. *Prog. Nucl. Magn. Reson. Spectrosc.* 31:63–105.
- [61] Dayie, K., G. Wagner, and J. Lefevre. 1996. Theory and practice of nuclear spin relaxation in proteins. *Annu. Rev. Phys. Chem.* 47:243–282.

- [62] Fischer, M., A. Majumdar, and E. Zuiderweg. 1998. Protein NMR relaxation: theory, applications and outlook. *Prog. Nucl. Magn. Reson. Spectrosc.* 33:207–272.
- [63] Ishima, R., and D. Torchia. 2000. Protein dynamics from NMR. *Nat. Struct. Biol.* 7:740–743.
- [64] Jarymowycz, V., and M. Stone. 2006. Fast time scale dynamics of protein backbones: NMR relaxation methods, applications, and functional consequences. *Chem. Rev.* 106:1624–1671.
- [65] Kay, L. 1998. Protein dynamics from NMR. *Nat. Struct. Biol.* 5:513–517.
- [66] Kay, L. 1998. Protein dynamics from NMR. *Biochem. Cell Biol.* 76:145–152.
- [67] Kern, D., and E. Zuiderweg. 2003. The role of dynamics in allosteric regulation. *Curr. Opin. Struct. Biol.* 13:748–757.
- [68] Palmer, A. 1997. Probing molecular motion by NMR. *Curr. Opin. Struct. Biol.* 7:732–737.
- [69] Palmer, A. 2001. NMR probes of molecular dynamics: Overview and comparison with other techniques. *Annu. Rev. Biophys. Biomolec. Struct.* 30:129–155.
- [70] Palmer, A. 2004. NMR characterization of the dynamics of biomacromolecules. *Chem. Rev.* 104:3623–3640.
- [71] Palmer, A., J. Williams, and A. McDermott. 1996. Nuclear magnetic resonance studies of biopolymer dynamics. *J. Phys. Chem.* 100:13293–13310.
- [72] Peng, J., and G. Wagner. 1994. Investigation of protein motions via relaxation measurements. In Nuclear Magnetic Resonance, PT C, volume 239 of *Methods Enzymol.* Academic Press Inc, 525 B street, suite 1900, San Diego, CA 92101-4495, 563–596.

- [73] Spyropoulos, L. 2005. Thermodynamic interpretation of protein dynamics from NMR relaxation measurements. *Protein Pept. Lett.* 12:235–240.
- [74] Spyropoulos, L., and B. Sykes. 2001. Thermodynamic insights into proteins from NMR spin relaxation studies. *Curr. Opin. Struct. Biol.* 11:555–559.
- [75] Stone, M. 2001. NMR relaxation studies of the role of conformational entropy in protein stability and ligand binding. *Accounts Chem. Res.* 34:379–388.
- [76] Purcell, E., H. Torrey, and R. Pound. 1946. Resonance absorption by nuclear magnetic moments in a solid. *Physical Review* 69:37–38.
- [77] Bloch, F., W. Hansen, and M. Packard. 1946. Nuclear induction. *Physical Review* 69:127.
- [78] d’Auvergne, E. J., and P. R. Gooley. 2008. Optimisation of NMR dynamic models I. Minimisation algorithms and their performance within the model-free and Brownian rotational diffusion spaces. *J. Biomol. NMR* 40:107–119.
- [79] Levitt, M. 2008. Spin dynamics: basics of nuclear magnetic resonance. 2nd edition. John Wiley & Sons, Chichester, U.K.
- [80] Cavanagh, J., Fairbrother, W.J., Palmer, A.G., And Skelton, N. 1996. Protein NMR spectroscopy: principles and practice. Academic Press, New York, N.Y.
- [81] Freeman, R. 2003. Spin choreography: basic steps in high resolution NMR. Oxford University Press, Oxford, U.K.
- [82] Eliezer, D., J. Yao, H. Dyson, and P. Wright. 1998. Structural and dynamic characterization of partially folded states of apomyoglobin and implications for protein folding. *Nat. Struct. Biol.* 5:148–155.

- [83] Slupsky, C. M., L. Spyropoulos, V. K. Booth, B. D. Sykes, and M. P. Crump. 2007. Probing nascent structures in peptides using natural abundance C-13 NMR relaxation and reduced spectral density mapping. *Proteins* 67:18–30.
- [84] Muhandiram, D., T. Yamazaki, B. Sykes, and L. Kay. 1995. Measurement of H-2 T-1 and T-1ρ relaxation-times in uniformly C-13-labeled and fractionally H-2-labeled proteins in solution. *J. Am. Chem. Soc.* 117:11536–11544.
- [85] Tugarinov, V., J. Ollerenshaw, and L. Kay. 2005. Probing side-chain dynamics in high molecular weight proteins by deuterium NMR spin relaxation: An application to an 82-kDa enzyme. *J. Am. Chem. Soc.* 127:8214–8225.
- [86] Lee, A., P. Flynn, and A. Wand. 1999. Comparison of H-2 and C-13 NMR relaxation techniques for the study of protein methyl group dynamics in solution. *J. Am. Chem. Soc.* 121:2891–2902.
- [87] Farrow, N., R. Muhandiram, A. Singer, S. Pascal, C. Kay, G. Gish, S. Shoelson, T. Pawson, J. FormanKay, and L. Kay. 1994. Backbone dynamics of a free and a phosphopeptide-complexed SRC homology-2 domain studied by N-15 NMR relaxation. *Biochemistry* 33:5984–6003.
- [88] Ropars, V., S. Bouguet-Bonnet, D. Auguin, P. Barthe, D. Canet, and C. Roume-stand. 2007. Unraveling protein dynamics through fast spectral density mapping. *J. Biomol. NMR* 37:159–177.
- [89] Clore, G., P. Driscoll, P. Wingfield, and A. Gronenborn. 1990. Analysis of the backbone dynamics of interleukin-1-beta using 2-dimensional inverse detected heteronuclear N-15-H-1 NMR-spectroscopy. *Biochemistry* 29:7387–7401.
- [90] Copie, V., J. Battles, J. Schwab, and D. Torchia. 1996. Secondary structure of beta-hydroxydecanoyl thiol ester dehydrase, a 39-kDa protein, derived from

H-alpha, C-alpha, C-beta and CO signal assignments and the chemical shift index: Comparison with the crystal structure. *J. Biomol. NMR* 7:335–340.

- [91] Kay, L., D. Torchia, and A. Bax. 1989. Backbone dynamics of proteins as studied by N-15 inverse detected heteronuclear NMR-spectroscopy - application to staphylococcal nuclease. *Biochemistry* 28:8972–8979.
- [92] Cheng, J., C. Lepre, S. Chambers, J. Fulghum, J. Thomson, and J. Moore. 1993. N-15 NMR relaxation studies of the FK506 binding-protein - backbone dynamics of the uncomplexed receptor. *Biochemistry* 32:9000–9010.
- [93] Williams, K., N. Farrow, C. Deber, and L. Kay. 1996. Structure and dynamics of bacteriophage IKe major coat protein in MPG Micelles by solution NMR. *Biochemistry* 35:5145–5157.
- [94] Yan, C., R. Digate, and R. Guiles. 1999. NMR studies of the structure and dynamics of peptide E, an endogenous opioid peptide that binds with high affinity to multiple opioid receptor subtypes. *Biopolymers* 49:55–70.
- [95] Song, X.-j., P. F. Flynn, K. A. Sharp, and A. J. Wand. 2007. Temperature dependence of fast dynamics in proteins. *Biophys. J.* 92:L43–L45.
- [96] Lee, A., K. Sharp, J. Kranz, X. Song, and A. Wand. 2002. Temperature dependence of the internal dynamics of a calmodulin-peptide complex. *Biochemistry* 41:13814–13825.
- [97] Kallick, D., M. Tessmer, C. Watts, and C. Li. 1995. The use of dodecylphosphocholine micelles in solution NMR. *J. Magn. Reson. Ser. B* 109:60–65.
- [98] Jones, M.N., And Chapman, D. 1994. Micelles, monolayers and biomembranes. Wiley-Liss, New York, N.Y.



- [99] Krishnan, V., and M. Cosman. 1998. An empirical relationship between rotational correlation time and solvent accessible surface area. *J. Biomol. NMR* 12:177–182.
- [100] de la Torre, J., M. Huertas, and B. Carrasco. 2000. HYDRONMR: Prediction of NMR relaxation of globular proteins from atomic-level structures and hydrodynamic calculations. *J. Magn. Reson.* 147:138–146.
- [101] Lee, D., C. Hilty, G. Wider, and K. Wuthrich. 2006. Effective rotational correlation times of proteins from NMR relaxation interference. *J. Magn. Reson.* 178:72–76.
- [102] Kojima, C., A. Ono, M. Kainosho, and T. James. 1999. Quantitative measurement of transverse and longitudinal cross-correlation between C-13-H-1 dipolar interaction and C-13 chemical shift anisotropy: Application to a C-13-labeled DNA duplex. *J. Magn. Reson.* 136:169–175.
- [103] Kroenke, C., J. Loria, L. Lee, M. Rance, and A. Palmer. 1998. Longitudinal and transverse H-1-N-15 dipolar N-15 chemical shift anisotropy relaxation interference: Unambiguous determination of rotational diffusion tensors and chemical exchange effects in biological macromolecules. *J. Am. Chem. Soc.* 120:7905–7915.
- [104] Luginbuhl, P., and K. Wuthrich. 2002. Semi-classical nuclear spin relaxation theory revisited for use with biological macromolecules. *Prog. Nucl. Magn. Reson. Spectrosc.* 40:199–247.
- [105] Clore, G., A. Szabo, A. Bax, L. Kay, P. Driscoll, and A. Gronenborn. 1990. Deviations from the simple 2-parameter model-free approach to the interpretation of N-15 nuclear magnetic-relaxation of proteins. *J. Am. Chem. Soc.* 112:4989–4991.

- [106] Mandel, A., M. Akke, and A. Palmer. 1996. Dynamics of ribonuclease H: Temperature dependence of motions on multiple time scales. *Biochemistry* 35:16009–16023.
- [107] Yang, D., Y. Mok, J. FormanKay, N. Farrow, and L. Kay. 1997. Contributions to protein entropy and heat capacity from bond vector motions measured by NMR spin relaxation. *J. Mol. Biol.* 272:790–804.
- [108] Goodman, J., M. Pagel, and M. Stone. 2000. Relationships between protein structure and dynamics from a database of NMR-derived backbone order parameters. *J. Mol. Biol.* 295:963–978.
- [109] Redfield, C., J. Boyd, L. Smith, R. Smith, and C. Dobson. 1992. Loop mobility in a 4-helix-bundle protein - N-15 NMR relaxation measurements on human interleukin-4. *Biochemistry* 31:10431–10437.
- [110] Mittag, T., and J. D. Forman-Kay. 2007. Atomic-level characterization of disordered protein ensembles. *Curr. Opin. Struct. Biol.* 17:3–14.
- [111] Wright, P., and H. Dyson. 1999. Intrinsically unstructured proteins: Re-assessing the protein structure-function paradigm. *J. Mol. Biol.* 293:321–331.
- [112] Csizmok, V., I. C. Felli, P. Tompa, L. Banci, and I. Bertini. 2008. Structural and Dynamic Characterization of Intrinsically Disordered Human Securin by NMR Spectroscopy. *J. Am. Chem. Soc.* 130:16873–16879.
- [113] Danielsson, J., L. Liljedahl, E. Barany-Wallje, P. Sonderby, L. H. Kristensen, M. A. Martinez-Yamout, H. J. Dyson, P. E. Wright, F. M. Poulsen, L. Maler, A. Graslund, and B. B. Kragelund. 2008. The Intrinsically Disordered RNR Inhibitor Sml1 Is a Dynamic Dimer. *Biochemistry* 47:13428–13437.

- [114] Zhao, X., B. Georgieva, A. Chabes, V. Domkin, J. Ippel, J. Schleucher, S. Wijmenga, L. Thelander, and R. Rothstein. 2000. Mutational and structural analyses of the ribonucleotide reductase inhibitor Sml1 define its Rnr1 interaction domain whose inactivation allows suppression of Mec1 and Rad53 lethality. *Mol. Cell. Biol.* 20:9076–9083.
- [115] Duvignaud, J.-B., C. Savard, R. Fromentin, N. Majeau, D. Leclerc, and S. M. Gagne. 2009. Structure and dynamics of the N-terminal half of hepatitis C virus core protein: An intrinsically unstructured protein. *Biochem. Biophys. Res. Commun.* 378:27–31.
- [116] Buevich, A., and J. Baum. 1999. Dynamics of unfolded proteins: Incorporation of distributions of correlation times in the model free analysis of NMR relaxation data. *J. Am. Chem. Soc.* 121:8671–8672.
- [117] Johnson, E., S. A. Showalter, and R. Bruschweiler. 2008. A multifaceted approach to the interpretation of NMR order parameters: A case study of a dynamic alpha-helix. *J. Phys. Chem. B* 112:6203–6210.
- [118] Bremi, T., and R. Bruschweiler. 1997. Locally anisotropic internal polypeptide backbone dynamics by NMR relaxation. *J. Am. Chem. Soc.* 119:6672–6673.
- [119] Zhang, F., and R. Bruschweiler. 2002. Contact model for the prediction of NMR N-H order parameters in globular proteins. *J. Am. Chem. Soc.* 124:12654–12655.
- [120] Torchia, D., Nicholson, L., Cole, H., And Kay, L. 1993. Heteronuclear NMR studies of the molecular dynamics of staphylococcal nuclease. CRC Press, Inc., Boca Raton, FL, 109–219.
- [121] Palmer, A., M. Rance, and P. Wright. 1991. Intramolecular motions of a zinc finger DNA-binding domain from XFIN characterized by proton-detected nat-

- ural abundance C-12 heteronuclear NMR-spectroscopy. *J. Am. Chem. Soc.* 113:4371–4380.
- [122] Liu, L., and C. Deber. 1998. Uncoupling hydrophobicity and helicity in transmembrane segments - alpha-helical propensities of the amino acids in non-polar environments. *J. Biol. Chem.* 273:23645–23648.
- [123] Jones, D., W. Taylor, and J. Thornton. 1994. A model recognition approach to the prediction of all-helical membrane-protein structure and topology. *Biochemistry* 33:3038–3049.
- [124] Murtazina, R., B. Booth, B. Bullis, D. Singh, and L. Fliegel. 2001. Functional analysis of polar amino-acid residues in membrane associated regions of the NHE1 isoform of the mammalian Na<sup>+</sup>/H<sup>+</sup> exchanger. *Eur. J. Biochem.* 268:4674–4685.
- [125] Zhang, Y., R. Lewis, R. Hodges, and R. Mcelhaney. 1995. Interaction of a peptide model of a hydrophobic transmembrane alpha-helical segment of a membrane-protein with phosphatidylethanolamine bilayers - differential scanning calorimetric and fourier-transform infrared spectroscopic studies. *Biophys. J.* 68:847–857.
- [126] Kay, L., P. Keifer, and T. Saarinen. 1992. Pure absorption gradient enhanced heteronuclear single quantum correlation spectroscopy with improved sensitivity. *J. Am. Chem. Soc.* 114:10663–10665.
- [127] Taylor, J. 1997. *An Introduction To Error Analysis*. 2nd edition. University Science Books, Sausalito, CA.
- [128] d’Auvergne, E. J., and P. R. Gooley. 2008. Optimisation of NMR dynamic models II. A new methodology for the dual optimisation of the model-free parameters and the Brownian rotational diffusion tensor. *J. Biomol. NMR* 40:121–133.

- [129] Abragam, A. 1961. *The Principles of Nuclear Magnetism*. Clarendon Press, Oxford.
- [130] Hiyama, Y., C. Niu, J. Silverton, A. Bavoso, and D. Torchia. 1988. Determination of N-15 chemical-shift tensor via N-15-H-2 dipolar coupling in BOC-GLYCYLGLYCYL[N-15]glycine benzyl ester. *J. Am. Chem. Soc.* 110:2378–2383.
- [131] Spyropoulos, L. 2006. A suite of Mathematica notebooks for the analysis of protein main chain N-15 NMR relaxation data. *J. Biomol. NMR* 36:215–224.
- [132] Mandel, A., M. Akke, and A. Palmer. 1995. Backbone dynamics of Escherichia coli ribonuclease HI - correlations with structure and function in an active enzyme. *J. Mol. Biol.* 246:144–163.
- [133] d’Auvergne, E., and P. Gooley. 2003. The use of model selection in the model-free analysis of protein dynamics. *J. Biomol. NMR* 25:25–39.
- [134] Peng, J., and G. Wagner. 1995. Frequency spectrum of NH bonds in eglin c from spectral density mapping at multiple fields. *Biochemistry* 34:16733–16752.
- [135] Mulder, F., A. Mittermaier, B. Hon, F. Dahlquist, and L. Kay. 2001. Studying excited states of proteins by NMR spectroscopy. *Nat. Struct. Biol.* 8:932–935.
- [136] Langelaan, D. N., E. M. Bebbington, T. Reddy, and J. K. Rainey. 2009. Structural Insight into G-Protein Coupled Receptor Binding by Apelin. *Biochemistry* 48:537–548.
- [137] Lee, B. L., X. Li, Y. Liu, B. D. Sykes, and L. Fliegel. 2009. Structural and Functional Analysis of Transmembrane XI of the NHE1 Isoform of the Na<sup>+</sup>/H<sup>+</sup> Exchanger. *J. Biol. Chem.* 284:11546–11556.

- [138] Mayer, U., and C. Nussleinvolhard. 1988. A group of genes required for pattern-formation in the ventral ectoderm of the drosophila embryo. *Genes Dev.* 2:1496–1511.
- [139] Lee, J., S. Urban, C. Garvey, and M. Freeman. 2001. Regulated intracellular ligand transport and proteolysis control EGF signal activation in *Drosophila*. *Cell* 107:161–171.
- [140] Urban, S., J. Lee, and M. Freeman. 2001. *Drosophila* Rhomboid-1 defines a family of putative intramembrane serine proteases. *Cell* 107:173–182.
- [141] Strisovsky, K., H. J. Sharpe, and M. Freeman. 2009. Sequence-Specific Intramembrane Proteolysis: Identification of a Recognition Motif in Rhomboid Substrates. *Mol. Cell* 36:1048–1059.
- [142] Stevenson, L. G., K. Strisovsky, K. M. Clemmer, S. Bhatt, M. Freeman, and P. N. Rather. 2007. Rhomboid protease AarA mediates quorum-sensing in *Providencia stuartii* by activating TatA of the twin-arginine translocase. *Proc. Natl. Acad. Sci. U. S. A.* 104:1003–1008.
- [143] Nicolle, L. 2002. Resistant pathogens in urinary tract infections. *J. Am. Geriatr. Soc.* 50:S230–S235.
- [144] Baker, R. P., R. Wijetilaka, and S. Urban. 2006. Two *Plasmodium* rhomboid proteases preferentially cleave different adhesins implicated in all invasive stages of malaria. *PLoS Pathog.* 2:922–932.
- [145] McQuibban, G., S. Saurya, and M. Freeman. 2003. Mitochondrial membrane remodelling regulated by a conserved rhomboid protease. *Nature* 423:537–541.
- [146] Cipolat, S., T. Rudka, D. Hartmann, V. Costa, L. Serneels, K. Craessaerts, K. Metzger, C. Frezza, W. Annaert, L. D’Adamio, C. Derks, T. Dejaegere,

- L. Pellegrini, R. D’Hooge, L. Scorrano, and B. De Strooper. 2006. Mitochondrial rhomboid PARL regulates cytochrome c release during apoptosis via OPA1-dependent cristae remodeling. *Cell* 126:163–175.
- [147] Lemberg, M., J. Menendez, A. Misik, M. Garcia, C. Koth, and M. Freeman. 2005. Mechanism of intramembrane proteolysis investigated with purified rhomboid proteases. *Embo J.* 24:464–472.
- [148] Baker, R. P., K. Young, L. Feng, Y. Shi, and S. Urban. 2007. Enzymatic analysis of a rhomboid intramembrane protease implicates transmembrane helix 5 as the lateral substrate gate. *Proc. Natl. Acad. Sci. U. S. A.* 104:8257–8262.
- [149] White, S. H. 2006. Rhomboid intramembrane protease structures galore! *Nat. Struct. Mol. Biol.* 13:1049–1051.
- [150] Vinothkumar, K. R., K. Strisovsky, A. Andreeva, Y. Christova, S. Verhelst, and M. Freeman. 2010. The structural basis for catalysis and substrate specificity of a rhomboid protease. *Embo J.* 29:3797–3809.
- [151] Kaiser, E., R. Colescot, C. Bossinge, and P. Cook. 1970. Color test for detection of free terminal amino groups in solid-phase synthesis of peptides. *Anal. Biochem.* 34:595–&.
- [152] Kaiser, E., C. Bossinger, R. Colecott, and D. Olsen. 1980. Color test for terminal prolyl residues in the solid-phase synthesis of peptides. *Anal. Chim. Acta* 118:149–151.
- [153] Lei, X., K. Ahn, L. Zhu, I. Ubarretxena-Belandia, and Y.-M. Li. 2008. Soluble Oligomers of the Intramembrane Serine Protease YqgP Are Catalytically Active in the Absence of Detergents. *Biochemistry* 47:11920–11929.

- [154] Kirin, S. I., F. Noor, N. Metzler-Nolte, and W. Mier. 2007. Manual solid-phase peptide synthesis of metallocene-peptide bioconjugates. *J. Chem. Educ.* 84:108–111.
- [155] Eswarakumar, V., I. Lax, and J. Schlessinger. 2005. Cellular signaling by fibroblast growth factor receptors. *Cytokine Growth Factor Rev.* 16:139–149.
- [156] Ornitz, D. 2005. FGF signaling in the developing endochondral skeleton. *Cytokine Growth Factor Rev.* 16:205–213.
- [157] Chen, H., J. Ma, W. Li, A. V. Eliseenkova, C. Xu, T. A. Neubert, W. T. Miller, and M. Mohammadi. 2007. A molecular brake in the kinase hinge region regulates the activity of receptor tyrosine kinases. *Mol. Cell* 27:717–730.
- [158] Johnson, D., and L. Williams. 1993. Structural and functional diversity in the FGF receptor multigene family. *Adv. Cancer Res.* 60:1–41.
- [159] Partanen, J., S. Vainikka, and K. Alitalo. 1993. Structural and functional specificity of FGF receptors. *Philos. Trans. R. Soc. Lond. Ser. B-Biol. Sci.* 340:297–303.
- [160] Chellaiah, A., D. McEwen, S. Werner, J. Xu, and D. Ornitz. 1994. Fibroblast growth-factor receptor (FGFR)-3 - alternative splicing in immunoglobulin-like domain-III creates a receptor highly specific for acidic FGF FGF-1. *J. Biol. Chem.* 269:11620–11627.
- [161] Basilico, C., and D. Moscatelli. 1992. The FGF family of growth-factors and oncogenes. *Adv. Cancer Res.* 59:115–165.
- [162] Francomano, C., R. Deluna, T. Hefferon, G. Bellus, C. Turner, E. Taylor, D. Meyers, S. Blanton, J. Murray, I. McIntosh, and J. Hecht. 1994. Localiza-



- tion of the achondroplasia gene to the distal 2.5 MB of human-chromosome-4P. *Hum. Mol. Genet.* 3:787–792.
- [163] Shiang, R., L. Thompson, Y. Zhu, D. Church, T. Fielder, M. Bocian, S. Winokur, and J. Wasmuth. 1994. Mutations in the transmembrane domain of FGFR3 cause the most common genetic form of dwarfism, achondroplasia. *Cell* 78:335–342.
- [164] Rousseau, F., J. Bonaventure, L. Legeaimallet, A. Pelet, J. Rozet, P. Maroteaux, M. Lemerrer, and A. Munnich. 1994. Mutations in the gene encoding fibroblast growth-factor receptor-3 in achondroplasia. *Nature* 371:252–254.
- [165] Bellus, G. A., T. W. Hefferon, R. I. Ortiz de Luna, J. T. Hecht, W. A. Horton, M. Machado, I. Kaitila, I. McIntosh, and C. A. Francomano. 1995. Achondroplasia is defined by recurrent G380R mutations of FGFR3. *American journal of human genetics* 56:368–73.
- [166] Jones, K. 1988. Smith's recognizable patterns of human malformation. 4th edition. WB Saunders, Philadelphia.
- [167] Gorlin, R., M. Cohen, and L. Levin. 1990. Syndromes of the head and neck. 3rd edition. Oxford University Press, New York.
- [168] Murdoch, J., B. Walker, J. Hall, H. Abbey, K. Smith, and V. McKusick. 1970. Achondroplasia - a genetic and statistical survey. *Ann. Hum. Genet.* 33:227–&.
- [169] Oberklaid, F., D. Danks, F. Jensen, L. Stace, and S. Rosshandler. 1979. Achondroplasia and hypochondroplasia - comments on frequency, mutation-rate, and radiological features in skull and spine. *J. Med. Genet.* 16:140–146.

- [170] Stoll, C., B. Dott, M. Roth, and Y. Alembik. 1989. Birth prevalence rates of skeletal dysplasias. *Clin. Genet.* 35:88–92.
- [171] Hecht, J., C. Francomano, W. Horton, and J. Annegers. 1987. Mortality in achondroplasia. *Am. J. Hum. Genet.* 41:454–464.
- [172] Webster, M. K., and D. J. Donoghue. 1996. Constitutive activation of fibroblast growth factor receptor 3 by the transmembrane domain point mutation found in achondroplasia. *The EMBO journal* 15:520–7.
- [173] You, M., E. Li, and K. Hristova. 2006. The achondroplasia mutation does not alter the dimerization energetics of the fibroblast growth factor receptor 3 transmembrane domain. *Biochemistry* 45:5551–6.
- [174] Schrödinger, LLC. 2010. The PyMOL molecular graphics system, version 1.3r1.
- [175] Marrink, S. J., A. H. de Vries, and A. E. Mark. 2004. Coarse grained model for semiquantitative lipid simulations. *The Journal of Physical Chemistry B* 108:750–760.
- [176] Bond, P. J., and M. S. P. Sansom. 2006. Insertion and assembly of membrane proteins via simulation. *Journal of the American Chemical Society* 128:2697–2704.
- [177] Bond, P. J., J. Holyoake, A. Ivetac, S. Khalid, and M. S. P. Sansom. 2007. Coarse-grained molecular dynamics simulations of membrane proteins and peptides. *J. Struct. Biol.* 157:593–605.
- [178] Lindahl, E., B. Hess, and D. van der Spoel. 2001. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *J. Mol. Model.* 7:306–317.
- [179] Monticelli, L., S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, and

- S.-J. Marrink. 2008. The MARTINI coarse-grained force field: Extension to proteins. *J. Chem. Theory Comput.* 4:819–834.
- [180] Marrink, S. J., H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. de Vries. 2007. The martini force field: coarse grained model for biomolecular simulations. *The Journal of Physical Chemistry B* 111:7812–7824. PMID: 17569554.
- [181] Berendsen, H., J. Postma, W. Vangunsteren, A. Dinola, and J. Haak. 1984. Molecular-dynamics with coupling to an external bath. *J. Chem. Phys.* 81:3684–3690.
- [182] Humphrey, W., A. Dalke, and K. Schulten. 1996. VMD: Visual molecular dynamics. *J. Mol. Graph.* 14:33–&.
- [183] Michaud-Agrawal, N., E. J. Denning, T. B. Woolf, and O. Beckstein. 2011. Md-analysis: A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry* :n/a–n/a.
- [184] Psachoulia, E., P. W. Fowler, P. J. Bond, and M. S. P. Sansom. 2008. Helix-helix interactions in membrane proteins: Coarse-grained simulations of glycophorin A helix dimerization. *Biochemistry* 47:10503–10512. PMID: 18783247.
- [185] Sengupta, D., and S. J. Marrink. 2010. Lipid-mediated interactions tune the association of glycoporphin a helix and its disruptive mutants in membranes. *Phys. Chem. Chem. Phys.* 12:12987–12996.
- [186] MacKenzie, K. R., J. H. Prestegard, and D. M. Engelman. 1997. A Transmembrane Helix Dimer: Structure and Implications. *Science* 276:131–133.
- [187] Smith, S. O., D. Song, S. Shekar, M. Groesbeek, M. Ziliox, and S. Aimoto. 2001. Structure of the transmembrane dimer interface of glycoporphin A in membrane bilayers. *Biochemistry* 40:6553–6558. PMID: 11380249.

- [188] Finger, C., C. Escher, and D. Schneider. 2009. The Single Transmembrane Domains of Human Receptor Tyrosine Kinases Encode Self-Interactions. *Sci. Signal.* 2:ra56–.
- [189] Treutlein, H. R., M. A. Lemmon, D. M. Engelman, and A. Brunger. 1992. The glycophorin A transmembrane domain dimer: Sequence-specific propensity for a right-handed supercoil of helices. *Biochemistry* 31:12726–12732. PMID: 1463744.
- [190] Meyers, G. A., S. J. Orlow, I. R. Munro, K. A. Przylepa, and E. W. Jabs. 1995. Fibroblast growth factor receptor 3 (FGFR3) transmembrane mutation in Crouzon syndrome with acanthosis nigricans. *Nature genetics* 11:462–4.
- [191] Martínez-Frías, M. L., C. A. de Frutos, E. Bermejo, and M. A. Nieto. 2010. Review of the recently defined molecular mechanisms underlying thanatophoric dysplasia and their potential therapeutic implications for achondroplasia. *American journal of medical genetics. Part A* 152A:245–55.
- [192] Peng, W. C., X. Lin, and J. Torres. 2009. The strong dimerization of the transmembrane domain of the fibroblast growth factor receptor (FGFR) is modulated by C-terminal juxtamembrane residues. *Protein Science* 18:450–9.
- [193] You, M., J. Spangler, E. Li, X. Han, P. Ghosh, and K. Hristova. 2007. Effect of pathogenic cysteine mutations on fgfr3 transmembrane domain dimerization in detergents and lipid bilayers. *Biochemistry* 46:11039–11046. PMID: 17845056.
- [194] Daura, X., K. Gademann, B. Jaun, D. Seebach, W. van Gunsteren, and A. Mark. 1999. Peptide folding: When simulation meets experiment. *Angew. Chem.-Int. Edit.* 38:236–240.
- [195] Bondar, A.-N., C. del Val, and S. H. White. 2009. Rhomboid Protease Dynamics and Lipid Interactions. *Structure* 17:395–405.

- [196] Vostrikov, V. V., B. A. Hall, D. V. Greathouse, R. E. Koeppe, and M. S. P. Sansom. 2010. Changes in transmembrane helix alignment by arginine residues revealed by solid-state NMR experiments and coarse-grained MD simulations. *Journal of the American Chemical Society* 132:5803–5811.
- [197] Moriki, T., H. Maruyama, and I. Maruyama. 2001. Activation of preformed EGF receptor dimers by ligand-induced rotation of the transmembrane domain. *J. Mol. Biol.* 311:1011–1026.
- [198] Beevers, A. J., A. Damianoglou, J. Oates, A. Rodger, and A. M. Dixon. 2010. Sequence-Dependent Oligomerization of the Neu Transmembrane Domain Suggests Inhibition of “Conformational Switching” by an Oncogenic Mutant. *Biochemistry* 49:2811–2820.
- [199] Urban, S., and M. Wolfe. 2005. Reconstitution of intramembrane proteolysis in vitro reveals that pure rhomboid is sufficient for catalysis and specificity. *Proc. Natl. Acad. Sci. U. S. A.* 102:1883–1888.
- [200] Urban, S. 2006. Rhomboid proteins: conserved membrane proteases with divergent biological functions. *Genes Dev.* 20:3054–3068.
- [201] Tzeng, J., B. L. Lee, B. D. Sykes, and L. Fliegel. 2010. Structural and Functional Analysis of Transmembrane Segment VI of the NHE1 Isoform of the Na<sup>+</sup>/H<sup>+</sup> Exchanger. *J. Biol. Chem.* 285:36656–36665.
- [202] Laederich, M. B., and W. A. Horton. 2010. Achondroplasia: pathogenesis and implications for future treatment. *Curr. Opin. Pediatr.* 22:516–523.