ABSTRACT

LAPIN, JOEL. Advanced Computational Techniques for Structure Determination of Membrane Proteins by Solid-state NMR. (Under the direction of Dr. Alexander Nevzorov).

Utilizing solid-state NMR spectroscopy of aligned membrane proteins has the potential to yield high-resolution structures for these important biological constituents. Generally, upon acquisition of their NMR spectra, the resonances must be 1) assigned and then 2) the structure must be calculated while satisfying the experimental NMR restraints. Both processes are, in general, affected by ambiguity and degeneracy in the interpretation of the spectra, owing in part to broad linewidths and an insufficient number of NMR measurements. For spectroscopic assignment we have developed an automated algorithm that can assign 2D homonuclear and heteronuclear spectra based on the crosspeak pattern from the spin-exchange experiments. By treating the spectral contours as a "pseudopotential", the main peak assignment can be scored based on how well the calculated crosspeaks match the experimental intensity. The best score yields the most plausible assignment which best fits the spectra. The developed Monte Carlo/Simulated annealing algorithm to find the highest-scoring main peak order is first exemplified on synthetic spectra, and then demonstrated on two experimental datasets, i.e. Pf1 coat protein reconstituted in magnetically aligned bicelles and within the entire Pf1 bacteriophage supramolecular assembly. In the former case the previous assignment that was established via selective labeling experiments was recovered in an automated fashion. For the latter dataset, a new assignment was suggested by the algorithm, simultaneously satisfying both the selective labeling spectra and the assumed helical pattern for the crosspeaks. For structure determination we have significantly improved a previous algorithm which has resulted in more accurate structures, as well as developed a protocol to utilize Rosetta bioinformatics scoring functions to screen the fitted solutions yielding the most plausible structures. Since 3D NMR spectra are not yet routinely available, this protocol was demonstrated on synthetic spectra generated from PDBs 2gb1, a soluble protein, and 4a2n, a membrane protein. The results provide a road map for future structure calculations. Based on the trials with the synthetic spectra, and using a new pulse sequence for correlating $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings with ^{15}N CSA, the protocol has been applied on the dataset derived from triple-resonance NMR spectra of Pf1 coat protein in magnetically aligned bicelles. Structure calculations were carried out without biasing the ϕ, ψ torsion search, as was previously necessary for 2D spectra, and produced a

consensus alpha-helical solution. Finally, simulations were utilized for NMR pulse sequence optimization, producing spectra with superior ^{1}H-^{15}N dipolar coupling linewidths. A Monte Carlo search protocol implemented on a GPU computer was used to extensively simulate NMR experiments in order to optimize the pulse phases and durations that make up a pulse sequence. Predicted pulse sequences were tested on an NAL crystal and Leucine labeled Pf1 coat protein in magnetically aligned bicelles. The resulting pulse sequence, termed ROULETTE-1, resulted in linewidths 18% sharper than the SAMPI4 pulse sequence on the crystal, and 14% sharper for the Pf1 sample.

Advanced Computational Techniques for Structure Determination of Membrane Proteins by Solid-state NMR

by
Joel Lapin

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Chemistry

Raleigh, North Carolina

2020

APPROVED BY:

_____	_____
Alexander Nevzorov Chair of Advisory Committee	Stefan Franzen
_____	_____
Alex I. Smirnov	Thomas Theis

BIOGRAPHY

Joel Lapin was born on May 25, 1990 in Evanston, Illinois. He grew up in Deerfield, Illinois with his 2 older sisters, Marni and Ashley, and his parents Greg and Jill Lapin. He attended Deerfield public schools all the way up through receiving his high school diploma from Deerfield high school in 2008. He graduated with a BS in Biology from Illinois State University in 2012, and an MS in Materials Science from North Carolina State University in 2017. He joined Dr. Alexander Nevzorov's lab in August 2017, where he has pursued his interests at the interface of Physical Chemistry and computational science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction

Nuclear Magnetic Resonance is an exceptionally versatile technique that allows one to obtain structural and dynamic information about biological macromolecules at atomic resolution. NMR spectroscopy takes advantage of the magnetic properties of various atomic nuclei having non-zero spins. In proteins these nuclei could include ^{1}H, ^{14}N, ^{15}N, ^{13}C, which are all present in the polypeptide backbone. In general, when a magnetic field B_0 is applied, a nuclear spin's energy will split based on whether the spin's angular momentum is aligned or opposed to the field. This is referred to as the Zeeman effect. The Zeeman Hamiltonian can be written as:

$$\hat{H}_Z = -\hat{\mu} \cdot \vec{B}_0 \qquad (Eq.\,1.1)$$

Here the magnetic dipole operator $\hat{\mu}$ is a product of the gyromagnetic ratio, γ, and the spin angular moment operator $\hat{I_z}$. Choosing the magnetic field along the z-axis, one can write:

$$\hat{H}_Z = -\gamma \hat{I}_z B_0 \qquad (Eq.\,1.2)$$

For $S = 1/2$ spins:

$$\hat{I}_z|+\rangle = \frac{1}{2}\hbar|+\rangle \qquad (Eq.\,1.3)$$

$$\hat{I}_z|-\rangle = -\frac{1}{2}\hbar|-\rangle \qquad (Eq.\,1.4)$$

Projection angular momentum operator $\hat{I}_z$, when applied to spin up ($|+\rangle$ or α) or spin down ($|-\rangle$ or β) states, will yield the expectation values $\pm\frac{1}{2}\hbar$ (Eqs. 1.3 and 1.4). Thus, for a spin ½ nucleus, the application of the magnetic field causes a splitting between the energy levels corresponding to spins' z-projection being aligned along or opposite to the magnetic field (cf. Fig. 1.1a).

$$E_{\pm 1/2} = \pm\frac{1}{2}\gamma\hbar B_0 \qquad (Eq.\,1.5)$$

$$\Delta E = \gamma\hbar B_0 = \hbar\omega \qquad (Eq.\,1.6)$$

Introduced in Eq. 1.6 is the Larmor frequency ω, which is specific to the chemical identity of the spin, via the gyromagnetic ratio.

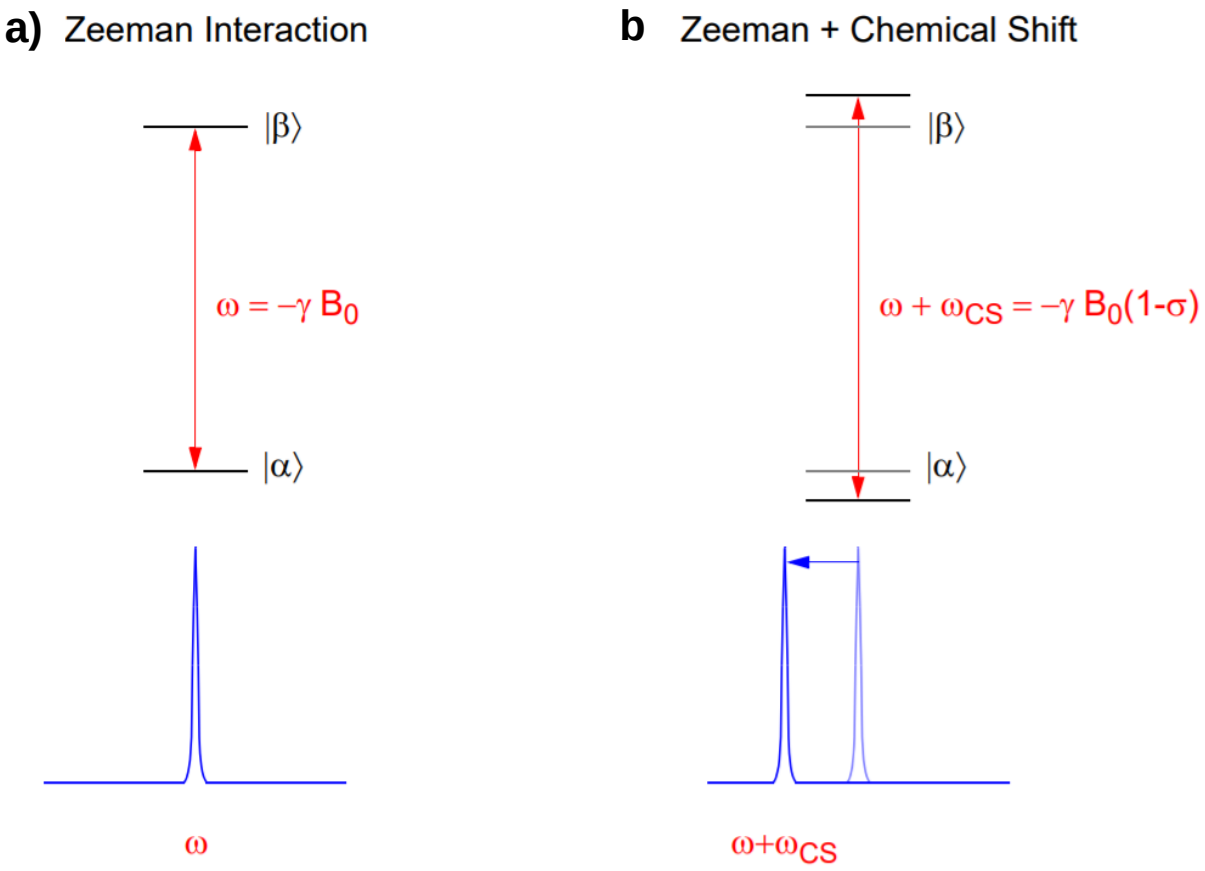


Figure 1.1. Splitting of the α and β energy levels of a spin in a magnetic field B_0, proportional to the Larmor frequency ω. The Zeeman interaction alone in (a) leads to a splitting given by eqs. 1.5 and 1.6. Inclusion of chemical shift interactions in (b) causes variations in the energy gap due to changes in the local environment, as in eq. 1.9. Figure obtained from Ref. (Verel, 2013).

The incredible usefulness and versatility of NMR spectroscopy is due in large part to the perturbations arising from the local electronic environment surrounding each nuclear spin. The application of B_0 affects the motions of the electrons around their associated nucleus, which generates a local magnetic field. This local field diminishes (in accordance with Lenz's law) the effect of B_0 and is referred to as the chemical shift interaction. This interaction is represented by the product of a chemical shift tensor σ^k with B_0.

$$\hat{H}_{CS}^{k} = -\gamma_k \hat{I}_z \sigma^k B_0 \qquad (Eq. 1.7)$$

$$\hat{H}_{CS}^{k} = -\omega(\sigma_{xz}^{k}\hat{I}_{kx} + \sigma_{yz}^{k}\hat{I}_{ky} + \sigma_{zz}^{k}\hat{I}_{kz}) \qquad (Eq.1.8)$$

Under a sufficiently high B_0 field, the Hamiltonian can be approximated to contain only the σ_{zz} component (the high field approximation).

$$\hat{H}_{CS}^{k} = -\omega\sigma_{zz}^{k}\hat{I}_{kz} \qquad (Eq.1.9)$$

Combining this with the original Zeeman Hamiltonian yields a small, but experimentally measurable effect on the nuclear spins.

$$\hat{H}_{Z}^{k} + \hat{H}_{CS}^{k} = \omega\hat{I}_{kz} - \sigma_{zz}^{k}\hat{I}_{kz}\omega = (1 - \sigma_{zz}^{k})\omega\hat{I}_{kz} \qquad (Eq.1.10)$$

Measurements of chemical shifts, on the order of Hz, are reported in parts per million (ppm) due to their minor magnitude in comparison to the spectrometer frequency, which is on the order of ~10^2 MHz. The chemical shift interaction can reveal information about the local electronic environment for the nuclear spin. Conversely, if the chemical environment is already known, chemical shifts can reveal the orientation of its tensor relative to the magnetic field in macroscopically oriented samples, which is useful in interpreting spectra of protein backbones.

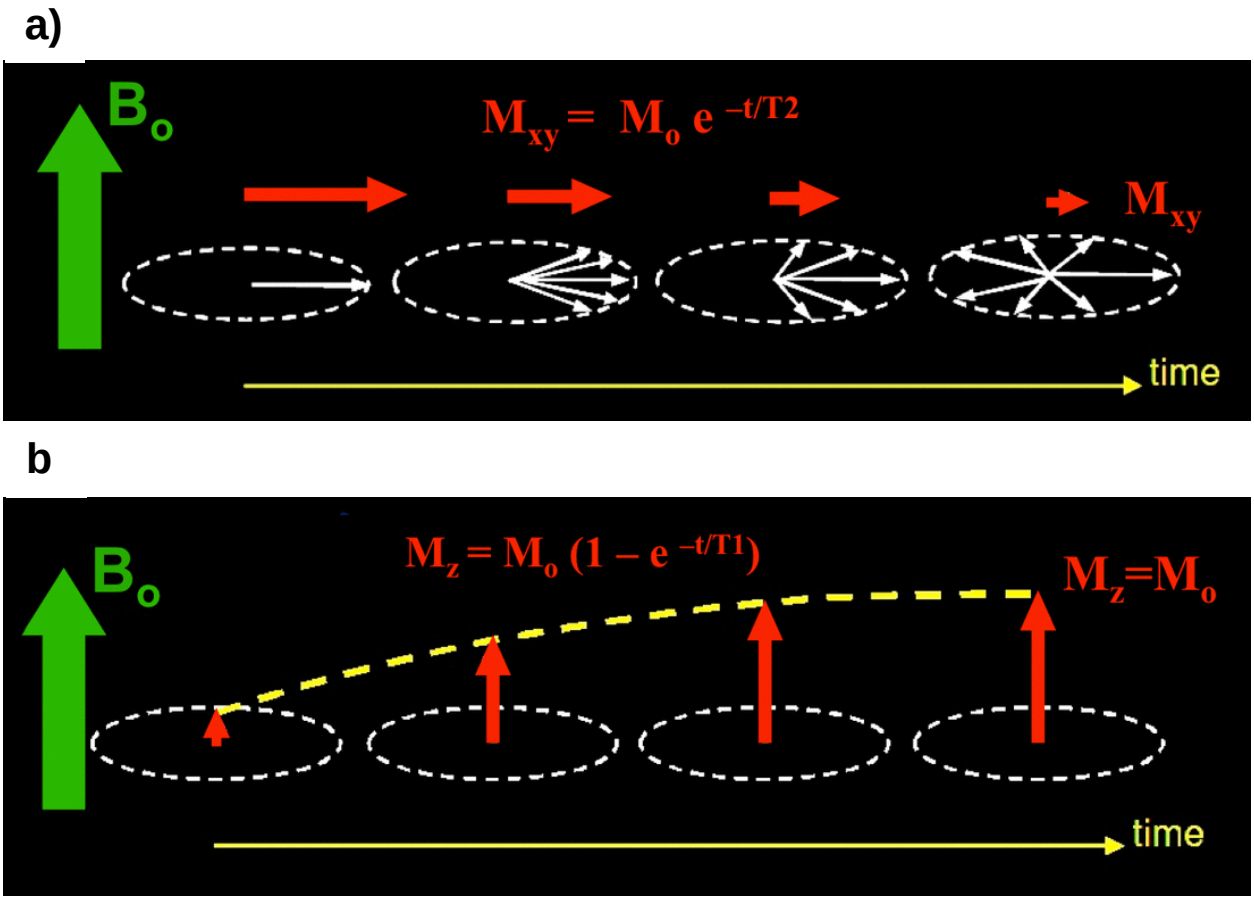


Figure 1.2. T2 and T1 relaxation in NMR. T2 relaxation, shown in (a), involves the loss of transverse magnetization as spins lose phase coherence. T1 relaxation in (b) shows the gradual growth of longitudinal magnetization under the influence of magnetic field B_0. Figure obtained from ref. (Elster 2020).

Central to NMR spectroscopy is the consideration of sources of relaxation in a spin system. The relaxation rate will ultimately determine the sharpness of measured resonances in an acquired NMR spectrum, commonly referred to as spectroscopic resolution. The transverse (T_2), or spin-spin, relaxation refers to the decay of the transverse magnetization (in the x-y plane) due to the loss of synchronicity of the individual precessing nuclear spins.

$$M_{xy}(t) = M_{xy}(0)\mathrm{e}^{-\frac{t}{T_2}} \qquad (Eq.\,1.11)$$

After applying a 90° rf-pulse, the nuclear spins that were previously aligned with B_0 (in the z-direction) now precess, at the Larmor frequency, in the xy-plane at the Larmor frequency (cf. Fig. 2). Syncronicity of the precession frequency will gradually be lost, largely by several mechanisms.

Spin-spin relaxation is the loss of synchronicity in the precessing frequencies of the individual spins due to their interaction, typically via magnetic dipole-dipole interaction. Even in the absence of spin-spin relaxation, spatial and temporal inhomogeneities in the magnetic field will cause each nuclear spin within the sample to fall out of phase coherence with each other. Moreover, each precessing nuclear spin generates its own fluctuating local magnetic field, and while much smaller in scale to B_0, these varying fields will be felt by other nearby coupled spins. The ultimate result is for the individual spins to start precessing at slightly different frequencies, which will tend to dissipate the overall measured signal, and leads to line broadening in the spectrum. Spin relaxation can also occur via spin-lattice relaxation, which is also known as longitudinal relaxation. Longitudinal relaxation refers to the proclivity of a spin system to establish its equilibrium Boltzmann distribution between the spin-up (α) state and the spin-down (β) state by exchanging its Zeeman energy quanta with the surroundings. Either after the application of a strong external magnetic field B_0 on a static sample, or after an additional rf-pulse that establishes a transverse magnetization, the average of the spin system's angular momentum will grow in the z-direction towards its equilibrium state. For instance, following a 90-degree pulse, the z-magnetization will relax back to equilibrium as:

$$M_z(t) = M_z(0)\left(1 - e^{-\frac{t}{T1}}\right) \qquad (Eq.\,1.12)$$

Longitudinal relaxation occurs because the spin system is exchanging energy with its surrounding lattice as it establishes Boltzmann equilibrium with its surroundings, which is a relatively slow process.

These aspects of NMR that allow for determination of high resolution chemical information can be very effectively utilized in solid-state NMR (ssNMR) for structure determination of aligned proteins. Over the last two decades, ssNMR of uniaxially aligned samples has emerged as a useful tool for structure determination of membrane proteins incorporated in planar, lipid-rich hydrated bilayers (McDonnell and Opella, 1993; Opella, S.J. et al., 1999; Wang et al., 2001; Marassi and Opella, 2003; Traaseth et al., 2006; Traaseth et al., 2009; Sharma et al., 2010; Verardi et al., 2011; Gayen et al., 2013; Gleason et al., 2013; Yamamoto et al., 2013). With the help of specialized sequences of rf-pulses, the NMR resonances along the backbone information about their local orientation with respect to the external magnetic field, B_0, via the

frequencies of their spectral resonances. Before the structure can be calculated one must find an accurate spectroscopic assignment. Spectroscopic assignment in NMR of proteins provides the mapping of spectral resonances to their sequential positions within the amino-acid primary chain (protein backbone). Spectroscopic assignment thus represents a most essential step in the determination of structure and dynamics of soluble and membrane proteins (Castellani et al., 2002; Etzkorn et al., 2007; Heise et al., Khitrin et al., 2011; McDermott, 2009, Park et al., 2010; Tycko et al., 2006).

The assignment process could be considerably simplified by devising an automated algorithm that would identify the crosspeaks via their intensity, imposing the selective labeling constraints, and scoring the possible assignment solutions in order to find the best fit of all the experimental spectra. Automation has the benefit not only for speeding up the process of sequential assignment, but also for justifying the assignment based on additional objective criteria. Spectral peaks are defined by their intensity above a certain cutoff relative to the noise level. For samples exhibiting suboptimal resolution a justification for the correct peak assignment can be based on mapping the calculated crosspeak pattern onto the spectrum and covering as much experimental peak intensity as possible. Conversely, the calculated crosspeaks that lie in the areas of little or no intensity may indicate incorrect sequential assignment. The automated approach can easily incorporate all the aforementioned constraints while searching for the possible peak assignments. Ideally, such an algorithm would need a minimal experimental input and be robust enough to converge to a single solution. Realistically, however, the final assignment step needs extra discernment to evaluate the viability of each of the possible solutions based on certain post-fitting scoring criteria.

Chapter 2 describes a protocol for applying an automated algorithm to find the best scoring assignments on 2D solid-state spectra, as well as post-program filtering of the solutions. This protocol is exemplified on synthetic spectra, as well as two real spectra with selective labeling experiments.

After an NMR spectrum of a membrane protein has been assigned its structure can be calculated. Structure determination from angular restraints relies on the measured frequencies to determine the possible peptide plane orientations along the polypeptide chain. In the mapping of resonance frequencies to peptide plane orientations, finding a consensus structure is primarily hindered by the degeneracy of the possible solutions. Using a simplex minimizer to find the best

ϕ/ψ torsions along the backbone demonstrated that with three NMR dimensions erroneous solutions may arise due a combination of experimental uncertainty and angular degeneracy for the possible peptide plane orientations (Yin and Nevzorov, 2011). Presumably, if a sufficiently large set of the possible structural solutions to the NMR spectra is obtained, statistically it will likely contain a number of "true" structures corresponding to the native fold of the protein. The use of bioinformatics restraints can then be applied for separating the most plausible, low RMSD solutions from the less accurate structures.

A protocol employing Rosetta scoring functions is presented in Chapter 3, which distinguishes the "true" structures from those that violate physical principles of protein folding, which would result in unlikely conformations while still satisfying NMR angular restraints. Since at present experimental solid-state spectra correlating three independent spectroscopic dimensions are not widely available, the protocol was exemplified by using synthetic spectra. The method was tested on synthetic datasets generated from a soluble protein (PDB 2gb1) and the first two transmembrane helices of a membrane protein (PDB 4a2n). Structural solutions were obtained at various degrees of experimental error or uncertainty (also termed as "noise") in each NMR dimension. Experimental error was simulated by randomizing the NMR resonances, calculated from the known three-dimensional structures, in order to illustrate the performance of the algorithm in the presence of non-ideal peptide plane geometry, variable chemical shift tensor orientations, and uncertainty in the peak positions arising from spectral resolution. Rosetta scoring was used to determine the consensus structures for these proteins which were then compared to the original structures used to generate the NMR angular restraints.

Extensive many-spin NMR simulations were utilized in Chapter 4 for the purpose of creating new pulse sequences which provide line narrowing in the dipolar (indirect) dimensions of multidimensional NMR spectra. The line narrowing is achieved here by effective cancelling out of the *homonuclear* dipolar terms in the average Hamiltonian, which otherwise would significantly broaden the most informative *heteronuclear* dipolar couplings, thus negatively affecting spectroscopic resolution. By utilizing a Monte Carlo protocol, similar to that used for automated assignment, pulse sequence parameters could be sampled extensively and selected based on the sharpest linewidths obtained in silica. The resulting pulse sequences were experimentally tested for measurement of ^{1}H-^{15}N dipolar couplings in two ways; 1) on a 300 MHz spectrometer for an N-acetyl Leucine crystal; 2) on a 500 MHz spectrometer for Leucine labelled Pf1 coat protein

reconstituted in magnetically aligned bicelles. In each case the best-performing pulse sequence, termed ROULETTE-1, outperformed the previously experimentally proven pulse sequence, SAMPI4, in terms of the average dipolar linewidths. This work is fully extendable to optimization of other NMR experiments, including $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings, which could be of great use in structure calculations.

Chapter 5 extends the structure fitting algorithm to the recently obtained three dimensional NMR data. This extension of the algorithm was motivated by the development of new pulse sequences to enable measurements of a third NMR dimension along a protein backbone. Experimental triple-resonance NMR experiments that include measurements of the chiral $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings were carried out on Pf1 coat protein in bicelles (Awosanya et al., 2020). As was previously predicted theoretically, structure calculations with Rosetta filtering yielded 10 structural solutions which resulted in a tilted, kinked alpha-helix. In comparison to 2D structural fitting, the addition of a third dimension greatly narrows down the distribution of the possible torsion-angle solutions.

1.2 References

Awosanya, E.O., Lapin, J., Nevzorov, A.A. (2020) NMR "Crystallography" for Uniformly (13C, 15N)-Labeled Oriented Membrane Proteins, Angewandte Chemie 132:3582-3585

Castellani, F., van Rossum, B., Diehl, A., Schubert, M., Rehbein, K., Oschkinat, H. (2002) Structure of a protein determined by solid-state magic-angle-spinning NMR spectroscopy, Nature 420:98-102

Elster LLC, "What is T2 Relaxation?", mriquestions.com, Elster LLC, 2020, https://mriquestions.com/what-is-t2.html

Etzkorn, M., Martell, S., Andronesi, O. C., Seidel, K., Engelhard, M., Baldus, M. (2007) Secondary structure, dynamics, and topology of a seven-helix receptor in native membranes, studied by solid-state NMR spectroscopy, Angewandte Chemie-International Edition 46:459-462

Gayen, A., Banigan, J.R., Traaseth, N.J. (2013) Ligand-Induced Conformational Changes of the Multidrug Resistance Transporter EmrE Probed by Oriented Solid-State NMR Spectroscopy, Angewandte Chemie International Edition 52:10321-10324

Gleason, N.J., Vostrikov, V.V., Greathouse, D.V., Koeppe, R.E. (2013) Buried lysine, but not arginine, titrates and alters transmembrane helix tilt, Proceedings of the National Academy of Sciences, 110:1692-1695

Heise, H.; Seidel, K.; Etzkorn, M.; Becker, S.; Baldus, M. (2005) 3D NMR spectroscopy for resonance assignment and structure elucidation of proteins under MAS: novel pulse schemes and sensitivity considerations, J. Magn. Reson. 173:64-74

Khitrin, A. K.; Xu, J. D.; Ramamoorthy (2011) Cross-correlations. between low-gamma nuclei in solids via a common dipolar bath, Journal of Magnetic Resonance 212:95-101

Marassi, F.M., Opella, S.J. (2003) Simultaneous assignment and structure determination of a membrane protein from NMR orientational restraints, Protein Science 12:403-411

McDermott (2009) Structure and dynamics of membrane proteins by magic angle spinning solid-state NMR, Annual Review of Biophysics 38:385-403

McDonnell, P.A., Shon, K., Kim, Y., Opella, S.J. (1993) fd Coat protein structure in membrane environments, JMR 233:447-463

Opella, S.J., Marassi, F.M., Gesell, J.J., Valente, A.P., Kim, Y., Oblatt-Montal, M., Montal, M. (1999) Structures of the M2 channel-lining segments from nicotinic acetylcholine **and** NMDA receptors by NMR spectroscopy, Nature Structural and Molecular Biology 6:374

Park, S. H.; Marassi, F. M.; Black, D.; Opella, S. J. (2010) Structure and dynamics of the membrane-bound form of Pf1 coat protein: implications of structural rearrangement for virus assembly, Biophysical Journal 99:1465-1474

Sharma, M., Yi, M., Dong, H., Qin, H., Peterson, E., Busath, D.D., Cross, T.A. (2010) Insight into the mechanism of the influenza A proton channel from a structure in a lipid bilayer, Science 330:509-512

Traaseth, N.J., Buffy, J.J., Zamoon, J., Veglia, G. (2006) Structural dynamics and topology of phospholamban in oriented lipid bilayers using multidimensional solid-state NMR, Biochemistry 45:13827-13834

Traaseth, N.J., Shi, L., Verardi, R., Mullen, D.G., Barany, G., Veglia, G. (2009) Structure and topology of monomeric phospholamban in lipid membranes determined by a hybrid solution and solid-state NMR approach, Proceedings of the National Academy of Sciences 106:10165-10170

Tycko, R. (2006) Molecular structure of amyloid fibrils: insights from solid-state NMR, Quarterly Reviews of Biophysics 39:1-55

Verardi, R., Shi, L., Traaseth, N.J., Walsh, N., Veglia, G. (2011) Structural topology of phospholamban pentamer in lipid bilayers by a hybrid solution and solid-state NMR method, Proceedings of the National Academy of Sciences 108:9101-9106

Verel, Rene. "Solid-State NMR Spectroscopy - An Introduction." *Ethz.ch*, ETH, 27 Oct. 2013, ethz.ch/content/dam/ethz/special-interest/chab/icb/van-bokhoven-group-dam/coursework/Characterization-Techniques/2015/solid-state-nmr.pdf.

Wang, J., Kim, S., Kovacs, F., Cross, T.A. (2001) Structure of the transmembrane region of the M2 protein H^+ channel, Protein Science 10:2241-2250

Yamamoto, K., Durr, U.H., Xu, J., Im, S.C., Waskell, L., Ramamoorthy, A. (2013) Dynamic Interaction Between Membrane-Bound Full-Length Cytochrome P450 and Cytochrome b_5 Observed by Solid-State NMR Spectroscopy, Scientific reports, 3:2538

Yin, Y., Nevzorov, A.A. (2011) Structure determination in "shiftless" solid state NMR of oriented protein samples, JMR, 212:64-73

CHAPTER 2: AUTOMATED ASSIGNMENT OF NMR SPECTRA OF MACROSCOPICALLY ORIENTED PROTEINS USING SIMULATED ANNEALING

(Chapter based on publication: Lapin, J., Nevzorov, A.A. (2018) Automated Assignment of NMR Spectra of Macroscopically Oriented Proteins Using Simulated Annealing, J. Magn. Reson. 293:104-114)

2.1 Introduction

In general terms, the process for assigning the backbone resonances can be defined as matching the main peak assignment and the resulting cross-peak pattern with the experimentally measured intensity in the cross-peak regions. Two sequential (i.e. in the order they appear within the backbone) main peaks in an NMR spectrum having the coordinates (x_1, y_1) and (x_2, y_2) will yield cross peaks (x_1, y_2) and (x_2, y_1). Every peak ordering (i.e. assignment) will give rise to an essentially unique crosspeak intensity distribution based on the sequential connectivity within the protein backbone. Insufficient resolution of crosspeaks in the spectrum makes spectroscopic assignment ambiguous. This problem is especially paramount in oriented-sample NMR exhibiting linewidths on the order of 1 ppm or even greater (Nevzorov, 2011). To resolve such ambiguities, more experimental information may be required which involves measuring more than one spin-exchange NMR spectrum or adding another spectroscopic dimension.

When using the spin-exchanged SAMPI4X spectrum alone (Tang et al., 2011), the cross peaks between the main peaks that have similar ^{1}H-^{15}N dipolar couplings may overlap with the greater intensity of the main peaks. By combining it with the homonuclear spin-exchange spectrum, these crosspeaks can be separated from the main peaks and, thus, are more likely resolved. Additional constraints to aid in the assignment include selective labeling experiments for specific amino acids which are distributed throughout the protein backbone at defined locations. By invoking selective labeling experiments, the peaks that belong to the same type of amino acid can only take on certain peak assignments, thus restricting the number of possible assignments.

Previous successful attempts made at devising automated assignment algorithms were based on a particular NMR technique used and the experimental data requirements. Most programs rely on discerning algorithms that can detect the positions of real crosspeaks (Wurz and Guntert, 2017; Smith, 2017). The widely utilized CYANA software (Wurz and Guntert, 2017; Buchner and

Guntert, 2015) for automated assignment of solution NMR spectra creates two NMR signal lists: (*i*) measured signals using a peak detection algorithm and (*ii*) expected signals using atom connectivity criteria. The program then uses the FLYA algorithm (Schmidt and Guntert, 2012) to find an optimal mapping between the measured and expected peaks. Empirical data are integrated into the algorithm to aid in the mapping using the previously measured spectra of similar biological systems. The search for the best assignment is carried out using an evolutionary algorithm. Another recent approach (Tycko and Hu, 2010; Hu et al., 2011) uses measured crosspeak signal lists with a Monte Carlo/simulated annealing (MCSA) protocol (Metropolis and Ulam, 1949) to optimize the assignment.

By contrast, the method described in the present Chapter forgoes identifying crosspeak signals *a priori*, which is especially beneficial for spectra exhibiting relatively low resolution. The main peak signals are identified from a non-exchanged SAMPI4 heteronuclear spectrum (Nevzorov and Opella, 2007), and subsequently used to generate the possible crosspeaks. The peak assignment is then scored based on the pseudoenergy of the positions that the calculated crosspeaks occupy. The sole connectivity criterion for a crosspeak is based on spin exchange between the nearest-neighbor main peaks as obtained by the spin-exchanged SAMPI4X spectrum, i.e. for the (i, $i\pm1$) pairs. This approach is particularly suitable for ssNMR spectra containing only inter-residue signals from the backbone amide ^{15}N spins. Spectral resolution, however, constitutes the main limiting factor for obtaining a unique assignment, regardless of the method used. The main benefit of our *ab initio* approach is in its potential to produce an assignment with relatively little experimental input and in the presence of overlapping spectral peaks.

The automated assignment program is first exemplified on synthetic spectra, generated to closely resemble the ^{15}N-^{15}N homonuclear and ^{15}N-^{1}H^{15}N heteronuclear spin-exchange SAMPI4X experiments. The program will then be applied to assigning experimental spectra having the same spectroscopic dimensions. The first two spectra correspond to the 20-residue transmembrane domain of Pf1 coat protein reconstituted in magnetically aligned bicelles whose assignment was previously established by various methods (Park et al., 2010; Tang et al., 2011). The second system was the significantly more challenging Pf1 bacteriophage, consisting of 46 residues in length (Thiriot et al., 2005) and exhibiting significant spectral crowding. Additional post-fitting scoring criteria were utilized for scoring and filtering out the appropriate assignment solutions.

2.2 Methods

2.2.1. Programming languages

Python scripts were written and executed with the Spyder 3.2.5 development environment. Programs written in C were compiled on a GCC compiler with O3 level of compiler optimization. Compilation and execution were run in the UNIX virtual environment, *Cygwin*, for Windows. Pre-program processing was done on the scripts entitled *readraw.py* and *readrawphage.py*. The raw *.fid* NMR spectrum, produced by nmrPipe, was read in as a binary file. The pre-program script writes a CSV input file for the MCSA program. Two programs for MCSA were written. The first one was written for synthetic systems entitled *synthetic11.c*. This program did not need the CSV input file since all system parameters were declared inside the program, but did have an option to read in a CSV peak assignment instead of randomization. The other MCSA program, entitled *real25.c*, was used exclusively for experimental spectra. All MCSA trials were run on a Dell Latitude Laptop with an intel CORE™ i5 processor. Post-program processing was done with the script entitled *anal3.py*.

Included in the Supplementary material section is the program to run synthetic trials, *synthetic11.c*, and its complementary plotting script, *plotsynth.py*. All other programs and scripts are available upon request.

2.2.2. Peak assignment scoring and spectral editing

The raw data from a 2D NMR spectrum are read into matrix arrays. The data are truncated to the spectral region that needs to be assigned. The spectrum is normalized so that the highest peak intensity is equal to 1.0. During the automated assignment process, the fit of a peak assignment to the experimental spectrum must be scored objectively. For any particular peak ordering, the predicted crosspeaks are back-calculated and compared to the experimental spectrum. Since a Monte Carlo algorithm seeks the lowest score, the spectrum is inverted by multiplying the spectral intensity by the factor -1.0 so that the most intense areas have the lowest, or most negative values. By treating the inverted spectral intensity as a “pseudopotential” the predicted crosspeak distribution is scored based on how well it fits the experimental intensity distribution. The correct peak assignment is distinguished from the rest of the possible assignments by the goodness of fit to the intensities of its crosspeaks, i.e. the lowest pseudoenergy. The total score for the “energy” E is the sum of the pseudopotentials from the individual crosspeaks:

$$E = \sum_{i=1}^{n-1} [S(x_{i,}, y_{i+1}) + S(x_{i+1}, y_i)] \qquad (eq.\ 2.1)$$

where $S(x, y)$ is the surface function of the inverted experimental spectra and n is the total number of main peaks.

2.2.3. Setting up the fitting

Selective labeling is declared through a string of one-letter amino acid codes. In addition to the letters for the 20 amino acid types, the program also utilizes letters X and Z. The letter X refers to an amino acid that was not selectively labeled. This means that an index labeled X has no restrictions for the shifts/couplings that can be assigned to it. The letter Z prevents an index from being swapped with another peak during the MCSA fitting. This effectively couples an index with a specific main peak and permanently fixes its assignment. A residue index should be designated as Z only if there is absolute certainty for its chemical shift and dipolar coupling in the spectrum. The Z designation can also be useful to anchor residues at the beginning and/or end of the sequence to be assigned. For Pf1 coat protein reconstituted in magnetically aligned bicelles only the transmembrane domain G23-M42 was fit, while for Pf1 bacteriophage a contiguous segment A7-R44 was used.

If the algorithm is applied directly to the original spectra then the results may be affected by the presence of intense main peaks, whose intensity is about an order of magnitude greater than the average crosspeak. In this case, the algorithm may incorrectly favor a peak assignment whose shifts or couplings are close together, so that the predicted crosspeaks may appear close to the main peak intensity. Therefore, the experimental spectra have been edited to reduce the intensity from the main peaks. The main peak attenuation can be done in two ways. The first way, which was used for Pf1 coat protein in bicelles, was to generate a synthetic spectrum of 2-dimensional Gaussian surfaces at the positions of the main peaks, and then subtract them from the original spectrum, *viz.*:

$$S(x, y) = A * \exp\left(-\left(\frac{x - x_0}{\sqrt{2}\sigma_x}\right)^2 - \left(\frac{y - y_0}{\sqrt{2}\sigma_y}\right)^2\right) \qquad (Eq.\ 2.2)$$

The form of Equation (2.2) is modeled after a normal distribution. The individual parameters in Eq. (2.2) are determined by fitting the experimental intensity distribution for well-resolved peaks. The parameter *A* refers to the amplitude of the peak centered at (x_0, y_0). The parameters σ_x and σ_y determine the spread of the peak in the *x* and *y* spectroscopic dimensions, respectively. Only an average set of parameters was used for every main peak, since it is nearly impossible to individually parameterize all the peaks in the spectrum due to their overlap. For Pf1 coat protein reconstituted in magnetically aligned bicelles the Gaussian parameters were found by fitting several fully resolved peaks in the spectrum, such as G37 and S41.

The second method for the attenuation of main peaks is to subtract the SLF spectrum (Nevzorov and Opella, 2007) from the spin-exchanged SLF spectrum (Nevzorov, 2008) after proper scaling. This method was illustrated for Pf1 bacteriophage. If done correctly, this method is more appropriate than the Gaussian subtraction since the non-exchange SLF spectrum only contains the main peaks and their true shapes. The intensity of the non-exchange spectrum should be scaled before subtraction to match the peak intensity in the spin-exchange spectra. For bacteriophage the scaling factor was chosen so that there would be minimal remnant intensity at the positions of main peaks after subtraction. In addition, extra care has been taken in order not to over-edit the spectrum. This is necessary to ensure that the remaining cross-peaks would have no major distortions before applying the MCSA assignment algorithm.

2.2.4. The annealing schedule

There are a number of ways one can tailor the cooling schedule as specified by the inverse temperature *W*. In general, too rapid a cooling may cause the system to get trapped in a local minimum. Too slow cooling may prevent the program from running to completion in a reasonable amount of time, i.e. arrive at a high enough *W* value to consider the search sufficient. In the present work an exponential cooling schedule has been used (Kirkpatrick et al., 1983).

$$W = W_{max}(1 - \alpha^n) \qquad (eq.\,2.3)$$

The inverse temperature *W* thus asymptotically approaches W_{max}. The parameter *n* is the number of times that *W* is incremented. The parameter α determines the shape of the cooling schedule, and is typically set between 0.8-0.9. A smaller value of α will make *W* approach W_{max} more

quickly, a larger value of α will give the cooling schedule a longer tail. For this experiment α was set to 0.85 and *W* was always incremented until the final fraction $\frac{W}{W_{max}}$ was 0.99. Typically, the value of *n* was incremented up to 1000 and at each value of *W* there were as many as 10^8 random steps.

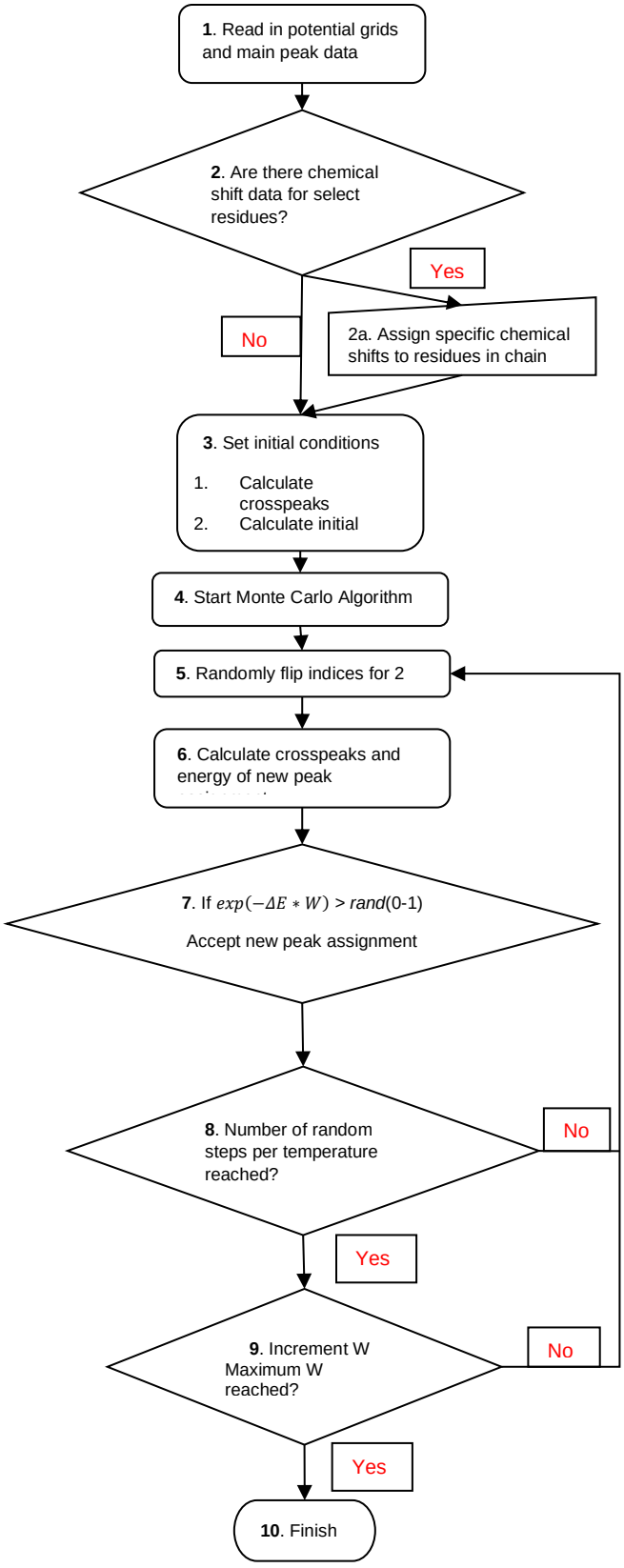
2.2.5. The MCSA algorithm

Figure 2.1 displays the logic flow of the algorithm for automated assignment of experimental spectra as implemented in program *real25.c*. The highly similar code for assigning synthetic spectra, *synthetic11.c*, is provided in the Supplementary material section.

1) Read in a CSV header file that contains the list of main peak shifts/dipolar couplings, axes limits, and the matrices of spectral surfaces.
2) If selective labeling information is available, then assign specific chemical shifts and dipolar couplings to specific amino acid indices. This will ultimately restrict the peak indices that can be exchanged out during the MCSA run. Residues can either be assigned to specific chemical shifts/dipolar couplings at runtime, or preordered with their shifts in the CSV input file.
3) Seed the algorithm by starting with a random peak assignment, calculating the crosspeaks, the initial score *E*, and setting the inverse annealing temperature *W* to a very low value (very high temperature).
4) Start the Monte Carlo algorithm by entering loops for increasing *W* (incremented for sufficiently slow cooling) and the number of random index flips for every value of *W*.
5) Randomly exchange positions of any two peaks. If selective labeling constraints are imposed, not all amino acid indices will be available to every chemical shift and dipolar coupling value.
6) Calculate the crosspeak pattern resulting from the new peak assignment and then calculate the sum of the pseudopotential energies (*E*) from those crosspeaks, Eq. (2.1).
7) Calculate the change in the pseudoenergy score (ΔE), generate a random number from 0-1, and apply the Metropolis criterion: if $exp(-\Delta E * W) > rand$(0-1) then accept the new assignment, else reject the new assignment and continue. Whether accepted or rejected, evaluate the new ranking assignment for top-score assignment solutions.
8) Has the number of random flips *n* reached its maximum? If no, then go back to step 5. If yes, then increment *W* and set *n* to 0.

9) Has the final *W* been reached? If no, then increment *W* and go back to step 5. If yes, then exit the MCSA loops. The number of *W* increments is determined manually. It should be chosen to ensure a sufficiently slow cooling schedule, as well as a reasonable runtime for the program. It was found that the best practice would be to cool down until 99% of the new moves were rejected for a specific temperature step.
10) Save several tens of thousands of the top-scoring peak assignments to a file. The main peak assignment alone is sufficient for a post-fitting MCSA program to recalculate the crosspeaks and their energies.

Figure 2.1 (continued). Flowchart for the MCSA Algorithm showing the main steps: initiation of the fitting, generation of a new assignment step, calculation of the pseudoenergy, and the Metropolis acceptance criterion for each reassignment step. Note that tens of thousands of top-score possible assignment solutions are retained, which are further sorted out in the post-fitting steps. Figure obtained from Lapin and Nevzorov, 2018, JMR, 293:104-114.


1. Read in potential grids and main peak data
2. Are there chemical shift data for select residues?
Yes
No
2a. Assign specific chemical shifts to residues in chain
3. Set initial conditions
1. Calculate crosspeaks
2. Calculate initial
4. Start Monte Carlo Algorithm
5. Randomly flip indices for 2
6. Calculate crosspeaks and energy of new peak
7. If $exp(-\Delta E * W)$ > rand(0-1)
Accept new peak assignment
8. Number of random steps per temperature reached?
No
Yes
9. Increment W
Maximum W reached?
No
Yes
10. Finish

2.2.6. Filtering out multiple assignment solutions

The output from the MCSA program consists of only the list of top main peak assignments. There are no crosspeaks, pseudopotentials, or energies; these values can be quickly recovered from the peak assignment and the spectral surface. The program is stratified to do the most processing-intensive part in the fast, compiled programming c-language, and then save the data analysis for the slower but much more development-friendly Python computer language. The post-fitting processing is necessary because the limited resolution and/or imperfect editing may cause the lowest pseudopotential solution to be different from the anticipated peak assignment, especially without any residue-type constraints. Saving a large number of top scoring solutions from the MCSA program increases the likelihood that the correct solution, even if it has a higher energy, is retained within the output. There are several criteria by which one can filter the output to distinguish the most plausible assignment(s) from the rest. For instance, if an assignment produces a crosspeak that lies in a location of little or no experimental intensity, we denote this as an "orphan crosspeak" and filter out such an assignment. This involves defining a cutoff intensity value, above which a calculated crosspeak is considered orphan. A correct solution should have no orphan crosspeaks in either the homonuclear or heteronuclear spin-exchange spectra. That is to say, each experimentally valid crosspeak should fall within measurable intensity in the spectrum. Imperfect main-peak editing may complicate implementing this criterion since some cross-peak intensity may be subtracted from the real spectrum. Therefore, it is best to set the cutoff values just above the (negative) intensity of the weakest crosspeak. This filtering criterion is built into the MCSA program during the saving of top solutions and its implementation takes place in between steps 7 and 8, i.e. after the new assignment is scored.

In addition, selecting for specific crosspeaks to match as much of the experimental intensity as possible was found to be the easiest way to filter out a plethora of possible assignment solutions down to a manageable number. While unnecessary for the relatively simple spectra of Pf1 in bicelles, this method was effectively used for the Pf1 bacteriophage where 38,000 outputs resulted after filtering out the orphan crosspeaks. Moreover, since the structure for Pf1 coat protein is mostly alpha-helical (Thiriot et al., 2004), an additional scoring criterion was implemented alongside the orphan peak criterion. Namely, each (i, i+1) assignment received a score of 1.0 if the adjacent main peaks (loosely) satisfied the so-called "PISA pie" pattern (Marassi and Opella, 2002). Computationally this was achieved by calculating the angle between the two vectors

originating from the average center of the spectrum (selected at 177.2 ppm and 7.3 kHz) and ending at the adjacent resonances and taking into account the clockwise sense of rotation between the resonances in the PISA wheel pattern (Marassi and Opella, 2002). Since an experimental peak pattern may well deviate from the theoretical (i.e. ideal) PISA wheel, any resulting angle between 90 and 180 degrees received the score of 1.0; otherwise, the score was 0.0. The final assignment solutions were selected based on the criterion of covering as many experimental cross peaks as possible. This involved an iterative process which continued until there were few enough resulting peak assignments that a single candidate could be selected to represent the final (or consensus) solution.

2.3 Results

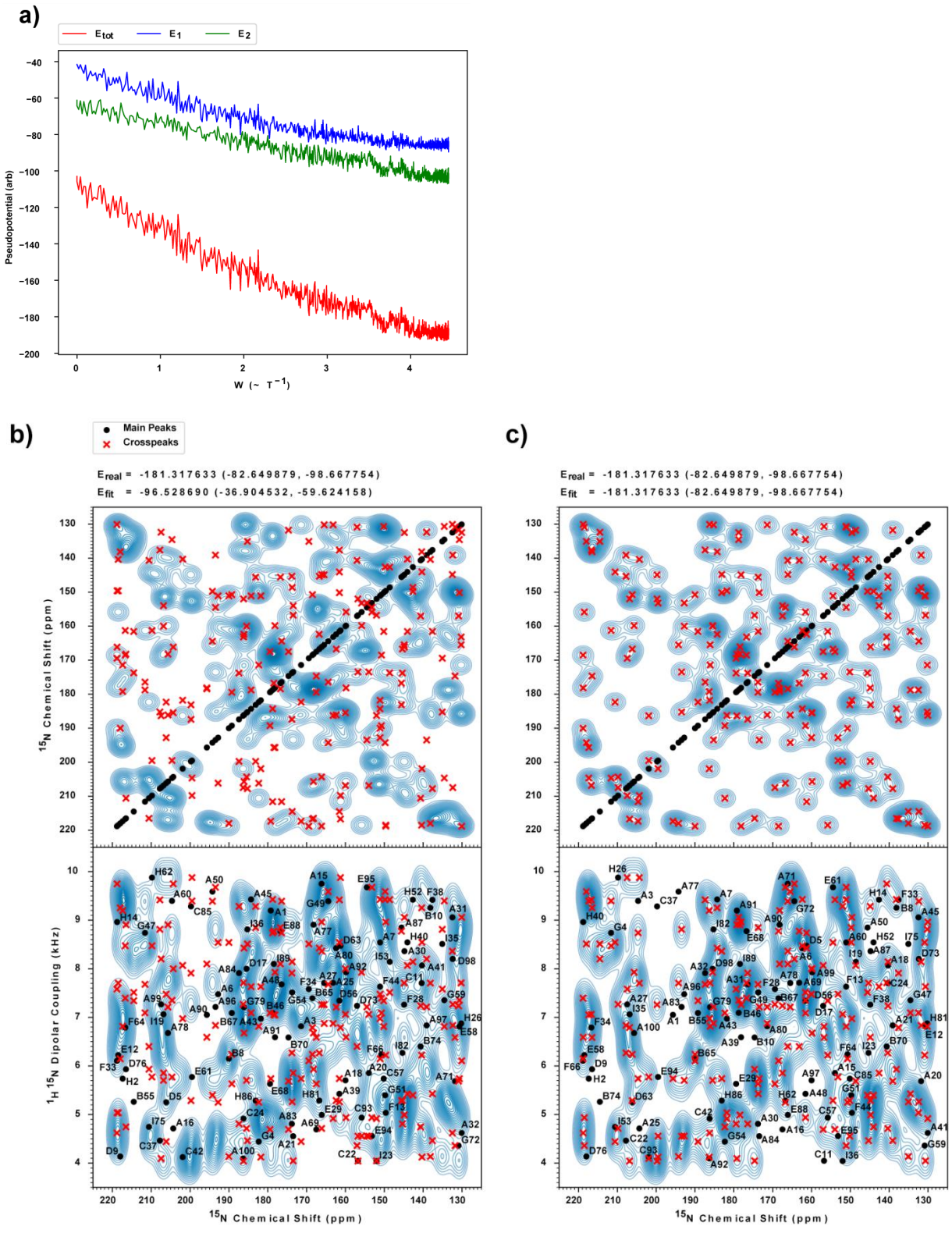
2.3.1. Assignment of synthetic spectra

The algorithm was first tested on simulated ^{15}N-^{15}N homonuclear exchange and ^{1}H^{15}N-^{15}N dipolar SLF spectra with the crosspeaks modeled by equation (2.2). No main peaks were included in generating the pseudoenergy landscape function. The spectrum was generated on a 1000-by-1000 point grid. The Gaussian lineshape parameters (in the format $[A, \sigma_x, \sigma_y]$) were [0.25, 2.0, 2.0] and [0.25, 1.5, 0.25] for the homonuclear exchange and spin exchanged SLF spectra, respectively. These parameters were chosen to reasonably approximate the shape of crosspeaks in real spectra, such as for Pf1 coat protein. Figure 2.2 illustrates the performance of MCSA algorithm for assigning a synthetic spectrum containing 100 peaks with some selectively labeled residues. In this run there were 9 labeled groups, in which one group had 36 peaks, and the other eight groups had 8 peaks each. The main peaks were randomly generated with the chemical shifts ranging within 130-220 ppm, and the corresponding dipolar couplings were generated within 4-10 kHz. The starting order of peaks was shuffled prior to application of MCSA by flipping the peak indices 10^4 times, cf. Fig. 2.2A. The inverse temperature parameter W was incremented 10^3 times, starting from a value of 10^{-8} and ending at 4.5. Per every value of W there were 10^8 random steps taken. The peak assignments were filtered during MCSA to have no orphan crosspeaks above the -0.1 contours. The grouping of the main peaks was chosen to reflect 8 hypothetical selective labeling experiments that constrain the possible peak positions for the 8 amino acids. The final group represents the remaining residues not accounted for in the selective labeling experiments (these indices have the X designation per our notation, see Methods).

During the MCSA run, the pseudoenergy decreases and starts to level off at higher *W* values, cf. Fig. 2.2A, where most of the sampling is done until the correct solution is found. Figure 2.2b shows the crosspeak positions in the starting assignment. This random assignment has a very high energy as compared to the target energy. Strikingly, upon completion of the MCSA run, the original peak assignment was fully recovered. As can be seen from Fig. 2.2c, all the fitted crosspeaks fill out the spectrum perfectly and are entirely consistent with the intensity distribution. Thus, despite significant spectral overlap of the crosspeaks the algorithm was able to recover the correct assignment.

We should note that direct enumeration of all the possible solutions would have resulted in an astronomical number of samplings, $(8!)^8 36! = 2.6\times10^{78}$, which would have been unfeasible. Over multiple MCSA runs on synthetic spectra, occasional failures to recover the original assignment was the result of either insufficient sampling, or excessive spectral overlap resulting in lower energy solutions than the actual solution. The former issue was solved by increasing the number of random steps up to practical limits. The latter issue can be solved by either more stringent orphan peak filtering or including more selective labeling groups.

Figure 2.2: Assignment of synthetic spectra containing arbitrarily generated 100 peaks. The spectra are assigned using arbitrary selective labeling for 8 residues, each group having 8 peaks. The remaining 36 peaks are effectively the "unlabeled" group. **a.)** MCSA profile of pseudopotential energy vs. W (inverse temperature). The three lines represent the individual pseudoenergies calculated for the two spin-exchange spectra and their sum as indicated. Pseudoenergy E_1 corresponds to the top (homonuclear exchange) spectrum and E_2 to the bottom (heteronuclear spin-exchange SLF). **b.)** Crosspeaks for the initial assignment, i.e. prior to starting MCSA, are provided for direct comparison with the final solution. The top contour plot corresponds to a homonuclear ^{15}N-^{15}N exchange experiment and the bottom spectrum reflects the heteronuclear spin-exchange SLF experiment such as SAMPI4X. The spectra were calculated from an arbitrary peak assignment and then the order of the main peaks was thoroughly reshuffled to provide the initial state for the MCSA run. The main peak labels are included in the bottom spectra; the letter denotes the label for the residue group and its corresponding number in the peak assignment. The net pseudoenergies are displayed above the plots. The initial energy is much higher than for the real assignment, as evident by the number of crosspeaks that fall completely outside the contours. **c.** The final solution after the MCSA run. Despite significant spectral overlap, the back-calculated solution is identical to the original assignment. Figure obtained from Lapin and Nevzorov, 2018, JMR, 293:104-114.


a)
E tot
E 1
E 2
Pseudopotential (arb)
W (~ T −1)
b)
Main Peaks
Crosspeaks
E real = -181.317633 (-82.649879, -98.667754)
E fit = -96.528690 (-36.904532, -59.624158)
15N Chemical Shift (ppm)
1H 15N Dipolar Coupling (kHz)
15N Chemical Shift (ppm)
c)
E real = -181.317633 (-82.649879, -98.667754)
E fit = -181.317633 (-82.649879, -98.667754)
15N Chemical Shift (ppm)

2.3.2. Automated spectroscopic assignment of Pf1 coat protein spectra in magnetically aligned bicelles.

Both the homonuclear and heteronuclear SAMPI4X spin-exchange spectra previously published in ref. (Tang et al., 2012) were edited by the synthetic peak subtraction method as described above. The peak parameters were determined using a surface fitting script *surf_fit.py*. The Gaussian lineshape parameters were found to be [0.300, 0.884, 0.884] for the homonuclear ^{15}N exchange and [0.400, 0.964, 0.112] for the spin-exchanged SAMPI4X spectra. The algorithm was run for both unrestrained and restrained (i.e. with a set of selectively labeled residues) trials. For the restrained trial, the resonances for the A, G, I, L, and M residues were fixed (but not in a particular order) by using the previous selectively labeled spectra (Opella et al., 2008). The parameter W was incremented 10^4 times, starting from 10^{-8} and ending at 10. There were 10^4 random steps taken per each value of W. The runtime of the fit was 225 seconds. The results were filtered down only to the assignments yielding the fewest orphan peaks, using peak cutoff values of -0.2 and -0.05 for the homonuclear exchange and heteronuclear exchange SAMPI4X spectra, respectively.

The algorithm was run both with and without selective labeling input (i.e. unrestrained), saving as many as 10,000 outputs for each run. Figure 2.3 shows the program output for the top scores of both runs. The unrestrained run shown in Fig. 2.3a represents the global minimum score for this spectral surface (E = -33.60 for the two spectra). This solution, while being lower in energy, however, does not satisfy the previous selective labeling restraints (Opella et al., 2008). The top 10,000 scores were saved, ranging from -33.6 to -32.9. Figure 2.3b is the lowest score peak assignment obtained under the restrictions that labeled residues 'AGILM' can occupy certain, pre-selected main peak positions, and has resulted in a higher pseudoenergy E = -28.64. With orphan peak filtering 113 solutions resulted in pseudoenergies ranging from -28.6 to -24.5. The second top solution differs from the original solution only by the swapping the peaks for residues I32 and I39. Notably, the top scoring peak assignment for the restrained run of Fig. 2.3b is identical to the original assignment provided by Park et al. (Park et al., 2011).

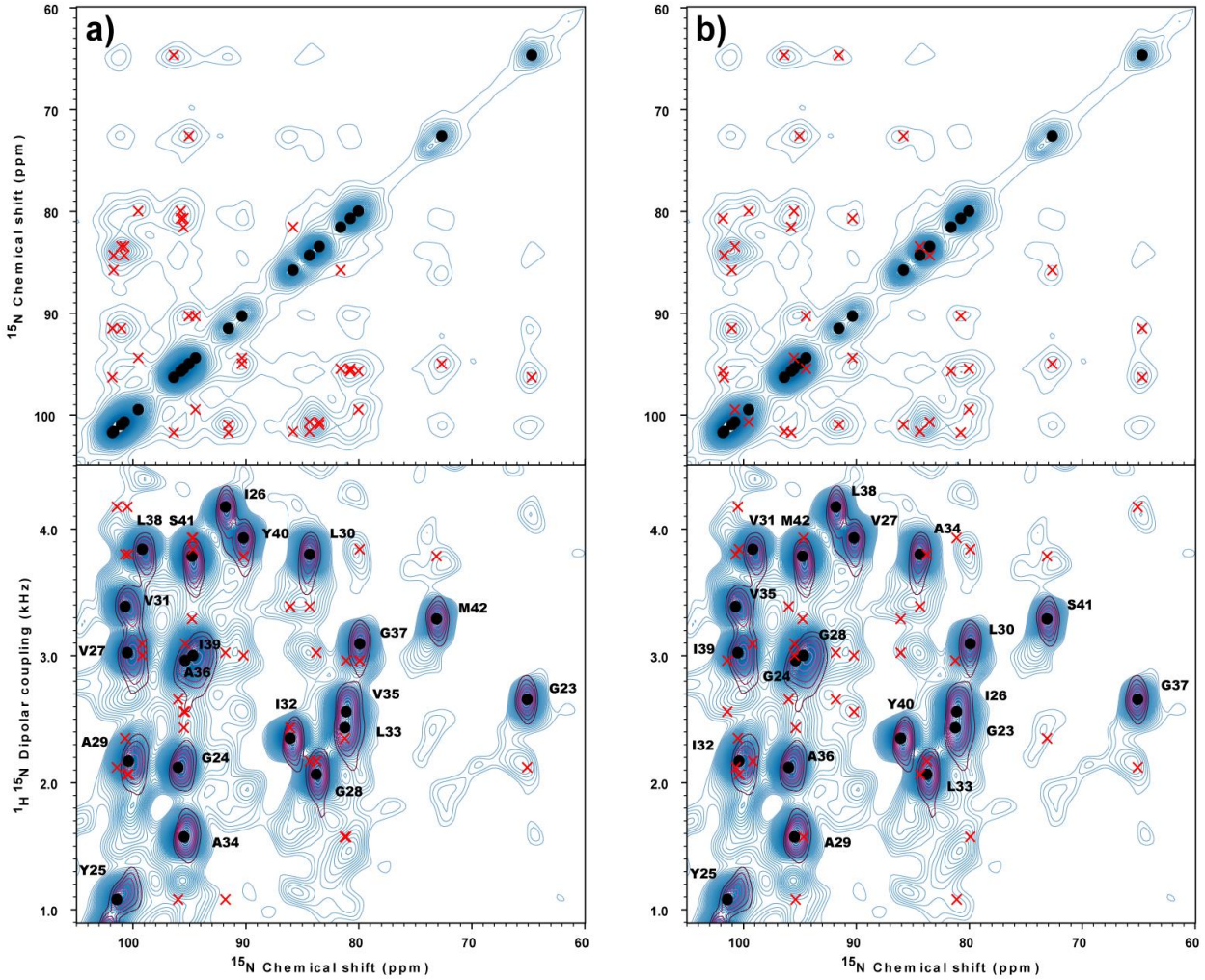


Figure 2.3: Automated assignment of ssNMR spectra of Pf1 coat protein reconstituted in magnetically aligned bicelles and acquired at 500 MHz ^{1}H frequency. **a.)** Unrestrained run, i.e. only the data from a uniformly labeled sample were used, in which any main peak can take on any index, without orphan peak filtering. **b.)** Restrained run with selectively labeled groups of residues 'AGILM'. The main peaks are denoted by black circles with labels, while the crosspeaks are represented by red crosses. The recovered peak assignment is identical to the previous assignment (Park et al., 2010). The total pseudoenergy score for the unrestrained fit was -33.60 (E_1= -18.20, E_2 = -15.40) and for AGILM-restricted run the pseudoenergy is -28.64 (E_1 = -14.53, E_2 = -14.11). Figure obtained from Lapin and Nevzorov, 2018, JMR, 293:104-114.

2.3.3. Assignment of Pf1 Bacteriophage Spectra

For Pf1 bacteriophage, only the spin-exchange PISEMA spectra (Knox et al., 2010) was used to score peak assignments as it exhibits superior resolution as compared to the homonuclear ^{15}N-^{15}N exchange spectrum. Due to lower signal-to-noise ratio, the latter was used only for filtering against the presence of orphan crosspeaks, and the MCSA algorithm was instructed to save only the top solutions with the fewest orphan crosspeaks. Similar to Pf1 coat protein in bicelles, the orphan crosspeaks were defined as those above -0.1 intensity threshold; up

to 8 orphans were tolerated in the homonuclear spin-exchange spectrum; 1 orphan were tolerated in the spin-exchange PISEMA spectrum. The regular PISEMA spectrum having only the main peak intensity was subtracted from the spin-exchanged PISEMA spectrum for the main peak attenuation. The scaling factor for subtracting the non-exchange spectrum was 0.7. The algorithm was run with 8 residues "AGIKLMRTV" fixed, but not in a particular order, as per previous experimental data for selectively labeled samples (Tang et al., 2012). Six residues were completely fixed and given the Z designation: V8 (169.03 ppm, 4.64 kHz), I12 (179.84 ppm, 5.55 kHz), T13 (211.35 ppm, 7.96 kHz), D14 (201.83 ppm, 9.93 kHz), L43 (191.90 ppm, 9.80 kHz), and R44 (138.36 ppm, 6.37 kHz). These designations were chosen for the peaks having little ambiguity for their assignment. The parameter W was incremented 1000 times, starting from 10^{-8} and ending at 5.0 with 10^6 steps per W. The total program runtime was 16,400 seconds. Solutions were filtered in an iterative process by selecting for the presence of crosspeaks in intense areas, and excluding the crosspeaks lying beyond the contour edges. The process continued until the best solution was found.

It should be noted that Pf1 bacteriophage spectra exhibited heavy degeneracy of possible assignments in the top scoring solutions, even though the majority of the amino acids were selectively labeled. There were 38,088 retained solutions, ranging in their pseudoenergies between E = -60 and E = -48. The top assignment having the composite pseudoenergy E = -56.0 and resulting from the filtering based on the number of orphan/missed peaks and Pf1 helicity (cf. section 6 in Methods) is represented in Figure 2.4B and compared to the original assignment (Thiriot et al., 2005; Knox et al., 2010) in Table 2.1. Note that the score of the original assignment by Thiriot et al. has a significantly higher pseudoenergy, E = -49.98. In contrasting the two assignments, it becomes readily apparent that the original assignment contains a number of crosspeaks that fall outside the experimental cross-peak contours, i.e. they are essentially orphan cross peaks, cf. Fig. 2.4A. The main program precluded such assignments from being saved. Figure 2.4B represents the best peak assignment found under the restraints imposed by selective labeling and filtering criteria. Relaxing of selective-residue restraints can allow for even better pseudoenergy scores, but the assigned peaks for some residues would be in contradiction with the selective labeling experimental data and the predominantly helical wheel pattern expected for Pf1 (results not shown). As can be seen from Table 2.1, 18 out of 38 assignments are identical to the previous assignments (Thiriot et al., 2005; Knox et al., 2010). If not counting the permutations of

the resonances corresponding to the same aminoacid type (e.g. G or V) as well as highly overlapping peaks, 32 (or 84%) of the previous 38 assignments have been confirmed, cf. Figs. 2.4A and 1.4B. The differences between the original peak assignment and the assignment obtained by the automated algorithm merit additional investigation possibly involving additional selectively labeled spectra. It is apparent, however, that the automated algorithm provides much better coverage of the experimental crosspeak intensity.

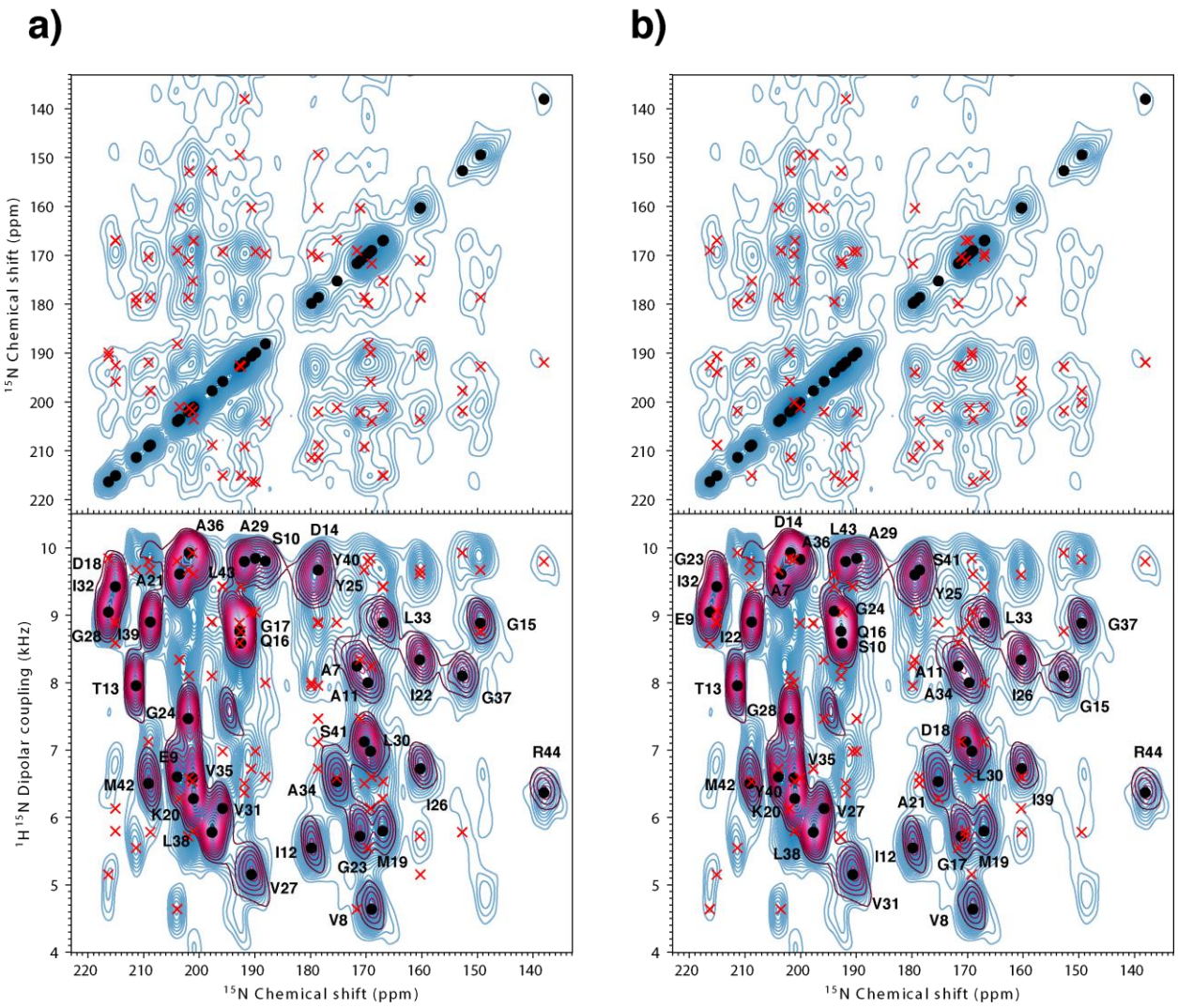


Figure 2.4: Automated assignment of ssNMR spectra of Pf1 bacteriophage acquired at 500 MHz ^{1}H frequency. **a.)** The original assignment (Thiriot et al., 2005; Knox et al. 2010) and **b.)** The assignment obtained using MCSA and post-fitting criteria for the orphan/missed cross peaks and the expected helical wheel pattern. Bottom contour plot is an overlay of the PISEMA spectrum (red contours) and the spin-exchange PISEMA spectrum (blue contours). Main peak labels are included in the latter. In addition to the sequential assignment, the new and original assignments also differ in positions of certain main peaks in ambiguous areas of the non-exchanged spectrum. For example, two main peaks in part (a) occupy the same positions (i.e. D14 and Y40), whereas they have been separated in part (b) to be commensurate with the areas of underlying intensity. The highest contour level (least negative) is set to -0.1, which was used as the cutoff intensity for filtering, above which noise artifacts become prominent in both spectra. Figure obtained from Lapin and Nevzorov, 2018, JMR, 293:104-114.

Table 2.1: Comparison of Pf1 bacteriophage assignments with the original assignment. (Thiriot et al., 2005; Knox et al., 2010). Different assignments are given in boldface. Table obtained from Lapin and Nevzorov, 2018, JMR, 293:104-114.

Residue	^{15}N Chemical shift (ppm) (Original)	^{1}H^{15}N Dipolar coupling (kHz) (Original)	^{15}N Chemical shift (ppm) (New)	^{1}H^{15}N Dipolar coupling (kHz) (New)
A7	**172**	**8.25**	**204**	**9.61**
V8	169	4.64	169	4.64
E9	**204**	**6.60**	**216**	**9.05**
S10	**188**	**9.80**	**193**	**8.59**
A11	170	8.00	172	8.25
I12	180	5.55	180	5.55
T13	211	7.96	211	7.96
D14	**179**	**9.67**	**202**	**9.93**
G15	**150**	**8.88**	**153**	**8.10**
Q16	193	8.76	193	8.76
G17	**193**	**8.59**	**153**	**8.10**
D18	**215**	**9.43**	**193**	**8.76**
M19	**167**	**5.80**	**171**	**5.72**
K20	201	6.28	201	6.28
A21	**204**	**9.61**	**175**	**6.53**
I22	**160**	**8.34**	**209**	**8.90**
G23	**171**	**5.72**	**215**	**9.43**
G24	**202**	**7.47**	**194**	**9.06**
Y25	179	9.67	179	9.60
I26	160	6.72	160	8.34
V27	**191**	**5.15**	**196**	**6.13**
G28	**216**	**9.05**	**202**	**7.47**
A29	190	9.85	190	9.85
L30	169	6.98	169	6.98
V31	**196**	**6.13**	**191**	**5.15**

Table 2.1 (continued)

I32	215	9.43	215	9.43
L33	167	8.89	167	8.89
A34	**175**	**6.53**	**170**	**8.00**
V35	201	6.59	201	6.58
A36	202	9.93	200	9.83
G37	**153**	**8.10**	**149**	**8.88**
L38	198	5.78	198	5.78
I39	**209**	**8.90**	**160**	**6.72**
Y40	**179**	**9.67**	**204**	**6.60**
S41	**170**	**7.12**	**179**	**9.67**
M42	209	6.50	209	6.50
L43	192	9.80	192	9.80
R44	138	6.37	138	6.37

2.4 Discussion

Spectroscopic assignment of the synthetic spectra has provided a proof of concept for the viability of the MCSA algorithm. At the same time, they have served for optimizing the annealing schedule with full control of the spectral parameters and prior knowledge of the real peak assignment. However, the fitting of synthetic spectra is considerably simplified by not including the main peak intensity and noise, both of which are expected to always be present in real spectra. As discussed in the Methods section, one of the main challenges in fitting real experimental spectra is to reduce the main peak intensity by editing them in order to avoid preferential clustering of the cross peaks near the main peaks by the MCSA algorithm. This involves subtracting out the main peak intensity, which is often an order of magnitude greater than the crosspeak intensity, thus assisting the algorithm in finding the lowest energy solutions preferentially in the areas of crosspeak intensity, but not that of the main peaks. More rigorous procedures and standards may need to be established for attenuating the main peak intensity.

In general, there are two common pitfalls to a successful recovery of the original peak assignment. The first is that the system could be so large that a sufficient amount of sampling could

not be achieved in a reasonable amount of time. This is particularly paramount for uniformly labeled trials with many possibilities for peak assignment. The limit on uniformly labeled spectra (containing two 2D experiments) currently appears to be around 50 peaks due to the vast sampling demands (e.g. 50! = $3x10^{64}$ overall direct sampling possibilities). However, when selective labeling of certain residues is employed, up to 100 peaks can be assigned. In this case, the selective labeling should cover about two-thirds of the peaks in the sequence but knowledge of the exact relative order for the select peaks within each group is not necessary. These results are useful with regard to demonstrating the upper bounds for the algorithm at the current CPU speeds.

The algorithm is always instructed to search for the assignment having the lowest pseudoenergy. The lowest energy solution(s) may turn out to be very different from the expected peak assignment. These energy-biased solutions may even contradict intuitive assignments of an NMR spectrum and contain a few orphan crosspeaks. This problem is circumvented here by applying concurrent filtering schemes. Without much increase in runtime, the orphan crosspeak criterion precluded many such lower-score solutions, often rendering the correct solution at or near the lowest pseudoenergy score. Orphan crosspeaks were specified in the program by a maximum number of tolerated orphan cross peaks and a cutoff value, i.e. the lowest contour level of experimental intensity that a crosspeak must fit within.

Selective labeling not only greatly reduces the total permutations for the peak assignments, but also effectively eliminates unwanted crosspeaks arising in crowded regions of the spectrum. This greatly reduces the likelihood of lower-energy solutions that could be very different from the real or expected assignment. The distribution of selectively labeled residues throughout the "sequence" in the simulation of Fig. 2.2 was chosen to reflect a realistic situation. The results suggested that having fewer than 8 groups out of the total of 100 residues would make recovering the original assignment nearly impossible when two two-dimensional spectra are used for assignment. In general, selective labeling experiments have been found to be a very effective tool for mitigating misassignments in the presence of considerable spectral overlap.

Solid-state NMR spectra of Pf1 coat protein reconstituted in magnetically aligned bicelles were used for benchmarking the method for real experimental data. The transmembrane domain of Pf1 is a relatively small spin system (20 residues) with a previous sequential assignment available (Opella et al., 2008) that served as a target. Without a general sense of the correct peak assignment, it would have been difficult to ascertain that an unrestrained MCSA run was not

sufficient to solve the problem. While the overall score from the unrestrained run is lower than from the original assignment, many cross peaks do not fit the homonuclear ^{15}N-^{15}N spin exchange spectrum adequately, cf. Fig. 2.3A. Due to the spectral overlap, the most intense crosspeak region is roughly from 78 to 86 ppm and from 94 to 103 ppm. Indeed, this region is made up of multiple peaks, but the unrestrained algorithm attempted to fit as many peaks inside these intense contours as possible, in spite of other peaks being vacated. Additionally, the heteronuclear SAMPI4X spectrum on the bottom of Fig. 2.3A has an orphan crosspeak near (92 ppm, 1.1 kHz). It is clear that placing a crosspeak in a crowded but more intense region would outweigh the cost of placing a different crosspeak in an area of less intensity. The pseudoenergies for the top 10,000 peak assignments were between -33.60 and -32.90. Thus, the real assignment score (-28.64) would have been unlikely to rank anywhere near the low score without selective labeling restraints.

The addition of even a single prevalent residue in the amino acid sequence for selective labeling restraints greatly reduces the number of possible peak assignments, and increases the ranking of the real assignment's score relative to the minimum pseudoenergy. Several combinations of residues were used for selective labeling. Ultimately, a minimum of 5 selectively labeled residues was necessary to render the original assignment (Park et al., 2010) as the top scoring assignment for the transmembrane domain of Pf1. An additional restraint was to exclude assignments with orphan crosspeaks. This is a very effective strategy for precluding many high scoring, but unrealistic solutions. For instance, while the unrestrained run collected 10,000 degenerate solutions, the filter yielded only 113 possible assignments but with a much broader range of scores. The correct solution from Fig. 2.2B had no orphan crosspeaks in the heteronuclear SAMPI4 spectrum, but two orphan peaks in the homonuclear ^{15}N-^{15}N exchange spectrum. We have attributed these orphan crosspeaks to excessive main peak subtraction during the editing. This highlights the necessity of optimizing the editing scheme during pre-program processing and close scrutiny of crosspeaks near the positions of main peaks during the filtering. Amongst the 113 solutions, the top 2 best scores were the original assignment and an additional solution differed from the original only by interchanging two residues (I32 and I39).

Using fewer selectively labeled residues often recovered the original assignment with a lower-ranking (i.e. higher) MCSA score. In such a scenario extra post-program filtering criteria would be needed to justify a consensus solution. This could include visual filtering for the presence (or absence) of specific crosspeaks or assuming a pre-defined peak pattern such as PISA wheels.

The success of automated assignment for the synthetic and Pf1 bicelle spectra suggest that selective labeling is still the best tool for finding the correct solution and possibly obviating the need for post-fitting procedures. While experimentally time-consuming and expensive, a series of selective labeling experiments can be effectively used to aid the disambiguation process, especially in the presence of significant spectral overlap.

Solid-state NMR spectra of Pf1 bacteriophage have represented a larger, more challenging spin system with 38 residues exhibiting highly overlapping peaks, both main and cross. Even though the PISEMA spectrum is used to identify the main peak positions and subtract them from the spin-exchanged PISEMA spectrum, some areas of peak overlap remained. The degeneracy of assignment solution was much greater for bacteriophage than for the transmembrane domain of Pf1 in bicelles, so that ranking the possible assignment solutions based only on the pseudopotential score did not suffice, even with 8 amino acids selectively labeled and 6 anchored residues. Post-fitting processing in the bacteriophage system consisted of filtering outputs for the presence or absence of specific crosspeaks. Namely, the solutions were selected in order to cover as much experimental intensity as possible while generally satisfying the PISA wheel pattern. The great volume of degenerate scoring solutions makes using selective labeling restraints even more essential.

While confirming 84% of the original assignment, our suggested reassignment of Pf1 phage spectra improves upon the original because it is devoid of orphan crosspeaks, which better corresponds to the experimental spin-exchange NMR spectra. Post-processing criteria were chosen to heavily favor the presence of the crosspeaks that more uniformly fit the spread of intensity in both the homonuclear exchange and spin-exchanged PISEMA spectra. There still are questionable areas of the assignment, however, which we feel are inconsistent with the spectrum. Inconsistencies include too many crosspeaks in one area, or a lack thereof in other, above-noise areas. Unlike the synthetic spectra, real spectra may contain additional peaks that may misguide the nearest-neighbor or (i, i+1) assignment algorithm. This may include the intensity from secondary crosspeaks, or (i, i+2), noise, or exchange peaks with the ^{15}N-containing side chains of specific amino acids such as Arginine and Lysine. Fitting one area of the spectrum often compromised the coverage in other areas. Additional selective labeling data (especially for Y and D residues) would be very useful in resolving such ambiguities in NMR assignment of Pf1 phage.

2.5 References

Buchner, L.; Guntert, P. (2015) Systematic Evaluation of Combined Automated NOE Assignment and Structure Calculation with CYANA, Journal of Biomolecular Nmr 62:81-95

Hu, K. N.; Qiang, W.; Tycko, R. (2011) A General Monte Carlo/Simulated Annealing Algorithm for Resonance Assignment in NMR of Uniformly Labeled Biopolymers, Journal of Biomolecular Nmr 50: 267-276

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983) Optimization by Simulated Annealing, Science 220:671-680

Knox, R. W.; Lu, G. J.; Opella, S. J.; Nevzorov, A. A. (2010) A Resonance Assignment Method for Oriented-Sample Solid-State NMR of Proteins, Journal of the American Chemical Society 132:8255-8257

Lapin, J., Nevzorov, A.A. (2018) Automated Assignment of NMR Spectra of Macroscopically Oriented Proteins Using Simulated Annealing, J. Magn. Reson. 293:104-114

Marassi, F. M.; Opella, S. J. (2002) Dilute Spin-Exchange Assignment of Solid-State NMR Spectra of Oriented Proteins: Acetylcholine M2 in Bilayers, J Biomol NMR 23:239-242

Metropolis, N.; Ulam, S. (1949) The Monte Carlo Method, ournal of the American Statistical Association 44:335-341

Nevzorov, A. A.; Opella, S. J. (2007) Selective Averaging for High-Resolution Solid-State NMR Spectroscopy of Aligned Samples, J. Magn. Reson. 185:59-70

Nevzorov, A. A. (2008) Mismatched Hartmann-Hahn Conditions Cause Proton-Mediated Intermolecular Magnetization Transfer Between Dilute Low Spin Nuclei in NMR of Static Solids, J. Am. Chem. Soc 130:11282–11283

Nevzorov, A. A. (2011) Ergodicity and Efficiency in Cross-Polarization of NMR of Static Solids, J. Magn. Reson. 209:161-166

Opella, S. J.; Zeri, A. C.; Park, S. H. (2008) Structure, Dynamics, and Assembly of Filamentous Bacteriophages by Nuclear Magnetic Resonance Spectroscopy, Annu. Rev. Phys. Chem. 59:635–657

Park, S. H.; Marassi, F. M.; Black, D.; Opella, S. J. (2010) Structure and Dynamics of the Membrane-Bound Form of Pf1 Coat Protein: Implications of Structural Rearrangement for Virus Assembly, Biophysical Journal 99:1465-1474

Schmidt, E.; Guntert, P. (2012) A New Algorithm for Reliable and General NMR Resonance Assignment, J Am Chem Soc 134:12817-12829

Smith, A. A. (2017) INFOS: Spectrum Fitting Software for NMR Analysis, Journal of Biomolecular Nmr 67:77-94

Tang, W. X.; Knox, R. W.; Nevzorov, A. A. A (2012) A Spectroscopic Assignment Technique for Membrane Proteins Reconstituted in Magnetically Aligned Bicelles, Journal of Biomolecular Nmr 54:307-316

Thiriot, D. S.; Nevzorov, A. A.; Zagyanskiy, L.; Wu, C. H.; Opella, S. J. (2004) Structure of the Coat Protein in Pf1 Bacteriophage Determined by Solid-State NMR Spectroscopy, J Mol Biol 341:869-879

Thiriot, D. S.; Nevzorov, A. A.; Opella, S. J. (2005) Structural Basis of the Temperature Transition of Pf1 Bacteriophage, Protein Sci. 14:1064-1070

Tycko, R.; Hu, K. N. (2010) A Monte Carlo/Simulated Annealing Algorithm for Sequential Resonance Assignment in Solid State NMR of Uniformly Labeled Proteins with Magic-Angle Spinning, Journal of Magnetic Resonance 205:304-314

Wurz, J. M.; Guntert, P. (2017) Peak Picking Multidimensional NMR Spectra with the Contour Geometry Based Algorithm CYPICK, Journal of Biomolecular Nmr 67:63-76

CHAPTER 3: VALIDATION OF PROTEIN BACKBONE STRUCTURES CALCULATED FROM NMR ANGULAR RESTRAINTS

(Chapter based on publication: Lapin, J., Nevzorov, A.A. (2019) Validation of protein backbone structures calculated from NMR angular restraints using Rosetta, J. Biomol. NMR 1-19)

3.1 Introduction

After the NMR spectra of a membrane protein have been accurately assigned, its structure needs to be calculated. Here we utilize a fixed-geometry repeating unit within a protein, termed the peptide plane to represent the possible structural conformations for the protein backbone. A peptide plane is outlined by four atoms in the backbone: an α-carbon, C_α^i, the amide proton *H*, the next α-carbon, C_α^{i+1}, and finally the carbonyl oxygen, *O*. Additionally, contained within the outlined atoms in the plane are the carbonyl carbon C' of the *i*'th residue, and the nitrogen atom, N, which belongs to the *i*+1'th residue. Figure 3.1 shows three peptide planes, each one outlined in red. The orientation of the main magnetic field, B_0, relative to an arbitrarily chosen molecular frame associated with the peptide plane is given by two spherical angles: β, the longitudinal angle, and α, the azimuthal angle. These angles determine the measured frequencies for ^{15}N CSA, ^{1}H-^{15}N dipolar couplings, and any other in-plane resonances. By contrast, for the chiral $C_\alpha H_\alpha$ bond, which is located at the juncture of two adjacent peptide planes, its orientation with respect to B_0 additionally depends on the ϕ torsion angle between the two planes.

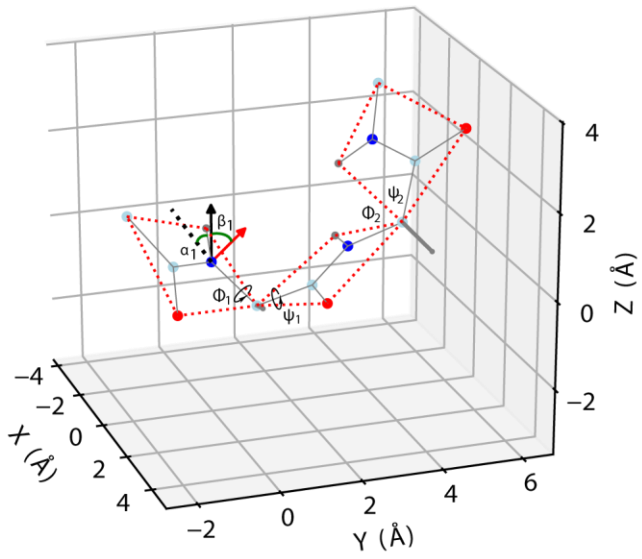


Figure 3.1
An idealized representation for the backbone of a tripeptide unit. The edges of the peptide planes are outlined by the dotted red lines. The main atoms are color coded as: nitrogen in dark blue, carbon in light blue, oxygen in red, and hydrogen in gray. The orientation of the magnetic field B_0 relative to the molecular frame associated with the peptide plane is defined by two spherical angles β, α. The spherical angle β is measured from the external magnetic field, (black arrow) B_0, to the peptide plane normal (red arrow), which forms the z-axis for the molecular frame. The azimuthal angle α is measured between the NH bond, which forms the x-axis of the molecular frame, and the projection of B_0 onto the peptide plane (black dotted line). The relative orientations of the peptide planes are given by the pairs of torsion angles (ϕ_1, ψ_1) and (ϕ_2, ψ_2). Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

Structure determination from angular restraints relies on the measured frequencies to determine the possible peptide plane orientations along a polypeptide. In the mapping of resonance frequencies to peptide plane orientations, finding a consensus structure is primarily hindered by the degeneracy of the possible solutions. For instance, in the case of two experimental NMR dimensions, i.e. ^{15}N CSA and ^{1}H-^{15}N dipolar couplings there can be as many as 16 peptide plane orientations (pairs of β, α) consistent with each resonance (Bertram et al., 2003). Adding a third NMR dimension, namely $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ DC's, significantly reduces the degeneracy (Yin and Nevzorov, 2011).

Another challenge for the structure determination process is experimental uncertainty, which can be thought of as ambiguity in interpreting the positions of the resonances. Such uncertainty can be due to insufficient resolution and slight variations in the dipolar scaling factors arising in composite decoupling pulse sequences (Wu et al., 1994; Dvinskikh et al., 2006; Nevzorov and Opella, 2007). In addition, all resonances are fitted and then back-calculated assuming ideal peptide plane geometry and constant (average) chemical shift tensor principal

values and orientations. In reality, minor deviations from both the ideal peptide plane geometry (Chellapa et al., 2015) and the CSA tensor orientation and its principal values can arise depending on the amino acid type and local structure (Cornilescu et al., 2000; Saito et al., 2010). This may interfere with the interpretation of angular anisotropy of the NMR resonances used during the structural fitting.

A possible strategy to circumvent these problems is to develop a robust automated algorithm for finding a multitude of the possible structural solutions which could then be screened afterwards. One such previously developed structure fitting algorithm walks along the backbone and finds ϕ/ψ pairs that best-fit the adjacent peptide plane resonances (Yin and Nevzorov, 2011). If at a certain residue the resonance frequencies cannot be fit within a desired tolerance, this may indicate implausible orientational solutions found for a previous residue(s). For such a scenario, a step-back or restart was implemented. In the presence of experimental uncertainty, the probability of misfit resonances increased greatly, even at a relatively low value of ±10 Hz. Moreover, by trying to fit a single resonance at a time certain peaks may be over-fit, which may also result in erroneous ϕ/ψ pairs. In other words, the closest fit to the NMR resonance for the current residue may worsen the fit to the subsequent resonances, thus decreasing the average fit quality for the whole spectrum. Such erroneous ϕ/ψ pairs are often imperceptible by their fit to the spectrum alone and, thus, may go undetected. In the presence of experimental uncertainty, a plethora of possible structural solutions must be obtained in the hope that the solution set would still contain low-RMSD structures which could be distinguished from the rest.

It should be noted that structural solutions having large structural RMSDs (relative to the "correct" structure) could still yield good-quality fits to the spectrum. Nevertheless, while these structures may fit the experimental spectrum well, their overall conformations could violate basic protein folding principles and result in implausible topology. For example, the calculated torsion angles could occupy forbidden areas of the Ramachandran plot, or the backbone and side chains could overlap with each other or be too close in space. Moreover, there could be unstructured folds where compact secondary structure is expected, or the fold could be inconsistent with the protein's specific physical environment, such as hydrophobic side chains exposed to an aqueous environment, or vice versa, hydrophilic side chains contained in a hydrophobic lipid membrane interior. While such erroneous structures could violate any number of these conditions, the correct structure should satisfy them all. The Rosetta software package is a bioinformatic tool that has

been very successful in *de novo* modeling of proteins (Chaudhury et al., 2010). The popularity of Rosetta in structural biology is due to its ease of use and the comprehensive, well parameterized scoring functions. These scoring functions include numerous score terms that combine fundamental thermodynamic principles with statistical data compiled from real PDB structures. Moreover, the scoring terms can be made context-dependent, some specifically for soluble proteins and others for proteins immersed in a membrane environment. With the aid of Rosetta scoring terms, a post-fitting protocol can augment the NMR structural fitting algorithm to distinguish the reasonable structures from the ones that violate the aforementioned biophysical principles.

3.2 Analytical framework: Calculating protein structures in the spherical basis

For most angular-dependent NMR observables the orientation of the magnetic field B_0 relative to each peptide plane (molecular frame) is most efficiently represented in the irreducible spherical basis. As was previously mentioned, two spherical angles, β and α, specify the orientation of B_0 relative to a molecular frame associated with a given peptide plane. The three-dimensional irreducible row vector $\vec{Y}$ can be constructed using the following form:

$$\vec{Y}(\beta,\alpha) = \left(-\frac{sin\beta}{\sqrt{2}}e^{i\alpha} \;\; cos\beta \;\; \frac{sin\beta}{\sqrt{2}}e^{-i\alpha}\right) \qquad (eq.\,3.1)$$

To convert the orientation vector into a scalar NMR observable, such as CSA or DC, we invoke an interaction tensor, M, which must be additionally transformed from the molecular frame (M) into its principal axis system (P). This transformation can be expressed by the Wigner rotation matrix $D(\Omega_{\mathrm{MP}})$, and any angular-dependent NMR observable can be written in the following generic form:

$$\nu = \vec{Y}(\beta,\alpha)\cdot[D(\Omega_{\mathrm{MP}})\cdot M\cdot D^{+}(\Omega_{\mathrm{MP}})]\cdot\vec{Y}^{+}(\beta,\alpha) \qquad (eq.\,3.2)$$

Here the superscript "$^{+}$" denotes the Hermitian conjugate. To propagate the orientation vector $\vec{Y}$ from residue *n* to residue *n+1*, a propagator matrix is applied:

$$\vec{Y}(\beta_{n+1}, \alpha_{n+1}) = \vec{Y}(\beta_n, \alpha_n) P(\phi_n, \psi_n, \omega_n) \qquad (eq. 3.3)$$

The propagator consists of the product of three Wigner rotation matrices, which explicitly contains fixed and variable angular parameters along the protein backbone (Yin and Nevzorov, 2011):

$$P(\phi_n, \psi_n, \omega_n) = D\left(\alpha_{NC_\alpha}, \phi_n, \gamma_{tetra}\right) D(0, -\psi_n - 180°, 0) D\left(-\alpha_{NC'C_\alpha}, 180° - \omega_n, -90° - \gamma_{HNC'}\right) \quad (eq. 3.4)$$

Here the angles $\alpha_{NC_\alpha} = 151.8°$, $\gamma_{tetra} = 110.5°$, $\alpha_{NC'C_\alpha} = 115.6°$, $\gamma_{HNC'} = 119.5°$, and $\omega = 180°$, can be treated as constants assuming an ideal peptide plane geometry. Calculating the $C_\alpha H_\alpha$ DC requires a different transformation, viz.:

$$\overrightarrow{Y_{CH}} = \overrightarrow{Y_n} \cdot D\left(270° - \alpha_{NC_\alpha}, \phi_n - 60°, 90° - \gamma_{tetra}\right) \cdot D(0, -90°, 0) \qquad (eq. 3.5)$$

From which the $C_\alpha H_\alpha$ DC can be explicitly calculated from the second element of the vector $\overrightarrow{Y_{CH}}(2)$ as:

$$\nu_{CH} = \chi_{CH} \frac{3\left[\overrightarrow{Y_{CH}}(2)\right]^2 - 1}{2} \qquad (eq. 3.6)$$

where χ_{CH} is the DC constant for the $C_\alpha H_\alpha$ coupling interaction (see below). Assuming a constant peptide plane geometry, only two variables are contained in each propagator, namely the dihedral angles ϕ_n, ψ_n. Equations (3.1-3.5) represent a recursive recipe for mapping the backbone torsions ϕ/ψ onto the NMR observables, each specified by its own interaction tensor M.

The CSA interaction tensor for ^{15}N depends on its three principal components $\sigma_{11}, \sigma_{22}, \sigma_{33}$. Although the values of the principal components vary from residue to residue in real proteins, average tensor values are used in the calculations, and are written in the irreducible basis as:

$$M = \begin{pmatrix} 0.5 * (\sigma_{11} + \sigma_{22}) & 0 & 0.5 * (\sigma_{22} - \sigma_{11}) \\ 0 & \sigma_{33} & 0 \\ 0.5 * (\sigma_{22} - \sigma_{11}) & 0 & 0.5 * (\sigma_{11} + \sigma_{22}) \end{pmatrix} \qquad (eq. 3.7)$$

For Glycine these values (in ppm) are: (41, 64, 215); and (64, 77, 222) for all other residue types. For the transformation of the matrix into the principal axis of the tensor, $D(\Omega_{MP})$, the σ_{33} axis of the CSA tensor was assumed to be first rotated by 18.5˚ off the NH bond within the peptide plane and then tilted by 25˚ off the plane normal about the σ_{22} axis These values have been established experimentally in both solid-state and solution NMR (Lee et al., 1998; Cornilescu et al., 2000). Possible variability in the tensor orientation along the protein is considered in the results and discussion sections. The DC interaction tensors differ only by their coupling constants, χ, each having the same form for the inner matrix M:

$$M = \chi \begin{pmatrix} 0.25 & 0 & -0.75 \\ 0 & -0.5 & 0 \\ -0.75 & 0 & 0.25 \end{pmatrix} \qquad (eq. 3.8)$$

where the coupling constant χ between any two spins 1 and 2 is given by:

$$\chi = \frac{\mu_0 \gamma_1 \gamma_2 h}{16\pi^3 r_{12}^3} \, (Hz) \qquad (eq. 3.9)$$

Here γ_1, γ_2 are the gyromagnetic ratios of the two interacting spins, and r_{12} is the distance between the spins. For ^{1}H-^{15}N the coupling constant is 9,965.4 Hz, and for $C_\alpha H_\alpha$ it is 23,334.7 Hz.

In order to make the matrix-matrix multiplications more efficient, the quantities of equations (3.2-3.5) can be transformed from the *Y* into the *Q*-basis, which diagonalizes the variable part for the rotations involving $\phi, \psi,$ and ω. The *Q*-transformations can be carried out as follows:

$$T_Q = e^{-i\left(3\pi/2 - \alpha_{NC'C_\alpha} - \gamma_{HNC'}\right)L_y} \cdot e^{i\pi/2 L_x} \qquad (eq. 3.10)$$

$$M_Q = T_Q \cdot M \cdot T_Q^+ \qquad (eq. 3.11)$$

yielding the following relations between the "*Y*" and "*Q*" bases:

$$\vec{Q} = \vec{Y} \cdot e^{-i(3\pi/2 - \alpha_{HNC_\alpha})L_y} \cdot e^{-i\pi/2 L_x} \qquad (eq. 3.12)$$

Here the matrices L_x, L_y are given by the rank-1 angular momentum operators:

$$L_x = \frac{1}{\sqrt{2}}\begin{pmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{pmatrix}, L_y = \frac{i}{\sqrt{2}}\begin{pmatrix}0 & -1 & 0\\ 1 & 0 & -1\\ 0 & 1 & 0\end{pmatrix} \qquad (eq.3.13)$$

Since the operations for calculating NMR observables are performed most frequently during the execution of the program, transforming the *Y*-basis into the diagonal *Q*-basis results in a 2-fold improvement in runtime. The *Q*-basis can be transformed back into the *Y*-basis using the inverse matrices corresponding to the operations given by Equations (3.10-3.12).

3.3 Methods

3.3.1 Structure Fitting Algorithm Flowchart

All computer codes and scripts can be provided upon request to the authors. The algorithm for structure calculations has been implemented in the C programming language, which significantly speeds up the calculations *vs.* the previous code (Yin and Nevzorov, 2011) that was written in Matlab (Mathworks®). It should be noted that implementing the new algorithm in an interpreted computer language becomes prohibitively slow due to the sampling demands of the simplex solver while simultaneously fitting two or more peptide planes. The algorithm flowchart is displayed in figure 3.2, having the following basic steps:

(1) The input spectrum consists of a list of NMR resonance targets. A spectrum can either be simulated (option 1A) or read in (option 1B) from a file that includes the starting orientation, β_1, α_1, and the ϕ/ψ torsion angles for every residue.

(2) The master loop administers the calculation of kb_{max} structures, which continues until the last structure is reached, i.e. $kb=kb_{max}$.

(3) For each calculation, "experimental uncertainty" or "noise" can be included in the resonance targets by adding a random number, chosen from a uniform distribution between $-noise_{max}$ and $+noise_{max}$, to each dimension in the spectrum. This effectively puts the resonance to be fitted inside a cube (for three spectral dimensions) with side lengths equal to $2*noise_{max}$. Simulated noise should be utilized on any spectrum, synthetic or real, in which there is uncertainty assumed in the peak center positions. All tolerances must be reset since they may have been altered in the previous run (see **8N,Y**). Due to the random noise added at the beginning, all alterations to tolerances only apply to that run.

(4) Enter loop for calculating structure. Start at first residue, $N=0$.

(5) Is the current residue the first peptide plane, i.e. N=0?

(6) If at the first residue (**6Y**), search for the starting orientation (angles β, α) of the magnetic field relative to the first peptide plane. Then convert the *Y*-basis to the *Q*-basis. Proceed to the torsion angle search (**6N**). If not at the beginning of the sequence then bypass **6Y** and proceed directly to **6N**. Set the tolerance for the current residue. Search for ϕ/ψ's using a simplex search function in a loop that runs *simits* iterations. The number of torsion angles to be searched at once will be the 2**stride*; *stride* is the number of planes searched at once. Score the calculated resonances to the target spectrum (*z0*). To speed up the program, break the simplex loop if *z0* is less than *tol2* (optional).

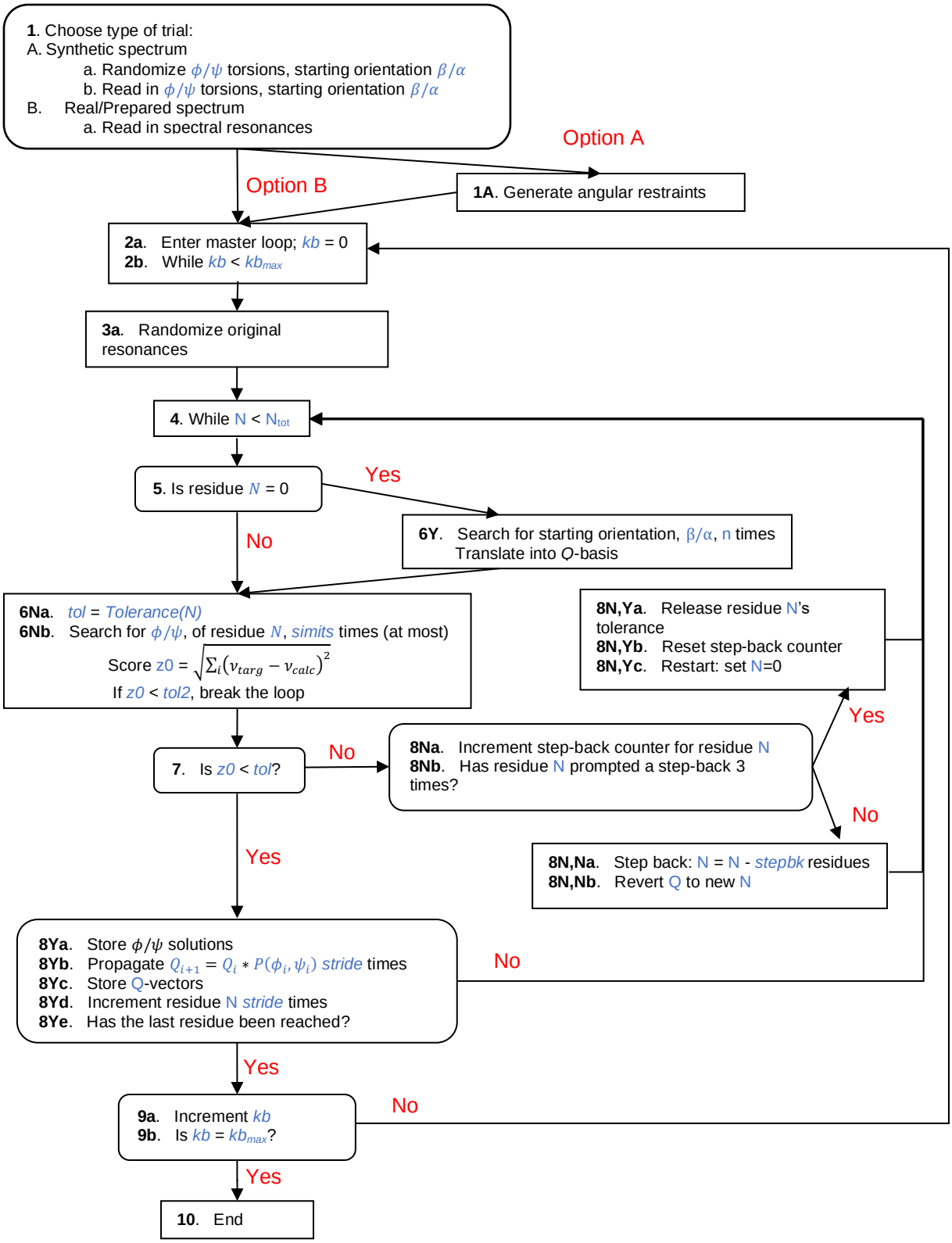
(7) Is the score to the spectrum, *z0*, below the tolerance for residue *N*?

(8) If *z0* < *tol* (**8Y**) then store the torsion angles, propagate the orientation vector *Q* from residue *N* to residue *N*+*stride*, and increment *N*. Store the *Q*-vectors. If no *z0* was found below the tolerance (**8N**) increment the step-back counter for residue *N*. If residue *N* fails to pass its tolerance 3 times (**8N,Y**), release its tolerance, reset the step-back counter, and step-back to the first residue. Otherwise (**8N,N**) step-back to residue *N-stepbk* and revert Q_N to $Q_{N\text{-}stepbk}$. If residue *N* is the last in the sequence, proceed to **9;** otherwise, go back to **4**.

(9) Structure calculation for structure *kb* is complete. Store the starting orientation and torsions. Increment *kb*. If $kb=kb_{max}$, exit the main loop and proceed to **10**. Otherwise go back to **2**.

(10) Finish the program. Print any pertinent information to the output files (torsion angles, atomic coordinates in the PDB format).

Figure 3.2
Flowchart for the structure calculation algorithm. Variables in the program are represented with blue text. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.


1. Choose type of trial:
A. Synthetic spectrum
a. Randomize ϕ/ψ torsions, starting orientation β/α
b. Read in ϕ/ψ torsions, starting orientation β/α
B. Real/Prepared spectrum
a. Read in spectral resonances
Option A
Option B
1A. Generate angular restraints
2a. Enter master loop; kb = 0
2b. While kb < kb_{max}
3a. Randomize original resonances
4. While N < N_{tot}
5. Is residue N = 0
Yes
No
6Y. Search for starting orientation, β/α, n times
Translate into Q-basis
6Na. tol = Tolerance(N)
6Nb. Search for ϕ/ψ, of residue N, simits times (at most)
Score z0 = $\sqrt{\sum_i (v_{targ} - v_{calc})^2}$
If z0 < tol2, break the loop
8N,Ya. Release residue N's tolerance
8N,Yb. Reset step-back counter
8N,Yc. Restart: set N=0
Yes
7. Is z0 < tol?
No
8Na. Increment step-back counter for residue N
8Nb. Has residue N prompted a step-back 3 times?
No
Yes
8N,Na. Step back: N = N - stepbk residues
8N,Nb. Revert Q to new N
Yes
8Ya. Store ϕ/ψ solutions
8Yb. Propagate $Q_{i+1} = Q_i * P(\phi_i, \psi_i)$ stride times
8Yc. Store Q-vectors
8Yd. Increment residue N stride times
8Ye. Has the last residue been reached?
No
Yes
9a. Increment kb
9b. Is kb = kb_{max}?
No
Yes
10. End

The quality of the output structures will depend on the size of the protein, the number of planes being evaluated per iteration, and the noise level being added to the spectral targets. The number of planes that are fit simultaneously is set by the variable *stride*. Since the simplex function is the bottleneck for program runtime, *simits* should not exceed an amount necessary to reliably find the lowest ϕ/ψ solutions for the majority of the runtime. With a larger value for *stride*, more simplex iterations will be necessary to ensure the lowest solution is found for every residue. The number of residues in the step-back move is set by *stepbk*, which was always set to 5 for these calculations. The experimental uncertainty level is set by *noisiness* and corresponds to the maximum deviation (in Hz) from the "true" spectral target in each dimension. The higher *stride* and *noisiness* are, the higher the tolerance criteria, *tol*, should be set. The program can be sped up by setting *tol2* at a value that below which there can be confidence that the simplex has already found the lowest function value for the current residue. Determining appropriate values for *tol* and *tol2* requires trial and error with the program. The general procedure used to find the right values involves setting *tol* high and *tol2* low, and gradually increasing and decreasing the two values, respectively, up to the point that they start compromising the quality of the structural solutions. Finally the number of structural solutions can be set by *kb_max*.

For the calculation of synthetic spectra, input csv files were prepared with all relevant information. A script was written in python, *torsions.py*, to read in the coordinates of a PDB file and calculate the ϕ/ψ angles for every residue. The ϕ/ψ torsions were written to a csv file whose top line was a single integer for the number of residues in the protein, the second line contained the one-letter amino acid codes for every residue, and the third and fourth lines contained the list of ϕ and ψ torsion angles, respectively. With the backbone torsion angles and a specified pair of angles (β, α) for the starting orientation, the spectrum was calculated using equations 3.1-3.6 assuming ideal peptide plane geometry. Angles (β, α) were selected for membrane protein *4a2n* so that the helices spanning the membrane would be on average oriented along the z-axis, which represents the most biologically realistic orientation for transmembrane helices. There were multiple outputs for the program. The specific angles and parameters used in the calculations of the peptide plane, from Eq. 3.4-3.9, were written to a csv file named *angles.csv*, from which the calculations could be reproduced in post-fitting evaluation of the results. The results of the structure calculations were written to a file named *output.csv*. The structure of the csv file started with a comment line at the top, which included specific details for the simulation. The second line

contained 2 integers, the first of which was the number of residues in the protein being measured, and the second was the total number of structural solutions calculated during the program run (*kb_max*). The third line was the one-letter amino acid sequence. All remaining lines are successive structural solutions that include 3 lines per solution: the first line is the (β, α) starting orientation solution obtained for that particular structure, and the second and third lines are the lists of ϕ and ψ torsions, respectively. For example, if a program run collected 1,000 solutions then the output file would contain 3,003 lines total. Finally the program writes another output in the same structure as *output.csv* containing only the orientation and ϕ/ψ torsions of the real system from which the spectrum was generated, named *real.csv*.

Program runtime largely depended on the value for *simits* used in the simulation. With the simplex function being the execution bottleneck in the program, running with *stride=1* was roughly an order of magnitude faster than *stride=2*. For single-plane fitting the runtime per structural solution on a desktop computer equipped with i7 2GHz Intel^{TM} processor ranged between 5-30 seconds, largely depending on how many step-back moves were made for a particular solution. For fitting two planes at once the runtime was 60-220 seconds per solution.

3.3.2 Calculations of spectral and structural RMSDs

The figure of merit for spectral fitting is the magnitude of deviations from the target resonances, which can be calculated for a set of kb_{max} solutions as:

$$\overline{\Delta v_k^2} = \frac{\sum_j^{kb_{max}} \sum_i^N \frac{\left(|C_{i,k} - T_{i,k}|\right)^2}{N}}{kb_{max}} \qquad (Eq. 3.14)$$

The results of Eq. 3.14, $\overline{\Delta v_k^2}$, for each NMR dimension are then used to calculate the spectral RMSD of the fit to the spectrum (in Hz) as:

$$\Delta X = \sqrt{\sum_k^M \overline{\Delta v_k^2}} \ (Hz) \qquad (Eq. 3.15)$$

Here $C_{i,k}$ and $T_{i,k}$ are the calculated and target values for M-dimensional resonance, i in an N-residue protein in the k'th spectroscopic dimension. The average is taken over all calculated

structures. In Eq. 3.15 the magnitude is taken over the M dimensions of the NMR spectrum being fit. In this work kb_{max} was always 1000 and M is 3 (corresponding to ^{15}N CSA, and ^{1}H-^{15}N and $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings).

Structural RMSD calculations have been carried out using the python script *plotstruct.py*. This script reads in the orientation of the first peptide plane and the torsion angles, and calculates the N-residue structure into an (3N)x3 array of points for atoms along the backbone. The script then reads in the torsions from *real.csv*, generates the "true" (PDB) backbone structure coordinates in the same size array and calculates the structural RMSD using the Kabsch algorithm (Kabsch, 1976). Structural RMSD calculations are composed of a rigid translation to bring each structure's center of mass into coincidence, and a rigid rotation to minimize the distance of all corresponding points in two structures.

3.3.3 Rosetta post-fit filtering of structures

PyRosetta was used in order to utilize the Rosetta scoring functions. Inside the python scripts *score_output.py*, *membrane_score_output.py* the protein was assembled in a Rosetta *pose* object, using the output ϕ/ψ torsions from the program. The *pose* object was then scored in both full atom and centroid (coarse grained side chain representation) forms. The full atom function used for both soluble and membrane proteins was *talaris2013* (Kuhlman et al., 2000; Kuhlman et al., 2003; Rohl et al., 2004; Leaver-Fay et al., 2013; O'Meara et al., 2015). The centroid function used for soluble proteins was *score3* (Rohl et al., 2004), and for membrane proteins it was *mpframework_cen_2006* (Alford et al., 2015). In order to further customize score functions, the values for the individual terms of each score function were written to file for all outputs from the original program. The python script *rosetta_scores.py* read in the unweighted individual terms and scores and recombined them with custom weights, effectively creating new custom scoring functions. The program also was used to scan different combinations of score terms in order to develop the most effective scoring function.

PyRosetta was used to distinguish the low-RMSD structures from the high ones. This involved multiple custom scoring functions that filtered results in two steps. The first filtering tier was a coarse search from which the top 20 solutions were extracted. The second tier applied a slightly different scoring function to sort out and rank those top 20 solutions, after which the top

10 solutions were reported as the final answer. The scoring functions were customized by finding the scoring terms, both full atom and centroid, that best correlated with structural RMSD in the two proteins being tested. Combinations of the scoring terms were scanned for those that produced the lowest median structural RMSDs in their top solutions. Ultimately the consensus scoring functions, i.e. the terms and their weights, were determined based on those that produced reasonable final strcutural RMSDs at all noise levels.

Table 3.1: Custom Rosetta score functions for soluble (s) and membrane (m) proteins. Table obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

Score Function	Term	Weight
1s	*rg*	1.0
1s	*env*	1.0
1s	*cbeta*	1.0
2s	*fa_atr*	0.2
2s	*fa_sol*	1.0
1m	*mp_env*	1.0
1m	*Rama*	0.5
2m	*fa_atr*	0.2
2m	*hbond_sr_bb*	1.0
2m	*p_aa_pp*	1.0
2m	*fa_mpsolv*	1.0

Table 3.1 lists the terms of the 2 scoring functions for soluble proteins, denoted with the letter ‘s’. Each scoring function is constructed by summing up the individual terms with the relative weights as given. These scoring terms are specific to soluble proteins (Leaver-Fay et al., 2013; O’Meara et al., 2015) and thus cannot be used for proteins in a membrane environment (although most terms have analogs for membrane proteins and vice versa). Analysis on 2gb1 solutions from the program found that term *rg* had the best structural RMSD correlation amongst all the structural solutions. This represents a statistical term favoring compact structures, i.e. having minimal radius of gyration. Terms *env* and *cbeta* also favor compact structures, and had the next best correlations.

Term *env* is a solvation term for every residue based on their hydrophobicity, and *cbeta* is a solvation term correcting for excluded volume. Score function 1s contains all centroid terms, whereas score function 2s only contains full-atom terms. Term *fa_rep* is the Lennard-Jones repulsion energy between pairs of atoms, and *fa_solv* is the Lazaridis-Karplus solvation energy. The two scoring functions used for membrane proteins are denoted with the letter 'm' in Table 3.1. Term *mp_env* is a statistical term describing the likelihood of a specific residue being at the calculated depth in the modeled membrane; *rama* refers to the backbone torsion-angle preferences by residue type on the Ramachandran map. Once again, the second scoring function consists of terms that score full-atom structures and contain a mix of physical and statistical criteria. Term *fa_atr* refers to the Lennard-Jones attractive energy; *hbond_sr_bb* scores the hydrogen bonding between backbone atoms; *p_aa_pp*, similar to *rama*, is the probability for a given amino acid to have a given ϕ/ψ torsion pair; *fa_mpsolv* scores pairs of residues based on their depth in lipid bilayer. Scoring terms *mp_env* and *fa_mpsolv* are specific to membrane environments and available from Rosetta's membrane modeling package *RosettaMP* (Leman et al., 2014; Alford et al., 2015).

3.4 Results

3.4.1 Fitting of synthetic data for GB1 protein

All 55 residues of the soluble GB1 protein (PDB ID 2gb1) were used for generating the spectrum consisting of ^{15}N CSA, and ^{1}H-^{15}N and $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings. The first peptide plane orientation was arbitrarily set to $(\beta, \alpha) = (0.5, 1.0)$. There were 1000 structural solutions collected at various experimental uncertainty ("noise") levels of 10, 30, and 50 Hz and *stride* number (either 1 or 2 planes). For the three noise levels *tol* was 50, 70, and 90 Hz respectively, and *tol2* was always 20 Hz. When *stride*=1 (single-plane fitting), *simits* was set to 100, and increased to 1000 when *stride*=2 (two-plane fitting). The structural RMSDs of 1000 back-calculated structures were binned as histograms shown in Fig. 3.3. The structural RMSDs were calculated relative to a "true" structure having the torsions angles calculated from the PDB coordinates. For each experimental uncertainty ("noise" level) the 1000 structural solutions were put through the Rosetta post-fitting protocol as described in the Methods section. Using the 1000 sets of ϕ/ψ torsions, the proteins were reconstructed in PyRosetta and then scored. Figure 3.4 shows the resulting scores from the 2 custom Rosetta scoring functions, 1s and 2s, with the 20

lowest scoring structures in blue. The top 10 solutions for each noise level are depicted in Fig. 3.5.

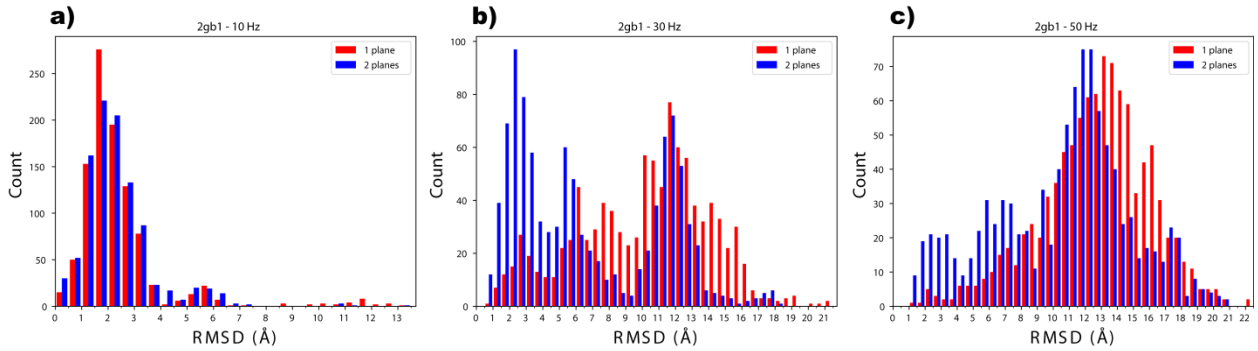


Figure 3.3
Structural RMSD histograms (relative to the original PDB structure coordinates measured in Å) for 1000 structures of protein 2gb1 back-calculated from 3 synthetic NMR angular restraints per plane (^{15}N CSA, ^{1}H-^{15}N, $^{1}H_{\alpha}$-$^{13}C\alpha$ DC's) with experimental uncertainty of: a) 10, b) 30, and c) 50 Hz. The results of runs involving fitting one plane at a time (*stride=1*) are shown in red while the histograms obtained from fitting two planes at a time (*stride=2*) are shown in blue. Each histogram contains 1000 runs total for each value of *stride*. Histograms are binned in 0.5Å steps. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

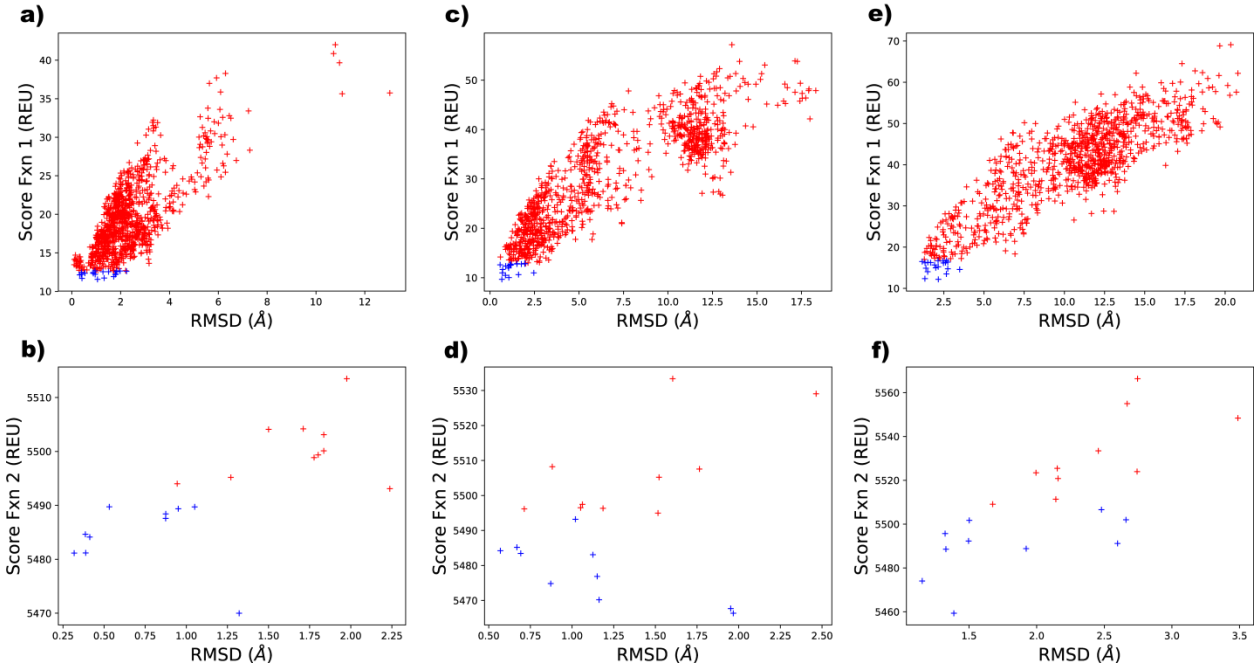


Figure 3.4
Scatter plots for structural RMSDs vs. Rosetta scores for 2gb1. Plots a/b, c/d. and e/f, correspond to noise levels of 10, 30, and 50 Hz, respectively. The y-axes represent the Rosetta function scores, in REU (Rosetta energy units). Top plots result from Score Function 1s applied to all 1000 structural solutions from the program output. Scatter points in blue are the lowest 20 scoring structures for the Score Function 1s, which are not necessarily the 20 lowest structural RMSD values. Bottom plots result from Score Function 2s applied to the 20 lowest scoring structural solutions from the previous scoring function. Scatter points in blue are the 10 lowest scoring structures. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

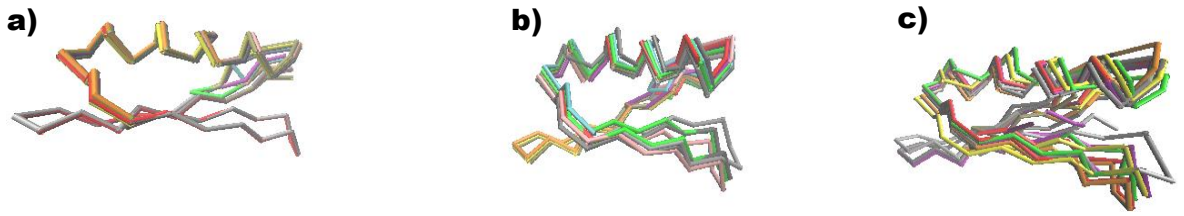


Figure 3.5
Overlays of top 10 solutions for soluble protein 2gb1 for noise levels a) 10, b) 30, and c) 50 Hz. The 10 solutions for each noise level correspond to the structural RMSD values listed in Table 3.3. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

3.4.2 Fitting of synthetic data for a transmembrane helical hairpin (pdb id 4a2n)

The first two transmembrane helices of integral membrane methyltransferase protein (PDB id 4a2n) (Yang et al., 2011) were used to calculate the spectrum, totaling 64 residues. The starting orientation, $(\beta, \alpha) = (0.0, 3.14)$, was chosen to orient the helices to be approximately parallel with the membrane normal. All the same noise levels, values of *stride*, and *simits* as for 2gb1 were used for 4a2n. For the three noise levels *tol* was 30 Hz, 50 Hz, 70 Hz, and *tol2* was 5 Hz, 20 Hz, and 20 Hz, respectively. The back-calculated structures have been analyzed and filtered similarly to the GB1 protein, albeit using the scoring functions for membrane proteins. The results are shown in Figs. 3.7-3.8.

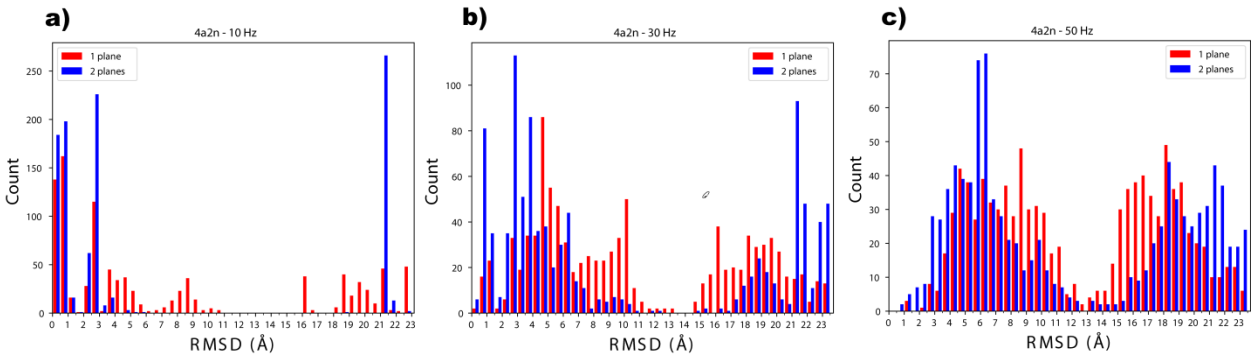


Figure 3.6
Structural RMSD histograms (relative to the original structure) for 1000 structures of protein 4a2n back-calculated from 3 synthetic NMR angular restraints per plane (^{15}N CSA, ^{1}H-^{15}N, ^{1}H$_{\alpha}$-^{13}Cα DC's) with experimental uncertainty of: a) 10, b) 30, and c) 50 Hz. The results of runs involving fitting one plane at a time (*stride=1*) are shown in red while the histograms obtained from fitting two planes at a time (*stride=2*) are shown in blue. Each histogram contains 1000 runs total for each value of *stride*. Histograms are binned in 0.5Å steps. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

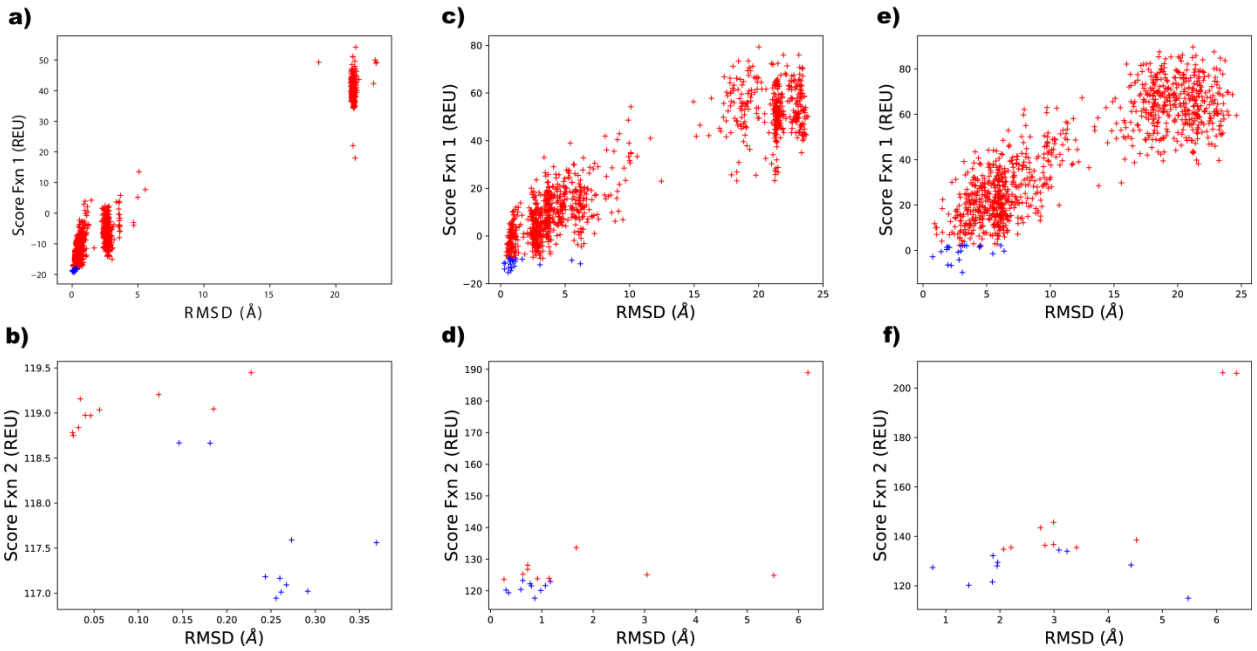


Figure 3.7
Scatter plots for structural RMSDs vs. Rosetta scores for 4a2n. Plots a/b, c/d. and e/f, correspond to noise levels of 10, 30, and 50 Hz, respectively. The y-axes represent the Rosetta function scores, in REU (Rosetta energy units). Top plots result from Score Function 1m applied to all 1000 structural solutions from the program output. Scatter points in blue are the lowest 20 scoring structures for the Score Function 1m, which are not necessarily the 20 lowest structural RMSD values. Bottom plots result from Score Function 2m applied to the 20 lowest scoring structural solutions from the previous scoring function. Scatter points in blue are the 10 lowest scoring structures. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

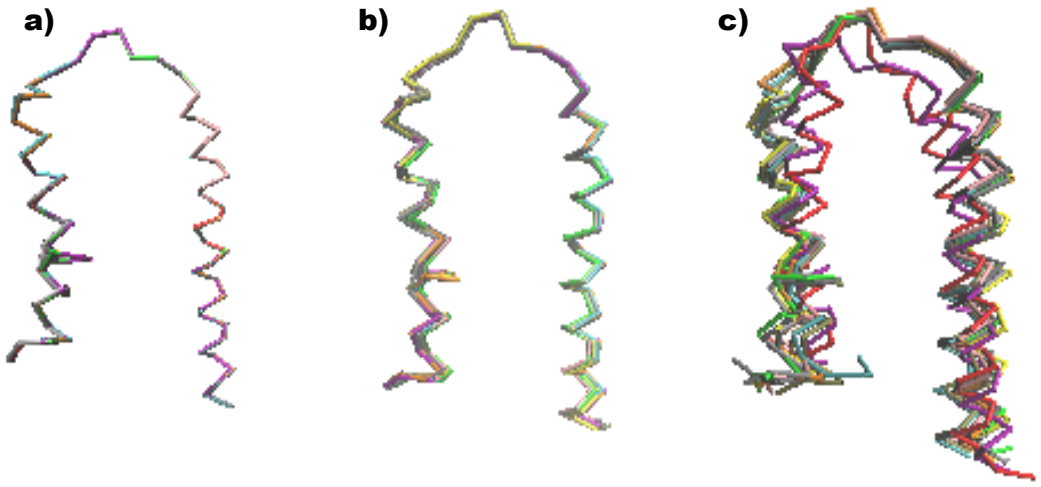


Figure 3.8
Overlays of 10 top-scoring structures for membrane protein 4a2n for a) 10, b) 30, and c) 50 Hz of experimental uncertainty, *stride*=2. The 10 solutions in each picture correspond to the structural RMSD values in Table 3.3. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

3.4.3 Variations in the tensor orientation

A separate trial was performed to quantify the effects for the potentially variable orientation of the σ_{33} axis of the CSA tensor relative to the NH bond. For this trial, the three-dimensional 2gb1 spectrum was generated by randomly varying the σ_{33} angle off of the NH bond within a uniform distribution from 18˚ to 19˚ for every residue; in the previous trials, this angle was set to 18.5˚ for all residues. The calculated resonances deviated from those calculated by assuming a constant σ_{33} angle by a maximum of 79.7 Hz with a mean of 25.3 Hz per resonance. This mean deviation most closely resembles the trials of Fig. 3.3c run with 50 Hz of experimental uncertainty,. The structures were then back-calculated by assuming a constant average value for the σ_{33} angle of 18.5º. Figure 3.9 shows the histogram of the structural RMSD values alongside the results for 2gb1 with 50 Hz of experimental uncertainty for direct comparison (cf. Fig. 3.3c). As can be seen from Fig. 3.9, the distributions of the structural solutions are similar. Moreover, structural RMSDs well below 2 Å are still obtainable even if the spectra generated with a variable σ_{33} angle are fitted assuming a constant (average) CSA tensor orientation. Thus, the uncertainty in the tensor orientation can be adequately represented by using a uniform experimental uncertainty for the spectral positions.

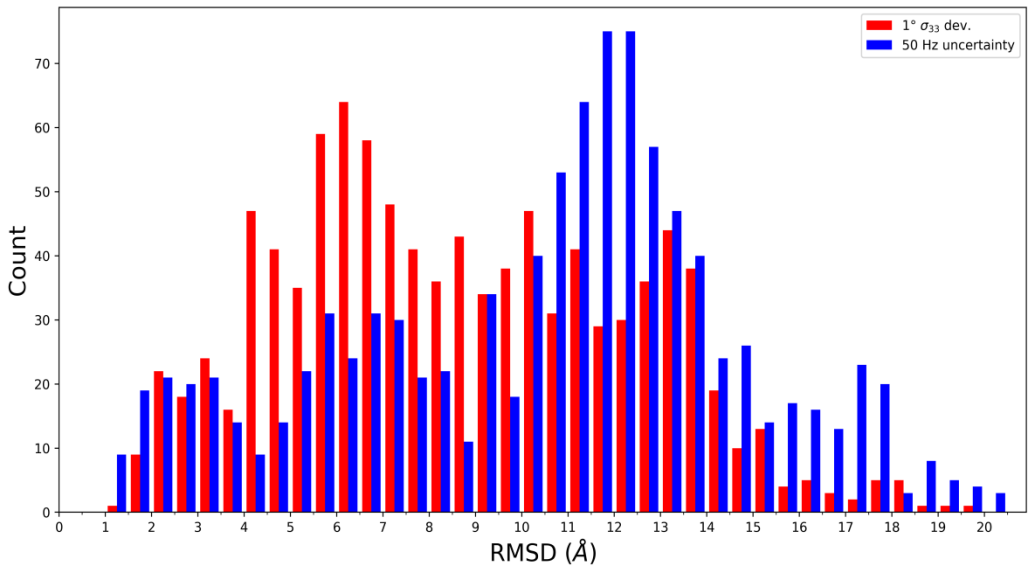


Figure 3.9
Structural RMSD histogram for the synthetic spectra of 2gb1 generated by using a random uniform distribution of the σ_{33} angle within 1˚, centered at 18.5˚. One thousand structures are back calculated on this spectrum using a fixed angle of 18.5º (red). The histogram includes 1000 structures obtained from a spectrum generated by adding random uniform noise of ±50 Hz (from Fig. 3.3c) and is shown in blue for direct comparison. Figure obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

3.5 Discussion

The highly correlated nature of angular restraints results in extreme ruggedness in the spectral RMSD landscape, which may render a global minimization search problematic. While the popular software package Xplor-NIH (Schweiters et al., 2003) predicts structure via molecular dynamics augmented with experimental restraint potentials, such as distance restraints and implicit solvation potentials (Tian et al, 2014; Tian et al. 2017), our algorithm outlined in Fig. 3.2 attempts a *de novo* structure prediction from the NMR angular restraints alone. Such a heavy reliance on the angular restraints necessitates at least three (3) NMR restraints per peptide plane in order to obtain a convergent set of structures. Peptide planes with less than 3 restraints greatly increase the degeneracy of structural solutions, making the correct structure less likely to be found. Due to the above reasons, a sequential walk along the backbone appears at present to be a more practical method for solving a structure entirely from NMR angular restraints. Using the simplex solver results in the correct torsion angles for the majority of residues, even when 50 Hz of experimental uncertainty is present throughout the spectrum. Yet miscalculating only a few pairs of the torsion angles, especially those in turn regions between the secondary structure elements, may result in the calculated structures being very different from the real structure. Even though some segments

of the calculated structure may closely match parts of the real structure, the overall tertiary fold can still have a high structural RMSD value. Despite this potential pitfall our algorithm is capable of sampling a vast conformational space thereby retaining the relevant torsion angles along the backbone. In some cases, occasionally missed torsional pairs along the backbone can be mitigated by selecting compensatory ϕ/ψ torsions in the subsequent residues that would largely return the subsequent plane(s) to the correct orientation.

A notable modification of the algorithm as compared to the previous work (Yin and Nevzorov, 2011) involves the capability of fitting NMR resonances simultaneously for two peptide planes (4 torsion angles) instead of fitting a single plane at a time. As previously mentioned, a potential pitfall for any sequential structure fitting algorithm is that only a few incorrectly determined torsion angles may compromise the overall fold and yield high structural RMSDs to the actual structure. This can occur even if the fit to the spectrum is acceptable. A plausible reason for the discord between the spectral RMSD vs. structural RMSD is that incorrect torsions can provide better fits to the spectrum for certain residues (especially if experimental uncertainty is large), thus being more likely selected by the minimization algorithm. Adding more residues to be fit simultaneously helps in rejecting the (incorrect) solutions that may fit the *i*'th residue very closely while being inconsistent with the NMR data for the residues downstream (i.e. *i*+1 and *i*+2) due to the incompatible orientations of their peptide planes. It was determined that increasing the number of simultaneously fitted planes in the simplex solver to just two largely avoids over-fitting certain NMR resonances while being computationally feasible. The solver could always find dozens of structural solutions within 2 Å structural RMSD relative to the starting structure (among 1000 iterations). Adding a third plane to the solver required an order of magnitude more simplex iterations to reliably find the correct solution, which considerably slowed down the execution time while not appreciably affecting the efficacy of the search.

Table 3.2: Spectral RMSDs of fits using one vs. two-plane method. Table obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

Protein	2GB1		4A2N	
Experimental uncertainty (Hz)	1 Plane	2 Planes	1 Plane	2 Planes
10	86	37	165	38
30	543	125	246	119
50	628	212	366	277

All values are in Hz

A direct comparison between the above two methods of fitting is presented in Table 3.2 and Figures 3.3 and 3.6. With the (minor) exception of 4a2n at 50 Hz experimental error, the spectral RMSDs (in Hz) for the one-plane method are more than double that for the 2 planes. Since these values are averages of 1000 structures in each case, it can be concluded that multiple planes consistently improve the fit to the spectrum. More importantly, this should lead to solutions with lower structural RMSDs to the PDB structures. Although there is no guarantee that a single structural solution that better fits the spectrum will have a lower structural RMSD to the PDB coordinates, on average the distribution of structural RMSDs shifts towards lower values as compared to under-fitted spectra. This can be seen in the histograms of Figures 3.3 and 3.6 from the intensity levels on the left hand side (< 2Å RMSD). Owing to their low structural RMSDs, the fraction of these solutions relative to the total number of possible structural solutions determines the success rate for the algorithm. At every noise level there is consistently a higher proportion of sub-2Å solutions for the two-plane fitting than for the single plane fitting among the 1000 calculated structures. While the improvement is marginal at 10 Hz uncertainty level for 2gb1 (cf. Fig. 3.3a), fitting two planes at a time has proven to have a more pronounced effect when the experimental uncertainty is increased (Figures 3.3b,c), where the ratio of low-RMSD solutions for two- vs. single-plane fitting exceeds 5:1. The same trend persists for 4a2n (Fig. 3.5): at 50 Hz of experimental uncertainty there are roughly twice as many “correct” solutions for 4a2n when two peptide planes are fit instead of one. These results suggest that fitting two planes simultaneously in the simplex solver is highly beneficial for retaining good structural solutions without considerably increasing the program runtime.

Nevertheless, the histograms in Figures 3.3 and 3.6 show that even at the lowest noise levels, the majority of the solutions are still expected to have greater than 2Å structural RMSD, thus being unacceptable. At the highest noise levels, less than 1% of the solutions contain acceptable structures. Since all these solutions fit the CSA and dipolar couplings almost equally well, it is clear that fitting the angular restraints cannot by itself ensure correctness of the calculated structure, except, perhaps single helices with highly constrained torsion angles (Thiriot et al., 2004). The obtained 1000 structural solutions would still likely contain a number of acceptable structures, at which point the problem becomes to distinguish the realistic structural solutions from the implausible ones. Through various Rosetta scoring terms, which are compiled from a plethora of statistics derived from real structures, one can identify the most likely global folds amongst the structures calculated from the spectral fit.

The scatter plots of Fig. 3.4 demonstrate that the custom Rosetta scoring functions for 2gb1 are capable of filtering out the most implausible solutions. Although some low RMSD structures may have a higher/worse Rosetta score, it is important to note that the inverse generally does not happen, i.e. high-RMSD structures do not yield lowest/best Rosetta scores. For Score Function 1s the overall correlations between structural RMSD and score are strong, but these correlations alone do not constitute a sole reliable metric for obtaining the final results. The primary goal for the first scoring function is to separate out the most unrealistic solutions, while retaining low to moderate RMSD structures. Therefore, the first selection of solutions (20 for these calculations) may need to be subjected to the second scoring function. The distribution of structural RMSDs in the top 20 solutions can be seen in Figures 3.4b,d,f. For each noise level the Score Function 1s solutions contained structures over 2Å. Score Function 2s appears to have a stronger correlation between the structural RMSD (in Å) and Rosetta score, and filters out the remaining unrealistic structures among these 20 solutions from the realistic, low RMSD ones. The results for 2gb1 in Table 3.3 shows that applying different custom Rosetta scoring functions in series can be very effective at all noise levels, with the majority of structures selected in the final list having very low structural RMSD values. As the experimental uncertainty increases, a number of higher-RMSD structures may still end up among the final 10 solutions. Neither 10 nor 30 Hz experimental uncertainty contains any structures having RMSDs greater than 2Å, but for 50 Hz there are three such structures (in between 2-3Å).

Table 3.3: Structural RMSD of top 10 solutions for 2gb1 and 4a2n. Table obtained from Lapin and Nevzorov, 2019, J. Biomol. NMR, 1-19.

	2gb1			4a2n		
Uncertainty Level	10 Hz	30 Hz	50 Hz	10 Hz	30 Hz	50 Hz
1	**1.32**	**1.97**	**1.39**	**0.26**	**0.86**	**5.48**
2	**0.32**	**1.95**	**1.15**	**0.26**	**0.35**	**1.43**
3	**0.39**	**1.16**	**1.33**	**0.29**	**0.98**	**1.86**
4	**0.41**	**0.87**	**1.92**	**0.27**	**0.30**	**0.76**
5	**0.39**	**1.15**	**2.60**	**0.26**	**0.59**	**1.95**
6	**0.87**	**1.13**	**1.50**	**0.24**	**0.80**	**4.43**
7	**0.87**	**0.69**	**1.32**	**0.37**	**1.08**	**1.96**
8	**0.95**	**0.57**	**1.50**	**0.27**	**0.77**	**1.87**
9	**1.05**	**0.67**	**2.66**	**0.18**	**1.17**	**3.24**
10	**0.53**	**1.02**	**2.48**	**0.15**	**0.63**	**3.09**

The results for 4a2n are very similar to that of 2gb1: the post-fitting Rosetta protocol yielded 10 solutions with largely acceptable structural RMSDs at all experimental uncertainty levels. One noticeable difference is that the final top scoring solution for 50 Hz has a significantly higher RMSD of 5.48 Å (cf. Table 3.3); additionally there were three other structures present having RMSD> 2Å. Closer observation of Fig. 3.7f shows that a number of lower RMSD structures, between 2-3.5Å, were spurned in favor of a few high RMSD structures with low score function 2 scores. Overall, most of the structures from 50 Hz were acceptable, yielding a consensus structure. Another feature of the 4a2n trials that distinguishes it from that of 2gb1 is that the RMSDs of the structural solutions are distributed less continuously. Figures 3.6a,c,e show that Rosetta segregates the structural solutions essentially into two groups with higher and lower RMSD values. A possible reason for this segregation is that 4a2n could be considered structurally simpler than 2gb1. In general, 4a2n has 2 secondary structural elements (α-helices) separated by a single turn, whereas 2gb1 has 1 helix and 2 β-sheets that are separated by 3 turn regions overall. It is likely that the turn regions pose consequential obstacles for the algorithm, in that they determine the relative orientations of the larger secondary structure elements. Nevertheless, the post-fitting Rosetta screening protocol still correctly identified the low-RMSD structures amongst the 1000 total solutions.

In general, the utilization of the *mp_env* and *rama* terms are especially useful for screening the calculated structures of membrane proteins. Frequently, in solving for 4a2n structure, the algorithm would fail to orient the second helix back into the membrane, yielding one long, straight helix. The term *mp_env* describes the burial depth into the membrane and thus heavily penalizes

such a structure where a hydrophobic α-helix is exposed to a soluble environment. The *rama* term (and similar term *p_aa_pp*) is also indispensible in filtering out physically impossible structures and providing a simultaneous validation for the calculated torsion angles to comply with the Ramachandran map. Finally, in this study it was shown that two scoring functions were sufficient to achieve the end goal of separating low RMSD structures from the high ones. It is possible that additional scoring functions in the protocol may be necessary to achieve better scoring of the structural solutions calculated from NMR angular restraints.

3.6 References

Alford, R.F., Leman, J.K., Weitzner, B.D., Duran, A.M., Tilley, D.C., Elazar, A., Gray, J.J. (2015) An Integrated Framework Advancing Membrane Protein Modeling and Design, PLOS Computational Biology 1-23

Bertram, R., Asbury, T., Fabiola, F., Quine, J.R., Cross, T.A., Chapman, M.S. (2003) Atomic Refinement with Correlated Solid-State NMR Restraints, JMR 163:300-309

Chaudhury, S., Lyskov, S., Gray, J.J. (2010) PyRosetta: A Script-Based Interface for Implementing Molecular Modeling Algorithm Using Rosetta, Bioinformatics 26:689-691

Chellapa, G.D., Rose, G.D. (2015) On Interpretation of Protein X-ray Structures: Planarity of the Peptide Unit, Proteins 83:1687-1692

Cornilescu, G., Bax, A. (2000) Measurement of Proton, Nitrogen, and Carbonyl Chemical Shielding Anisotropies in a Protein Dissolved in a Dilute Liquid Crystalline Phase, J. Am. Chem. Soc. 122:10143-10154

Dvinskikh, S., Yamamoto, K., Ramamoorthy, A. (2006) Heteronuclear Isotropic Mixing Separated Local Field NMR Spectroscopy, J. Chem. Phys. 125:034507

Kabsch W. (1976) A Solution for the Best Rotation to Relate Two Sets of Vectors, Acta Crystallographica A32:922-923

Kuhlman, B. and D. Baker (2000) Native Protein Sequences are Close to Optimal for their Structures, Proceedings of the National Academy of Sciences of the United States of America, 97(19):10383-10388.

Kuhlman, B., Dantas, G., Ireton, G. C., Varani, G., Stoddard, B. L., Baker, D. (2003) Design of a Novel Globular Protein Fold with Atomic-Level Accuracy, Science 302(5649):1364-1368

Lapin, J., Nevzorov, A.A. (2019) Validation of Protein Backbone Structures Calculated from NMR Angular Restraints Using Rosetta, J. Biomol. NMR 1-19

Leaver-Fay, A., O'Meara, M. J., Tyka, M., Jacak, R., Song, Y., Kellogg, E. H., Thompson, J., Davis, I. W., Pache, R. A., Lyskov, S., Gray, J. J., Kortemme, T., Richardson, J. S., Havranek, J. J., Snoeyink, J., Baker, D., Kuhlman, B. (2013) Scientific Benchmarks for Guiding Macromolecular Energy Function Improvement, Methods in enzymology 523:109

Lee, D.K., Wittebort, R.J., Ramamoorthy, A. (1998) Characterization of ^{15}N Chemical Shift and ^{1}H-^{15}N Dipolar Coupling Interactions in a Peptide Bond of Uniaxially Oriented and Polycrystalline Samples by One-Dimensional Dipolar Chemical Shift Solid-State NMR Spectroscopy, JACS 120: 8868-8874

Leman, J.K., Ulmschneider, M.B., Gray, J.J. (2014) Computational Modeling of Membrane Proteins, Proteins Struct. Funct. Bioinf 83:1-24

Nevzorov, A.A., Opella, S.J. (2007) Selective Averaging for High-Resolution Solid-State NMR Spectroscopy of Aligned Samples, JMR 185:59-70

O'Meara, M. J., Leaver-Fay, A., Tyka, M., Stein, A., Houlihan, K., DiMaio, F., Bradley, P., Kortemme, T., Baker, D., Snoeyink, J. (2015) A Combined Covalent-Electrostatic Model of Hydrogen Bonding Improves Structure Prediction with Rosetta. Journal of Chemical Theory and Computation, Journal of Chemical Theory and Computation 11:609-622

Rohl, C. A., Strauss, C. EM., Misura, K. MS., Baker, D. (2004) Protein Structure Prediction Using Rosetta, Methods in enzymology, 383:66-93

Saito, H., Ando, I., Ramamoorthy, A. (2010) Chemical Shift Tensor – the Heart of NMR: Insights into Biological Aspects of Proteins, Prog. Nucl Magn Reson Spectrosc. 57:181-228

Schwieters, C.D., Kuszewski, J.J., Tjandra, N., Clore, G.M. (2003) The Xplor-NIH NMR Molecular Structure Determination Package, JMR 160:65-73

Thiriot, D.S., Nevzorov, A.A., Opella, S.J. (2004) Structural Basis of the Temperature Transition of Pf1 Bacteriophage, Protein Science 14:1064-1070

Tian, Y., Schwieters, C.D., Opella, S.J, Marassi, F.M. (2014) A Practical Implicit Solvent Potential for NMR Structure Calculation, J. Magn. Reson. 243:54-64

Tian, Y., Schwieters, C.D., Opella, S.J, Marassi, F.M. (2017) High Quality NMR Structures: A New Force Field with Implicit Water and Membrane Solvation for Xplor-NIH, J. Biomol. NMR 67:35–49

Wu, C.H., Ramamoorthy, A., Opella, S.J. (1994) High-Resolution Heteronuclear Dipolar Solid-State NMR Spectroscopy, JMR 109:270-272

Yang, J., Kulkarni, K., Manolaridis, I., Zhang, Z., Dodd, R.B., Mas-Droux, C., Barford, D. (2011) Mechanism of Isoprenylcysteine Carboxyl Methylation from the Crystal Structure of the Integral Membrane Methyltransferase ICMT, Molecular Cell 44:997-1004

Yin, Y., Nevzorov, A.A. (2011) Structure Determination in “Shiftless” Solid State NMR of Oriented Protein Samples, JMR 212:64-73

CHAPTER 4: DE NOVO NMR PULSE SEQUENCE DESIGN USING MONTE-CARLO OPTIMIZATION TECHNIQUES

(Chapter based on publication: Lapin, J., Nevzorov, A.A. (2020) J. Magn. Reson., 310)

4.1 Introduction

Orientationally dependent heteronuclear dipolar couplings (DC) provide direct experimental input for structural elucidation in solid-state NMR of membrane proteins. Along the protein backbone the possible DCs between NMR active nuclei include ^{1}H-^{15}N, $^{13}C_{\alpha}$-$^{1}H_{\alpha}$, and ^{13}C-^{15}N DCs. The value of a DC is determined by the angle that the internuclear bond of interest makes with the external magnetic field, B_0, which provides angular restraints for structure calculations. Efficient pulse sequences correlating DCs between three or more spins are necessary for structure determination of membrane proteins in their native-like lipid environments. High-resolution Separated Local Field (SLF) pulse sequences such as PISEMA, SAMPI4, HIMSELF, and the more recent 2_n-SEMA (Wu et al., 1994; Dvinskikh et al., 2005; Dvinskikh et al., 2006; Nevzorov et al., 2003; Sinha et al., 2007; Nevzorov et al. 2007; Gopinath et al. 2009; Jayanthi et al., 2010; Jayanthi et al., 2010), yield sufficiently sharp linewidths over a broad range of dipolar interactions, and have been shown to be robust over a wide range of proton carrier frequencies. For dilute low spins (^{15}N) surrounded by a "spin bath" of protons (^{1}H), the main goal in developing such sequences is to selectively evolve the heteronuclear DCs, while simultaneously decoupling the ^{1}H-^{1}H homonuclear DCs. In order to accomplish the evolution of heteronuclear DCs, as in PISEMA (Wu et al., 1994), the low-spin channel (S/^{15}N channel) has one phase alteration, i.e. a 180˚ x-pulse followed by a -180˚ x-pulse. On the high-spin channel (I/^{1}H channel), PISEMA uses off-resonance Lee-Goldburg irradiation (Lee et al., 1965; Bielecki et al., 1990) via tilting (spin-locking) the ^{1}H spins at the magic angle, $\theta = 54.7°$. The subsequently developed Sandwich-Assisted Magnetic MicroscopY pulse sequence (SAMMY) (Nevzorov et al., 2003) keeps the protons spin-locked either along the x-axis or along the z-axis with on-resonance pulses, which makes this sequence less sensitive to proton carrier offsets than PISEMA. This is accomplished by using a $(\pi/2)_y$ pulse, followed by a $-(\pi/2)_y$ pulse, between which the power is turned off on the ^{1}H channel. The SAMPI4 pulse sequence further improves on SAMMY by decreasing the number of phase transients on the low spin channel, and subtracting $\pi/4$ pulse duration from each subdwell to compensate for the

finite π/2 pulses (Nevzorov et al. 2007). For both PISEMA and SAMPI4 the dwell time and the scaling factors can be derived theoretically. These pulse sequences have been shown to dramatically decrease dipolar linewidths (by approximately an order of magnitude) thus increasing sensitivity in 2D heteronuclear NMR experiments of uniaxially aligned samples.

All previous SLF pulse sequences only utilize quadrature phases. That is, the aforementioned pulse sequences, PISEMA and SAMPI4 utilize only phases of 0˚, 90˚, 180˚, or 270˚. Other heteronuclear DC experiments involve more pulses in their architecture, such as BLEW-12 (Burum et al., 1981) which consists of twelve back-to-back pulses, and the related HIMSELF sequence (Dvinskikh et al., 2006). Still, those sequences are restricted to quadrature phases, which may limit their full potential. Average-Hamiltonian theory has been recently utilized (Cui et al., 2018) to optimize the pulse phases and rotation angles to find the radiofrequency (rf) pulse cycles which decouple the homonuclear dipolar couplings while evolving the chemical shift and heteronuclear dipolar interactions. However, analytical treatment of the Hamiltonian transformations involving multiple non-quadrature phases and arbitrary rotation angles can become prohibitively complex. In this case, average-Hamiltonian treatment to find the optimal pulse sequence parameters becomes a rather complicated task. To overcome the challenges of a rigorous analytical treatment, numerical spin simulations can easily and accurately predict spin evolution under any pulse sequence architecture and associated pulse parameters, thus permitting exploration of new NMR experiments involving non-quadrature phases and arbitrary rotation angles for spin magnetization. For instance, the decoupling pulse sequence DUMBO (Sakellariou et al., 2000) represents a notable example of computer-aided pulse sequence design.

Herein we describe a protocol to find *de novo* pulse sequences using numerical optimizations performed on a relatively large 8-spin system, derived from the coordinates of a polyalanine amino acid sequence. The simulation method consists of the construction of the Hamiltonian, propagation of the density matrix, and scoring of the resulting Fourier transformed spectra. The optimizations are carried out by the Monte-Carlo Simulated Annealing (MCSA) protocol, which allows for sampling over a large sequence parameter space that includes pulse phases and timings. The automated algorithm is instructed to find a pulse sequence that maximizes peak sharpness, while correctly rendering a pre-defined set of three (3) dipolar splittings. This constitutes a *de novo* approach for pulse sequence design from first principles. The top scoring pulse sequences were run on an N-acetyl Leucine (NAL) crystal in a 300 MHz NMR spectrometer,

and the spectral linewidths were compared to that for the SAMPI4 pulse sequence. The best performing pulse sequence was then re-optimized for a sample of ^{15}N Leucine labeled Pf1 coat protein reconstituted in magnetically aligned bicelles at 500 MHz ^{1}H NMR frequency. This method of numerical optimization is greatly versatile and potentially applicable to any NMR experiment that can be simulated with reasonable accuracy and speed.

4.2 Analytical Framework

In general, the outcome of any pulse sequence can be simulated using the Liouville-von Neumann equation describing the evolution of the density matrix ρ for a spin system evolving under the Hamiltonian, H_n. One can write for each subdwell, n:

$$\frac{d\rho}{dt} = -\frac{i}{\hbar}[H_n, \rho] \qquad \text{(Eq. 4.1)}$$

The density matrix $\rho(t)$ can be successively propagated after each time subdwell Δt_n using the unitary transformations of the initial density matrix.

$$\rho(t_n) = e^{-iH_n\Delta t_n}\rho(t_{n-1})e^{iH_n\Delta t_n} \qquad (Eq.\,4.2)$$

The real part of the time-domain signal can be calculated from the density matrix as:

$$S(t) = Trace\big(S_x\rho(t)\big) \qquad (Eq.\,4.3)$$

where S_x is the detector matrix for transverse magnetization of the low (S)-spins. The Hamiltonian H for a system of low (S/^{15}N) and high (I/^{1}H) spins along the backbone of a protein can be separated into single-spin terms involving the Zeeman interactions of the spins with the main B_0 and radiofrequency B_1 fields and the two-spin dipolar terms. The single-spin rf Hamiltonian terms for an arbitrary phase of each pulse can be written as:

$$H_{rf}^{S} = \omega_i^S[\cos\big(\phi_i^S\big)\,S_x + \sin\big(\phi_i^S\big)\,S_y] \qquad (Eq.\,4.4a)$$

$$H_{rf}^{I} = \omega_i^I[\cos(\phi_i^I)\,I_x + \sin(\phi_i^I)\,I_y] \qquad (Eq.\,4.4b)$$

$$H_{rf} = \sum_{spins} \left(H_{rf}^{S} + H_{rf}^{I}\right) \qquad (Eq. 4.5)$$

Let us consider for simplicity a system of one low spin and (N - 1) high spins. The two-body terms for the heteronuclear dipolar interactions are given by:

$$H^{(IS)} = \sum_{i=2}^{N} a_{1i} S_Z I_Z^{(i)} \qquad (Eq. 4.6)$$

and the homonuclear dipolar interactions among the high spins are:

$$H_{zz} = \sum_{i<j}^{N} \frac{a_{ij}}{2}\left[3I_z^{(i)}I_z^{(j)} - \left(I^{(i)}I^{(j)}\right)\right] \qquad (Eq. 4.7)$$

Here the dipolar coupling constants between spin pairs can be written as (in the units of rad×s^{-1}):

$$a_{ij} = \frac{\gamma_i \gamma_j \hbar (3\cos^2\theta - 1)}{r_{ij}^3} \qquad (Eq. 4.8)$$

where θ is the angle between the interspin vector and z-axis.

In most general terms, the goal of any rational SLF pulse sequence design is to find a periodic train of pulses with specific phases and timings that would ensure the evolution of the heteronuclear terms yielding accurate dipolar doublet splittings, a_{1i} (cf. Eq. 4.6), while concomitantly canceling the homonuclear terms (Eq. 4.7). Theoretical design of such pulse sequence usually starts with considering average Hamiltonians (Haeberlen et al., 1968; Leskes et al., 2010; Brinkmann et al., 2016) up to the second-order approximation, followed by practical considerations including the probe efficiency, spectrometer dead time, magnetic field homogeneity, and other experimental limitations. Additional considerations may include proton frequency offsets, such as in the case of the PISEMA pulse sequence, which would result in spin-locking the protons away from magic angle, or phase transients, which generally preclude the use of too many subdwells due to accumulation of error over longer t_1 dwells.

Numerical spin simulations are an invaluable tool for the prediction and rationalization of experimental results, owing to the well-developed mathematical formalism which accurately simulates many-body spin systems. In addition to the available software packages for carrying out spin simulations (Bak et al., 2000; Veshtort et al., 2006; Hogben et al., 2011), Eq. (4.2) can be relatively easily custom-coded in a high-level programming language in order to simulate a particular NMR experiment and optimize its parameters. However, the density matrix representation of spin systems exponentially increases with the system size, with the dimensions of the density matrix and the Hamiltonians scaling as 2^N x 2^N with the number of spins N. This restricts the practical spin system size that can be used in the calculations due to both memory and time constraints. Thus, in the present work, we have restricted ourselves to 8 spins total, with the proton coordinates constrained within a 3 Å sphere centered at a ^{15}N atom in a polyalanine molecule. A dual-GPU (NVidia 1080Ti) processor has been employed to perform the most computationally intensive matrix operations, which reduces the time required to simulate a single experiment by more than an order of magnitude as compared to a CPU calculation.

The ability to rapidly simulate individual NMR experiments permits an exhaustive search over a large parameter space. First, a desired outcome function must be defined, so that the resulting spectrum can be objectively scored (as defined in the subsequent section). The search then seeks to optimize the parameters for that outcome. Desirable characteristics for an experiment probing ^{1}H-^{15}N dipolar couplings include peak height and/or peak sharpness, i.e. the full width at half height (FWHH), and the appearance of the dipolar doublets at the locations either equal or proportional to the "true" couplings between the single ^{15}N spin and the directly bonded ^{1}H. The latter quantity is closely related to the so-called scaling factor (*scf*) arising in complex pulse sequences, and its robustness is critical for the accurate determination of the angular restraints in the SLF experiments. The major parameters defining an NMR pulse are the pulse phases, timings, and the rf power amplitudes. Optimization commences by selecting the initial sequence parameters for each subdwell, simulating the experiment, measuring the peak heights and locations for three different dipolar couplings, changing the parameters and repeating the process until the best pulse sequence is found. The optimization is carried out by the MCSA protocol (Metropolis et al., 1949). MCSA is a broadly applicable optimization technique that can efficiently search over numerous parameters that may vary in the extent of their effect on the outcome. To avoid spurious solutions for a narrow range of DCs, additional steps need to be taken to ensure the robustness of the pulse

sequence. This can include simulating dipolar spectra with different maximum DC values at each optimization step and verifying their relative positions along the frequency axis. Such an approach would eliminate pulse sequences that would be optimized to evolve dipolar couplings within a narrow frequency range. Another strategy to ensure robustness is to include random chemical shift perturbations (σ) for each of the protons (and also for the ^{15}N spins). This is added to the Hamiltonian as a sum over the individual chemical shift terms, viz.:

$$H_\sigma = \omega_0 \sum_i^N \sigma_i I_z \qquad (Eq. 4.9)$$

where ω_0 is the carrier frequency in MHz, and σ_i are the individual chemical shift perturbations. It is important to emphasize that including different chemical shifts and several dipolar couplings in the simulation is critical in order to ensure robustness of the final sequence.

Most of the previous pulse sequences were designed with symmetry considerations to resemble the canonical transformation for a matrix *A, i.e.* $A' = TAT^{-1}$. That is, each pulse that forms a subdwell would include a mirrored, counterpart pulse that would differ by a 180˚ phase. The same phase mirroring strategy is applied here for even vs. odd total t_1 dwells. For example, the odd dwell on the low S-spin channel in SAMPI4 starts with a 0˚ pulse, and ends with a 180˚ pulse, while the even dwell starts with a 180˚ pulse and ends with a 0˚ phase. Symmetry is reflected in the timings as well, with the first subdwell having the same timing as the last subdwell, the second subdwell the same as second to last, etc. In the current simulations we have imposed either full symmetry (FS), as described for SAMPI4 above, or partial symmetry (PS) in which the first and the last subdwell pulse phases would still differ by 180˚, but the even and odd dwells were essentially treated independently from one another. In general, the simulation can let all parameters float freely within a certain range, taking no symmetry into consideration. The more symmetry that is invoked, however, the fewer free parameters there are in the pulse sequence, thus making it faster for the MCSA protocol to find a reasonably good scoring solution. The phase space can be searched with a mixture of symmetry, by forcing some channels/subdwells to be mirror images of each other, while letting others assume phase values independently.

4.3 Methods

4.3.1. Software and Hardware

All calculations were carried out on an Alienware Aurora RB desktop computer with an Intel® Core™ I7-9700K processor operating at 3.60 GHz. The computer was custom equipped with two NVIDIA GeForce GTX 1080 Ti GPUs, on which the spectral simulations were carried out. It should be noted that without GPU computing, the simulations would have been prohibitively slow and severely limited in the sampling capability. For an 8-spin system in which the matrix dimensions were 256 x 256, sampling of a single SLF experiment was ~20 times faster on GPU than on the CPU (the relative speedup over CPU is expected to increase even more with the matrix size).

The script responsible for administering the MCSA loop and spectral simulation will be provided upon request to the authors. The code was specifically optimized for calculations on a PC equipped with dual GPU processors, which are concurrently used in the simulations when assembling the even and odd dwell Hamiltonians, as well as when propagating the density matrix. The simulation was GPU optimized with help from Python's Cupy library, which provides the functionality of Python's popular NumPy library, but with functions that are GPU compatible. All Python scripts were executed using the open-source data science platform Anaconda©.

4.3.2. Simulation details

As stated in the Introduction, the DC constants have been calculated using the coordinates of a polyalanine alpha-helix. Distant ^{1}H spins have been truncated beyond a 3 Å radius of the backbone amide Nitrogen, resulting in a system of 8 spins (with matrix dimension size 256 x 256). A series of arbitrary chemical shifts were included in the H_σ term of the Hamiltonian to account for the possible proton carrier offsets. The carrier frequency ω_0 was set to 500 MHz for protons and 50.7 MHz for the nitrogen spins.

The program was optimized at a ω_{rf} field amplitude of 58.14 kHz, in order to match the experimental value calibrated for the NAL single crystal sample. The rf amplitudes for each subdwell were modulated by the binary W_i variable, which in an experiment would correspond to irradiation being either power off, 0, or power on, 1. The full Hamiltonian for each subdwell is the sum of Eqs. (4.5-4.7) and (4.9).

$$H_{tot} = H_{\sigma} + H_{rf} + H^{(IS)} + H_{zz} \qquad (Eq. 4.10)$$

After being assembled via Eq. 4.10, the Hamiltonian, H_{tot}, is diagonalized to yield the eigenvalue matrix, D, and the eigenvector matrix, V. This decomposition is used to create the propagator matrix, P_n for each subdwell:

$$P_n = V_n \, e^{-iD_n \Delta t_n} \, V_n^{\dagger} \qquad (Eq. 4.11)$$

and then the overall propagator is assembled as a matrix product over all subdwells:

$$P = \prod_n P_n \qquad (Eq. 4.12)$$

Note that the even and odd dwells can have their own propagators. The manipulations with the Hamiltonian were GPU optimized due to the numerous linear algebra operations, such as diagonalization and matrix-matrix multiplication, necessary to implement Eqs. (4.11) and (4.12). The simulation then proceeds to the propagation of the FID (also GPU optimized), where the FID is calculated via Eqs. (4.1-4.3). Prior to the Fourier transformation, the FID vector, G, is multiplied elementwise by an exponential apodization function:

$$G'(t) = G(t) \exp(-t \, \pi \, d\nu) \qquad (Eq. 4.13)$$

The linebroadening parameter $d\nu$ was set to 50 Hz for all simulations. The Fourier transform of G' is matched to a frequency range f, based on the total dwell time, which is treated as the sum of all individual subdwell durations:

$$dwell = \sum_n t_n \qquad (Eq. 4.14)$$

In each simulation, 512 t_1 dwells have been propagated and the calculated FID was zero-filled to 2048 points to provide sufficient digitization of the Fourier-transformed spectra. The frequency range is given by the corresponding re-scaled Nyquist frequencies (in Hz) as:

$$f_{Nyquist} = \pm \frac{1}{2 * dwell * scf} \qquad (Eq. 4.15)$$

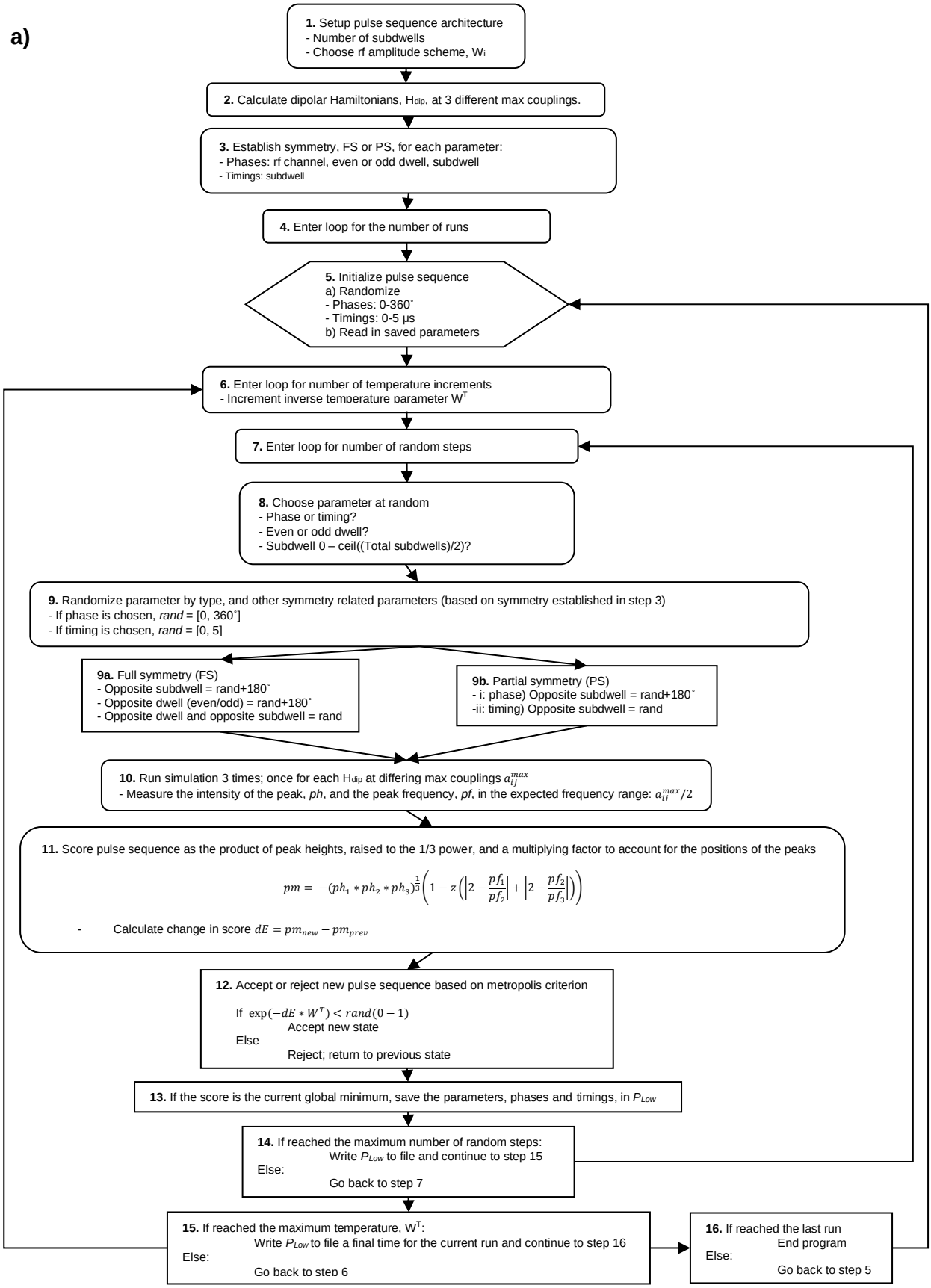
where *scf* is the apparent scaling factor for the pulse sequence that needs to be calibrated by comparing the apparent DCs to the known SLF sequences (see simulation results). The magnitudes of the apparent DCs are then evaluated as to whether they would yield the ratios of their relative positions corresponding to the pre-defined couplings (set to 20, 10, and 5 kHz or 4:2:1 in the present work). The relative intensity of each peak (which is inversely related to its linewidth) is also evaluated and included in the overall scoring function as described in the next section.

4.3.3. The MCSA Algorithm

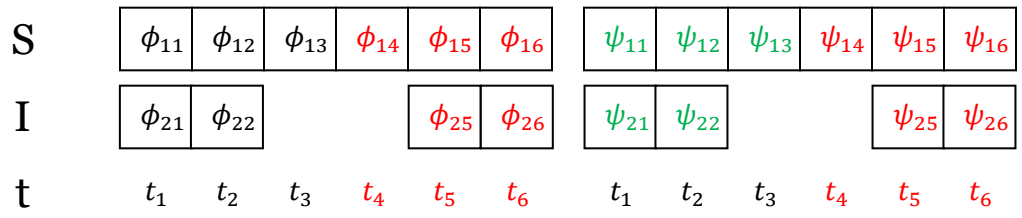
The MCSA protocol uses the dipolar spectrum simulation described in the previous section as the fundamental block for sampling the parameter space. More specifically, the peak amplitudes in the calculated spectrum and their locations are used to score a pulse sequence. The parameter search is then conducted using the Metropolis criterion in order to find a global minimum for the scoring function as defined below, always accepting a smaller proportion of higher energy moves as temperature decreases. The flowchart of Fig. 4.1 shows the logic flow for the algorithm.

Figure 4.1. a) Flowchart for MCSA algorithm applied to pulse sequence optimization. b) Block diagram for a pulse sequence architecture showing the individual subdwell parameters. Each row belongs to a specific spin channel, S (^{15}N) or I (^{1}H), and across the channel are the individual subdwells. Parameters ϕ,ψ are the phases for the odd and dwells, respectively. For FS symmetry, only parameters denoted by the black text are varied throughout the MCSA run; for PS symmetry black and green text are variable; the red text parameters are always fixed by minimal symmetry considerations. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

a)


1. Setup pulse sequence architecture
- Number of subdwells
- Choose rf amplitude scheme, Wi
2. Calculate dipolar Hamiltonians, Hdip, at 3 different max couplings.
3. Establish symmetry, FS or PS, for each parameter:
- Phases: rf channel, even or odd dwell, subdwell
- Timings: subdwell
4. Enter loop for the number of runs
5. Initialize pulse sequence
a) Randomize
- Phases: 0-360°
- Timings: 0-5 μs
b) Read in saved parameters
6. Enter loop for number of temperature increments
- Increment inverse temperature parameter W^T
7. Enter loop for number of random steps
8. Choose parameter at random
- Phase or timing?
- Even or odd dwell?
- Subdwell 0 – ceil((Total subdwells)/2)?
9. Randomize parameter by type, and other symmetry related parameters (based on symmetry established in step 3)
- If phase is chosen, rand = [0, 360°]
- If timing is chosen, rand = [0, 5]
9a. Full symmetry (FS)
- Opposite subdwell = rand+180°
- Opposite dwell (even/odd) = rand+180°
- Opposite dwell and opposite subdwell = rand
9b. Partial symmetry (PS)
- i: phase) Opposite subdwell = rand+180°
-ii: timing) Opposite subdwell = rand
10. Run simulation 3 times; once for each Hdip at differing max couplings a_{ij}^{max}
- Measure the intensity of the peak, ph, and the peak frequency, pf, in the expected frequency range: $a_{ii}^{max}/2$
11. Score pulse sequence as the product of peak heights, raised to the 1/3 power, and a multiplying factor to account for the positions of the peaks
$$pm = -(ph_1 * ph_2 * ph_3)^{\frac{1}{3}}\left(1 - z\left(\left|2 - \frac{pf_1}{pf_2}\right| + \left|2 - \frac{pf_2}{pf_3}\right|\right)\right)$$
- Calculate change in score $dE = pm_{new} - pm_{prev}$
12. Accept or reject new pulse sequence based on metropolis criterion
If $\exp(-dE * W^T) < rand(0-1)$
Accept new state
Else
Reject; return to previous state
13. If the score is the current global minimum, save the parameters, phases and timings, in P_{Low}
14. If reached the maximum number of random steps:
Write P_{Low} to file and continue to step 15
Else:
Go back to step 7
15. If reached the maximum temperature, W^T:
Write P_{Low} to file a final time for the current run and continue to step 16
Else:
Go back to step 6
16. If reached the last run
End program
Else:
Go back to step 5

b)


S
I
t
ϕ_{11} ϕ_{12} ϕ_{13} ϕ_{14} ϕ_{15} ϕ_{16}
ψ_{11} ψ_{12} ψ_{13} ψ_{14} ψ_{15} ψ_{16}
ϕ_{21} ϕ_{22} ϕ_{25} ϕ_{26}
ψ_{21} ψ_{22} ψ_{25} ψ_{26}
t_1 t_2 t_3 t_4 t_5 t_6
t_1 t_2 t_3 t_4 t_5 t_6

In the present work, the program is initiated by setting manually the general pulse architecture and amplitude scheme (e.g. starting from the original SAMPI4 ordering of pulses), which in turn determines the variable space used in the simulation (i.e. the number of phases and timings). For PS, the total number of parameters varied during the MCSA run would then be one-half the number of subdwells times 5; 80% of the parameters belong to the phases and the rest to timings (for FS there are half as many varied phase parameters). For phases there are two channels, one for the high spins I and one for the low spins S. The two channels have both even and odd dwells. The symmetry is defined prior to starting the MCSA optimization process, which is represented in step 3 of Fig. 4.1a. While the phase parameters can in general have either FS (step 9a) or PS (step 9b), timings can only be PS. In principle, the rf frequency amplitudes for each channel and subdwell could also be considered parameters. Due to the inherent rise/fall times of the NMR amplifiers and their non-linearity, however, continuously varying the rf power between the subdwells would be impractical. Therefore, the values for the rf amplitudes were chosen to be either 0 or ω_{rf}. In addition, continuously varying the rf amplitudes would often result in a search that would never converge to a global minimum. Therefore, for the majority of the simulations we used the fixed power scheme such as used by SAMPI4, in which the power is turned off during the third and fourth subdwells on the ^{1}H-channel only.

The calculation of the individual terms in the Hamiltonian is a rather time consuming operation, but it only needs to be completed once (step 2), allowing the simulation to change the parameters and continue using largely the same Hamiltonian matrices. The only Hamiltonian term that changes with the pulse sequence is H_{rf}; however, the matrices I_x, S_x only need to be calculated once, initially, and then H_{rf} is simply scaled per Eqs. 4.4a,b. When one considers only a single dipolar doublet between the ^{15}N and its nearest ^{1}H, optimizing the pulse sequence carries a risk of finding a singularity for a specific dipolar doublet. This problem is assuaged here by running the simulation consecutively for three different DCs, thus ensuring that all resonances are properly evolved and decoupled. To calculate an intermediate score in the MCSA protocol, the simulation is, therefore, run three times for each random step. The three different precalculated $H^{(IS)}$ matrices are used in step 10, where the NMR experiment is simulated.

In this work the three maximum couplings were arbitrarily set to 5, 10, and 20 kHz. Since the generated pulse sequences may all have different scaling factors, the apparent DC frequencies could deviate from their true values. Thus, instead of their exact dipolar frequencies, only the ratios

of the DC frequencies for the three peaks are factored into the score. The intensities from the three simulations (ph_i) are multiplied together and raised to the one-third power, which is considered as the raw score, termed *pmraw*. Using the product instead of the mean ensures that no single peak is optimized at the exclusion of the others. The final scoring function, termed *pm*, then becomes:

$$pm = -(ph_1\ ph_2\ ph_3)^{\frac{1}{3}}\left(1 - z\left(\left|2 - \frac{pf_1}{pf_2}\right| + \left|2 - \frac{pf_2}{pf_3}\right|\right)\right) \qquad (Eq.\ 4.16)$$

The ratios of the peak frequencies are accounted for by the pf_i terms. Parameter z (0.5 in this work) is used to increase or decrease the penalty for violating the expected peak ratios. The score in Eq. 4.16 will be lowest (most negative) when the peak intensities are collectively high and the both ratios of the successive dipolar frequencies are as close to 2.0 as possible.

The starting parameters for an MCSA run must be initialized, either at random or from a fixed set of parameters that was previously saved to a file. If the initialization of the parameters is done randomly, the starting values are chosen within the same range where parameters would be sampled. For phases this range is [0,360˚], which represents all possible angles that a pulse phase can assume. For timings this range was chosen to be 0-5, in the multiples of 90-degree pulse durations, i.e. $t90 = \frac{\pi}{2\omega_{rf}}$ (in microseconds). This range was chosen based on the previous SAMMY and SAMPI4 subdwells (Nevzorov and Opella, 2007).

All MCSA protocols have two main loops: one for the “temperature” decrements and the other for random steps taken at each temperature. The temperature loop sets a unitless “inverse temperature” parameter, W^T, to be used in the Metropolis criterion when accepting or rejecting new steps. Since W^T is proportional to inverse temperature, it is increasing throughout the simulation, corresponding to cooling of the system. In this work, given the relative magnitudes of the scoring function (typically ranging from 0 to -30) it was sufficient to increment W^T from 1-8. The strategy for finding the best pulse sequences often stressed running numerous shorter, rather than fewer longer, si0mulations. Simulations were initially run with 100 temperature increments and 200 random steps per temperature, for a total of 20,000 random steps. Further refinement, when desired, was then run starting from each MCSA run’s saved parameters for 10,000 additional steps.

Random steps are taken by choosing a particular parameter and then randomizing its value from a uniform distribution of its designated range. Two random integers are generated to index a specific parameter. The first random integer, *ind1*, is from 0 through 4. This integer represented one of the possible five (5) choices; 0: low-spin S-channel, odd dwell; 1: high-spin I-channel, odd dwell; 2: low-spin S-channel, even dwell; 3: high-spin I-channel, even dwell; 4: timings. The second random integer, *ind2*, chooses the subdwell to be sampled. Due to the symmetry restrictions across subdwells, the maximum value for *ind2* only needs to be half the number of subdwells (rounded up if there is an odd number of subdwells). At a minimum each parameter's symmetry was designated as PS. This means that setting the value for a subdwell in the first half of the dwell would also set the value for the mirrored subdwell in the second half of the dwell. Running all parameters with PS cuts down the true number of variables from 30 to 15, or 5 times the maximum value for *ind2*. For FS, the true number of variables becomes 9 since there is the same number of variables for timings (3), but half as many for phases (6). The variables for phases are further decreased because FS relates the subdwell variables between the even and odd dwells for each spin channel.

Once a parameter is specified by *ind1* and *ind2*, it is assigned a new value *rand* (step 9). As previously stated the range of *rand* is specific to its parameter type (phase or timing), and is drawn from a uniform distribution. The symmetry related parameters are automatically varied as well (step 9a,b). For phases, the mirrored subdwell is set to *rand*+180°. If FS, then the opposite dwell for that channel (even or odd) is also set to *rand*+180°, and lastly the parameter that has both opposite dwell and mirrored subdwell is set to *rand*. For timings the mirrored subdwell is set to *rand*.

The simulation is repeated three times for the new set of parameters, each simulation having a different heteronuclear dipolar Hamiltonian $H^{(IS)}$ that corresponds to a specific maximum coupling, and then is assigned a score. All the other steps that follow are standard MCSA procedure. Step 12 applies the Metropolis criterion for *dE*, which is the difference between the new score and the previous score. The scoring parameters are saved to file after every W^T increment completes. After all MCSA runs are completed the program terminates. The parameters can either be used for further refinement or converted into the final pulse sequence format for implementation on an NMR spectrometer.

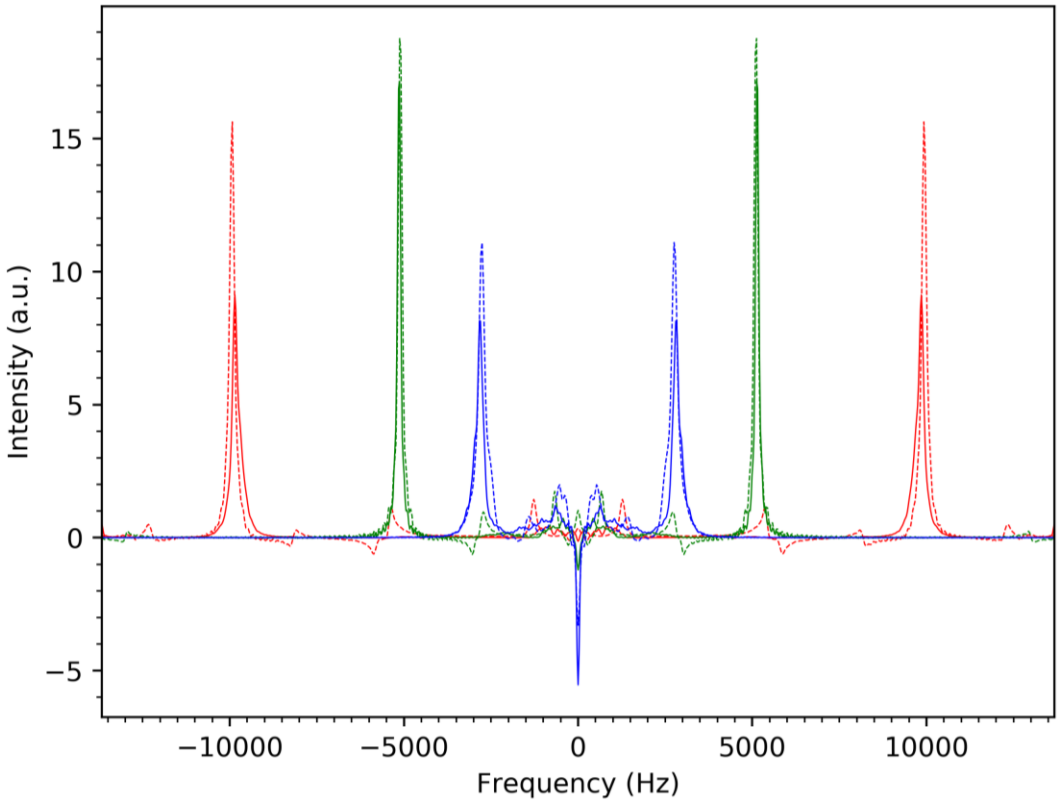
4.3.4. NMR experiments

NMR experiments for the NAL crystal (18 mg) were carried out on a Bruker 300 MHz spectrometer operated by Topsin 3.2 software and using a static Doty Scientific™ 7 mm coil HX probe. The B_1 field was calibrated at 58.14 kHz and 2 scans were accumulated for each of 200 t_1 points, each having the architecture shown in Fig. 4.1b with the individual subdwell phases and timings calculated by the optimization, cf. Table 4.1 below. Experiments for the selectively Leu-labeled Pf1 coat protein in bicelles (ca. 4 mg) were carried out on a Bruker 500 MHz spectrometer operated by Bruker 2.0 software and using a static 5 mm coil HCN Bruker E-free™ probe. The B_1 field was calibrated at 49.5 kHz and 256 scans were co-added. Spectra were processed using NMRPipe (Delaglio et al., 1995). Linewidth calculations were performed using the signal function in Python's Scipy library. The acquisition parameters were held constant between all experiments runs.

4.4 Results

4.4.1. MCSA simulations

MCSA trials were often run in succession, with the final scores typically ranging from -10 to -30. The resulting family of computationally optimized pulse sequences has been termed "ROULETTE" (Random Optimization Using the Liouville Equation Tailored To Experiment). Compared to SAMPI4, whose simulated *pmraw* score was -10.59, the top experimentally performing ROULETTE pulse sequence, termed ROULETTE-1, had a *pmraw* score of -14.43. A direct comparison between the simulations of this pulse sequence and SAMPI4 is provided in Fig. 4.2.



"True" frequency (Hz)	2500	5000	10000
SAMPI4	2830	5127	9853
ROULETTE-1	2755	5122	9926

Figure 4.2 MCSA-optimized simulation for the ROULETTE-1 pulse sequence (dashed lines), vs. the simulation for the SAMPI4 pulse sequence (solid lines) shown across the ^{1}H-^{15}N DC dimension. The sets of three different NH DCs set in each simulation are color-coded, with red corresponding to 20 kHz, green for 10 kHz, and blue for 5 kHz full DC. The raw scores of SAMPI4 vs. ROULETTE-1 are -10.59 and -14.43, respectively. The empirically determined scaling factors for the SAMPI4 and ROULETTE-1 are 0.709 and 0.587, respectively. The accompanying table displays the calculated values for the apparent frequencies for each simulated experiment with the scaling factors taken into account. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

In silica, ROULETTE-1 has a *pmraw* score roughly 33% greater than that of SAMPI4. Since the scaling factors can vary widely between pulse sequences, retaining the correct relative ratios of the DC frequencies is most important.

The scaling factors for the ROULETTE sequences, *scf*, were calculated as the slope in a linear regression between the expected frequencies (from the maximum DCs used in the simulations) and the actual measured (or apparent) frequencies resulting from the simulated spectrum. With the dwell defined per Eq. 4.14, the numerical *scf* for SAMPI4 was 0.709 (which is to be compared with the theoretical result of 8×1.0927/12 = 0.7285 from Nevzorov et al., 2007) and for ROULETTE-1 *scf*=0.587. This ensured that the resonances for a given maximum coupling

would all appear at approximately the same locations across the different pulse sequences, cf. Fig. 4.2. The simulations also predict that, while the two pulse sequences match the expected frequencies for the intermediate couplings (e.g. 10 kHz), the ROULETTE-1 resonances are closer to the "true" couplings for both the higher (20 kHz) and lower (5 kHz) couplings than for SAMPI4.

4.4.2. Experimental results for NAL crystal at 300 MHz ^{1}H frequency.

More than one hundred (100) ROULETTE-family pulse sequences were tested experimentally using a single NAL crystal. The top seventeen pulse sequences that improved on the mean dipolar linewidths, μ's, are shown for comparison with SAMPI4, and each other, in Table 4.1.

Table 4.1: Phases, timings, and mean linewidths for ROULETTE (1) optimized sequences vs. SAMPI4 (S). Table obtained from Lapin and Nevzorov, 2020, JMR, 310.

Exp. #	ϕ_{11}	ψ_{11}	ϕ_{12}	ψ_{12}	ϕ_{13}	ψ_{13}	ϕ_{21}	ψ_{21}	ϕ_{22}	ψ_{22}	t_1 (μs)	t_2 (μs)	t_3 (μs)	μ (Hz)
S	0	180	0	180	0	180	0	180	90	270	15.05	4.30	6.45	213
1	60	255	168	314	118	358	89	139	184	47	15.74	4.58	4.57	178
2	209	64	294	147	126	138	259	323	355	227	15.16	5.05	2.75	168
3	157	164	68	234	10	285	132	215	41	112	14.03	4.98	1.94	173
4	339	120	43	230	47	53	354	61	267	149	7.41	5.56	1.40	183
5	208	70	294	143	190	148	188	259	281	167	14.42	4.94	2.73	187
6	76	292	340	224	332	213	26	35	313	104	7.55	4.77	3.40	188
7	153	336	79	72	295	182	105	15	3	301	13.78	5.43	1.66	191
8	195	88	279	157	188	265	165	233	258	138	15.16	5.17	2.86	193
9	195	70	261	149	91	261	189	259	284	162	14.95	5.38	2.46	195
10	181	1	135	315	163	343	127	307	222	42	16.21	4.89	6.31	197
11	235	36	300	106	17	205	137	64	47	155	15.36	5.11	3.37	199
12	149	303	80	239	342	121	161	83	71	172	15.34	5.16	3.49	199
13	342	115	43	231	56	120	354	63	267	149	7.44	5.56	1.18	202
14	25	221	293	287	119	51	45	129	323	31	13.68	5.39	1.55	204
15	162	149	102	213	319	275	145	230	68	128	14.25	5.72	1.99	205
16	36	243	140	296	81	21	49	118	139	29	15.68	4.94	4.04	212
17	26	146	149	8	332	206	21	225	113	305	16.30	5.14	5.40	212

In Table 4.1, only the phases and timings are shown for subdwells 1-3; the values for subdwells 4-6 are determined by the symmetry relationships as described in the Methods section and Fig. 4.1b.

Furthermore, since the rf amplitude on the ^{1}H-channel is off during subdwells 3 and 4, the corresponding phases need not be specified.

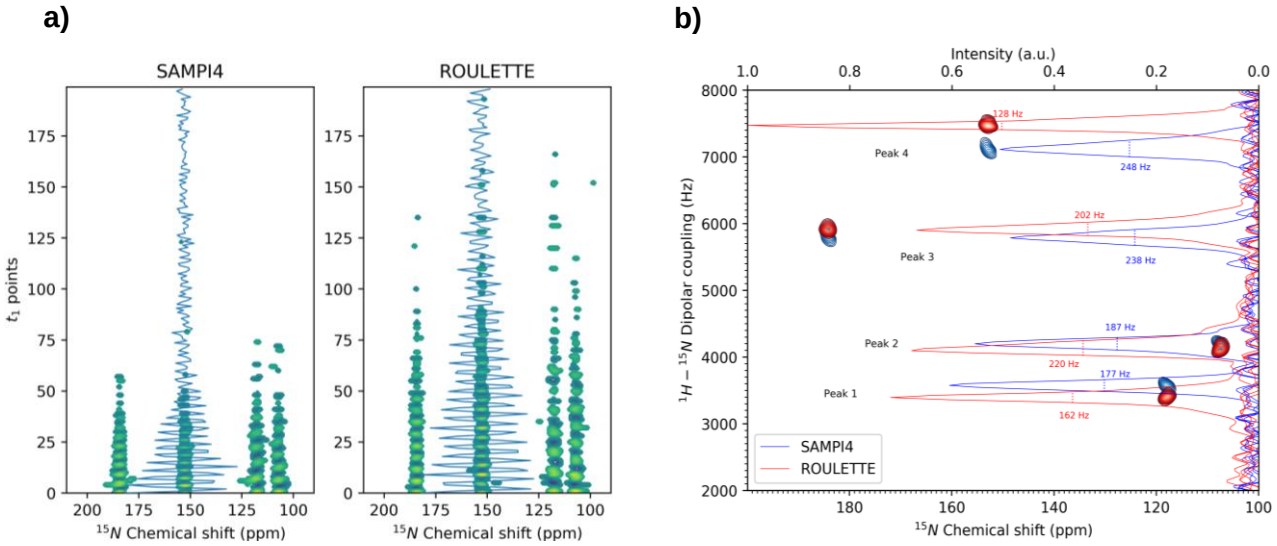


Figure 4.3. a) Side by side comparison of the interferograms for SAMPI4 and ROULETTE-1 experiments measured for the NAL crystal at 300 MHz ^{1}H frequency. The oscillations in the t_1 dimension through the 4th peak are shown in blue b) An overlay of 2D SAMPI4 (blue) and ROULETTE-1 (red) spectra for all 4 NAL peaks. The additional horizontal axis on the top corresponds to the normalized ^{1}H-^{15}N DC spectra for each peak. The corresponding linewidths, calculated as FWHH, are listed beside each peak. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

A side by side comparison of the interferograms and spectra for SAMPI4 and the best performing ROULETTE pulse sequence, i.e. ROULETTE-1, is presented in tabFig. 4.3. The interferogram in Fig. 4.3a shows much more persistent dipolar oscillations for all peaks in ROULETTE-1 as compared to SAMPI4. For instance, for peak 4 the signal has completely dissipated in SAMPI4 after ~75 t_1 points, while the signal still remains prominent even at ~125 t_1 points in ROULETTE-1. In Fig. 4.3b the SLF spectra for the two experiments are shown superimposed, along with respective FWHH linewidth for each peak. As predicted by the simulations presented in Fig. 4.2, ROULETTE-1 yields greater observed dipolar couplings than SAMPI4 at the higher frequencies and slightly lesser couplings at the lower frequencies. The simulations also suggest that ROULETTE-1 measured couplings are closer to the true couplings of the backbone NH bonds. The distributions of linewidths for the individual DC peaks present in NAL are shown for the seventeen best ROULETTE experiments in Fig. 4.4. The linewidth values for SAMPI4 are shown as horizontal lines for comparison with the ROULETTE experiments.

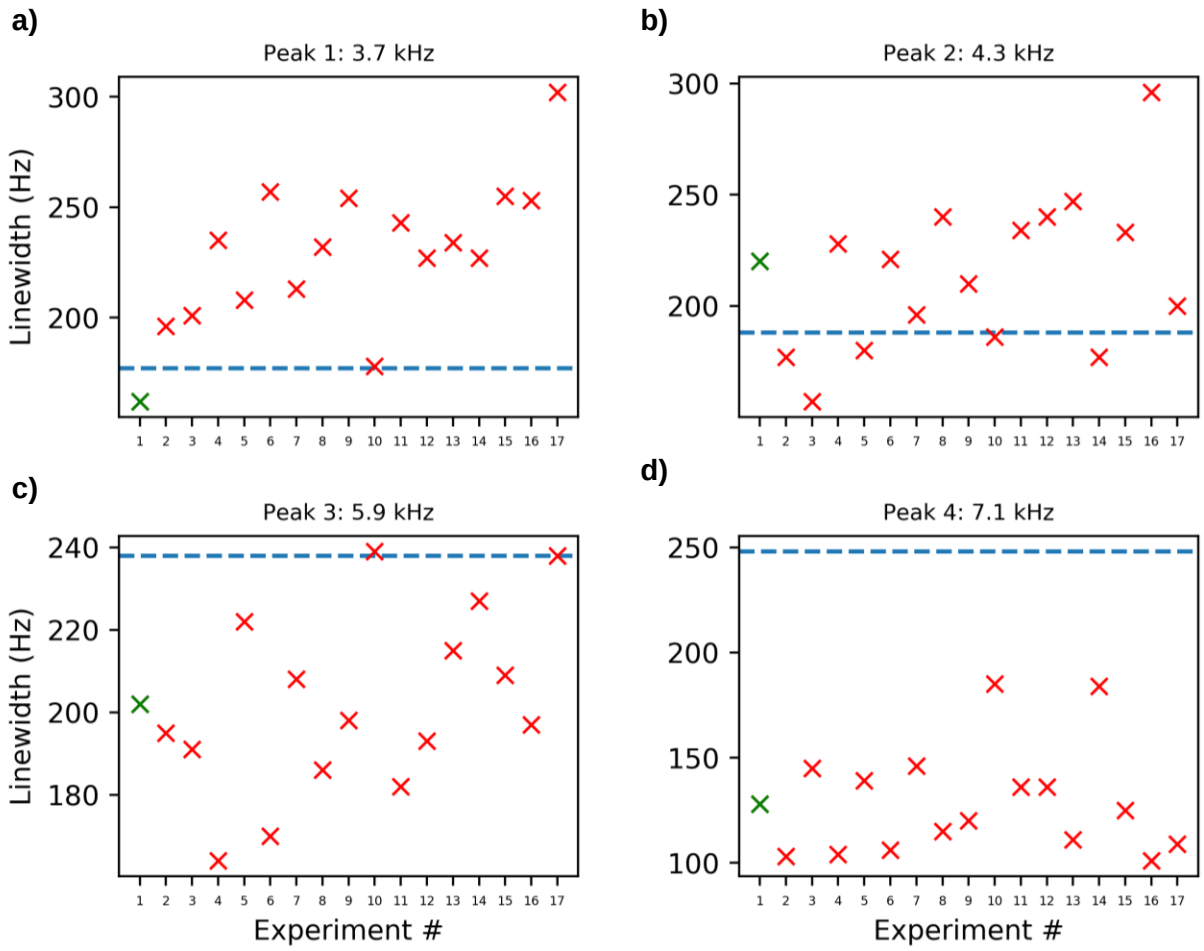


Figure 4.4. Linewidths of the 17 best optimized ROULETTE pulse sequences for NAL crystal as compared to SAMPI4. Experiments are denoted by their relative number shown on the x-axis; ROULETTE-1 is highlighted by the green crosses. The individual plots (a-d) correspond to the peaks 1-4 in NAL. The linewidth values for SAMPI4 are represented by the horizontal dashed lines for comparison with the optimized experiments. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

While all experiments included in the plots had better *average* linewidths than SAMPI4, they do not necessarily improve over SAMPI4 for *every* peak. The optimized pulse sequences perform particularly well at high couplings, cf. Fig. 4.4 (c,d), while SAMPI4 better resolves the lower couplings Fig. 4.4 (a,b). The mean linewidth amongst the 4 peaks for ROULETTE-1 was 178 Hz, which is 18% sharper as compared to 212.75 Hz for SAMPI4, with a marked improvement over SAMPI4 for the largest DCs, cf. Fig. 4.4d.

4.4.3. Robustness of the scaling factor with respect to ^{1}H frequency offsets

To further evaluate robustness of the ROULETTE optimized pulse sequences, a series of experiments were performed at various ^{1}H carrier offset frequencies. A common pitfall for composite-pulse sequences is the potential variability in the scaling factor, which could alter the measured DC values, thus affecting subsequent structure calculations. Previous experiments have shown that the scaling factor can be sensitive to ^{1}H carrier frequency (Nevzorov et al., 2003; Nevzorov et al., 2007; Yamamoto et al., 2005; Gen et al., 2000). Figure 4.5a shows the sensitivity of the measured DCs to the proton carrier frequency for ROULETTE-1.

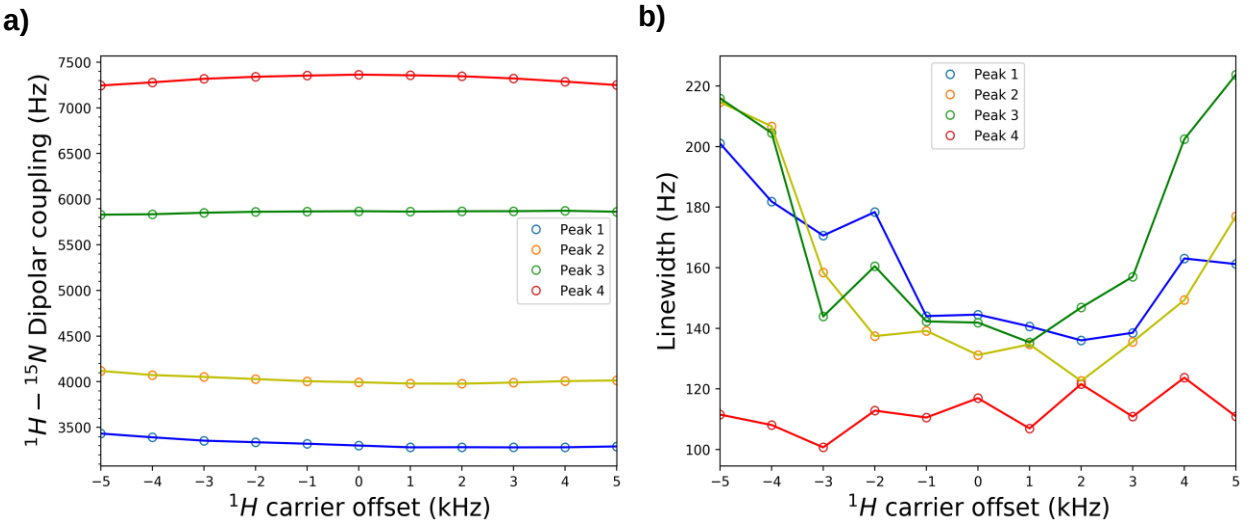


Figure 4.5. Sensitivity of ROULETTE-1's DCs and linewidths to ^{1}H carrier frequency for 4 peaks of NAL crystal. The 0 kHz point corresponds to the optimum carrier frequency of 300.1267 MHz. The other 10 experiments were performed at ^{1}H carrier frequencies above and below the optimum center frequency, in increments of 1 kHz. For peaks 1-4 in a), the ranges from the minimum to the maximum DC are 151, 140, 43, and 119 Hz, respectively (see legend for the color coding of peaks). b). Sensitivity of the linewidths to the offset for the 4 peaks (same color coding for peaks as in a). Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

Relative to the DC range of roughly 3.5 kHz, the deviations in the measured coupling from the optimal carrier ^{1}H frequency of 300.1267 Hz are small, and comparable to the respective variances in SAMPI4 (Nevzorov et al., 2007). Figure 4.5b shows the distribution of the measured linewidths at the same carrier frequencies. Peaks 1-3 tend to have the sharpest linewidths around the center carrier frequency, while the linewidths for peak 4 do not vary appreciably.

To verify that the optimized pulse sequence is still effective at other crystal orientations, the sample was rotated and the ROULETTE-1 sequence was compared to SAMPI4.

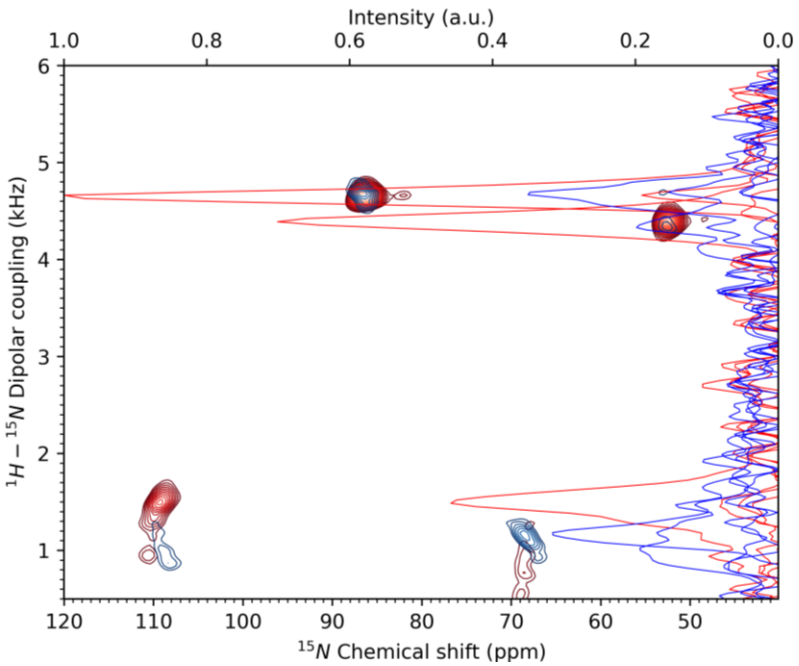


Figure 4.6. ROULETTE-1 (red) vs. SAMPI4 (blue), for an alternate NAL crystal orientation. Compared to the previous orientation, shown in Fig. 4.3b, the DCs for the 4 peaks have much lower frequencies. The lower x-axis corresponds to the chemical shift anisotropy, while the normalized intensities for dipolar slices, taken at the respective chemical shifts of the 4 peaks, are measured on the top x-axis. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

The second crystal orientation, shown in Fig. 4.6, is significantly more challenging than the one pictured in Fig. 4.3. Two of the four couplings have significantly lower DC frequencies than the original orientation, and are generally more difficult to evolve than moderate to high DCs. Nevertheless, it can be seen from Fig. 4.6 that ROULETTE-1 still yields much better signal to noise ratio than SAMPI4 at this crystal orientation, especially for the larger couplings.

4.4.4. Pf1 spectra at 500 MHz ^{1}H frequency

The SAMPI4 and refined ROULETTE-1 pulse sequence were then run on Leucine labeled Pf1 coat protein reconstituted in magnetically aligned bicelles at 500 MHz ^{1}H frequency. To conform with the calibrated pulses at 500 MHz, the sequence was refined starting from the previous phases and timings, but using $\omega_{rf} = 49.5\ kHz$ instead of $58.14\ kHz$. This resulted in a sequence with only small phase differences (within 20˚) relative to the original ROULETTE-1 parameters that were optimized for 300 MHz ^{1}H frequency. The differences can be seen in Table 4.2 where the two pulse sequences are listed by their phases and timings.

Table 4.2: Comparison of ROULETTE-1 pulse sequences, optimized at ω_{rf} = 49.5 kHz vs. 58.14 kHz. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

Exp.	ϕ_{11}	ψ_{11}	ϕ_{12}	ψ_{12}	ϕ_{13}	ψ_{13}	ϕ_{21}	ψ_{21}	ϕ_{22}	ψ_{22}	t_1 (µs)	t_2 (µs)	t_3 (µs)
ω_{rf} = 49.5 kHz	60	250	159	309	136	9	91	146	186	53	17.34	5.20	4.41
ω_{rf} = 58.14 kHz	60	255	168	314	118	358	89	139	184	47	15.74	4.58	4.57

The pulse sequence at 58.14 kHz has the same parameters as the ROULETTE-1 sequence from Table 4.1. The greatest deviation between the two pulse sequences is in the third odd subdwell for the ^{15}N channel (ϕ_{13}), where the subdwell phase at 49.5 kHz is ca. 18˚ more than the corresponding phase at 58.14 kHz. Naturally the timings are longer with a lower rf amplitude, but the ratios between the durations t_1, t_2, and t_3 remain roughly the same, albeit with a greater disparity between t_2 and t_3, as compared to 58.14 kHz. The comparison of SAMPI4 and the ROULETTE-1 sequence refined at ω_{rf}=49.5 kHz at 500 MHz ^{1}H frequency is pictured in Fig. 4.7.

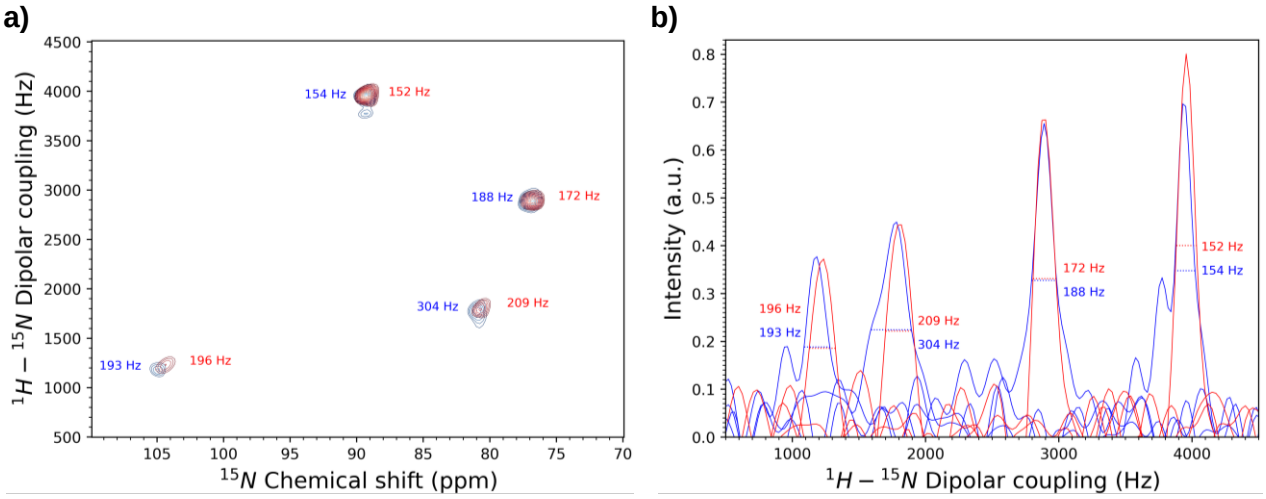


Figure 4.7. Comparison of SAMPI4 (blue) and the re-optimized ROULETTE-1 (red) pulse sequence run on ^{15}N Leu-labeled Pf1 coat protein at 500 MHz ^{1}H frequency. a) The 2D spectra with the ^{1}H-^{15}N DC linewidths labeled next to each peak. b) Dipolar slices taken at the chemical shifts of each of the 4 peaks. Linewidth (FWHH) values are also displayed in Hz beside each peak. The mean peak width is 210 Hz for SAMPI4 and 182 Hz for the optimized pulse sequence, which is 14% sharper. This gain in resolution proportionally increases the signal-to-noise ratio from 17:1 for SAMPI4 to 20:1 for ROULETTE-1. Figure obtained from Lapin and Nevzorov, 2020, JMR, 310.

The improvement in linewidths for ROULETTE-1 over SAMPI4 is slightly less for Pf1 than is observed for the NAL crystal (14% vs. 18%). However, this improvement in linewidths yields a corresponding increase in the signal to noise ratio from 17:1 in SAMPI4 to 20:1 for ROULETTE-1, as measured by NMRPipe. Moreover, the ROULETTE-1 lineshapes appear free of the

additional artifacts such as “shoulders and “humps” that are otherwise present in the SAMPI4 dipolar slices for Pf1, cf. Fig. 4.7.

4.5 Discussion

The present work addresses the following three main aspects: application of MCSA to optimize NMR pulse sequences by computer simulations; experimental validation of the optimized pulse sequences using a static sample consisting of NAL crystal, and the performance of the pulse sequence on a much more dynamic biological system, i.e. Pf1 coat protein reconstituted in magnetically aligned bicelles. It was shown that an MCSA-guided parameter search can optimize a pulse sequence by minimizing an appropriately chosen scoring function (the product of peak intensities and the ratios of the DCs, cf. Eq. 4.16). Rarely did a sufficiently long simulation fail to find a score that was better than that for SAMPI4 (often scores were greater than 1.5x that of SAMPI4). It was also rare for the simulation to converge to the same set of parameters in subsequent runs, which suggests that there are numerous local minima in the energy surface produced by the scoring function, Eq. 4.16. The best practice was to run the optimizations multiple times with different starting parameters so that the optimum parameter set would be largely determined by random chance, hence the pulse sequence name, “ROULETTE”. For this purpose the program was written to run numerous MCSA simulations in succession, cf. step 4 in Fig. 4.1a. In that way multiple solutions would be obtained; the promising solutions could be further refined and the bad ones rejected. It is still important to note that a number of runs produced false positives, i.e. sharp peaks obtained in silica that failed to produce sharper resonances than SAMPI4 experimentally.

Running numerous shorter simulations helped identify the simulation parameters that will more likely yield sharper linewidths experimentally. The majority of the optimizations were run using the same architecture and rf amplitude scheme as SAMPI4. Increasing the number of subdwells not only increases the number of the parameters to be optimized in the simulations, but also provides more flexibility for the algorithm to find solutions with lower scores. Additional simulations contained as many as 15 subdwells, which allowed lower *pm* scores to be obtained (results not shown). Unfortunately, these results were not confirmed experimentally when run on NAL, often producing spectra with no discernable peaks. This is likely the result of the increased number of phase transients in between subdwells. It should be noted that increased number of

subdwells can be of use when designing pulse sequences with continuous phase modulation (Sakellariou et al., 2000). This type of pulse sequence optimization is only effective when the difference in phases between the adjacent subdwells is small. The optimization method described in this paper, by contrast, allows adjacent subdwells to assume any phase, from 0 to 360˚.

About 15% of the tested pulse sequences have succeeded in improving on SAMPI4's mean linewidth and are listed in Table 4.1. Only one of the pulse sequences, ROULETTE-10, was constructed with FS symmetry, while the rest were only PS. A consistent pattern that can seen in the optimized pulse sequences arises from the timings, which tend to be roughly in a 3:1:1 ratio for subdwells 1-3, respectively (with a few exceptions). Amongst all sequences, each phase appears to be able to assume the entire range of values, i.e. 0˚-360˚. Surprisingly, none of the optimizations have lead to the original SAMPI4 values, even when FS was imposed. Even more surprising is that there were no clusters of pulse sequences in which all phases were similar to each other (to within 10˚). Overall, these results suggest that there may exist multiple ways of decoupling the ^{1}H-^{1}H dipolar interactions from ^{1}H-^{15}N interactions, especially when non-quadrature phases and arbitrary subdwell durations are employed.

The departures of the measured frequencies at low and, especially, high dipolar couplings between SAMPI4 and ROULETTE-1 (Fig. 4.3b) is largely in agreement with the simulations in Fig. 4.2. The simulations predict that ROULETTE-1 has greater fidelity in evolving the local NH couplings than SAMPI4, which is likely a consequence of the assumption of infinitely sharp refocusing 90-degree pulses utilized when deriving the scaling factor for SAMPI4 (Nevzorov et al., 2007). In addition, SAMPI4 is known to underestimate the values for larger (>7 kHz) dipolar couplings (Nevzorov et al., 2007), especially at lower B_1 rf fields (<60 kHz). By contrast, the ROULETTE sequences have been optimized without such delta-pulse approximation. This feature of the ROULETTE sequence could potentially result in more reliable measurements of the dipolar couplings for structure calculations with less uncertainty in determining the backbone folds of oriented membrane proteins (Yin and Nevzorov, 2011; Lapin and Nevzorov, 2019).

Additional control experiments, cf. Section 3.4.3 in Results, further demonstrate the robustness of the ROULETTE-1 pulse sequence. A pulse sequence whose scaling factor is highly variable with the ^{1}H carrier frequency requires an extra optimization step in order to find the optimal frequency to achieve ^{1}H-^{1}H decoupling for all heteronuclear dipolar couplings, which may be difficult. It was shown that ROULETTE-1 has a largely consistent scaling factor for DCs in the

relevant ^{1}H-^{15}N frequency range while maintaining sharp dipolar linewidths. The ROULETTE-1 pulse sequence was also tested at an alternate crystal orientation exhibiting much smaller DCs. Overall, the results show that the GPU-optimized ROULETTE-1 can effectively evolve a wide range of DCs at various crystal orientations.

As an application to biological samples, the ROULETTE-1 pulse sequence was tested on ^{15}N Leucine labeled Pf1 coat protein reconstituted in magnetically aligned bicelles. The average improvement in linewidths over SAMPI4 was 14%, which is a slightly lesser improvement as compared to the tests on the NAL crystal. This may be due to the imperfections arising from the protein dynamics and greater B_1 field inhomogeneity over a larger sample volume (180 uL). Another possible explanation for the difference in results between the NAL crystal and Pf1 in bicelles is that ROULETTE-1 has not been experimentally optimized with the Pf1 sample at 500 MHz ^{1}H frequency. Whereas a single SLF experiment on the NAL crystal took only 30 minutes, a 1D experiment on the ^{15}N Leu-labeled Pf1 sample would take 44 hours. This imposes practical limits on the feasibility of screening different optimizations/refinements using biological samples. However, the ROULETTE-1 spectrum of Pf1 coat protein still compares more favorably to SAMPI4 in the sense that in the former the dipolar peaks are more symmetric and are free of "shoulders". Although these features do not appear to appreciably influence the linewidths measured at half height, the extra intensity at the bases of the SAMPI4 peaks may potentially complicate overall spectral resolution and, therefore, the subsequent structural analysis.

4.6 Recent work on pulse sequence development

Continuing work on pulse sequence design has been aimed at (1) increasing agreement between simulation and experiment, i.e. reducing false positives, (2) generalizing the approach by measuring ^{1}H-^{15}N DC for NAL at 500 MHz ^{1}H frequency, and (3) optimizing $C_{\alpha}H_{\alpha}$ dipolar linewidths for NAL. Whereas ^{1}H-^{15}N spectra for NAL was previously run exclusively at 300 MHz ^{1}H frequency, subsequent trials at 500 MHz have revealed practical limitations of running complex pulse sequences on the existing 300 MHz spectrometer. There were many pulse sequences that despite showing sharp spectra in silica, when run on the 300 MHz spectrometer either had very poor linewidths, or completely failed to render observable peaks altogether. Furthermore, pulse sequences that successfully rendered all four individual peaks in the NAL crystal were a mere fraction amongst all the sequences tested. The most important finding was that, when implemented

on a 500 MHz spectrometer, there appears to be few, if any of these false positives between the simulations and experiment. It is likely that the lower frequency spectrometer is only appropriate for simpler pulse sequences such as PISEMA, or SAMPI4, whereas for complex computer-generated pulse sequences the hardware was not able to adequately produce non-quadrature phases with sufficiently short phase transients between them. Results give further confidence in the reliability computational pulse sequence design, and in the adequacy of the simulation methods used.

The target function (i.e. figure of merit) in (Eq. 4.16) was originally determined for convenience. The equation in its totality selects for the most intense spectra that are effectively decoupling homonuclear and heternuclear interactions. Spectral amplitude is a desirable quantity for two reasons; it can be measured easily and unambiguously, and it is a smoothly increasing quantity. In the context of Monte Carlo simulations, a negative value increasing in amplitude is very desirable for minimization. Unfortunately, the amplitudes/intensities between different simulated spectra are difficult to compare directly. In order to account for this, the FID was multiplied by the total dwell time. These simulations tended to favor sub-optimal sequences with unrealistic, short scaling factors. An important addition to this program was the inclusion of the computational peak width. This presents two challenges: optimal peak width is a value declining in magnitude, and the difficulty of defining peak width between peaks with variable shapes. Therefore, a new figure of merit was designed to find the lowest linewidths, and was constructed with the following modifications. The product of peak intensity is replaced by the product of peak widths, raised to the number of probed resonances.

$$PW = \left(\prod_{i}^{n} pw_i \right)^{\frac{1}{n}} \qquad (Eq. 4.17)$$

Since the spectra are calculated prior to the application of a scaling factor, the calculated peak widths reflect that for an *scf* of 1. True peak width, i.e. with a scaling factor, is approximated according to (Eq. 4.18).

$$pw^{true} = \frac{\nu_{true}}{\nu_{calc}} pw^{calc} \qquad (Eq. 4.18)$$

A minimum peak width must be defined in order to prevent the peak widths of small artifacts from being counted. For ^{1}H-^{15}N, this value was 85 Hz, and for $C_{\alpha}H_{\alpha}$ it was 150 Hz. The peak frequency factor was modified from that of the absolute differences of (Eq. 4.16) to a gaussian/normal distribution shape for the difference between the expected and measured frequencies, with deviation z. This function is also inverted, rising to a value of Ω since the optimization involves minimizing a positive number.

$$fac = \prod_{j}^{n-1} \Omega - (\Omega - 1) \exp\left(-\left(\frac{\left(rat_j - \frac{pf_j}{pf_{j+1}} \right)}{z} \right)^2 \right) \qquad (Eq. 4.19)$$

where rat_j is the expected ratio between the frequencies of peak *j* and peak *j+1*. Ideally *fac* will be 1, but otherwise will increase the magnitude of the final score. Finally, the product of the peak intensities is subtracted from (Eq. 4.17) as a discount to the score, ensuring that peak width and peak height are optimized jointly.

$$score = fac \cdot \left(PW - wt \left(\prod_{i}^{n} ph_i \right)^{\frac{1}{n}} \right) \qquad (Eq. 4.20)$$

Parameter *wt* has the purpose of reconciling the disparate scales of *PW* (10^2-10^4) and *ph* (10^0-10^1). The upshot of this scoring function is to ensure both sharp and intense spectra, of course still with properly decoupled interactions, enforced by *fac*. For Ω a value of 3 was sufficient to penalize possible singularities, and wt was set to 100. Deviation *z* was set to 0.834.

Pulse sequences for ^{1}H-^{15}N DC were newly optimized for comparison to SAMPI4 on NAL at 500 MHz ^{1}H freuqencies. These optimizations, in addition to phases and timings, also optimized ω_{rf} amplitudes. For the purposes of the simulation these variables were considered as boolean values, i.e. either on at rf frequency ω_{rf} or off at 0. This stands in contrast to the previous optimizations which optimized only phases and timings, under the SAMPI4 power scheme with a hole for subdwells 3 and 4 on the I-channel. This yielded a new champion pulse sequence, whose parameters are listed in table 4.3

Table 4.3: ROULETTE-2.0 and ROULETTE-CAHA, optimized at ω_{rf} = 61 kHz

Exp.	ϕ_{11}	ψ_{11}	ϕ_{12}	ψ_{12}	ϕ_{13}	ψ_{13}	ϕ_{21}	ψ_{21}	ϕ_{22}	ψ_{22}	ϕ_{23}	ψ_{23}	t_1 (μs)	t_2 (μs)	t_3 (μs)
ROUL.-2.0	201	176	190	152	141	273	248	154	288	165	182	73	*10.99*	*3.19*	6.46
ROUL.-CAHA	126	347	208	83	279	45	251	3	167	87	158	120	6.90	3.44	2.59

The most notable difference from previous optimized sequences is that this sequence is fully "windowless", i.e. the power on the I-channel remains "on" for all subdwells, whereas it was "off" for subdwells 3 and 4 in the previous scheme(s).

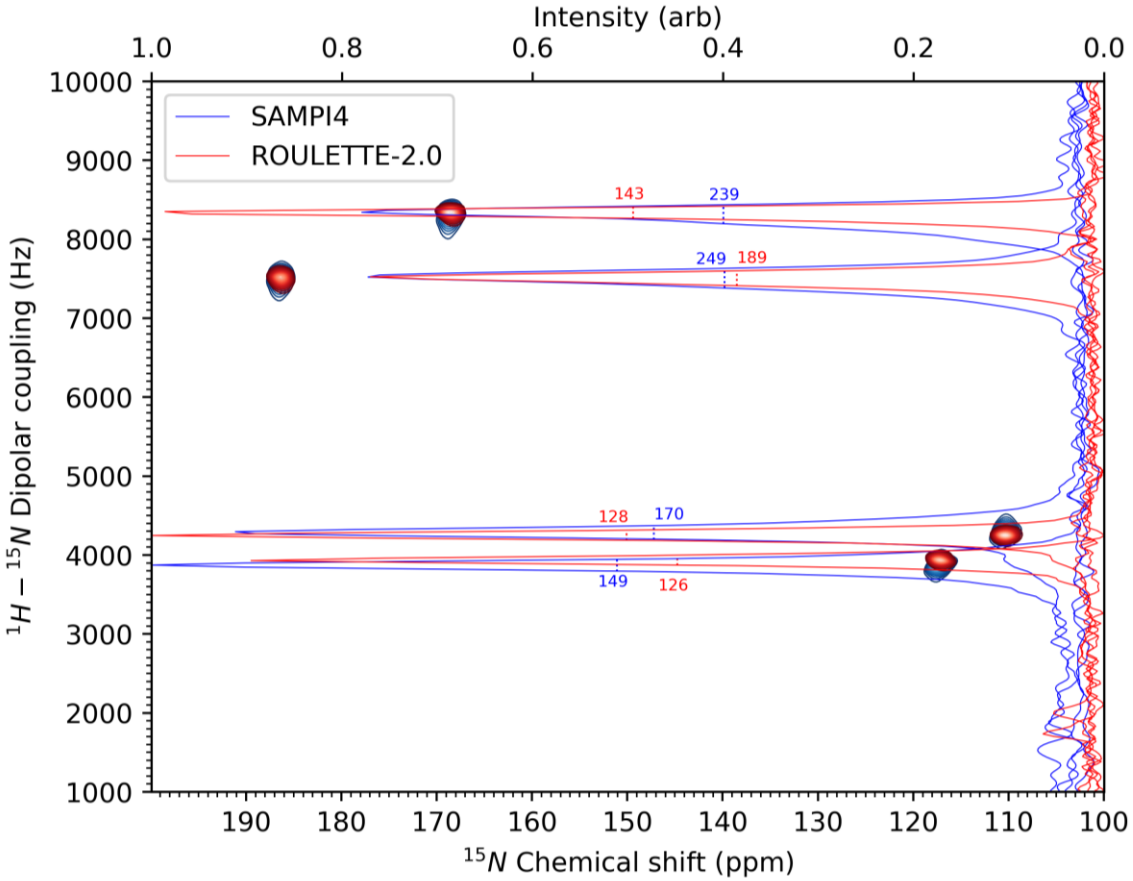


Figure 4.8. Pulse sequences SAMPI4 vs. ROULETTE-2.0 (from table 4.3); the dipolar ^{1}H-^{15}N dimension is correlated with ^{15}N CSA. Mean linewidth at half height in the ^{1}H-^{15}N dimension amongst the 4 peaks is 201.75 and 146.5 Hz for SAMPI4 and ROULETTE-2.0, respectively (yielding 32% gain in resolution).

It should be noted that implementing the newly optimized pulse sequences on the 500 MHz spectrometer having three dedicated amplifiers for all three channels (^{1}H, ^{13}C, ^{15}N) has increased the disparity in the linewidths between SAMPI4 and the computationally optimized pulse sequence, with the latter now having a 32% sharper mean linewidth, as compared to 18% sharper

on the 300 MHz spectrometer with a dual amplifier. Furthermore, whereas there was a notable disparity in the frequency of high and low dipolar couplings between the ROULETTE pulse sequences and SAMPI4, now the measured couplings nearly perfectly overlap. The prior disparity appears to be further evidence of the limitations of the more basic 300 MHz spectrometer to accurately measure computationally optimized pulse sequences.

As a further demonstration of the method, the ROULETTE approach has been extended to optimization of $C_\alpha H_\alpha$ dipolar linewidths in NAL. As with ^{1}H-^{15}N DC, pulse sequences were optimized for a 6 subdwell sequence architecture. Optimization of $C_\alpha H_\alpha$ pulse sequences had several differences with respect to the previous ^{1}H-^{15}N optimization. The spin system had to be modified such that it was centered on an C_a atom (still based on an alanine residue), surrounded by three other S-spins (nearby carbons) and four I-spins (hydrogens). As such, these simulations were slower due to the inclusion of an additional spin, resulting in a nine-spin system. Owing to a coupling constant more than twice as large as ^{15}N, the dipolar splittings were measured all the way up to 40 kHz (or +/-20 kHz). Specifically 4 couplings were probed; 40, 30, 20, and 10 kHz. In general, simulated $C_\alpha H_\alpha$ spectra were in all cases predicted to be less sharp than simulations with ^{1}H-^{15}N, especially for the smallest coupling of 10 kHz. N-acetyl Leucine contains 4 measurable $C_\alpha H_\alpha$ resonances, which were easily identified by the presence of a peak splitting when measured without ^{15}N decoupling.

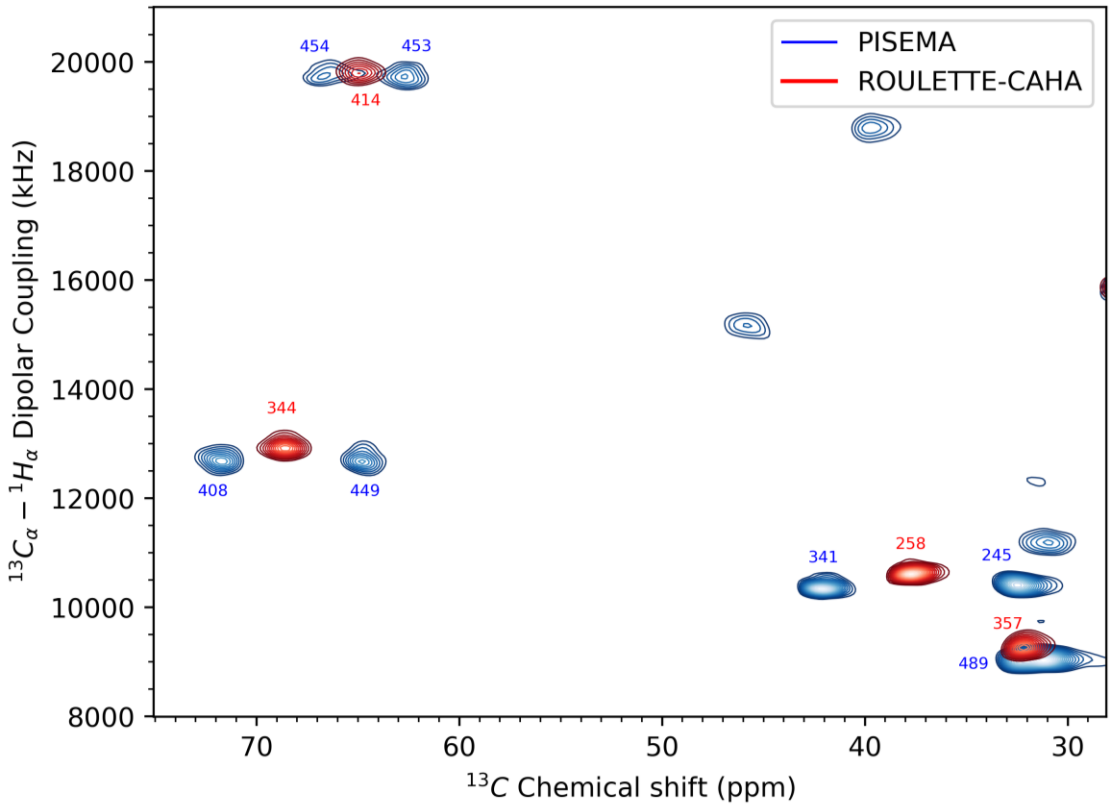


Figure 4.9. Overlay of PISEMA and ROULETTE-CAHA (table 4.3) spectra of NAL crystal at 500 MHz ^{1}H frequency. The $C_\alpha H_\alpha$ DC dimension, along the y-axis, is correlated with ^{13}C CSA, along the x-axis. The PISEMA (blue) experiment was run without ^{15}N decoupling, such that doublet peaks show up along the direct dimension; ROULETTE-CAHA (red), by contrast, was run with ^{15}N decoupling. All relevant resonances are marked with their corresponding linewidth (in Hz) in the DC dimension. The unmarked peaks correspond to the aliphatic carbons.

The newly optimized sequence is termed ROULETTE-CAHA. Its parameters in Table 4.3 show that this is also a sequence for which the power on both rf channels is always on. The resulting spectrum of ROULETTE-CAHA is shown in Fig. 4.9 for comparison with PISEMA (Wu, 1994). The PISEMA spectrum, in blue, unambiguously displays the presence of 3 $C_\alpha H_\alpha$ peaks ranging from 10-20 kHz, as evident by the sharp doublets with the uncoupled ROULETTE-CAHA, in red, in between. The fourth peak, around (32 ppm, 9 kHz), failed to completely split for undecoupled PISEMA, but still can be identified by its coincidence with ROULETTE-CAHA. It can be seen that ROULETTE-CAHA has 15% sharper mean linewidths when compared to the sharper peaks of each of PISEMA's doublets. It should be noted that at the time of this writing, only a handful of sequences have been tested for $C_\alpha H_\alpha$ DC's, and there still an on-going work to find more optimal sequences.

4.7 References

Bak, M.; Rasmussen, J. T.; Nielsen, N. C. (2000) SIMPSON: A General Simulation Program for Solid-State NMR Spectroscopy, Journal of Magnetic Resonance 147, 296-330

Bielecki, A.; Kolbert, A. C.; de Groot, H. J. M.; Griffin, R. G.; Levitt, M. H. (1990) Frequency-Switched Lee-Goldburg Sequences in Solids, Adv. Magn. Reson. 14, 111

Brinkmann, A. (2016) Introduction to Average Hamiltonian Theory. I. Basics, Concepts in Magnetic Resonance Part A 45A

Burum, D. P.; Linder, M.; Ernst, R. R. (1981) Low-Power Multipulse Line Narrowing in Solid-State NMR, J. Magn. Reson. 44, 173-188

Cui, J. Y.; Li, J.; Liu, X. M.; Peng, X. H.; Fu, R. Q. (2018) Engineering Spin Hamiltonians Using Multiple Pulse Sequences in Solid State NMR Spectroscopy, Journal of Magnetic Resonance 294, 83-92

Delaglio, F.; Grzesiek, S.; Vuister, G. W.; Zhu, G.; Pfeifer, J.; Bax, A. (1995) NMRPipe: A Multidimensional Spectral Processing System Based on UNIX Pipes, J. Biomol. NMR 6, 277-293

Dvinskikh, S. V.; Sandström, D. (2005) Frequency Offset Refocused PISEMA-Type Sequences, J. Magn. Reson. 175, 163-169

Dvinskikh, S. V.; Yamamoto, K.; Ramamoorthy (2006) Heteronuclear Isotopic Mixing Separted Local Field NMR Spectroscopy, J. Chem. Phys. 125, 034507

Gan, Z. (2000) Spin Dynamics of Polarization Inversion Spin Exchange at the Magic Angle in Multiple Spin Systems, J. Magn. Reson. 143, 136-143

Gopinath, T.; Veglia, G. (2009) Sensitivity Enhancement in Static Solid-State NMR Experiments via Single- and Multiple-Quantum Dipolar Coherences, Journal of the American Chemical Society 131, 5754-5756

Haeberlen, U.; Waugh, J. S. (1968) Coherent Averaging Effects in Magnetic Resonance, Physical Review 175, 453-467

Hogben, H. J.; Krzystyniak, M.; Charnock, G. T. P.; Hore, P. J.; Kuprov, I. (2011) Spinach - A Software Library for Simulation of Spin Dynamics in Large Spin Systems, Journal of Magnetic Resonance 208, 179-194

Jayanthi, S.; Ramanathan, K. V. (2010) 2(n)-SEMA-A Robust Solid State Nuclear Magnetic Resonance Experiment for Measuring Heteronuclear Dipolar Couplings in Static Oriented Systems Using Effective Transverse Spin-Lock, Journal of Chemical Physics 132

Jayanthi, S.; Sinha, N.; Ramanathan, K. V. (2010) 2(4)-SEMA as a Sensitive and Offset Compensated SLF Sequence, Journal of Magnetic Resonance 207, 206-212

Lapin, J.; Nevzorov, A. A. (2019) Validation of Protein Backbone Structures Calculated from NMR Angular Restraints Using Rosetta, Journal of Biomolecular Nmr 73, 229-244

Lee, M.; Goldburg, W. I. (1965) Nuclear-Magnetic-Resonance Line Narrowing By a Rotating Rf Field, Physical Review 140, 1261-1271

Leskes, M.; Madhu, P. K.; Vega, S. (2010) Floquet Theory in Solid-State Nuclear Magnetic Resonance, Progress in Nuclear Magnetic Resonance Spectroscopy 57, 345-380

Metropolis, N.; Ulam, S. (1949) The Monte Carlo Method, Journal of the American Statistical Association 44, 335-341

Nevzorov, A. A.; Opella, S. J. (2003) A "Magic Sandwich" Pulse Sequence with Reduced Offset Dependence for High-Resolution Separated Local Field Spectroscopy, J. Magn. Reson. 164, 182-186

Nevzorov, A. A.; Opella, S. J. (2007) Selective Averaging for High-Resolution Solid-State NMR Spectroscopy of Aligned Samples, J. Magn. Reson. 185, 59-70

Sakellariou, D.; Lesage, A.; Hodgkinson, P.; Emsley, L. (2000) Homonuclear Dipolar Decoupling in Solid-State NMR Using Continuous Phase Modulation, Chem. Phys. Letts. 319, 253-260

Sinha, N.; Grant, C. V.; Park, S. H.; Brown, J. M.; Opella, S. J. (2007) Triple Resonance Experiments for Aligned Sample Solid-State NMR of C-13 and N-15 Labeled Proteins, Journal of Magnetic Resonance 186, 51-64

Veshtort, M.; Griffin, R. G. (2006) SPINEVOLUTION: A Powerful Tool for the Simulation of Solid and Liquid State NMR Experiments, Journal of Magnetic Resonance 178, 248-282

Wu, C. H.; Ramamoorthy, A.; Opella, S. J. (1994) High-Resolution Heteronuclear Dipolar Solid-State NMR Spectroscopy, J. Magn. Reson. A 109, 270-272.

Yamamoto, K.; Lee, D. K.; Ramamoorthy, A. (2005) Broadband-PISEMA Solid-State NMR Spectroscopy, Chem. Phys. Letts. 407, 289-293

Yin, Y. Y.; Nevzorov, A. A. (2011) A. Structure Determination in "Shiftless" Solid State NMR of Oriented Protein Samples, J. Magn. Reson. 212, 64-73

CHAPTER 5: NMR "CRYSTALLOGRAPHY" FOR UNIFORMLY (^{13}C, ^{15}N)-LABELED ORIENTED MEMBRANE PROTEINS

(Chapter based on publication: Awosanya, E.O., Lapin, J., Nevzorov, A.A. (2020) Angewandte Chemie Intl. Ed.)

5.1 Introduction

Structure elucidation of membrane proteins (MPs) in planar, native-like bilayers remains a central topic in modern structural biology. While being the undeniable leaders in structure determination of MPs, both X-ray crystallography and cryo-EM employ cryogenic temperatures, which may trap the proteins in biased structural conformations different from those encountered under the native conditions. Moreover, the fluid-like bilayer lipid bilayer environment has been shown to affect structure and function of membrane proteins (White and Wimley, 1999; Bechinger, 2000; Lee, 2011; Laganowsky et al., 2014; Hamilton et al., 2014; Martens et al., 2016). Solid-state NMR is a powerful spectroscopic tool, which yields atomic-level information about MPs under nearly physiological conditions (Traaseth et al., 2009; Shi et al., 2009; Cady et al., 2010; Sharma et al., 2010; Park et al., 2012; Renault et al., 2011; Wang et al., 2013; Elkins et al., 2017). In particular, the method of oriented-sample (OS) ssNMR involves the measurements of the orientationally dependent dipolar couplings, which provide angular restraints for structure calculations. These restraints are highly complementary to the distance restraints obtainable in solid-state Magic Angle Spinning (MAS) NMR where no preferred sample orientation is present and the angular information is, therefore, inaccessible. So far, only the ^{1}H-^{15}N dipolar couplings and ^{15}N chemical shift anisotropy (CSA) have been routinely measured in singly labeled protein samples by the high-resolution separated local field (SLF) techniques (Wu et al., 1994; Dvinskikh et al., 2005; Dvinskikh et al., 2006; Nevzorov and Opella, 2003; Nevzorov and Opella, 2007; Gopinath and Veglia, 2009; Jayanthi and Ramanathan, 2010, Jayanthi et al., 2010). Using the two-dimensional (^{1}H-^{15}N/^{15}N) OS NMR spectra, many MP structures have been determined (Traaseth et al., 2009; Sharma et al., 2010; Park et al., 2006; De Angelis et al., 2006). However, while being efficient for determining the tilts of short α-helical peptides within the membrane, the ^{1}H-^{15}N dipolar and ^{15}N CSA frequencies are by themselves insufficient for determining protein structures

of arbitrary topology. Structure determination of membrane proteins by OS NMR will likely require measurements of additional angular restraints, for which doubly (^{15}N, ^{13}C) labeled protein samples would need to be employed (Vosegaard and Nielsen, 2002). Structure calculations from fitting synthetic multidimensional OS NMR data (Yin and Nevzorov, 2011; Lapin and Nevzorov, 2019) have identified the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings as a critical third spectroscopic dimension, which would enable elucidation of the backbone folds of MPs without assuming any secondary structure elements *a priori*. The chiral nature of the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings in the predominantly L-aminoacids largely removes the orientational degeneracies associated with the second-rank ^{1}H-^{15}N dipolar and ^{15}N CSA interactions. However, for the uniformly doubly labeled samples the presence of the abundant ^{13}C spins represents an essential complication arising from the ^{13}C-^{13}C homonuclear couplings, which broaden NMR linewidths and, thus, negatively affect spectroscopic resolution. Previous attempts of assessing the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings in oriented membrane proteins involved either natural-abundance crystal samples (Gu and Opella, 1999) or selective incorporation of ($^{13}C_{\alpha}$, ^{15}N) labeled aminoacids (Sinha et al., 2007), which can be prohibitively expensive for larger proteins. Moreover, not all ($^{13}C_{\alpha}$, ^{15}N) isotopically labeled aminoacids are available commercially. Therefore, in order to measure $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings, radically new pulse sequences need to be developed that would be generally applicable to uniformly doubly labeled membrane proteins.

5.2 Results

Here we present a triple-resonance pulse sequence that enables detection of the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings at the ^{15}N amide sites for a uniaxially aligned protein NMR sample. The performance of the pulse sequence has been tested numerically and by using a uniformly (^{15}N, ^{13}C) doubly labeled Pf1 phage coat protein reconstituted in magnetically aligned bicelles. Pf1 coat protein, the major protein of Pf1 bacteriophage, represents an ideal test system for methods development in OS NMR since its transmembrane domain consists of a well-characterized α-helix, and its OS NMR spectra have been unambiguously assigned using both selective labeling and the PISA wheel patterns, (Park et al., 2010) and spectroscopic techniques (Tang et al., 2012; Lu and Opella, 2014). Moreover, recently the Pf1 spectroscopic assignment has been confirmed by an automated assignment algorithm (Lapin and Nevzorov, 2018). Our developed NMR pulse sequence shown in Fig. 5.1 consists of three blocks: (i) Evolution of ^{1}H-^{13}C couplings under both ^{1}H-^{1}H and ^{13}C-^{13}C

homonuclear decoupling; (ii) Proton-mediated magnetization transfer of ^{13}C dipolar evolution to the proximal ^{15}N sites; (iii) Detection at the ^{15}N spin sites. The first part is achieved by essentially applying two SAMPI4 blocks (Nevzorov and Opella, 2007) on both channels, hence the pulse sequence was named SAMPI4^2. In order to refocus the ^{13}C chemical shift evolution and the secular, i.e. I_zS_z , part of the ^1H-^{13}C dipolar Hamiltonian an additional refocusing π_y pulse has been inserted on the ^{13}C channel. The proton-mediated transfer under mismatched Hartmann-Hahn (MMHH) conditions (Nevzorov, 2008) provides selective magnetization transfer from the nearest $^1H_\alpha$-$^{13}C_\alpha$ couplings to the ^{15}N sites via the amide and $^1H_\alpha$ protons. The contributions from the dipolar interactions originating from the more distant ^{13}C spins are expected to be negligible. A series of optimizations and control experiments has been performed to confirm that all ^{15}N detected magnetization indeed originates from the ^{13}C spins. Bacteriophage represents an ideal test system for method development in OSNMR, because its transmembrane domain consists of a well characterized a-helix, and its OS-NMR spectra have been unambiguously assigned using both selective labeling and the PISA wheel patterns (Park et al., 2010) and spectroscopic techniques (Tang et al., 2012; Lu and Opella, 2014). Moreover, recently, the spectroscopic assignment of Pf1 has been confirmed by an automated assignment algorithm (Lapin and Nevzorov, 2018). Our developed NMR pulse sequence shown in Fig. 5.1 consists of three blocks: i) Evolution of ^1H–^{13}C couplings under both ^1H–^1H and ^{13}C–^{13}C homonuclear decoupling;

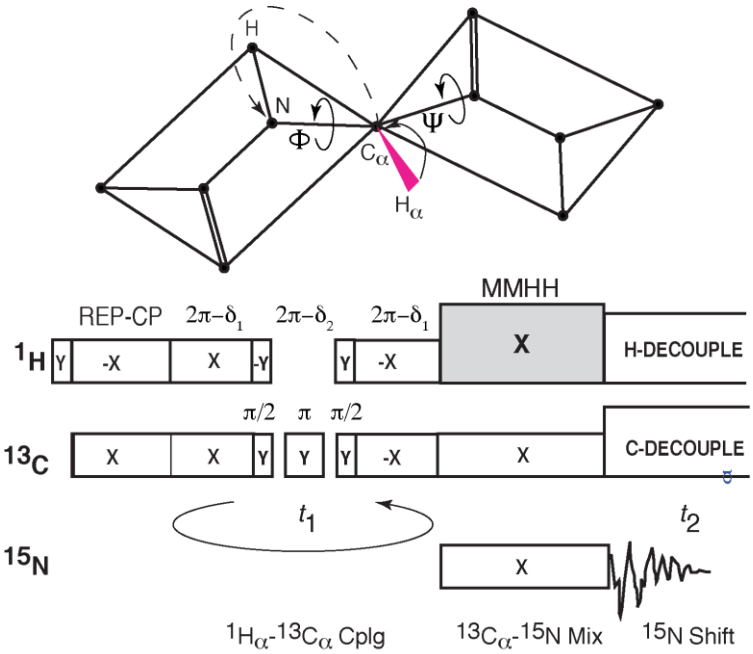


Figure 5.1. The developed SAMPI4[2] pulse sequence for detecting $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings at ^{15}N sites suitable for uniformly labeled (^{15}N, ^{13}C) aligned samples. Two peptide planes are included on top to illustrate the spin magnetization transfer pathway and the torsion angles Φ and Ψ. The timing corrections δ_1 and δ_2 correspond to the durations of $\pi/4$ and $\pi/2$ pulses, respectively. Figure obtained from Awosanya et al., 2020, Angewandte Chemie Intl. Ed.

Figure 5.2 shows an overlay of two-dimensional ($^{1}H_{\alpha}$-$^{13}C_{\alpha}$/^{15}N) SAMPI4[2] and (^{1}H-^{15}N/^{15}N) SAMPI4 spectra with ^{13}C SPINAL32 decoupling (Fung et al., 2000) added during the ^{15}N CSA acquisition. The ($^{1}H_{\alpha}$-$^{13}C_{\alpha}$ /^{15}N) SAMPI4[2] spectrum of Pf1 is shown in blue and the (^{1}H-^{15}N/^{15}N) SAMPI4 spectrum is shown in red. As an additional control, SAMPI4[2] was run to evolve the ^{1}H-^{15}N dipolar couplings under ^{13}C decoupling and gave almost identical results with SAMPI4 for both single crystal of n-acetyl Leucine and Pf1. As can be seen, the linewidths in the indirect (dipolar) dimension in the ($^{1}H_{\alpha}$-$^{13}C_{\alpha}$/^{15}N) SAMPI4[2] spectrum are wider than for its (^{1}H-^{15}N/^{15}N) SAMPI4 counterpart. This may be due to incomplete uniaxial diffusional averaging of the larger $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings at the perpendicular bicelle orientation. In addition, the relatively moderate rf powers (ca. 50 kHz) generated by the NMR probe may be insufficient for providing efficient simultaneous ^{13}C-^{13}C and ^{1}H-^{1}H decoupling. Many of the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ SAMPI4[2] peaks can be immediately assigned from the (^{1}H-^{15}N/^{15}N) SAMPI4 spectra by relating their corresponding ^{15}N chemical shifts. Assignments of some peaks required extra discernment. For instance, we have measured a selectively ($^{13}C_{\alpha}$, ^{15}N) Valine-labeled spectrum to delineate between the peaks in the crowded region at ca. 102 ppm (shown as brown contours in Fig. 5.2). This allowed us to identify

the Valine peaks and restrict the assignment possibilities for other residues. Other assignments can be resolved in a similar manner, i.e. by invoking selective labeling. Of special note are the Glycine ^{1}H-^{13}C peaks as the α-carbon is bonded to both H_α and H_β protons. The dipolar frequencies for such a three-spin system follow the "hypotenuse" relation in the doubly tilted rotating frame (Gan, 2000), *viz.*:

$$\nu_{^{1}H\text{-}^{13}C}(Gly) = \sqrt{\nu^2_{^{1}H_a\text{-}^{13}C_a} + \nu^2_{^{1}H_b\text{-}^{13}C_a}} \qquad \text{(Eq. 5.1)}$$

As a consequence, the Glycine ^{1}H-^{13}C dipolar couplings are typically larger than their corresponding ^{1}H-^{15}N interactions, which facilitates their assignment. The hypotenuse relation, Eq. (5.1), must also be taken into account during the structural fitting.

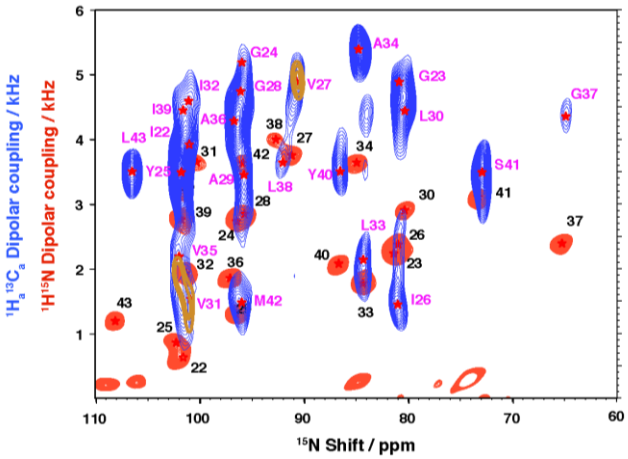


Figure 5.2. Overlay of ^{15}N-detected SLF ^{1}H-^{15}N (red contours) and $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings (blue contours) measured for doubly labeled sample of Pf1 coat protein reconstituted in magnetically aligned bicelles. Selectively labeled ($^{13}C_{\alpha}$, ^{15}N) Valine spectrum is shown in brown contours. Assignment of the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings is shown by the text in magenta. Figure obtained from Awosanya et al., 2020, Angewandte Chemie Intl. Ed.

In the case of Pf1 coat protein undergoing fast uniaxial rotational diffusion in the perpendicularly aligned bicelles, the observed dipolar couplings and chemical shifts are scaled by the effective order parameter, $-S_o/2$. Motions of the more dynamic (while structured) domains can potentially be accounted for in a manner similar to the residual dipolar couplings (Tolman et al., 1995), i.e. by introducing the additional local axial and rhombic components, which lies beyond the scope of the present paper. Ten thousand (10,000) structural solutions satisfying the three experimental angular restraints have been calculated using our recently developed structure fitting algorithm (Lapin and Nevzorov, 2019). Briefly, the algorithm finds iteratively the values for the two pairs of torsion angles Φ and Ψ that are consistent with the sets of NMR angular restraints measured for *three* adjacent peptide planes (in this case ^{15}N CSA, ^{1}H-^{15}N and, $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings). Fitting simultaneously the torsion angles for three adjacent planes instead of two alleviates the problem of overfitting some resonances at the expense of others and greatly reduces propagation of error. It should be emphasized that during the fitting the starting values for the torsion angles of Φ and Ψ have been randomized between -180^{o} and 180^{o} instead of biasing the search around the helical values. Figure 5.3A shows the distributions of the final (i.e. fitted) torsion angles over the Ramachandran map for 1,000 calculated structures. Shown in red are the values obtained from the fitting of three angular constraints per plane (denoted as 3D). Shown in blue for comparison are the fitted values from using just the two "traditional" ^{1}H-^{15}N dipolar and ^{15}N CSA NMR

dimensions (2D). As can be seen, the torsion angles from the 3D fit are predominantly clustered near the α-helical values, i.e. Φ=-61°; Ψ=-45°, whereas the final angles from the 2D fit are scattered over the much broader range of the Ramachandran map (while still yielding good fits to the NMR data). The next and final step was to perform a top-down filtering of the calculated 10,000 structures using three MP Rosetta scoring terms, cf. Fig. 5.3B. The first step in the filtering procedure included the membrane positioning scoring function, MPTMProj, (Yarov-Yarovoy et al., 2006) from which the top 500 solutions were collected. The filtering procedure then proceeded by scoring the remaining structural solutions for the allowed Ramachandran angles, p_aa_pp (Leaver-Fay et al., 2013), and choosing the top 50 solutions amongst the prior 500.

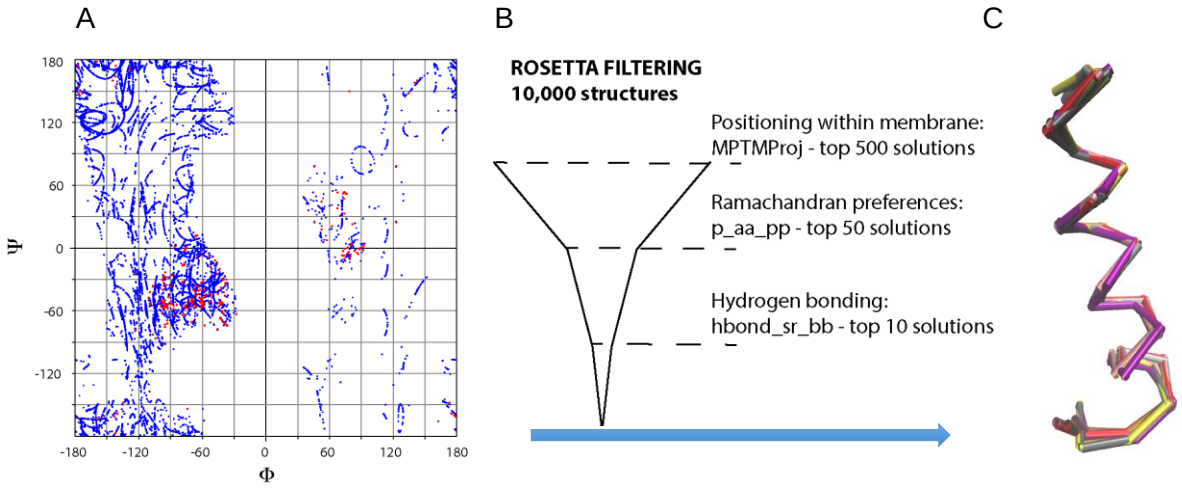


Figure 5.3. A. The resulting values of the torsion angles on the Ramachandran map for 2D vs 3D NMR data fit for 1,000 calculated structures. A clustering near the α-helical torsion angles is apparent for the 3D fit. No restriction on the torsion angles was assumed during the fitting. **B.** ROSETTA three-tier filtering protocol. **C.** Consensus family of ten (10) Pf1 TM structures determined from NMR. Figure obtained from Awosanya et al., 2020, Angewandte Chemie Intl. Ed.

Finally, the hydrogen bonding scoring function, hbond_sr_bb (Shmygelska and Levitt, 2009), was applied which resulted in the final 10 solutions shown in Fig. 5.3C. The final structure indeed results in an α-helix with a slightly unstructured N-terminal part containing the G23 and G24 residues. The slight unwinding of the helix at the N terminus is consistent with the pattern for the ^{1}H-^{15}N dipolar couplings for Pf1 reconstituted in perpendicularly aligned DMPC/DHPC bicelles (Opella et al., 2008), which demonstrate that these residues do not line up well with the "dipolar wave" of periodicity 3.6 (Mesleh and Opella, 2003). It should also be noted that a deliberate random switching of some of the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar assignments in Fig. 5.2 did not produce any

meaningful, high-scoring structures. Moreover, the fact that the measured $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings are consistent with the known α-helical structure of Pf1 provides an additional validation of our experimental method and the spectroscopic NMR assignment.

5.3 Conclusion

In conclusion, we have demonstrated, for the first time, that backbone folds of MPs can be experimentally determined by fitting simultaneously three NMR angular restraints per peptide plane and, notably, an α-helical configuration does not need to be assumed *a priori*. Including the third, chiral $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dimension allows one to search for the torsion solutions over the full range of the Ramachandran map without restricting the Φ and Ψ angles to an α-helical range. This new method is expected to greatly advance oriented-sample NMR as a "crystallography" tool in aligned, lipid-rich bilayers at near-physiological conditions.

5.4 References

Awosanya, E.O., Lapin, J., Nevzorov, A.A. (2020) NMR "Crystallography" for Uniformly (13C, 15N)-Labeled Oriented Membrane Proteins, Angewandte Chemie 132:3582-3585

Bechinger, B. (2000) Understanding peptide interactions with the lipid bilayer: a guide to membrane protein engineering, Curr Opin. Chem. Biol., 4, 639-644

Bertelsen, K., Paaske, B., Thogersen, L., Taikhorshid, E., Schiott, B., Skrydstrup, T., Nielsen, N.C., Vosegaard, T. (2009) Residue-Specific Information about the Dynamics of Antimicrobial Peptides from $^{1}H–^{15}N$ and ^{2}H Solid-State NMR Spectroscopy, J. Am. Chem. Soc., 131, 18335-18342

Cady, S.D., Schmidt-Rohr, K., Wang, J., Soto, C.S., DeGrado, W.F., Hong, M. (2010) Structure of the amantadine binding site of influenza M2 proton channels in lipid bilayers, Nature, 463, 689-692

De Angelis, A.A., Howell, S.C., Nevzorov, A.A., Opella, S.J. (2006) Structure Determination of a Membrane Protein with Two Trans-membrane Helices in Aligned Phospholipid Bicelles by Solid-State NMR Spectroscopy, J. Am. Chem. Soc., 128, 12256-12267

Dvinskikh, S.V., Sandstrom, D. (2005) Frequency offset refocused PISEMA-type sequences, J. Magn. Reson., 175, 163-169

Dvinskikh, S.V., Yamamoto, A., Ramamoorthy, A. (2006) Heteronuclear isotropic mixing separated local field NMR spectroscopy, J. Chem. Phys., 125, 034507

Elkins, M.R., Williams, J.K., Gelenter, M.D., Dai, P., Kwon, B., Sergeyev, I.V., Pentelute, B.L., Hong, M. (2017) Cholesterol-binding site of the influenza M2 protein in lipid bilayers from solid-state NMR, Proc. Natl. Acad. Sci. USA, 114, 12946-12951

Fung, B.M., Khitrin, A.K., Ermolaev, K. (2000) An Improved Broadband Decoupling Sequence for Liquid Crystals and Solids, J. Magn. Reson., 142, 97-101

Gan, Z., (2000) Spin Dynamics of Polarization Inversion Spin Exchange at the Magic Angle in Multiple Spin Systems, J. Magn. Reson., 143, 136-143

Gayen, A., Banigan, J.R., Traaseth, N.J. (2013) Ligand-Induced Conformational Changes of the Multidrug Resistance Transporter EmrE Probed by Oriented Solid-State NMR Spectroscopy, Angew. Chem. Int. Ed., 52, 10321-10324; (2013) Angew. Chem., 125, 10511-10514

Gleason, N.J., Vostrikov, V.V., Greathouse, D.V., Koeppe, R.E. (2013) Buried lysine, but not arginine, titrates and alters transmembrane helix tilt, Proc. Natl. Acad. Sci. USA, 110, 1692-1695

Gopinath, T., Veglia, G. (2009) Sensitivity Enhancement in Static Solid-State NMR Experiments via Single- and Multiple-Quantum Dipolar Coherences, J. Am. Chem. Soc., 131, 5754-5756

Gu, Z.T., Opella, S.J. (1999) Two- and Three-Dimensional $^{1}H/^{13}C$ PISEMA Experiments and Their Application to Backbone and Side Chain Sites of Amino Acids and Peptides, J. Magn. Reson., 140, 340-346

Hamilton, P.J., Belovich, A.N., Khelashvili, G., Saunders, C., Erreger, K., Javitch, J.A., Sitte, H.H., Weinstein, H., Matthies, H.J.G., Galli, A. (2014) PIP_2 regulates psychostimulant behaviors through its interaction with a membrane protein, Nat. Chem. Biol., 10, 582-589

Jayanthi, S., Ramanathan, K.V. (2010) 2*n*-SEMA-a robust solid state nuclear magnetic resonance experiment for measuring heteronuclear dipolar couplings in static oriented systems using effective transverse spin-lock, J. Chem. Phys., 132, 134501

Jayanthi, S., Sinha, N., Ramanathan, K.V. (2010) 2_4-SEMA as a sensitive and offset compensated SLF sequence, J. Magn. Reson., 207, 206-212

Laganowsky, A., Reading, E., Allison, T.M., Ulmschneider, M.B., Degiacomi, M.T., Baldwin, A.J., Robinson, C.V. (2014) Membrane proteins bind lipids selectively to modulate their structure and function, Nature, 510, 172-175

Lapin, J., Nevzorov, A.A. (2018) Automated assignment of NMR spectra of macroscopically oriented proteins using simulated annealing, J. Magn. Reson., 293, 104-114

Lapin, J., Nevzorov, A.A. (2019) Validation of protein backbone structures calculated from NMR angular restraints using Rosetta, J. Biomol. NMR, 73, 229-244

Leaver-Fay, A., O'Meara, M.J., Tyka, M., Jacak, R., Song, Y.F., Kellogg, E.H., Thompson, J., Davis, I.W., Pache, R.A., Lyskov, S., Gray, J.J., Kortemme, T., Richardson, J.S., Hayranek, J.J., Snoeyink, J., Baker, D., Kuhlman, B. (2013) Chapter Six - Scientific Benchmarks for Guiding Macromolecular Energy Function Improvement, Methods, in Protein Design, Vol. 523 (Ed,: A.E. Keating), Academic Press, San Diego, CA, pp. 109-143

Lee, A.G., (2011) Lipid–protein interactions, Soc. Trans., 39, 761-766

Lu, G.J., Opella, S.J. (2014) Resonance assignments of a membrane protein in phospholipid bilayers by combining multiple strategies of oriented sample solid-state NMR, J. Biomol. NMR, 58, 69-81

Mahalakshmi, R., Marassi, F.M. (2008) Orientation of the *Escherichia coli* Outer Membrane Protein OmpX in Phospholipid Bilayer Membranes Determined by Solid-State NMR, Biochemistry, 47, 6531-6538

Martens, C., Stein, R.A., Masureel, M., Roth, A., Mishra, S., Dawaliby, R., Konjinenberg, A., Sobott, C., Govaerts, C., Mchaourab, H.S. (2016) Lipids modulate the conformational dynamics of a secondary multidrug transporter, Nat. Struct. Mol. Biol., 23, 744-751

Mesleh, M.F., Opella, S.J. (2003) Dipolar Waves as NMR maps of helices in proteins, J. Magn. Reson., 163, 288-299

Mote, K.R., Gopinath, T., Traaseth, N.J., Kitchen, J., Gorskov, P.L., Brey, W.W., Veglia, G. (2011) Multidimensional oriented solid-state NMR experiments enable the sequential assignment of uniformly ^{15}N labeled integral membrane proteins in magnetically aligned lipid bilayers, J. Biomol. NMR, 51, 339-346

Murray, D.T., Li, C., Gao, F.P., Qin, H., Cross, T.A. (2014) Membrane Protein Structural Validation by Oriented Sample Solid-State NMR: Diacylglycerol Kinase, Biophys. J., 106, 1559-1569

Nevzorov, A.A., Opella, S.J. (2003) A "Magic Sandwich" pulse sequence with reduced offset dependence for high-resolution separated local field spectroscopy, J. Magn. Reson., 164, 182-186

Nevzorov, A.A., Opella, S.J. (2007) Selective averaging for high-resolution solid-state NMR spectroscopy of aligned samples, J. Magn. Reson., 185, 59-70

Nevzorov, A.A. (2008) Mismatched Hartmann−Hahn conditions cause proton-mediated intermolecular magnetization transfer between dilute low-spin nuclei in NMR of static solids, J. Am. Chem. Soc., 130, 11282-11283

Opella, S.J., Zeri, A.C., Park, S.H. (2008) Structure, dynamics, and assembly of filamentous bacteriophages by nuclear magnetic resonance spectroscopy, Annu. Rev. Phys. Chem., 59, 635-657

Park, S.H., De Angelis, A.A., Nevzorov, A.A., Wu, C.H., Opella, S.J. (2006) Three-dimensional structure of the transmembrane domain of Vpu from HIV-1 in aligned phospholipid bicelles, Biophys. J., 91, 3032-3042

Park, S.H., Marassi, M.F., Black, D., Opella, S.J. (2010) Structure and Dynamics of the Membrane-Bound Form of Pf1 Coat Protein: Implications of Structural Rearrangement for Virus Assembly, Biophys. J., 99, 1465-1474

Park, S.H., Das, B.B., Casagrande, F., Tian, Y., Nothnagel, H.J., Chu, M.N., Kiefer, H., Maier, K., De Angelis, A.A., Marassi, F.M., Opella, S.J. (2012) Structure of the chemokine receptor CXCR1 in phospholipid bilayers, Nature, 491, 779-783

Renault, M., Bos, M.P., Tommassen, J., Baldus, M. (2011) Solid-state NMR on a large multidomain integral membrane protein: the outer membrane protein assembly factor BamA, J. Am. Chem. Soc., 133, 4175-4177

Salnikov, E.S., Aisenbrev, C., Aussenac, F., Quari, O., Sarroui, H., Reiter, C., Tordo, P., Engelke, F., Bechinger, B. (2016) Membrane topologies of the PGLa antimicrobial peptide and a transmembrane anchor sequence by Dynamic Nuclear Polarization/solid-state NMR spectroscopy, Sci. Rep., 6, 20895

Sinha, N., Grant, C.V., Park, S.H., Brown, J.M., Opella, S.J. (2007) Triple resonance experiments for aligned sample solid-state NMR of ^{13}C and ^{15}N labeled proteins, J. Magn. Reson., 186, 51-64

Sharma, M., Yi, M.G., Dong, H., Qin, H.J., Peterson, E., Busath, D.D., Zhou, H.X., Cross, T.A. (2010) Insight into the Mechanism of the Influenza A Proton Channel from a Structure in a Lipid Bilayer, Science, 330, 509-512

Shi, L.C., Lake, E.M.R., Ahmed, M.A.M., Brown, L.S., Ladizhansky, V. (2009) Solid-state NMR study of proteorhodopsin in the lipid environment: Secondary structure and dynamics, Biochim. Biophys. Acta Biomembr., 1788, 2563-2574

Shmygelska, A., Levitt, M. (2009) Generalized ensemble methods for de novo structure prediction, Proc. Natl. Acad. Sci. USA, 106, 1415-1420

Tang, W.X., Knox, R.W., Nevzorov, A.A. (2012) A spectroscopic assignment technique for membrane proteins reconstituted in magnetically aligned bicelles, J. Biomol. NMR, 54, 307-316

Tolman, J.R., Flanagan, J.M., Kennedy, M.A., Prestegard, J.H. (1995) Nuclear magnetic dipole interactions in field-oriented proteins: information for structure determination in solution, Proc. Natl. Acad. Sci. USA, 92, 9279-9283

Traaseth, N.J., Shi, L., Verardi, R., Mullen, D.G., Barany, G., Veglia, G. (2009) Structure and topology of monomeric phospholamban in lipid membranes determined by a hybrid solution and solid-state NMR approach, Proc. Natl. Acad. Sci. USA, 106, 10165-10170

Vosegaard, T., Nielsen, N.C. (2002) Towards high-resolution solid-state NMR on large uniformly ^{15}N- and [^{13}C,^{15}N]-labeled membrane proteins in oriented lipid bilayers, J. Biomol. NMR, 22, 225-247

Walther, T.H., Grage, S.L., Roth, N., Ulrich, A.S. (2010) Membrane Alignment of the Pore-Forming Component $TatA_d$ of the Twin-Arginine Translocase from Bacillus subtilis Resolved by Solid-State NMR Spectroscopy, J. Am. Chem. Soc., 132, 15945-15956

Wang, S.L., Munro, R.A., Shi, L.C., Kawamura, I., Okitsu, T., Wada, A., Kim, S.Y., Jung, K.H., Brown, L.S., Ladizhansky, V. (2013) Solid-state NMR spectroscopy structure determination of a lipid-embedded heptahelical membrane protein, Nat. Methods, 10, 1007-1012

White, S.H., Wimley, W.C. (1999) Membrane protein folding and stability: physical principles, Annu. Rev. Biophys. Biomol. Struct. 28, 319-365

Wu, C.H., Ramamoorthy, A., Opella, S.J. (1994) Three-dimensional solid-state NMR experiment that correlates the chemical shift and dipolar coupling frequencies of two heteronuclei, J. Magn. Reason. Ser. A, 109, 270-272

Yamamoto, K., Gildenberg, M. Ahuja, S., Im, S.C., Pearcy, P., Waskell, L., Ramamoorthy, A. (2013) Probing the transmembrane structure and topology of microsomal cytochrome-p450 by solid-state NMR on temperature-resistant bicelles, Sci. Rep., 3, 2556

Yarov-Yarovoy, V., Schonbrun, J., Baker, D. (2006) Multipass membrane protein structure prediction using Rosetta, Proteins Struct. Funct. Bioinf., 62, 1010-1025

Yin, Y.Y., Nevzorov, A.A. (2011) Structure determination in "shiftless" solid state NMR of oriented protein samples, J. Magn. Reson., 212, 64-73

CHAPTER 6: CONCLUSIONS

We have developed an automated algorithm for the assignment of two sets of two dimensional ssNMR spectra of oriented membrane protein samples (Chapter 2). These include a homonuclear ^{15}N-^{15}N spin exchange experiment and a spin-exchange separated-local field experiment (PISEMA or SAMPI4X). The inverted spectral intensity was used as a pseudopotential energy landscape to quantitatively characterize the goodness of fit to a specific cross peak distribution and, consequently, possible assignment solutions. Despite a considerable size of the sampling space, the Monte-Carlo simulated annealing approach has proven to be effective in finding the lowest pseudoenergy solutions. The algorithm was successfully applied to a synthetic dataset and two sets of experimental NMR spectra of Pf1 coat protein reconstituted in magnetically aligned bicelles. While the fitting of the synthetic data and the relatively short transmembrane domain of Pf1 resulted in recovering the previous assignment, the highly overlapping peaks and the intense main peaks in the Pf1 phage spectra presented a challenge for the algorithm. In part, these challenges were successfully addressed by invoking main-peak editing and restraint criteria such as orphan and missed peak filtering together with the expected helical structure. It was shown that the previously reported assignment results in a much higher pseudoenergy exhibiting some missing/orphan cross peaks and an alternate possible reassignment was suggested.

While the current protocol only incorporates two spectroscopic dimensions for the fitting, i.e. ^{15}N chemical shifts and ^{1}H^{15}N dipolar couplings, it is fully extendable to including other spectroscopic dimensions such as ^{13}C-^{15}N or ^{1}H-^{13}C dipolar couplings when those become available. The incorporation of a third dimension could greatly decrease spectral overlap which could make misassignments less likely.

A critical analysis for the feasibility of *de novo* structure calculations using NMR angular restraints in uniaxially aligned samples has been presented (Chapter 3). The sequential-fitting structure calculation algorithm has been substantially improved to better handle experimental uncertainty. By doubling the number of ϕ/ψ torsion angles to be fitted at once, the output structural solutions have been more effectively funneled toward lower RMSD structures. The post-fitting filtering was made possible by using several custom Rosetta scoring functions including statistical and physical terms that correlate well with low structural RMSDs relative to the original protein structures. To test the robustness of the structure determination protocol, synthetic NMR

datasets derived from two proteins of different topology were employed. A set of 1000 calculated solutions has been reduced down to 10 acceptable solutions, of which the majority of structural RMSD scores were less than 2Å. Thus, without knowing the true structure *a priori*, it is possible to first calculate a large set of the structural solutions consistent with the NMR angular restraints and then filter them using the Rosetta scoring functions to obtain the most plausible structure(s).

Experimental error randomly added to the synthetic spectral resonances can adequately account for the deviations from constant CSA tensor orientation. Presumably, adding such random error can also encompass non-ideal peptide plane geometry, variability in the CSA tensor principal values, and insufficient spectral resolution. About 50 Hz of experimental error (or 1 ppm at 50 MHz ^{15}N NMR frequency) likely represents the maximum degree of uncertainty that could be tolerated in three-dimensional experiments to calculate structures reliably by using the method of ssNMR angular restraints and Rosetta-based filtering. For both protein structures considered, the maximum 50 Hz uncertainty yielded less than 3% of the 1000 total solutions that had a structural RMSD < 2Å. Greater experimental uncertainty in angular restraints and missing data would yield a highly diminishing proportion of acceptable solutions, which may necessitate adding a fourth dimension to the NMR spectrum, such as ^{13}C-^{15}N dipolar couplings to increase the likelihood of obtaining correct protein folds. Finally, the presented framework for de-novo structure determination of protein structures from NMR angular restraints calls for the necessity of the development of pulse sequences correlating ^{15}N spins with $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ and ^{13}C-^{15}N dipolar couplings for macroscopically aligned membrane proteins.

In Chapter 4, GPU-aided NMR simulations were utilized for the optimization of pulse sequences for evolving and narrowing of the ^{1}H-^{15}N dipolar couplings. This involved running exhaustive 8-spin spectral simulations with varying pulse sequence parameters and scoring the resulting spectra based on peak intensities and their relative dipolar couplings. The search for optimal parameters was implemented via a Monte-Carlo protocol, similar to the one utilized in Chapter 2 for automated assignment. An objective scoring function was devised in order to select for the most intense spectra (yielding narrowest linewidths) in which the measured signals are consistent with the local dipolar couplings of interest present in the spin system. Simulations consistently yielded pulse sequences having up to 50% more intense resonances than the existing SAMPI4 pulse sequence, which resulted in sharper linewidths. After optimizations in silica, the pulse sequences were successfully tested on an NAL crystal on a 300 MHz ^{1}H NMR frequency

spectrometer and the results were compared to SAMPI4. In a new pulse sequence, termed "ROULETTE-1", linewidths in the ^{1}H-^{15}N dimension were seen to be on average 18% sharper than those for SAMPI4. A biological sample of ^{15}N Leucine labeled Pf1 coat protein reconstituted in magnetically aligned bicelles was measured on a 500 MHz ^{1}H NMR frequency spectrometer. For this frequency and experimental sample conditions, ROULETTE-1 was re-optimized, which resulted in linewidths 14% sharper than for SAMPI4. This method of pulse sequence optimization is fully extendable to NMR experiments for other nuclei and dipolar couplings, such as the $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ couplings.

In Chapter 5, we have demonstrated the general applicability of our structure calculation program for experimental triple-resonance NMR spectra involving the measurements of the chiral $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings. The development of new pulse sequences to elucidate $^{1}H_{\alpha}$-$^{13}C_{\alpha}$ dipolar couplings along a protein backbone, in conjunction with Rosetta bioinformatics, allows for *de novo* structure determination of aligned membrane proteins, as was previously demonstrated in Chapter 3 on synthetic spectra. The developed structure calculation and filtering protocol was applied to triple-resonance data for Pf1 coat protein spectra reconstituted in magnetically aligned bicelles. This procedure has resulted in 10 consensus *de-novo* structures, which are in agreement with the previously assumed kinked alpha helix for the transmembrane domain of Pf1.

In summary, the presented computational tools greatly advance ssNMR of oriented samples as a powerful analytical technique, which allows one to obtain quantitative structural and dynamic information about membrane proteins in their native-like bilayer environments.

APPENDICES

Appendix A: Program for automated assignment algorithm on synthetically generated spectra

```
/*
The program as written is set to simulate a 2D spectra for 50 residues (NN and NH). There are
currently 5 selectively labeled groups, 6 groups total.
It should be ready to run as long as you have a gcc compiler. To compile via command line,
navigate to the directory that the code is in and type the following:

    gcc synthetic11.c -O3 -o run;

This creates an executable called "run.exe". To run that executable type:

    ./run.exe

You will see standard output reporting information from the MCSA run. Currently results are not
printed to file. If you want to see a plot of the final result
set the Global variable named "file" to 1, and then set three output paths starting after line
1038. This creates csv files that plotsynth.py can read in. You
will have to set the paths in that python script as well so that it can find the output files
from this program.

Commenting is generous throughout the program, so that it is as user friendly as possible. Any
questions can be directed to Joel Lapin, email jlapin@ncsu.edu.
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

////////////////////////////
///Global Variables////////
////////////////////////////
int
    amt = 2,                            // Amount of spectra to use in experiment. Can be 1 or 2
    len = 50,                           // How many residues are in the protein
    groups = 6,                         // Number of selectively labeled groups + 1
    groupings[] = {30,4,4,4,4,4},       // How many in each group. If the "groups" variable has x
numbers, there MUST be x numbers in this array. The numbers must sum to "len"
    sz = 1000,                          // Points per dimension on the potential grid, i.e. how
fine the grid is
    rnd = 1,                            // 1: Randomized peaks. 0: Read in peaks from a .csv file
    runs = 1,                           // How many runs you want to do. Could be prone to memory
leaks, best to leave this at 1
    steps = 1E4,                        // Number of random steps per W
    orphan = 1,                         // Flag for orphan peak filtering (1: on, 0: off); see
cut1, cut2, tolerance1, and tolerance2 variables
    tolerance1 = 0,                     // Number of orphan peaks to tolerate on spectrum 1
    tolerance2 = 0,                     // Number of orphan peaks to tolerate on spectrum 2
    refine = 0,                         // Raising the temperature at the end of a run to refine
the final state. Set to any integer values. Prolongs the runtime
    outp = 1,                           // 1 for extended (std)output, 0 for final results only
    file = 0,                           // 1 to print to file; 0 for stdout only
    movie = 0,                          // Printing the final state of each temperature to file
for visualization in a movie (1=on, 0=off). This feature is still under developement
    top = 100;                          // Maximum number of top scores to save
double
    Wmin = 1E-8,                        // Starting temperature. Usually best to keep it
negligibly small
    Wmax = 5E0,                         // Final W temperature target. The actual final value
will be: final_frac*Wmax
    pen = 0.2,                          // Penalty to the score for an orphan crosspeak.
Currently commented out in program. Best to leave this alone
    no_incr = 1E3,                      // Number of temperature increments
    alpha = 0.85,                       // Logarithmic factor in cooling schedule. Larger value
means slower cooling/longer tail, smaller value means faster cooling
    final_frac = 0.99,                  // Final fraction, W/Wmax, to run to completion
    cut1 = -0.1,                        // Cutoff intensity for orphan peaks on spectrum 1 (above
which peaks are designated orphans)
```

```
    cut2  = -0.1,                       // Cutoff intensity for orphan peaks on spectrum 2 (above
which peaks are designated orphans)

    // Gaussian parameters for spectral peaks
    mamp1 = 0.0,                        // Main peak amplitude on top spectrum (N-N if you are
running a 2D experiment
    mamp2 = 0.0,                        // Main peak amplitude on bottom spectrum (N-HN if you
are running a 2d experiment
    msigx1 = 1.0,                       // Main peak sigma in x dimension for spectrum 1
    msigy1 = 1.0,                       // Main peak sigma in y dimension for spectrum 1
    msigx2 = 1.0,                       // Main peak sigma in x dimension for spectrum 2
    msigy2 = 0.1,                       // Main peak sigma in y dimension for spectrum 2

    //Cross peaks
    camp1 = 0.25,                       // Cross peak amplitude in spectrum 1
    camp2 = 0.25,                       // Cross peak amplitude in spectrum 2
    csigx1 = 2.0,                       // Cross peak sigma in x dimension for spectrum 1
    csigy1 = 2.0,                       // Cross peak sigma in y dimension for spectrum 1
    csigx2 = 1.568955,                  // Cross peak sigma in x dimension for spectrum 2
    csigy2 = 0.274058,                  // Cross peak sigma in y dimension for spectrum 2

    **E,                                // Energy grid for spectrum 1
    **E2,                               // Energy grid for spectrum 2
    * rE_real,                         // Energies of individual crosspeaks from spectrum 1 for
real order
    * rE1_low,                         // Energies of individual crosspeaks from spectrum 1 for
lowest total energy order
    * rE2_real,                        // Energies of individual crosspeaks from spectrum 2 for
real order
    * rE2_low,                         // Energies of individual crosspeaks from spectrum 2 for
lowest total energy order
    E1_real,                            // Energy from spectrum 1 for real order
    E2_real,                            // Energy from spectrum 2 for real order
    Etot_real,                          // Total energy for real order
    E1_hold,                            // Energy from spectrum 1 for current order
    E1_new,                             // Energy from spectrum 1 for new order
    E2_hold,                            // Energy from spectrum 2 for current order
    E2_new,                             // Energy from spectrum 2 for new order
    Etot_hold,                          // Total energy for current order
    Etot_new,                           // Total energy for new order
    E1_low,                             // Energy from spectrum 1 for lowest energy order
    E2_low,                             // Energy from spectrum 2 for lowest energy order
    Etot_low,                           // Lowest measured total energy

    //First order variables: Saved for generating figure in publication
    E1_first,
    E2_first,
    Etot_first,

    elap,
    W,

    cs_min = 120,                       // Minimum desired chemcial shift - 10
    cs_max = 230,                       // Maximum desired chemical shift + 10
    dc_min = 3,                         // Minimum desired dipolar coupling - 1
    dc_max = 11;                        // Maximum desired dipolar coupling + 1

int i,j,k,l,m,n,p,
    accept_l,
    accept_h,
    reject,
    rtick = 0,
    tick = 0,
    ftick = 0,
    flor,
    ceel;

typedef struct // This struct holds important environment variables/parameters for the run
{
    int
        len,
```

```
        szr,
        szc;
    double
        **E,
        **pk,
        rmin,
        rmax,
        cmin,
        cmax;

} envobj;
typedef struct // This struct holds important parameter variables for generating a synthetic
spectrum
{
    double
        mamp,
        msigx,
        msigy,
        camp,
        csigx,
        csigy;
} parobj;
clock_t start_program, end_program;
////////////////////////////////////
///End global variables/////////////
////////////////////////////////////


//////////////////////////////
/// Functions/subroutines ///
//////////////////////////////
void xpeaks(double ** in, double **out)
{
    // Generate crosspeaks
    int x,y,z;
    double holder;

    for(x=0;x<(len-1);x++)
    {
        out[0][x] = in[0][x];
        out[1][x] = in[1][x+1];
        out[0][x+len-1] = in[0][x+1];
        out[1][x+len-1] = in[1][x];
    }
}
double* linspace(double lower, double upper, int points)
{
     // Create a vector from "lower" to "upper" with "points" number of points
    // Must deallocate memory if this function is called many times
    int x,y,z;
    double
        spacing,
        * output = (double*)malloc(points*sizeof(double));

    spacing = (upper-lower) / (double)(points-1);

    for(x=0;x<points;x++)
        output[x] = lower + (spacing*x);

    return output;

}
void eng(envobj ob, double **main, double **cross, parobj pob)
{
    // Generate synthetic spectrum with gaussians
    int x,y,z;
    double
        * xaxis = linspace(ob.cmax,ob.cmin,ob.szc),
        * yaxis = linspace(ob.rmax,ob.rmin,ob.szr);

    for(x=0;x<2*(ob.len-1);x++)
```

```
        for(y=0;y<ob.szr;y++)
            for(z=0;z<ob.szc;z++)
            {
                if(x<ob.len)
                    ob.E[y][z] -= pob.mamp * exp(-1*pow(((yaxis[y]-
main[0][x])/(pow(2,0.5)*pob.msigy)),2.0)) * exp(-1*pow(((xaxis[z]-
main[1][x])/(pow(2,0.5)*pob.msigx)),2.0));
                ob.E[y][z] -= pob.camp * exp(-1*pow(((yaxis[y]-
cross[0][x])/(pow(2,0.5)*pob.csigy)),2.0)) * exp(-1*pow(((xaxis[z]-
cross[1][x])/(pow(2,0.5)*pob.csigx)),2.0));
            }
    free(xaxis);
    free(yaxis);
}
double eng2(envobj ob, double **cross, double * rE)
{
    // Measure pseudoenergy, the slow way. Use to seed the algorithm
    int x,y,z;
    double xind, yind, E_tot, xspacing, yspacing;
    xspacing = (ob.cmax-ob.cmin) / (double)(ob.szc-1);
    yspacing = (ob.rmax-ob.rmin) / (double)(ob.szr-1);
    E_tot = 0;
    for(x=0;x<2*(ob.len-1);x++)
    {
        yind = round( ((ob.rmax-cross[0][x])/yspacing) );//Different than real
        xind = round( ((ob.cmax-cross[1][x])/xspacing) );
        // Octave code is E(y,x)
        rE[x] = ob.E[(int)yind][(int)xind];
        E_tot += rE[x];
    }
    return E_tot;
}
void xnumbers(int length, int * inds, int * num, int * xnums)
{
    /*
    This function, based on the index swap, will find the number of crosspeak indices that will
change their crosspeak (num),
    and their crosspeak indices (xnums). The output can be used as input to eng3.
    */
    //7 Scenarios
    if(inds[0]==0)//First Left end
    {
        if(inds[1]==1)//Second Adjacent
        {
            *num = 4;
            xnums[0]=0;xnums[1]=1;xnums[2]=0+length-1;xnums[3]=1+length-1;
        }
        else if(inds[1]==(length-1))//Second Right end
        {
            *num=4;
            xnums[0]=0;xnums[1]=length-2;xnums[2]=0+length-1;xnums[3]=(2*length)-3;
        }
        else//Second Middle
        {
            *num=6;
            xnums[0]=0;xnums[1]=inds[1]-1;xnums[2]=inds[1];xnums[3]=0+length-
1;xnums[4]=inds[1]+length-2;xnums[5]=inds[1]+length-1;
        }
    }
    else if(inds[0]==(length-2))//Adjacent to right end
    {
        *num=4;
        xnums[0]=length-3;xnums[1]=length-2;xnums[2]=(2*length)-4;xnums[3]=(2*length)-3;
    }
    else//First Middle
    {
        if(abs(inds[0]-inds[1])==1)//Second Adjacent
        {
            *num=6;
            xnums[0]=inds[0]-1;xnums[1]=inds[0];xnums[2]=inds[1];xnums[3]=inds[0]+length-
2;xnums[4]=inds[0]+length-1;xnums[5]=inds[1]+length-1;
```

```
        }
        else if(inds[1]==(length-1))//Second Right end
        {
            *num=6;
            xnums[0]=inds[0]-1;xnums[1]=inds[0];xnums[2]=inds[1]-1;xnums[3]=inds[0]+length-
2;xnums[4]=inds[0]+length-1;xnums[5]=inds[1]+length-2;
        }
        else//Both in the middle and not adjacent
        {
            *num=8;
            xnums[0]=inds[0]-1;xnums[1]=inds[0];xnums[2]=inds[1]-
1;xnums[3]=inds[1];xnums[4]=inds[0]+length-2;xnums[5]=inds[0]+length-1;xnums[6]=inds[1]+length-
2;xnums[7]=inds[1]+length-1;
        }
    }
}
void eng3(envobj ob, double **xpk, double * rEold, double * rEnew, double * Eold, double * Enew,
int num, int * xnums)
{
    // Speeds up energy calculation over eng2 by updating energy based on swap, instead of re-
calculating sum of all crosspeaks
    int x;
    double rind, cind;
    double rspac, cspac;

    rspac = (ob.rmax-ob.rmin) / (double)(ob.szr-1);
    cspac = (ob.cmax-ob.cmin) / (double)(ob.szc-1);

    //Must copy old crosspeak energies into new crosspeak energies (not too costly)
    for(x=0;x<(2*(ob.len-1));x++)
        rEnew[x] = rEold[x];

    *Enew = *Eold;
    for(x=0;x<num;x++)
    {
        rind = round( ((ob.rmax - xpk[0][xnums[x]])/rspac) );
        cind = round( ((ob.cmax - xpk[1][xnums[x]])/cspac) );

        *Enew -= rEold[xnums[x]];
        rEnew[xnums[x]] = 1.0*ob.E[(int)rind][(int)cind];*Enew += rEnew[xnums[x]];
    }
}
void swap_ind(double ** pks1old, double ** pks1new, double ** pks2old, double ** pks2new, int *
labels, int * inds)
{
    //Random step that swaps two indices. This is the preferred method for randomization
    int x, y, z;
    int
        indo = 10000,
        indn = 10000;
    double  hold;

    // Copy rpks into rpks2
    for(x=0;x<2;x++)
        for(y=0;y<len;y++)
        {
            pks1new[x][y] = pks1old[x][y];
            if(amt>1)
                pks2new[x][y] = pks2old[x][y];
        }

    while( (indo==indn) || (labels[indo]!=labels[indn]) )
    {
        indo = rand()%len;
        indn = rand()%len;
    }

    for(x=0;x<2;x++)
    {
        hold = pks1new[x][indo];
        pks1new[x][indo] = pks1new[x][indn];
```

```
            pks1new[x][indn] = hold;
            if(amt>1)
            {
                hold = pks2new[x][indo];
                pks2new[x][indo] = pks2new[x][indn];
                pks2new[x][indn] = hold;
            }
        }

        if(indo<indn)
        {
            inds[0] = indo;
            inds[1] = indn;
        }
        else
        {
            inds[0] = indn;
            inds[1] = indo;
        }
}
void rand_pks(double ** pks, double ** rpks, double ** rpks_hold, double ** pks2, double **
rpks2, double ** rpks2_hold)
{
    // Randomize peak order
    // Loops are the equivalent of Matlab's randperm()

    int
        ind,
        * used = (int*)malloc(len*sizeof(int)),
        x,y;

    for(x=0;x<len;x++)
    {
        ind = rand()%len;
        y=0;
        while(y<x)
        {
            if(ind == used[y])
            {
                ind = rand()%len;
                y=0;
            }
            else
                y++;
        }
        used[x] = ind;
        for(y=0;y<2;y++)
        {
            rpks[y][x] = pks[y][used[x]];
            rpks_hold[y][x] = rpks[y][x];
            if(amt>1)
            {
                rpks2[y][x] = pks2[y][used[x]];
                rpks2_hold[y][x] = rpks2[y][x];
            }
        }
    }
    free(used);
}
int* selectlab(envobj eob, int groups, int *groupings)
{
    // Grouping peaks by their selective labels
    int i,j,ind,
        * labels = (int*)calloc(eob.len,sizeof(int));
    for(i=0;i<groups;i++)
    {
        j=0;
        while(j<groupings[i])
        {
            ind = rand()%eob.len;
            if(labels[ind]==0)
```

```
            {
                labels[ind]=i+1;
                j++;
            }
        }
    }
    return labels;
}
int orfilt(envobj a, double * engs, double cut, int tol)
{
    // Orphan peak filtering which filters orders with "tol" number of crosspeaks above "cut"
intensity
    int i, tal;
    tal=0;
    for(i=0;i<2*(a.len-1);i++)
        if(engs[i]>cut)
        {
            tal++;
            if(tal>tol)
            {
                return 0;
                break;
            }
        }
    if(tal<=tol)
        return 1;
    else
        return 0;
}
//////////////////////////////
///End of functions/////////
//////////////////////////////



int main()
{
    // Timing
    start_program = clock();
    // Random number seeding
    srand(time(NULL));

    ////////////////////////////////////////////////////////////////////////////////
    ///Create a synthetic spectrum//////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////

    double endp = log(1-final_frac)/log(alpha);
    double
        * incr = linspace(Wmin, endp, no_incr),
        ** pks = (double**)malloc(2*sizeof(double*)),
        ** pks2 = (double**)malloc(len*sizeof(double*)),
        ** xpks = (double**)malloc(2*sizeof(double*)),
        ** xpks2 = (double**)malloc(2*sizeof(double*));

    for(i=0;i<2;i++)
    {
        pks[i] = (double*)malloc(len*sizeof(double));
        pks2[i] = (double*)malloc(len*sizeof(double));
        xpks[i] = (double*)malloc(2*(len-1)*sizeof(double));
        xpks2[i] = (double*)malloc(2*(len-1)*sizeof(double));
    }
    rE_real = (double*)malloc(2*(len-1)*sizeof(double));
    rE2_real = (double*)malloc(2*(len-1)*sizeof(double));

    envobj
        eob1,
        eob2;
    eob1.len = len;
    eob1.rmin = cs_min;//Change if you want to only run heteronuclear
    eob1.rmax = cs_max;
    eob1.szr = sz;
```

```
    eob1.cmin = cs_min;
    eob1.cmax = cs_max;
    eob1.szc = sz;
    if(amt>1)
    {
        eob2.len = len;
        eob2.rmin = dc_min;
        eob2.rmax = dc_max;
        eob2.szr = sz;
        eob2.cmin = cs_min;
        eob2.cmax = cs_max;
        eob2.szc = sz;
    }
    parobj
        pob1,
        pob2;
    pob1.mamp = mamp1;
    pob1.msigx = msigx1;
    pob1.msigy = msigy1;
    pob1.camp = camp1;
    pob1.csigx = csigx1;
    pob1.csigy = csigy1;
    if(amt>1)
    {
        pob2.mamp = mamp2;
        pob2.msigx = msigx2;
        pob2.msigy = msigy2;
        pob2.camp = camp2;
        pob2.csigx = csigx2;
        pob2.csigy = csigy2;
    }

    // Peak centers are generated here
    if(rnd==1)
    {
        i=0;
        while(i<eob1.len)
        {
            pks[0][i] = eob1.cmin + 10 + (double)(rand()%((int)eob1.cmax-(int)eob1.cmin-20)) +
((double)rand()/(double)RAND_MAX);
            pks[1][i] = pks[0][i];
            if(amt>1)
            {
                pks2[0][i] = eob2.rmin + 1 + (double)(rand()%((int)eob2.rmax-(int)eob2.rmin-2)) +
((double)rand()/(double)RAND_MAX);
                pks2[1][i] = pks[1][i];
            }
            i++;
        }
    }
    else // Read in peaks from csv file
    {
        FILE * pk = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project\\Output\\pkinput.csv","r");
        for(i=0;i<2;i++)
        {
            for(j=0;j<len;j++)
                fscanf(pk,"%lf,",&pks[i][j]);
            fscanf(pk,"\n");
        }
        if(amt>1)
        {
            fscanf(pk,"\n");
            for(i=0;i<2;i++)
            {
                for(j=0;j<len;j++)
                    fscanf(pk,"%lf,",&pks2[i][j]);
                fscanf(pk,"\n");
            }
        }
        fclose(pk);
```

```
}

// Selective Labeling
int * labels = selectlab(eob1, groups, groupings);

// Crosspeaks
xpeaks(pks,xpks);
if(amt>1)
    xpeaks(pks2,xpks2);

// Allocate for Energy grid
eob1.E = (double**)malloc(eob1.szr*sizeof(double*));
if(amt>1)
    eob2.E = (double**)malloc(eob2.szr*sizeof(double*));
for(i=0;i<eob1.szr;i++)
{
    eob1.E[i] = (double*)calloc(eob1.szc,sizeof(double));
    if(amt>1)
        eob2.E[i] = (double*)calloc(eob2.szc,sizeof(double));
}

// Energy grid
eng(eob1,pks, xpks, pob1);
if(amt>1)
    eng(eob2, pks2, xpks2, pob2);

//Editing Energy grid
/*
for(i=0;i<sz;i++)
{
    for(j=0;j<sz;j++)
    {
        if(E[i][j] > -0.00001)
            E[i][j] = pen;
        if(amt>1)
            if(E2[i][j] > -0.00001)
                E2[i][j] = pen;
    }
}
*/
// Initial score
E1_real = eng2(eob1, xpks, rE_real);
if(amt==1)
    Etot_real = E1_real;
else
{
    E2_real = eng2(eob2, xpks2, rE2_real);
    Etot_real = E1_real + E2_real;
}

////////////////////////////////////////////////////////////////////////////////
///BREAK////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////
/// Randomize the synthetic spectrum and try to recover the original peak order ///
////////////////////////////////////////////////////////////////////////////////

int
    num,
    * xnums = (int*)malloc(8*sizeof(int)),
    * inds = (int*)malloc(2*sizeof(int));
double
    ** rpks = (double**)malloc(2*sizeof(double*)),
    ** rpks_hold = (double**)malloc(2*sizeof(double*)),
    ** rpks2 = (double**)malloc(2*sizeof(double*)),
    ** rpks2_hold = (double**)malloc(2*sizeof(double*)),
    ** rxpks = (double**)malloc(2*sizeof(double*)),
    ** rxpks2 = (double**)malloc(2*sizeof(double*)),
```

```
    * rE_new = (double*)malloc(2*(len-1)*sizeof(double)),
    * rE_hold = (double*)malloc(2*(len-1)*sizeof(double)),
    * rE_low = (double*)malloc(2*(len-1)*sizeof(double)),
    * rE2_new = (double*)malloc(2*(len-1)*sizeof(double)),
    * rE2_hold = (double*)malloc(2*(len-1)*sizeof(double)),
    cur,
    acc_crit,
    dE,

    //First order
    ** pks1_first = (double**)malloc(2*sizeof(double*)),
    ** pks2_first = (double**)malloc(2*sizeof(double*)),
    * rE1_first = (double*)malloc(2*(len-1)*sizeof(double)),
    * rE2_first = (double*)malloc(2*(len-1)*sizeof(double));

for(i=0;i<2;i++)
{
    rpks[i] = (double*)calloc(len,sizeof(double));
    rpks_hold[i] = (double*)calloc(len,sizeof(double));
    rxpks[i] = (double*)malloc(2*(len-1)*sizeof(double));
    //First order
    pks1_first[i] = (double*)malloc(len*sizeof(double));

    if(amt>1)
    {
        rpks2[i] = (double*)calloc(len,sizeof(double));
        rpks2_hold[i] = (double*)calloc(len,sizeof(double));
        rxpks2[i] = (double*)malloc(2*(len-1)*sizeof(double));
        //First order
        pks2_first[i] = (double*)malloc(len*sizeof(double));
    }
}

//Saving variables
double
    ** E_save = (double**)malloc(top*sizeof(double*)),
    ** E_save_all = (double**)malloc(no_incr*sizeof(double*)),
    *** pks1_save = (double***)malloc(top*sizeof(double**)),
    *** pks2_save = (double***)malloc(top*sizeof(double**)),
    diff,
    diff2,

    //Movie
    **** seq = (double****)malloc(no_incr*sizeof(double***));

for(i=0;i<top;i++)
{
    E_save[i] = (double*)calloc(3,sizeof(double));
    pks1_save[i] = (double**)malloc(2*sizeof(double*));
    pks2_save[i] = (double**)malloc(2*sizeof(double*));
    for(j=0;j<2;j++)
    {
        pks1_save[i][j] = (double*)malloc(len*sizeof(double));
        pks2_save[i][j] = (double*)malloc(len*sizeof(double));
    }
}
for(i=0;i<no_incr;i++)
{
    E_save_all[i] = (double*)calloc(4,sizeof(double));
    if(movie==1)
    {
        seq[i] = (double***)malloc(2*sizeof(double**));
        for(j=0;j<2;j++)
        {
            seq[i][j] = (double**)malloc(2*sizeof(double*));
            for(k=0;k<2;k++)
                seq[i][j][k] = (double*)calloc(len,sizeof(double));
        }
    }
}
```

```
// Copy "real" peak order into variables to be randomized
for(k=0;k<runs;k++)
{
    for(i=0;i<2;i++)
        for(j=0;j<eob1.len;j++)
        {
            rpks[i][j] = pks[i][j];
            rpks_hold[i][j] = pks[i][j];
            if(amt>1)
            {
                rpks2[i][j] = pks2[i][j];
                rpks2_hold[i][j] = pks2[i][j];
            }
        }

    // Randomize peak order with 10,000 random steps
    for(i=0;i<10000;i++)
        swap_ind(rpks_hold, rpks_hold, rpks2_hold, rpks2_hold, labels, inds);

    // Generate crosspeaks
    xpeaks(rpks_hold, rxpks);
    if(amt>1)
        xpeaks(rpks2_hold, rxpks2);

    // Calculate pseudopotential of crosspeaks on E surface
    E1_hold = eng2(eob1, rxpks, rE_hold);
    if(amt==1)
        Etot_hold = E1_hold;
    else
    {
        E2_hold = eng2(eob2, rxpks2, rE2_hold);
        Etot_hold = E1_hold + E2_hold;
    }

    //First order
    E1_first = E1_hold;
    E2_first = E2_hold;
    Etot_first = Etot_hold;
    for(i=0;i<2;i++)
        for(j=0;j<2*(len-1);j++)
        {
            if(j<len)
            {
                pks1_first[i][j] = rpks_hold[i][j];
                if(amt>1)
                    pks2_first[i][j] = rpks2_hold[i][j];
            }
            if(i==0)
            {
                rE1_first[j] = rE_hold[j];
                if(amt>1)
                    rE2_first[j] = rE2_hold[j];
            }
        }

    W = Wmin;
    for(l=0;l<no_incr;l++)// Temperature loop
    {
        accept_h = 0;
        accept_l = 0;
        reject = 0;

        cur = 100*((double)l/(double)no_incr);

        if(outp==1)
        {
            if(runs>1)
                printf("Run: %d\n",k+1);
            if(refine>0)
                printf("Refine #: %d\n",tick);
            printf("W = %.3lf (%.0lf%% complete)\n",W,cur);
```

```
}

for(i=0;i<steps;i++)
{
    rtick = 0;
    ftick = 0;

    // New peak order
    //  - rpks will be the new try
    swap_ind(rpks_hold, rpks, rpks2_hold, rpks2, labels, inds);

    // Find crosspeak indices whose energy will be changed
    xnumbers(len, inds, &num, xnums);

    // Generate crosspeaks
    xpeaks(rpks,rxpks);
    if(amt>1)
        xpeaks(rpks2, rxpks2);

    // Compute pseudopotential
    eng3(eob1, rxpks, rE_hold, rE_new, &E1_hold, &E1_new, num, xnums);
    //printf("%f",E1_new);exit(0);
    if(amt==1)
        Etot_new = E1_new;
    else
    {
        eng3(eob2, rxpks2, rE2_hold, rE2_new, &E2_hold, &E2_new, num, xnums);
        Etot_new = E1_new + E2_new;
    }

    if( Etot_new < Etot_hold ) // Energy went down
    {
        E1_hold = E1_new;
        if(amt>1)
            E2_hold = E2_new;
        Etot_hold = Etot_new;
        for(n=0;n<2;n++)
        {
            for(j=0;j<(2*(len-1));j++)
            {
                rE_hold[j] = rE_new[j];
                if(amt>1)
                    rE2_hold[j] = rE2_new[j];
                if(j<len)
                {
                    rpks_hold[n][j] = rpks[n][j];
                    if(amt>1)
                        rpks2_hold[n][j] = rpks2[n][j];
                }
            }
        }
        accept_l++;
    }
    else if(Etot_new==Etot_hold)
    {
        acc_crit = (double)rand() / (double)RAND_MAX;
        if(acc_crit>0.5)
        {
            E1_hold = E1_new;
            if(amt>1)
                E2_hold = E2_new;
            Etot_hold = Etot_new;
            for(n=0;n<2;n++)
            {
                for(j=0;j<(2*(len-1));j++)
                {
                    rE_hold[j] = rE_new[j];
                    if(amt>1)
                        rE2_hold[j] = rE2_new[j];
                    if(j<len)
                    {
```

```
                                rpks_hold[n][j] = rpks[n][j];
                                if(amt>1)
                                    rpks2_hold[n][j] = rpks2[n][j];
                            }
                        }
                    }
                    //Following if/else statement is only so stdout numbers are roughly equal
                    acc_crit = (double)rand() / (double)RAND_MAX;
                    if(acc_crit>0.5)
                        accept_h++;
                    else
                        accept_l++;
                }
                else
                {
                    reject++;
                    rtick = 1;
                }
            }
            else // Energy went up  --> apply Metropolis criterion
            {
                acc_crit = (double)rand() / (double)RAND_MAX;
                dE = Etot_new - Etot_hold;
                if( exp(-1*dE*W) > acc_crit )
                {
                    E1_hold = E1_new;
                    if(amt>1)
                        E2_hold = E2_new;
                    Etot_hold = Etot_new;
                    for(n=0;n<2;n++)
                    {
                        for(j=0;j<(2*(len-1));j++)
                        {
                            rE_hold[j] = rE_new[j];
                            if(amt>1)
                                rE2_hold[j] = rE2_new[j];
                            if(j<len)
                            {
                                rpks_hold[n][j] = rpks[n][j];
                                if(amt>1)
                                    rpks2_hold[n][j] = rpks2[n][j];
                            }
                        }
                    }
                    accept_h++;
                }
                else
                {
                    reject++;
                    rtick = 1;
                }
            }

            // Filter for orphan peaks
            if(orphan==1)
            {
                if(amt>1)
                {
                    if( (orfilt(eob1, rE_new, cut1, tolerance1)==1) && (orfilt(eob2, rE2_new,
cut2, tolerance2)==1) )
                        ftick=1;
                }
                else
                {
                    if(orfilt(eob1,rE_new, cut1, tolerance1)==1)
                        ftick=1;
                }
            }


            // Sort Energy saving variables
```

```
                //  - Issue: same/equal numbers are not registered by the == logical operator
                //      - The result of that issue caused reverse orders to be saved adjacent
                //      - Used a workaround by calculating differences between E_hold and
E_save[m]
                diff = 1;
                diff2 = 1;
                m=-1;
                j=round((top-1)/2);
                flor = 0;
                ceel = top-1;
                if( (Etot_new<E_save[top-1][0]) && (ftick==1) )
                {
                    if(j==0)
                        m=j;
                    while( (j>0) && (j<(top-1)) )//(E_hold<=E_save[j]) && (j>-1) && (rtick==0) )
                    {
                        if(Etot_new<E_save[j][0])
                        {
                            diff = pow(pow(Etot_new-E_save[j][0],2.0),0.5); //If an "equal"
number sneaks in the backdoor
                            diff2 = pow(pow(Etot_new-E_save[j-1][0],2.0),0.5);
                            if( (Etot_new==E_save[j-1][0]) || (diff<1E-6) || (diff2<1E-6))
                            {
                                m=-1;
                                break;
                            }
                            if( Etot_new>E_save[j-1][0] )
                            {
                                m=j;
                                break;
                            }
                            ceel = j;
                            j = round((j+flor)/2);
                            if(j==0)
                                m=j;
                        }
                        else if(Etot_new>E_save[j][0])
                        {
                            diff = pow(pow(Etot_new-E_save[j][0],2.0),0.5); //If an "equal"
number sneaks in the backdoor
                            diff2 = pow(pow(Etot_new-E_save[j+1][0],2.0),0.5);
                            if( (Etot_new==E_save[j][0]) || (diff<1E-6) || (diff2<1E-6) )
                            {
                                m=-1;
                                break;
                            }
                            if( Etot_new<E_save[j+1][0] )
                            {
                                m=j+1;
                                break;
                            }
                            flor = j;
                            j = round((j+ceel)/2);
                        }
                        else
                        {
                            m=-1;
                            break;
                        }
                    }
                }
                if(m!=-1)
                {
                    for(n=(top-1);n>m;n--) // Move everything at that index downstream 1 spot
                    {
                        for(j=0;j<2;j++)
                        {
                            E_save[n][j] = E_save[n-1][j];
                            if(j<2)
                                for(p=0;p<len;p++)
                                {
```

```
                                    pks1_save[n][j][p] = pks1_save[n-1][j][p];
                                    if(amt>1)
                                        pks2_save[n][j][p] = pks2_save[n-1][j][p];
                                }
                            }
                        }
                        // Now insert the new value at index m
                        E_save[m][0] = Etot_new;
                        E_save[m][1] = E1_new;
                        if(amt>1)
                            E_save[m][2] = E2_new;
                        for(j=0;j<2;j++)
                        {
                            for(n=0;n<len;n++)
                            {
                                pks1_save[m][j][n] = rpks[j][n];
                                if(amt>1)
                                    pks2_save[m][j][n] = rpks2[j][n];
                            }
                        }
                    }
                    // End saving energies/peaks
                }

                // Save all pks_hold at end of each increment
                if(movie==1)
                {
                    for(i=0;i<2;i++)
                        for(j=0;j<len;j++)
                        {
                            seq[l][0][i][j] = rpks_hold[i][j];
                            seq[l][1][i][j] = rpks2_hold[i][j];
                        }
                }

                if(outp==1)
                {
                    printf("\n\n");
                    printf("    Steps accepted (E up)  : %7d\n",accept_h);
                    printf("    Steps accepted (E down): %7d\n",accept_l);
                    printf("    Steps rejected         : %7d\n",reject);
                    printf("    Current E: %11lf\n",Etot_hold);
                    printf("    Low E    : %11lf\n",E_save[0][0]);
                    printf("    Real E   : %11lf\n",Etot_real);
                    if(fabs(Etot_real-E_save[0][0])<1E-6)
                        printf("    DING!DING!DING!DING!DING!\n");
                    printf("\n\n");
                }
                // Post cycle refinement by increasing temperature (W)
                if( (l==no_incr-1) && (tick!=refine) )
                {
                    l = (int)(no_incr/10);
                    tick++;
                }

                //E_profile saving

E_save_all[l][0]=W;E_save_all[l][1]=Etot_hold;E_save_all[l][2]=E1_hold;E_save_all[l][3]=E2_hold;

                // Logarithmic cooling
                W = Wmax*(1 - pow(alpha,incr[l]));
            }
        }
        end_program = clock();

        elap = ((double)end_program-(double)start_program)/CLOCKS_PER_SEC;

        // Truncate saved scores (if applicable)
        k=0;
        while( E_save[k][0] != 0.0 )
        {
```

```
        k++;
        if(k==top)
            break;
    }
    top=k;

    // Print final results to file
    FILE * f;
    FILE * g;

    if(file==1)
    {
        //E_profile
        f = fopen("E_profile.csv","w");
        fprintf(f,"%lf,\n",no_incr);
        for(i=0;i<no_incr;i++)

fprintf(f,"%lf,%lf,%lf,%lf,\n",E_save_all[i][0],E_save_all[i][1],E_save_all[i][2],E_save_all[i][3
]);
        fclose(f);
        if(movie==1)
        {
            //Movie
            f = fopen("movie.csv","w");
            fprintf(f,"%d,\n",amt);
            fprintf(f,"%d,\n",(int)no_incr);
            for(i=0;i<no_incr;i++)
                for(j=0;j<amt;j++)
                    for(k=0;k<2;k++)
                    {
                        for(l=0;l<len;l++)
                            fprintf(f,"%lf,",seq[i][j][k][l]);
                        fprintf(f,"\n");
                    }
            fclose(f);
        }
        //Regular output
        f = fopen("outputsynth.csv","w");
        fprintf(f,"%d,\n",amt);
        fprintf(f,"%d,\n",top);
        //Real
        for(i=0;i<2;i++)
        {
            for(j=0;j<eob1.len;j++)
                fprintf(f,"%lf,",pks[i][j]);
            fprintf(f,"\n");
        }
        if(amt>1)
        {
            for(i=0;i<2;i++)
            {
                for(j=0;j<eob2.len;j++)
                    fprintf(f,"%lf,",pks2[i][j]);
                fprintf(f,"\n");
            }
        }
        //First
        for(i=0;i<2;i++)
        {
            for(j=0;j<eob1.len;j++)
                fprintf(f,"%lf,",pks1_first[i][j]);
            fprintf(f,"\n");
        }
        if(amt>1)
        {
            for(i=0;i<2;i++)
            {
                for(j=0;j<eob2.len;j++)
                    fprintf(f,"%lf,",pks2_first[i][j]);
                fprintf(f,"\n");
            }
```

```
}
//Top
for(i=0;i<top;i++)
{
    for(j=0;j<2;j++)
    {
        for(k=0;k<eob1.len;k++)
            fprintf(f,"%lf,",pks1_save[i][j][k]);
        fprintf(f,"\n");
    }
    if(amt>1)
    {
        for(j=0;j<2;j++)
        {
            for(k=0;k<eob2.len;k++)
                fprintf(f,"%lf,",pks2_save[i][j][k]);
            fprintf(f,"\n");
        }
    }
}
fclose(f);

g = fopen("inputsynth1.csv","w");

fprintf(g,"%d,\n",len);
for(i=0;i<3;i++)
{
        for(j=0;j<len;j++)
        {
            if(i<2)
                fprintf(g,"%lf,",pks1_save[0][i][j]);
            else
                fprintf(g,"%d,",labels[j]);
        }
        fprintf(g,"\n");
}
fprintf(g,"%lf,%lf,\n",eob1.rmax,eob1.rmin);
fprintf(g,"%lf,%lf,\n",eob1.cmax,eob1.cmin);
fprintf(g,"%d,%d,\n",eob1.szr,eob1.szc);
for(i=0;i<eob1.szr;i++)
{
    for(j=0;j<eob1.szc;j++)
        fprintf(g,"%.10lf,",eob1.E[i][j]);
    fprintf(g,"\n");
}
fclose(g);
if(amt>1)
{
    g = fopen("inputsynth2.csv","w");
    fprintf(g,"%d,\n",len);
    for(i=0;i<3;i++)
    {
        for(j=0;j<len;j++)
        {
            if(i<2)
                fprintf(g,"%lf,",pks2[i][j]);
            else
                fprintf(g,"%d,",labels[j]);
        }
        fprintf(g,"\n");
    }
    fprintf(g,"%lf,%lf,\n",eob2.rmax,eob2.rmin);
    fprintf(g,"%lf,%lf,\n",eob2.cmax,eob2.cmin);
    fprintf(g,"%d,%d,\n",eob2.szr,eob2.szc);
    for(i=0;i<eob2.szr;i++)
    {
        for(j=0;j<eob2.szc;j++)
            fprintf(g,"%.10lf,",eob2.E[i][j]);
        fprintf(g,"\n");
    }
    fclose(g);
```

```
        }
    }

    printf("Top Scores\n");
    for(i=0;i<top;i++)
        printf("%4d: %lf\n",i+1,E_save[i][0]);

    printf("\nTotal runtime: %.2lf seconds",elap);
}
```

Appendix B: Structure calculation program

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include </usr/include/gsl/gsl_rng.h>
#include </usr/include/gsl/gsl_randist.h>
#include </usr/include/gsl/gsl_matrix.h>
#include </usr/include/gsl/gsl_complex.h>
#include </usr/include/gsl/gsl_complex_math.h>
#include </usr/include/gsl/gsl_blas.h>
#include </usr/include/gsl/gsl_linalg.h>
#include </usr/include/gsl/gsl_multimin.h>
#include </usr/include/gsl/gsl_sort.h>

int
    rnd      = 4,
    sz       = 40,      // For rnd=1
    kb_max   = 1,
    dry_runs = 1,
    CH       = 1,
    CN       = 0,
    RAMA     = 1,
    stride   = 2,       // number of planes evaluated
    beating  = 1,       // This many times the simplex is stopped to check for tolerance
    simits   = 1000,    // At least this many simplex iterations to find a low score
    stepbk   = 5,
    file     = 1;
double
    tol       = 1E3,    // Proceed to next residue only if tol is passed; do not set above 1E10
    tol2      = 1E0,    // To speed up algorithm, stop search once score is below tol2
    noisiness = 0,
    THETA     = 2,
    PHI       = 1,

    phimin    = -85,    // for rnd=1
    phimax    =  -45,
    psimin    = -60,
    psimax    =  -20,
    PHI0,PSI0,dPHI,dPSI,dPHI3,dPSI3,
    PHI_max,PHI_min,PSI_max,PSI_min,
    ***surf_RAM,
    elapsed;
int
    RAM_crit[5];
double
    pi = 3.141592653589793238,
    d2r = 0.01745329251994329547,  // rad deg^-1
    wt = 50.80373831775702,//71.12523364485982,
    a_nc = 151.8,
    a_n = 12.5,
    tetra = 110.5,
    tetid,
    HNCa = 118.2,
    NCCa = 115.6,
    HNCo = 119.5,
    sigma[3] = {64, 77, 222},
```

```
    sigmagly[3] = {41, 64, 215},
    gamma1 = 18.5,
    gamma2 = 90,
    chi0 = 10545.76120, // Hz
    chi1 = 23334.73464537244, // Hz
    chi4 = 1007.172872312454, // Hz
    omega,
    angle1,
    angle2,
    sixty;

gsl_complex
    i;

gsl_matrix_complex
    *Dcs, *Dcsdag, *Mcs, *T_Q, *Tdag,
    *inner1, *inner2, *inner3, *inner1gly, *inner4, *inner1Q, *inner2Q, *inner3Q, *inner3PQ,
*inner4Q, *inner1glyQ, *ytetra, *eang12;

gsl_rng
    * r;

const gsl_multimin_fminimizer_type * TT;

clock_t
    begin,
    end;

// REMOVE
double e1av=0,e2av=0,e3av=0;
int uni = 0;


typedef struct
{
    int
        * resnum,
        lower,
        upper,
        ind,
        stride;
    double
        *** params,
        ** freq,
        ** target,
        answers[3],
        THETA0,
        PHI0;
    char
        * restyp;
    gsl_vector_complex
        * Q0;
} robj;
typedef struct
{
    int
        ind[3],
        typ[3];
    double
        ans[10],
        low[10],
        err;
    gsl_vector
        * guess;
} aobj;
void wigner(double one, double two, double three, gsl_matrix_complex * out)
{
    double
        cos_b,
        sin_b_sq_2;
    gsl_complex
```

```
        ex_ia,
        ex_ig,
        D33,D31,D23,D32;
    cos_b = cos(two);
    sin_b_sq_2 = sin(two)/sqrt(2);
    ex_ia = gsl_complex_exp(gsl_complex_mul_real(i,one));
    ex_ig = gsl_complex_exp(gsl_complex_mul_real(i,three));
    D33 = gsl_complex_mul(ex_ia,gsl_complex_mul_real(ex_ig,((1+cos_b)/2)));
    D31 = gsl_complex_mul(ex_ia,gsl_complex_mul_real(gsl_complex_conjugate(ex_ig),((1-
cos_b)/2)));
    D23 = gsl_complex_mul_real(ex_ig,sin_b_sq_2);
    D32 = gsl_complex_mul_real(ex_ia,sin_b_sq_2);
    gsl_matrix_complex_set(out,0,0,gsl_complex_conjugate(D33));
    gsl_matrix_complex_set(out,0,1,gsl_complex_mul_real(gsl_complex_conjugate(D32),-1));
    gsl_matrix_complex_set(out,0,2,gsl_complex_conjugate(D31));
    gsl_matrix_complex_set(out,1,0,gsl_complex_conjugate(D23));
    gsl_matrix_complex_set(out,1,1,gsl_complex_rect(cos_b,0.0));
    gsl_matrix_complex_set(out,1,2,gsl_complex_mul_real(D23,-1));
    gsl_matrix_complex_set(out,2,0,D31);
    gsl_matrix_complex_set(out,2,1,D32);
    gsl_matrix_complex_set(out,2,2,D33);
}
void M_csa(double s1, double s2, double s3, gsl_matrix_complex * out)
{
    gsl_matrix_complex_set(out,0,0,gsl_complex_rect((s1+s2)/2,0.0));
    gsl_matrix_complex_set(out,0,1,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,0,2,gsl_complex_rect((s2-s1)/2,0.0));
    gsl_matrix_complex_set(out,1,0,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,1,1,gsl_complex_rect(s3,0.0));
    gsl_matrix_complex_set(out,1,2,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,2,0,gsl_complex_rect((s2-s1)/2,0.0));
    gsl_matrix_complex_set(out,2,1,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,2,2,gsl_complex_rect((s1+s2)/2,0.0));
}
void M_dc(double Chi, gsl_matrix_complex * out)
{
    gsl_matrix_complex_set(out,0,0,gsl_complex_rect(-0.5*Chi,0.0));
    gsl_matrix_complex_set(out,0,1,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,0,2,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,1,0,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,1,1,gsl_complex_rect(1.0*Chi,0.0));
    gsl_matrix_complex_set(out,1,2,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,2,0,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,2,1,gsl_complex_rect(0.0,0.0));
    gsl_matrix_complex_set(out,2,2,gsl_complex_rect(-0.5*Chi,0.0));
}
void YyY(double theta, double phi, gsl_vector_complex * Y)
{
    gsl_complex
        holdcn,
        holdcn2;
    holdcn  = gsl_complex_exp(gsl_complex_rect(0,phi));
    holdcn2 = gsl_complex_mul_real(holdcn,(-sin(theta)/sqrt(2.0)));
    gsl_vector_complex_set(Y,0,holdcn2);
    gsl_vector_complex_set(Y,1,gsl_complex_rect(cos(theta),0));
    holdcn = gsl_complex_exp(gsl_complex_rect(0,-phi));
    holdcn2 = gsl_complex_mul_real(holdcn,(sin(theta)/sqrt(2.0)));
    gsl_vector_complex_set(Y,2,holdcn2);
}
void mat_mult(gsl_matrix_complex * x, gsl_matrix_complex * y, gsl_matrix_complex * result)
{
    gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,x,y,GSL_COMPLEX_ZERO,result);
    //gsl_blas_zsymm(CblasLeft,CblasUpper,GSL_COMPLEX_ONE,x,y,GSL_COMPLEX_ZERO,result);
}
void dagger(gsl_matrix_complex * in, gsl_matrix_complex * out)
{
    int m,n;
    for(m=0;m<3;m++)
        for(n=0;n<3;n++)

gsl_matrix_complex_set(out,m,n,gsl_complex_conjugate(gsl_matrix_complex_get(in,n,m)));
```

```
}
void daggerv(gsl_vector_complex * in, gsl_vector_complex * out)
{
    int m;
    for(m=0;m<3;m++)
        gsl_vector_complex_set(out,m,gsl_complex_conjugate(gsl_vector_complex_get(in,m)));
}
void matinv(gsl_matrix_complex * in, gsl_matrix_complex * out)
{
    int m;
    gsl_matrix_complex * copy = gsl_matrix_complex_alloc(3,3);
    gsl_permutation * p = gsl_permutation_alloc(3);
    gsl_matrix_complex_memcpy(copy,in);
    gsl_linalg_complex_LU_decomp(copy, p, &m);
    gsl_linalg_complex_LU_invert(copy, p, out);
    gsl_matrix_complex_free(copy);
    gsl_permutation_free(p);
}
void csatensor(double s1, double s2, double s3, int Q, gsl_matrix_complex * out)
{
    int m,n;
    gsl_matrix_complex
        * Mcs =  gsl_matrix_complex_alloc(3,3),
        * holdcm = gsl_matrix_complex_alloc(3,3);

    M_csa(s1, s2, s3, Mcs);
    gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,Dcs,Mcs,GSL_COMPLEX_ZERO,holdcm);//
inner1

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,holdcm,Dcsdag,GSL_COMPLEX_ZERO,out);//
inner1

    for(m=0;m<3;m++)
        for(n=0;n<3;n++)

gsl_matrix_complex_set(out,m,n,gsl_complex_mul_real(gsl_matrix_complex_get(out,m,n),wt));

    if(Q==1)
    {

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,out,Tdag,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,T_Q,holdcm,GSL_COMPLEX_ZERO,out);
    }
    gsl_matrix_complex_free(Mcs);gsl_matrix_complex_free(holdcm);
}
void csdc(robj * robo, gsl_vector_complex * startY, double ** parms, double ** freqs)
{
    int m,n;
    double hn, hp;
    gsl_matrix_complex
        * hold1 = gsl_matrix_complex_alloc(3,3),
        * hold2 = gsl_matrix_complex_alloc(3,3),
        * hold3 = gsl_matrix_complex_alloc(3,3);
    gsl_vector_complex
        * ypre = gsl_vector_complex_alloc(3),
        * ypost = gsl_vector_complex_alloc(3),
        * ych = gsl_vector_complex_alloc(3),
        * ydag = gsl_vector_complex_alloc(3),
        * holdv = gsl_vector_complex_alloc(3);
    gsl_complex
        hold;

    // First resonances
    daggerv(startY,ydag);
    if(robo->restyp[0]=='G')
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1gly,ydag,GSL_COMPLEX_ZERO,holdv);
    else
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1,ydag,GSL_COMPLEX_ZERO,holdv);
    gsl_blas_zdotu(startY,holdv,&hold);
    freqs[0][0] = GSL_REAL(hold)/*/wt*/;
```

```
    gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner2,ydag,GSL_COMPLEX_ZERO,holdv);
    gsl_blas_zdotu(startY,holdv,&hold);
    freqs[0][1] = fabs(GSL_REAL(hold));
    if(CN==1)
    {
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner4,ydag,GSL_COMPLEX_ZERO,holdv);
        gsl_blas_zdotu(startY,holdv,&hold);
        freqs[0][3] = fabs(GSL_REAL(hold));
    }

    // Main loop
    for(n=0;n<3;n++)
            gsl_vector_complex_set(ypre,n,gsl_vector_complex_get(startY,n));
    for(m=0;m<sz-1;m++)
    {
        if(CH==1)
        {
            wigner(0, -pi/2, 0, hold2);
            hp = 0;
            if(robo->restyp[m]=='G')
            {
                wigner(angle1, parms[m][0]+sixty, (pi/2)-tetid, hold1);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,hold1,hold2,GSL_COMPLEX_ZERO,hold3);
                gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,hold3,ypre,GSL_COMPLEX_ZERO,ych);
                hp = chi1*fabs((((3*pow(GSL_REAL(gsl_vector_complex_get(ych,1)),2.0))-1)/2.0));
            }
            wigner(angle1, parms[m][0]-sixty, (pi/2)-tetid, hold1);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,hold1,hold2,GSL_COMPLEX_ZERO,hold3);
            gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,hold3,ypre,GSL_COMPLEX_ZERO,ych);
            hn = chi1*fabs((((3*pow(GSL_REAL(gsl_vector_complex_get(ych,1)),2.0))-1)/2.0));
            freqs[m][2] = sqrt( (hp*hp) + (hn*hn) );
        }
        wigner(angle1,parms[m][0],tetra,hold1);
        wigner(0,(-1*parms[m][1])-pi,angle2,hold2);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,hold1,hold2,GSL_COMPLEX_ZERO,hold3);
        gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,hold3,ypre,GSL_COMPLEX_ZERO,ypost);
        for(n=0;n<3;n++)
            gsl_vector_complex_set(ypre,n,gsl_vector_complex_get(ypost,n));

        daggerv(ypost,ydag);
        if(robo->restyp[m+1]=='G')
            gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1gly,ydag,GSL_COMPLEX_ZERO,holdv);
        else
            gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1,ydag,GSL_COMPLEX_ZERO,holdv);
        gsl_blas_zdotu(ypost,holdv,&hold);
        freqs[m+1][0] = GSL_REAL(hold)/*wt*/;
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner2,ydag,GSL_COMPLEX_ZERO,holdv);
        gsl_blas_zdotu(ypost,holdv,&hold);
        freqs[m+1][1] = fabs(GSL_REAL(hold));
        if(CN==1)
        {
            gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner4,ydag,GSL_COMPLEX_ZERO,holdv);
            gsl_blas_zdotu(ypost,holdv,&hold);
            freqs[m+1][3] = fabs(GSL_REAL(hold));
        }
    }

gsl_matrix_complex_free(hold1);gsl_matrix_complex_free(hold2);gsl_matrix_complex_free(hold3);gsl_
vector_complex_free(ypre);

gsl_vector_complex_free(ypost);gsl_vector_complex_free(ydag);gsl_vector_complex_free(ych);gsl_vec
tor_complex_free(holdv);
}
void simplex(robj * robo, aobj * aob, int num, double (*func)(const gsl_vector * v, void * robo))
{
    gsl_multimin_fminimizer * s = NULL;
    gsl_vector
        *ss, // Updated answer
```

```
        *xx; // Initial guess
    gsl_multimin_function minex_func;
    size_t iter = 0;
    int m,status;
    double size;

    xx = gsl_vector_alloc(num);
    ss = gsl_vector_alloc(num);//holds the answer
    gsl_vector_set_all(ss,1.0);

    minex_func.n = num;
    minex_func.f = func;
    minex_func.params = robo;

    s = gsl_multimin_fminimizer_alloc(TT,num);
    gsl_multimin_fminimizer_set(s,&minex_func,aob->guess,ss);

    do
    {
        iter++;
        status = gsl_multimin_fminimizer_iterate(s); // Take a step

        if(status)
            break;

        size = gsl_multimin_fminimizer_size(s);
        status = gsl_multimin_test_size(size,1E-7);

        /*if(status==GSL_SUCCESS)
            printf("converged to minimum at\n",(gsl_vector_get(s->x,0)));

        printf("%5d %10.3e %10.3e f() = %7.3f size = %.3f\n",iter,gsl_vector_get(s-
>x,0),gsl_vector_get(s->x,1),s->fval,size);*/
    }
    while(status==GSL_CONTINUE && iter<1000);

    for(m=0;m<num;m++)
        aob->ans[m] = gsl_vector_get(s->x,m);
    aob->err = s->fval;

    gsl_vector_free(xx);gsl_vector_free(ss);gsl_multimin_fminimizer_free(s);
}
double plane_orientation_find(const gsl_vector * v, void * rob)
{
    robj * robo = (robj*)rob;
    int I = robo->ind;
    double e1,e2,e3,etot,
        PHI0 = gsl_vector_get(v,0),
        THETA0   = gsl_vector_get(v,1),
        cs, dc, cn;

    if( (PHI0>pi) || (PHI0<0) || (THETA0>(2*pi)) || (THETA0<0) )
        return 1E6;

    gsl_complex holdcn, holdcn2;
    gsl_vector_complex
        * Y0     = gsl_vector_complex_alloc(3),
        * Y0dag  = gsl_vector_complex_alloc(3),
        * holdcv = gsl_vector_complex_alloc(3);

    YyY(THETA0,PHI0,Y0);
    daggerv(Y0,Y0dag);
    if(robo->restyp[I]=='G')
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1gly,Y0dag,GSL_COMPLEX_ZERO,holdcv);
    else
        gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner1,   Y0dag,GSL_COMPLEX_ZERO,holdcv);
    gsl_blas_zdotu(Y0,holdcv,&holdcn);
    cs = GSL_REAL(holdcn);
    gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner2,Y0dag,GSL_COMPLEX_ZERO,holdcv);
    gsl_blas_zdotu(Y0,holdcv,&holdcn);
    dc = fabs(GSL_REAL(holdcn));
```

```
    gsl_blas_zgemv(CblasNoTrans,GSL_COMPLEX_ONE,inner4,Y0dag,GSL_COMPLEX_ZERO,holdcv);
    gsl_blas_zdotu(Y0,holdcv,&holdcn);
    cn = fabs(GSL_REAL(holdcn));

    e1 = (cs-robo->target[0][0]);
    e2 = dc-robo->target[0][1];
    etot = (e1*e1) + (e2*e2);
    if(CN==1)
    {
        e3 = cn-robo->target[0][3];
        etot += e3*e3;
    }
    etot = sqrt(etot);

    gsl_vector_complex_free(Y0);gsl_vector_complex_free(Y0dag);gsl_vector_complex_free(holdcv);

    if( (PHI0>2*pi) || (PHI0<0) || (THETA0>pi) || (THETA0<0) )
        etot=1E6;
    return etot;
}
void measurables(robj * R, gsl_vector_complex * Qin, double PHI, double PSI, gsl_vector_complex *
Qout, double * res)
{
    double hn, hp;
    gsl_complex
        cn0, cn1, cn2, cn3, cn4, cn5, cn6,
        Qi0, Qi1, Qi2;
    gsl_vector_complex
        * Q_i = gsl_vector_complex_alloc(3),
        * Q_i2 = gsl_vector_complex_alloc(3);

    int n = R->ind;

    cn0 = gsl_vector_complex_get(Qin,0);
    Qi0 = gsl_complex_mul(gsl_complex_exp(gsl_complex_rect(0.0,PHI)),cn0);
    Qi1 = gsl_vector_complex_get(Qin,1);
    cn0 = gsl_complex_conjugate(Qi0);// for some reason conjugate is taken of Qi1, NOT Qi0 as
intended
    Qi2 = gsl_complex_mul_real(cn0,-1.0);

gsl_vector_complex_set(Q_i,0,Qi0);gsl_vector_complex_set(Q_i,1,Qi1);gsl_vector_complex_set(Q_i,2,
Qi2);

    if(CH==1)
    {
        // CALC3
        hp = 0;
        if(R->restyp[n]=='G')
        {
            cn0 = gsl_matrix_complex_get(inner3PQ,0,0);cn1 =
gsl_matrix_complex_get(inner3PQ,1,1);
            cn2 = gsl_matrix_complex_get(inner3PQ,0,1);cn3 =
gsl_matrix_complex_get(inner3PQ,0,2);
            // Middle term
            cn4 =
gsl_complex_add(gsl_complex_mul(gsl_complex_sub(cn1,cn0),Qi1),gsl_complex_mul_real(gsl_complex_mu
l(cn2,Qi0),4.0));
            cn5 = gsl_complex_mul(Qi1,cn4);
            // Last term
            cn6 = gsl_complex_mul_real(gsl_complex_mul(gsl_complex_pow_real(Qi0,2.0),cn3),2.0);

            hp = fabs(GSL_REAL(gsl_complex_sub(gsl_complex_add(cn0,cn5),cn6)));
        }
        cn0 = gsl_matrix_complex_get(inner3Q,0,0);cn1 = gsl_matrix_complex_get(inner3Q,1,1);
        cn2 = gsl_matrix_complex_get(inner3Q,0,1);cn3 = gsl_matrix_complex_get(inner3Q,0,2);
        // Middle term
        cn4 =
gsl_complex_add(gsl_complex_mul(gsl_complex_sub(cn1,cn0),Qi1),gsl_complex_mul_real(gsl_complex_mu
l(cn2,Qi0),4.0));
        cn5 = gsl_complex_mul(Qi1,cn4);
        // Last term
```

```
        cn6 = gsl_complex_mul_real(gsl_complex_mul(gsl_complex_pow_real(Qi0,2.0),cn3),2.0);

        hn = fabs(GSL_REAL(gsl_complex_sub(gsl_complex_add(cn0,cn5),cn6)));

        // Exceptions (Glycines) for isotropic solution predictions
        //if(R->restyp[n]=='G')
        //  hn=0;

        res[2] = sqrt( (hn*hn) + (hp*hp) );
    }

    gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,ytetra,Q_i,GSL_COMPLEX_ZERO,Q_i2);

    Qi0 = gsl_complex_mul( gsl_complex_exp(gsl_complex_rect(0.0,-1*(pi+PSI))) ,
gsl_vector_complex_get(Q_i2,0) );
    Qi1 = gsl_vector_complex_get(Q_i2,1);
    cn0 = gsl_complex_conjugate(Qi0);
    Qi2 = gsl_complex_mul_real(cn0,-1.0);

    // CALC1
    if(R->restyp[n+1]=='G')
    {
        cn0 = gsl_matrix_complex_get(inner1glyQ,0,0);cn1 =
gsl_matrix_complex_get(inner1glyQ,1,1);
        cn2 = gsl_matrix_complex_get(inner1glyQ,0,1);cn3 =
gsl_matrix_complex_get(inner1glyQ,0,2);
    }
    else
    {
        cn0 = gsl_matrix_complex_get(inner1Q,0,0);cn1 = gsl_matrix_complex_get(inner1Q,1,1);
        cn2 = gsl_matrix_complex_get(inner1Q,0,1);cn3 = gsl_matrix_complex_get(inner1Q,0,2);
    }

    // Middle term
    cn4 =
gsl_complex_add(gsl_complex_mul(gsl_complex_sub(cn1,cn0),Qi1),gsl_complex_mul_real(gsl_complex_mu
l(cn2,Qi0),4.0));
    cn5 = gsl_complex_mul(Qi1,cn4);
    // Last term
    cn6 = gsl_complex_mul_real(gsl_complex_mul(gsl_complex_pow_real(Qi0,2.0),cn3),2.0);

    res[0] = GSL_REAL(gsl_complex_sub(gsl_complex_add(cn0,cn5),cn6));

    // CALC2
    cn0 = gsl_matrix_complex_get(inner2Q,0,0);cn1 = gsl_matrix_complex_get(inner2Q,1,1);
    cn2 = gsl_matrix_complex_get(inner2Q,0,1);cn3 = gsl_matrix_complex_get(inner2Q,0,2);

    // Middle term
    cn4 =
gsl_complex_add(gsl_complex_mul(gsl_complex_sub(cn1,cn0),Qi1),gsl_complex_mul_real(gsl_complex_mu
l(cn2,Qi0),4.0));
    cn5 = gsl_complex_mul(Qi1,cn4);
    // Last term
    cn6 = gsl_complex_mul_real(gsl_complex_mul(gsl_complex_pow_real(Qi0,2.0),cn3),2.0);

    res[1] = fabs(GSL_REAL(gsl_complex_sub(gsl_complex_add(cn0,cn5),cn6)));

    // CALC4
    cn0 = gsl_matrix_complex_get(inner4Q,0,0);cn1 = gsl_matrix_complex_get(inner4Q,1,1);
    cn2 = gsl_matrix_complex_get(inner4Q,0,1);cn3 = gsl_matrix_complex_get(inner4Q,0,2);

    // Middle term
    cn4 =
gsl_complex_add(gsl_complex_mul(gsl_complex_sub(cn1,cn0),Qi1),gsl_complex_mul_real(gsl_complex_mu
l(cn2,Qi0),4.0));
    cn5 = gsl_complex_mul(Qi1,cn4);
    // Last term
    cn6 = gsl_complex_mul_real(gsl_complex_mul(gsl_complex_pow_real(Qi0,2.0),cn3),2.0);

    res[3] = fabs(GSL_REAL(gsl_complex_sub(gsl_complex_add(cn0,cn5),cn6)));
```

```
gsl_vector_complex_set(Q_i,0,Qi0);gsl_vector_complex_set(Q_i,1,Qi1);gsl_vector_complex_set(Q_i,2,
Qi2);
    gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,eang12,Q_i,GSL_COMPLEX_ZERO,Qout);

    gsl_vector_complex_free(Q_i);gsl_vector_complex_free(Q_i2);
}
double sigmaQ2(const gsl_vector * v, void * rob)
{
    robj * robo = (robj*)rob;
    int
        m,o,I,J,
        n = robo->ind;
    double
        PHI,PSI,
        e1,e2,e3,e4,etot,E[4];

    if((RAMA==1))// && (n!=12) && (n!=36) && (n!=38) && (n!=48))
    {
        for(m=0;m<robo->stride;m++)
        {
            PHI = v->data[2*m];
            PSI = v->data[(2*m)+1];
            if(PHI>pi)
                PHI = -pi + fmod(PHI,pi);
            else if(PHI<-pi)
                PHI = pi + fmod(PHI,pi);
            if(PSI>pi)
                PSI = -pi + fmod(PSI,pi);
            else if(PSI<-pi)
                PSI = pi + fmod(PSI,pi);
            I = round((PHI+pi-0.01745329)/0.034906585);
            if(I==180)
                I-=1;
            J = round((PSI+pi-0.01745329)/0.034906585);
            if(J==180)
                J-=1;
            if(surf_RAM[RAM_crit[m]][I][J]<2E-3) // 5.05E-3
                return 1E5;
        }
    }
    gsl_vector_complex
        * Q_i = gsl_vector_complex_alloc(3),
        * Q_i2 = gsl_vector_complex_alloc(3);

    for(m=0;m<3;m++)
        gsl_vector_complex_set(Q_i, m, gsl_vector_complex_get(robo->Q0, m));
    etot=0;
    for(o=0;o<robo->stride;o++)
    {
        measurables(robo, Q_i, v->data[o*2], v->data[(o*2)+1], Q_i2, E);
        e1 = (E[0]-robo->target[robo->ind+1][0]);//*1.272820931368286;
        e2 = (E[1]-robo->target[robo->ind+1][1])/2;
        etot += (e1*e1) + (e2*e2);
        if(CH==1)
        {
            if(robo->ind!=100)
                e3 = (E[2]-robo->target[robo->ind][2])/4.0;
            else
                e3 = 1E-9;
            etot += e3*e3;
        }
        if(CN==1)
        {
            e4 = (E[3]-robo->target[robo->ind+1][3])*2;
            etot += e4*e4;
        }
        if(o<robo->stride-1)
            for(m=0;m<3;m++)
                gsl_vector_complex_set(Q_i, m, gsl_vector_complex_get(Q_i2, m));
        robo->ind++;
```

```
    }
    robo->ind=n;
    gsl_vector_complex_free(Q_i);gsl_vector_complex_free(Q_i2);
    //e1av += fabs(e1);e2av += fabs(e2);e3av += fabs(e3);
    //uni++;
    //printf("%9.0lf, %9.0lf, %9.0lf    (%d)\n",e1av/uni,e2av/uni,e3av/uni,uni);
    return sqrt(etot);
}
void noise(robj * robo, double radlim)
{
    int m;
    double radius, ang, ang2, dx, dy, dz;

    for(m=0;m<sz;m++)
    {
        robo->target[m][0] += gsl_ran_flat(r, -radlim, radlim);
        robo->target[m][1] += gsl_ran_flat(r, -radlim, radlim);
        if(CH==1)
            robo->target[m][2] += gsl_ran_flat(r, -radlim, radlim);
        if(CN==1)
            robo->target[m][3] += gsl_ran_flat(r, -radlim, radlim);
    }
}
void printdv(int length, double * A)
{
    int x;
    printf("\n");
    for(x=0;x<length;x++)
    {
        printf("%.5lf ", A[x]);
        printf("\n");
    }
}
void printda(int rows, int cols, double ** A)
{
    int x,y;
    printf("\n");
    for(x=0;x<rows;x++)
    {
        for(y=0;y<cols;y++)
            printf("%10.5lf   ",A[x][y]);
        printf("\n");
    }
}
void printcv(int length, gsl_vector_complex * A)
{
    int m;
    gsl_complex hold;
    for(m=0;m<length;m++)
    {
        hold = gsl_vector_complex_get(A,m);
        printf("%lf + %lfi\n",GSL_REAL(hold),GSL_IMAG(hold));
    }
}
void printcm(int rows, int columns, gsl_matrix_complex * A)
{
    int m,n;
    printf("\n");
    for(m=0;m<rows;m++)
    {
        for(n=0;n<columns;n++)
            printf("%10.5lf + %10.5lfi
",GSL_REAL(gsl_matrix_complex_get(A,m,n)),GSL_IMAG(gsl_matrix_complex_get(A,m,n)));
        printf("\n");
    }
    printf("\n");
}
void copydv(int length, double * one, double * two)
{
    int m;
```

```
    // two into one
    for(m=0;m<length;m++)
        one[m] = two[m];
}
int main()
{
    robj
        real,
        ran,
        sys;

    aobj
        A;

    int m,n,o,p;
    double temp[3];

    // Constants and Angles
    i=gsl_complex_rect(0.0,1.0);
    omega = pi;
    a_nc *= d2r;
    NCCa *= d2r;
    a_n *= d2r;
    tetra *= d2r;
    tetid = tetra;//acos(-1.0/3.0);
    HNCa *= d2r;
    HNCo *= d2r;
    gamma1 *= d2r;
    gamma2 *= d2r;
    angle1 = ((3*pi)/2)-HNCa;
    angle2 = ((3*pi)/2)-NCCa-HNCo;
    sixty = 60*pi/180;//acos((-tan(tetra/2.0))/tan(tetid));
    phimin*=d2r;phimax*=d2r;psimin*=d2r;psimax*=d2r;

    // Random Numbers
    srand(time(NULL));
    const gsl_rng_type * T;
    gsl_rng_env_setup();
    T = gsl_rng_default;
    r = gsl_rng_alloc(T);
    gsl_rng_set(r,rand());
    TT = gsl_multimin_fminimizer_nmsimplex2;

    // Allocation
    gsl_complex
        holdcn,
        holdcn2,
        holdcn3;

    gsl_vector_complex
        * y = gsl_vector_complex_alloc(3),
        * holdcv = gsl_vector_complex_alloc(3),
        * holdcv2 = gsl_vector_complex_alloc(3);

    gsl_matrix_complex
        * Mcsgly = gsl_matrix_complex_alloc(3,3),* Mdc = gsl_matrix_complex_alloc(3,3),
        * Ddc = gsl_matrix_complex_alloc(3,3),* Ddcinv = gsl_matrix_complex_alloc(3,3),
        * holdcm = gsl_matrix_complex_alloc(3,3),* holdcm2 = gsl_matrix_complex_alloc(3,3),*
holdcm3 = gsl_matrix_complex_alloc(3,3),
        * holdcm4 = gsl_matrix_complex_alloc(3,3),* holdcm5 = gsl_matrix_complex_alloc(3,3),*
holdcm6 = gsl_matrix_complex_alloc(3,3);
    Mcs = gsl_matrix_complex_alloc(3,3);Dcs = gsl_matrix_complex_alloc(3,3);Dcsdag =
gsl_matrix_complex_alloc(3,3);T_Q = gsl_matrix_complex_alloc(3,3);Tdag =
gsl_matrix_complex_alloc(3,3);
    inner1 = gsl_matrix_complex_alloc(3,3);inner1gly = gsl_matrix_complex_alloc(3,3);inner2 =
gsl_matrix_complex_calloc(3,3);inner3 = gsl_matrix_complex_alloc(3,3);inner4 =
gsl_matrix_complex_alloc(3,3);
    inner1Q = gsl_matrix_complex_alloc(3,3);inner1glyQ = gsl_matrix_complex_alloc(3,3);inner2Q =
gsl_matrix_complex_alloc(3,3);inner3Q = gsl_matrix_complex_alloc(3,3);inner3PQ =
gsl_matrix_complex_alloc(3,3);inner4Q = gsl_matrix_complex_alloc(3,3);
```

```
    ytetra = gsl_matrix_complex_alloc(3,3);real.Q0 = gsl_vector_complex_alloc(3);sys.Q0 =
gsl_vector_complex_alloc(3);

    // Constant matrices
    wigner(gamma1, pi/2, gamma2, Dcs);// 3 angles are omN in python
    dagger(Dcs, Dcsdag);// inner1
    M_csa(sigmagly[0], sigmagly[1], sigmagly[2], Mcsgly);// inner1gly

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,Mcsgly,Dcsdag,GSL_COMPLEX_ZERO,holdcm);/
/ inner1gly

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,Dcs,holdcm,GSL_COMPLEX_ONE,inner1gly);//
inner1gly
    gsl_matrix_complex_set(inner2,0,0,gsl_complex_rect(0.25,0));// inner2
    gsl_matrix_complex_set(inner2,0,2,gsl_complex_rect(-0.75,0));// inner2
    gsl_matrix_complex_set(inner2,1,1,gsl_complex_rect(-0.5,0));// inner2
    gsl_matrix_complex_set(inner2,2,0,gsl_complex_rect(-0.75,0));// inner2
    gsl_matrix_complex_set(inner2,2,2,gsl_complex_rect(0.25,0));// inner2
    gsl_matrix_complex_memcpy(inner3,inner2);// inner3
    wigner(-HNCa,0,0,holdcm);dagger(holdcm,holdcm2);// inner4

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner2,holdcm2,GSL_COMPLEX_ZERO,holdcm3)
;// inner4

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,holdcm,holdcm3,GSL_COMPLEX_ZERO,inner4);
// inner4
    for(m=0;m<3;m++)
        for(n=0;n<3;n++)
        {

gsl_matrix_complex_set(inner1gly,m,n,gsl_complex_mul_real(gsl_matrix_complex_get(inner1gly,m,n),w
t));

gsl_matrix_complex_set(inner2,m,n,gsl_complex_mul_real(gsl_matrix_complex_get(inner2,m,n),chi0));

gsl_matrix_complex_set(inner3,m,n,gsl_complex_mul_real(gsl_matrix_complex_get(inner3,m,n),chi1));

gsl_matrix_complex_set(inner4,m,n,gsl_complex_mul_real(gsl_matrix_complex_get(inner4,m,n),chi4));
        }

    // Q basis
    gsl_matrix_complex
        * eIx     = gsl_matrix_complex_alloc(3,3),* eang2 = gsl_matrix_complex_alloc(3,3),* eang1
= gsl_matrix_complex_alloc(3,3),* e_Ix = gsl_matrix_complex_alloc(3,3),
        * D_omega = gsl_matrix_complex_alloc(3,3);
    eang12  = gsl_matrix_complex_alloc(3,3);
    double
        // expm(i*(pi/2)*Ix)
        re1[][3] = {
            {0.5 , 0, -0.5},
            {0   , 0,    0},
            {-0.5, 0,  0.5}},
        im1[][3] = {
            {0          , 1.0/sqrt(2),          0},
            {1.0/sqrt(2), 0           , 1.0/sqrt(2)},
            {0          , 1.0/sqrt(2),          0}},
        // expm(-i*tetra*Ly): tetra=110.5
        re3[][3] = {
            {0.324896309370267,  -0.662327256766391,   0.675103690629734},
            {0.662327256766391,  -0.350207381259467,  -0.662327256766391},
            {0.675103690629734,   0.662327256766391,   0.324896309370266}},
        // expm(-i(angle1+angle2)*Iy)
        re4[][3] = {
            {3.414675230757E-3, 8.249866936897E-2,  9.965853247692E-1},
            {-8.249866936897E-2,-9.931706495385E-1, 8.249866936897E-2},
            {9.965853247692E-1, -8.249866936897E-2, 3.414675230757E-3}};

    wigner(-angle2-NCCa, 0*d2r, (3*pi/2)-HNCo, D_omega); // D_omega: Middle value can vary
    // expm(-i*angle1*Iz); angle1=2.64940980452739
    gsl_matrix_complex_set(eang1,0,0,gsl_complex_rect(-0.881303452064992,-0.472550764869054));
    gsl_matrix_complex_set(eang1,1,1,GSL_COMPLEX_ONE);
```

```
    gsl_matrix_complex_set(eang1,2,2,gsl_complex_rect(-0.881303452064992,0.472550764869054));
    // expm(-i*angle2*Iz): angle2=0.609119908946021
    gsl_matrix_complex_set(eang2,0,0,gsl_complex_rect(0.820151875873772,-0.572145873445516));
    gsl_matrix_complex_set(eang2,1,1,GSL_COMPLEX_ONE);
    gsl_matrix_complex_set(eang2,2,2,gsl_complex_rect(0.820151875873772,0.572145873445516));
    for(m=0;m<3;m++)
    {
        for(n=0;n<3;n++)
        {
            gsl_matrix_complex_set(eIx,m,n,gsl_complex_rect(re1[m][n],im1[m][n]));
            gsl_matrix_complex_set(e_Ix,m,n,gsl_complex_rect(re1[m][n],-1*im1[m][n]));
            gsl_matrix_complex_set(ytetra,m,n,gsl_complex_rect(re3[m][n],0.0));
            gsl_matrix_complex_set(eang12,m,n,gsl_complex_rect(re4[m][n],0.0));
        }
    }
    gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,eIx,eang2,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,holdcm,D_omega,GSL_COMPLEX_ZERO,T_Q);
    dagger(T_Q,Tdag);
    //inner1Q

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner1,Tdag,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,T_Q,holdcm,GSL_COMPLEX_ZERO,inner1Q);
    // inner1glyQ

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner1gly,Tdag,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,T_Q,holdcm,GSL_COMPLEX_ZERO,inner1glyQ);
    // inner2Q

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner2,Tdag,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,T_Q,holdcm,GSL_COMPLEX_ZERO,inner2Q);
    // inner3Q
    wigner(pi/3.0,(pi/2.0)-tetra,0,holdcm);dagger(holdcm,holdcm5);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,e_Ix,holdcm5,GSL_COMPLEX_ZERO,holdcm6);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner3,holdcm6,GSL_COMPLEX_ZERO,holdcm5)
;

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,eIx,holdcm5,GSL_COMPLEX_ZERO,holdcm6);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,holdcm,holdcm6,GSL_COMPLEX_ZERO,inner3Q)
;// inner3Q
    // inner3PQ
    wigner(-pi/3.0,(pi/2.0)-tetra,0,holdcm);dagger(holdcm,holdcm5);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,e_Ix,holdcm5,GSL_COMPLEX_ZERO,holdcm6);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner3,holdcm6,GSL_COMPLEX_ZERO,holdcm5)
;

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,eIx,holdcm5,GSL_COMPLEX_ZERO,holdcm6);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,holdcm,holdcm6,GSL_COMPLEX_ZERO,inner3PQ
);
    // inner4Q

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,inner4,Tdag,GSL_COMPLEX_ZERO,holdcm);

gsl_blas_zgemm(CblasNoTrans,CblasNoTrans,GSL_COMPLEX_ONE,T_Q,holdcm,GSL_COMPLEX_ZERO,inner4Q);


    // Read in ramachandran plot scores
    surf_RAM = (double***)malloc(4*sizeof(double**));
    // General: RAM_crit = 0
    FILE * f  =
fopen("C:\\Users\\joell\\Documents\\Octave\\Nevzorov\\Project2\\RAMA\\rama500general.data","r");
    surf_RAM[0] = (double**)malloc(180*sizeof(double*));
```

```
    for(m=0;m<180;m++)
    {
        surf_RAM[0][m] = (double*)malloc(180*sizeof(double));
        for(n=0;n<180;n++)
            fscanf(f,"%*lf %*lf %lf\n",&surf_RAM[0][m][n]);
    }
    fclose(f);
    // Proline: RAM_crit = 1
    f  =
fopen("C:\\Users\\joell\\Documents\\Octave\\Nevzorov\\Project2\\RAMA\\rama500pro.data","r");
    surf_RAM[1] = (double**)malloc(180*sizeof(double*));
    for(m=0;m<180;m++)
    {
        surf_RAM[1][m] = (double*)malloc(180*sizeof(double));
        for(n=0;n<180;n++)
            fscanf(f,"%*lf %*lf %lf\n",&surf_RAM[1][m][n]);
    }
    fclose(f);
    // Pre-Proline: RAM_crit = 2
    f  =
fopen("C:\\Users\\joell\\Documents\\Octave\\Nevzorov\\Project2\\RAMA\\rama500prepro.data","r");
    surf_RAM[2] = (double**)malloc(180*sizeof(double*));
    for(m=0;m<180;m++)
    {
        surf_RAM[2][m] = (double*)malloc(180*sizeof(double));
        for(n=0;n<180;n++)
            fscanf(f,"%*lf %*lf %lf\n",&surf_RAM[2][m][n]);
    }
    fclose(f);
    //Post-Glycine: RAM_crit = 3
    f  =
fopen("C:\\Users\\joell\\Documents\\Octave\\Nevzorov\\Project2\\RAMA\\rama500gly_sym.data","r");
    surf_RAM[3] = (double**)malloc(180*sizeof(double*));
    for(m=0;m<180;m++)
    {
        surf_RAM[3][m] = (double*)malloc(180*sizeof(double));
        for(n=0;n<180;n++)
            fscanf(f,"%*lf %*lf %lf\n",&surf_RAM[3][m][n]);
    }
    fclose(f);
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
    // Setting up real and random systems

    if(rnd==1) // Randomize angles
    {
        real.resnum = (int*)malloc(sz*sizeof(int));
        sys.resnum = (int*)malloc(sz*sizeof(int));
        real.restyp = (char*)malloc((sz+1)*sizeof(char));
        sys.restyp = (char*)malloc((sz+1)*sizeof(char));
        real.params = (double***)malloc(1*sizeof(double**));real.params[0] =
(double**)malloc((sz-1)*sizeof(double*));
        real.freq  = (double**)malloc(sz*sizeof(double*));
        sys.freq = (double**)malloc(sz*sizeof(double*));
        sys.target = (double**)malloc(sz*sizeof(double));
        for(m=0;m<sz;m++)
        {
            if(m<sz-1)
                real.params[0][m] = (double*)malloc(2*sizeof(double));
            real.freq[m]   = (double*)malloc(4*sizeof(double));
            sys.freq[m] = (double*)malloc(4*sizeof(double));
            sys.target[m] = (double*)malloc(4*sizeof(double));
        }
        real.THETA0 = gsl_ran_flat(r, 0, pi);
        real.PHI0   = gsl_ran_flat(r, 0, 2*pi);
        YyY(real.THETA0, real.PHI0, real.Q0);
        for(m=0;m<(sz-1);m++)
        {
            real.params[0][m][0] = gsl_ran_flat(r, phimin, phimax);
            real.params[0][m][1] = gsl_ran_flat(r, psimin, psimax);
            real.resnum[m] = m+1;
```

```
            real.restyp[m] = 'X';
            sys.resnum[m] = m+1;
            sys.restyp[m] = 'X';
        }
        real.resnum[sz-1]=sz;sys.resnum[sz-1]=sz;
        real.restyp[sz-1]='X';sys.restyp[sz-1]='X';real.restyp[sz]='X';sys.restyp[sz]='X';
        csdc(&real, real.Q0, real.params[0], real.freq);
    }
    else if((rnd==2)||(rnd==3)) // Or read in saved angles
    {
        if(THETA<0)
            gsl_ran_flat(r, 0, pi);
        else
            real.THETA0 = THETA;
        if(PHI<0)
            gsl_ran_flat(r, 0, 2*pi);
        else
            real.PHI0   = PHI;
        YyY(real.THETA0, real.PHI0, real.Q0);
        FILE * f = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Torsions\\4a2n.csv","r");
        fscanf(f,"%d,\n",&sz);
        real.resnum = (int*)malloc(sz*sizeof(int));
        sys.resnum = (int*)malloc(sz*sizeof(int));
        real.restyp = (char*)malloc((sz+1)*sizeof(char));
        sys.restyp = (char*)malloc((sz+1)*sizeof(char));
        real.params = (double***)malloc(1*sizeof(double**));real.params[0] =
(double**)malloc((sz-1)*sizeof(double*));
        real.freq  = (double**)malloc(sz*sizeof(double*));
        sys.freq = (double**)malloc(sz*sizeof(double*));
        sys.target = (double**)malloc(sz*sizeof(double));
        for(m=0;m<sz;m++)
        {
            if(m<sz-1)
                real.params[0][m] = (double*)malloc(2*sizeof(double));
            real.freq[m]   = (double*)malloc(4*sizeof(double));
            sys.freq[m] = (double*)malloc(4*sizeof(double));
            sys.target[m] = (double*)malloc(4*sizeof(double));
            real.resnum[m] = m+1;
            sys.resnum[m] = m+1;
            fscanf(f,"%c,",&real.restyp[m]);
            sys.restyp[m] = real.restyp[m];
        }
        fscanf(f,"%c\n",&real.restyp[sz]);
        sys.restyp[sz] = real.restyp[sz];
        for(m=0;m<sz-1;m++)
            fscanf(f,"%lf,",&real.params[0][m][0]);
        fscanf(f,"\n");
        for(m=0;m<sz-1;m++)
            fscanf(f,"%lf,",&real.params[0][m][1]);
        fclose(f);
        real.resnum[sz-1]=sz;sys.resnum[sz-1]=sz;
        csdc(&real, real.Q0, real.params[0], real.freq);
    }
    else
    {
        FILE * f = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\realtarg.csv","r");
        fscanf(f,"%*[^\n]\n",NULL);
        fscanf(f,"%d\n",&sz);
        real.restyp = (char*)malloc((sz+1)*sizeof(char));sys.restyp =
(char*)malloc((sz+1)*sizeof(char));
        real.freq = (double**)malloc(sz*sizeof(double*));
        sys.freq = (double**)malloc(sz*sizeof(double*));
        sys.target = (double**)malloc(sz*sizeof(double*));
        for(m=0;m<sz-1;m++)
        {
            fscanf(f,"%c,",&real.restyp[m]);
            sys.restyp[m] = real.restyp[m];
        }
        fscanf(f,"%c\n",&real.restyp[sz-1]);sys.restyp[sz-1] = real.restyp[sz-1];
```

```
        real.restyp[sz] = 'X';sys.restyp[sz] = 'X';
        for(m=0;m<sz;m++)
        {
            real.freq[m] = (double*)malloc(4*sizeof(double));
            sys.freq[m] = (double*)malloc(4*sizeof(double));
            sys.target[m] = (double*)malloc(4*sizeof(double));
            fscanf(f,"%lf,%lf,%lf,%lf\n", &real.freq[m][0], &real.freq[m][1], &real.freq[m][2],
&real.freq[m][3]);
        }
        fclose(f);
    }
    // Set inner1
    csatensor(sigma[0], sigma[1], sigma[2], 0, inner1);

    // Start constructing sys
    sys.params = (double***)malloc(kb_max*sizeof(double**));
    for(m=0;m<kb_max;m++)
    {
        sys.params[m] = (double**)malloc((sz-1)*sizeof(double*));
        for(n=0;n<sz-1;n++)
            sys.params[m][n] = (double*)malloc(2*sizeof(double));
    }

    int **stepbkV  = (int**)malloc(2*sizeof(int*));stepbkV[0] = (int*)calloc(sz-
1,sizeof(int));stepbkV[1] = (int*)calloc(sz-1,sizeof(int));
    double
        * tolerance_all = (double*)malloc(sz*sizeof(double));

    A.guess = gsl_vector_alloc(2);
    int
        S, kb=0, cnt, trials, nvar;
    double
        z_prev, theta00, phi00, z0, tolerance,
        sumrmsdf, sumcsrms, sumdcrms, sumchrms, sumcnrms,
        dummy[4],
        * tolerance_all_trial = (double*)malloc(sz*sizeof(double)),
        * CSRMS = (double*)calloc(kb_max,sizeof(double)), * DCRMS =
(double*)calloc(kb_max,sizeof(double)),
        * CHRMS = (double*)calloc(kb_max,sizeof(double)), * CNRMS =
(double*)calloc(kb_max,sizeof(double)),
        ** rmsdf = (double**)malloc(sz*sizeof(double)),
        ** freq_calc = (double**)malloc((sz-1)*sizeof(double*));
    if(CH==1)
        CHRMS = (double*)calloc(kb_max,sizeof(double));
    gsl_vector_complex
        * Y00, * Y0, * Q00, * Q0;
    gsl_matrix_complex
        * Q_all = gsl_matrix_complex_alloc(sz,3);
    Y00 = gsl_vector_complex_alloc(3); Y0  = gsl_vector_complex_alloc(3);
    Q00 = gsl_vector_complex_alloc(3); Q0  = gsl_vector_complex_alloc(3);
    for(m=0;m<kb_max;m++)
    {
        for(n=0;n<sz-1;n++)
        {
            if(m==0)
            {
                tolerance_all[n] = tol;
                rmsdf[n]         = (double*)malloc(kb_max*sizeof(double));
                freq_calc[n]     = (double*)calloc(4,sizeof(double));
            }
        }
    }
    tolerance_all[sz-1] = tol;/*RAM_crit[sz-1]=5;*/rmsdf[sz-1] =
(double*)malloc(kb_max*sizeof(double));freq_calc[sz-1] = (double*)calloc(4,sizeof(double));


    // Angle ranges
    PHI0  = 0;
    PSI0  = 0;
    dPHI  = 180;
    dPSI  = 180;
```

```
    dPHI3 = 40;
    dPSI3 = 40;
    PHI_max = (PHI0+dPHI)*d2r;
    PHI_min = (PHI0-dPHI)*d2r;
    PSI_max = (PSI0+dPSI)*d2r;
    PSI_min = (PSI0-dPSI)*d2r;

    // output.csv, freqs_RMS.csv
    FILE *fp, *fp2;
    if(file==1){
        fp = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\output.csv","w");
        fprintf(fp,"#Parameters: CH=%d; CN=%d; RAMA=%d; stride=%d; simits=%d; stepbk=%d;
tol=%.0lf; tol2=%.0lf; noisiness=%.0lf; BETA=%.2lf; ALPHA=%.2lf PHI0=%.0f PSI0=%.0f dPHI=%.0f
dPSI=%.0f\n",CH,CN,RAMA,stride,simits,stepbk,tol,tol2,noisiness,THETA,PHI,PHI0,PSI0,dPHI,dPSI);
        fprintf(fp,"%d,%d,\n",sz,kb_max);
        for(n=0;n<sz;n++)
            fprintf(fp,"%c,",sys.restyp[n]);
        fprintf(fp,"%c\n",sys.restyp[sz]);
        fclose(fp);
        fp2 = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\freqs_RMS.csv","w");
        fclose(fp2);
    }

    begin = clock();
    for(S=0;S<(kb_max+dry_runs-1);S++)
    {
        // Create targets with uncertainty
        fp = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\freqs_targ.csv","r");
        for(m=0;m<sz;m++)
        {
            for(n=0;n<4;n++)
            {
                sys.target[m][n] = real.freq[m][n];
                if(rnd==3)
                    fscanf(fp,"%lf,",&sys.target[m][n]);
            }
            if(rnd==3)
                fscanf(fp,"\n",NULL);
        }
        fclose(fp);
        if(rnd!=3)
            noise(&sys, noisiness);

        // Reset quantities that may have been altered during a run
        sys.stride = stride;
        nvar = 2*sys.stride;
        for(o=0;o<sz-1;o++){stepbkV[0][o] = 0;stepbkV[1][o] = 0;tolerance_all[o] = tol;}

        if(S<dry_runs)
            kb=0;
        else
            kb+=1;

        cnt=0;
        n=0;
        while(n<sz-1)
        {
            // If not the last residue, set nvar based on original stride
            if((sz-1-n)<sys.stride){
                sys.stride = sz-1-n;
                nvar = 2*sys.stride;
                gsl_vector_free(A.guess);
                A.guess = gsl_vector_alloc(nvar);}

            // Set the phi/psi torsion randomization range for residue n
            for(o=0;o<sys.stride;o++){
                if(sys.restyp[n+o+1]=='P')
                    RAM_crit[o] = 1;
```

```
                else if(sys.restyp[n+o+2]=='P')
                    RAM_crit[o] = 2;
                else if(sys.restyp[n+o]=='G')
                    RAM_crit[o] = 3;
                else
                    RAM_crit[o] = 0;}

            // Set inner1Q for residue n
            csatensor(sigma[0], sigma[1], sigma[2], 1, inner1Q);
            //if(n==7)
            //  csatensor(sigma[0], sigma[1], 240, 1, inner1Q);

            printf("\nn: %d | kb: %d\n",n,kb);
            sys.ind = n;
            if(n==0)
            {
                gsl_vector_free(A.guess);
                A.guess = gsl_vector_alloc(2);
                z_prev=1E7;
                for(trials=0;trials<100;trials++) // Few trials -> roll the dice on starting
orientation
                {
                    A.guess->data[0] = PHI; A.guess->data[1] = THETA;

gsl_vector_set(A.guess,0,2*pi*gsl_rng_uniform_pos(r));gsl_vector_set(A.guess,1,pi*gsl_rng_uniform
_pos(r));
                    simplex(&sys,&A,2,&plane_orientation_find);
                    if(A.err<z_prev){
                        z_prev = A.err;
                        sys.PHI0 = A.ans[0];
                        sys.THETA0 = A.ans[1];}
                }
                //sys.THETA0 = THETA;
                //sys.PHI0 = PHI;
                printf("Trying to initiate propagation...\n");
                printf("z_prev: %.20lf\n",z_prev);
                theta00 = sys.THETA0;
                phi00 = sys.PHI0;

                YyY(theta00,phi00,Y00);
                mat_mult(eang1,e_Ix,holdcm);
                gsl_blas_zgemv(CblasTrans,GSL_COMPLEX_ONE,holdcm,Y00,GSL_COMPLEX_ZERO,Q00);
                for(o=0;o<3;o++){
                    gsl_vector_complex_set(Q0,o,gsl_vector_complex_get(Q00,o));
                    gsl_matrix_complex_set(Q_all,0,o,gsl_vector_complex_get(Q00,o));}
                sys.stride = stride;
                nvar = 2*sys.stride;
                gsl_vector_free(A.guess);
                A.guess = gsl_vector_alloc(nvar);
            }

            z0 = 1E10;
            tolerance = tolerance_all[n];

            int in;
            while( (z0>tolerance) && (cnt<beating) )
            {
                sys.Q0 = Q0;
                for(o=0;o<simits;o++){
                    for(p=0;p<nvar;p++)
                        if(p%2==0)
                            gsl_vector_set(A.guess,p,gsl_ran_flat(r, PHI_min, PHI_max));
                        else
                            gsl_vector_set(A.guess,p,gsl_ran_flat(r, PSI_min, PSI_max));
                    simplex(&sys,&A,nvar,&sigmaQ2);
                    if(A.err < z0){
                        z0 = A.err;
                        in = o;
                        for(p=0;p<nvar;p++)
                            A.low[p] = A.ans[p];
                        if(z0<tol2)
```

```
                        break;}}

                cnt+=1;
            }

            if(z0<tolerance){
                if(S<dry_runs)
                    tolerance_all_trial[n] = z0;

                // This line returns a tolerance for indices in which it my have been erroneously
removed
                //if(z0<tol)
                //  for(o=0;o<sys.stride;o++)
                //      tolerance_all[n+o] = tol;

                rmsdf[n][kb] = z0;
                printf("z0    : %lf (%d)",z0,in);

                // Don't let angles go outside of -180 < x < 180
                // TODO: Put at end of program
                for(o=0;o<nvar;o++){
                    if(A.low[o] > pi)
                        A.low[o] = -pi + fmod(A.low[o],pi);
                    else if(A.low[o] < -pi)
                        A.low[o] = pi + fmod(A.low[o],pi);}

                for(o=0;o<3;o++)
                    gsl_vector_complex_set(holdcv, o, gsl_vector_complex_get(Q0, o));
                for(o=0;o<sys.stride;o++){
                    sys.params[S][n+o][0] = A.low[2*o];
                    sys.params[S][n+o][1] = A.low[(2*o)+1];
                    measurables(&sys, holdcv, sys.params[S][n+o][0], sys.params[S][n+o][1],
holdcv2, dummy);
                    for(p=0;p<3;p++){
                        gsl_matrix_complex_set(Q_all, n+o+1, p,
gsl_vector_complex_get(holdcv2,p));
                        gsl_vector_complex_set(holdcv, p, gsl_vector_complex_get(holdcv2, p));}}
                for(o=0;o<3;o++)
                    gsl_vector_complex_set(Q0, o, gsl_vector_complex_get(holdcv, o));
                cnt=0;

                n+=sys.stride;}
            else{
                // Residue n failed
                stepbkV[0][n]++;
                if(stepbkV[0][n]==3){ // Residue n has failed <#> times
                    stepbkV[1][n]++; // Residue n has arrested the search
                    if( (stepbkV[1][n] == 1) ){
                        tolerance_all[n] = 1E9; // Remove tolerance for residue n
                        tolerance_all[n+1] = 1E9;}
                    for(o=0;o<sz-1;o++){stepbkV[0][o]=0;} // Reset tally for next search
                    n=0;
                    printf("!!!RESTART!!!");}
                else{
                    n -= stepbk;
                    //tolerance_all[n] += 20;
                    printf("STEPBACK");}
                if(n>0){
                    for(o=0;o<3;o++)
                        gsl_vector_complex_set(Q0,o,gsl_matrix_complex_get(Q_all,n,o));
                    cnt=0;
                    sys.stride = stride;
                    nvar = 2*sys.stride;
                    gsl_vector_free(A.guess);
                    A.guess = gsl_vector_alloc(nvar);}
                else{

                    n = 0;
                    cnt = 0;
                    for(o=0;o<3;o++)
                        gsl_vector_complex_set(Q0,o,gsl_vector_complex_get(Q00,o));}}
```

```
        }

        if(S<dry_runs){
            for(m=0;m<sz;m++)
                tolerance_all[m] = tolerance_all_trial[m]+100;
            printf("\n\ndry run %d\n",S);}

        // Back-calculation of spectrum to check
        sys.THETA0 = theta00;
        sys.PHI0 = phi00;

        YyY(sys.THETA0,sys.PHI0,Y0);

        csdc(&sys,Y0,sys.params[S],freq_calc);

        sumrmsdf = 0;
        sumcsrms = 0;
        sumdcrms = 0;
        sumchrms = 0;
        sumcnrms = 0;
        for(m=0;m<sz;m++)
        {
            sumrmsdf += rmsdf[m][kb];
            sumcsrms += pow(sys.target[m][0]-freq_calc[m][0],2.0);
            sumdcrms += pow(sys.target[m][1]-freq_calc[m][1],2.0);
            if(CH==1)
                sumchrms += pow(sys.target[m][2]-freq_calc[m][2],2.0);
            if(CN==1)
                sumcnrms += pow(sys.target[m][3]-freq_calc[m][3],2.0);
        }
        CSRMS[kb] = sqrt(sumcsrms/(double)sz);
        DCRMS[kb] = sqrt(sumdcrms/(double)sz);
        if(CH==1)
            CHRMS[kb] = sqrt(sumchrms/(double)sz);
        if(CN==1)
            CNRMS[kb] = sqrt(sumcnrms/(double)sz);

        // Append to the end of output.csv, freqs_RMS.csv
        if(file==1)
        {
            fp = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\output.csv","a");
            fprintf(fp,"%.10lf,%.10lf\n",sys.THETA0,sys.PHI0);
            for(m=0;m<sz-1;m++)
                fprintf(fp,"%.10lf,",sys.params[kb][m][0]);
            fprintf(fp,"\n");
            for(m=0;m<sz-1;m++)
                fprintf(fp,"%.10lf,",sys.params[kb][m][1]);
            fprintf(fp,"\n");
            fclose(fp);
            fp2 = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\freqs_RMS.csv","a");
            fprintf(fp2,"%lf,%lf,%lf,%lf\n",CSRMS[kb],DCRMS[kb],CHRMS[kb],CNRMS[kb]);
            fclose(fp2);
        }

        printf("\n\none more structure done...\nkb: %d\n",kb);

        if(S>=dry_runs)
        {
            printf("Freq RMS for structure %d\n",kb);
            printf("%lf\n",sumrmsdf);
        }
    }
    end = clock();

    ////////////////////////// Statistics /////////////////////////////
    // Phi/Psi RMS
    double
        dif,
        *** difs = (double***)malloc(kb_max*sizeof(double**)),
```

```
        * phirms = (double*)calloc(kb_max,sizeof(double)),
        * psirms = (double*)calloc(kb_max,sizeof(double));
    if( (rnd==1) || (rnd==2) || (rnd==3) )
    {
        for(m=0;m<kb_max;m++)
        {
            difs[m] = (double**)malloc((sz-1)*sizeof(double*));
            for(n=0;n<sz-1;n++)
            {
                difs[m][n] = (double*)calloc(2,sizeof(double));
                for(o=0;o<2;o++)
                {
                    dif = (sys.params[m][n][o]-real.params[0][n][o])/d2r;
                    if( dif > 180 )
                        dif = dif - 360;
                    else if( dif < -180 )
                        dif = dif + 360;
                    difs[m][n][o] = dif;
                    if(o==0)
                        phirms[m] += pow(dif, 2.0);
                    else
                        psirms[m] += pow(dif, 2.0);
                }
            }
            phirms[m] /= (double)sz-1;
            phirms[m] = sqrt(phirms[m]);
            psirms[m] /= (double)sz-1;
            psirms[m] /= sqrt(psirms[m]);
        }
    }
    ////////////////////////// STDOUT ////////////////////////////////
    // Print frequencies for 1 run
    printf("\n\n");
    printf("res     CS (ppm) real| calc | diff       DC (Hz) real| calc | diff        CH (Hz)
real| calc | diff           RMS     \n");
    printf("-------------------------------------------------------------------------------------
--------------------------\n");
    for(m=0;m<sz-1;m++)
        printf("%3d:          %6.0lf|%6.0lf|%5.0lf             %6.0lf|%6.0lf|%5.0lf
%6.0lf|%6.0lf|%5.0lf        %6.2lf\n",m,sys.target[m][0], freq_calc[m][0], (freq_calc[m][0]-
sys.target[m][0]), sys.target[m][1], freq_calc[m][1], (freq_calc[m][1]-sys.target[m][1]),
sys.target[m][2], freq_calc[m][2], (freq_calc[m][2]-sys.target[m][2]), rmsdf[m][kb_max-1]);

    // Print CSRMS and DCRMS
    printf("\n\n");
    printf(" kb     CSRMS   DCRMS   CHRMS   CNRMS     PhiRMS   PsiRMS\n");
    printf("-----------------------------------------------------------\n");
    for(m=0;m<kb_max;m++)
        printf("%3d     %4.0lf    %4.0lf    %4.0lf    %4.0lf       %4.1lf
%4.1lf\n",m,CSRMS[m],DCRMS[m],CHRMS[m],CNRMS[m],phirms[m],psirms[m]);

    if( (rnd==1) || (rnd==2) || (rnd==3) )
    {
        // Print RAM RMS
        printf("\n\n");
        printf("Index          Phi real|  calc |  diff         Psi real|  calc |  diff\n");
        printf("-------------------------------------------------------------------------\n");
        for(m=0;m<sz-1;m++)
            printf("  %2d       %7.2lf|%7.2lf|%7.2lf
%7.2lf|%7.2lf|%7.2lf\n",m,real.params[0][m][0]/d2r,sys.params[0][m][0]/d2r,difs[kb_max-
1][m][0],real.params[0][m][1]/d2r,sys.params[0][m][1]/d2r,difs[kb_max-1][m][1]);
        printf("-------------------------------------------------------------------------\n");
        printf(" Phi RMS: %7.2lf    Psi RMS: %7.2lf\n",phirms[kb_max-1],psirms[kb_max-1]);
    }

    // Print runtime
    elapsed = ((double)end - (double)begin)/CLOCKS_PER_SEC;
    printf("\n\nElapsed time is %.3lf seconds\n",elapsed);

    ////////////////////////// FILE ////////////////////////////////
    if(file==1)
```

```
    {
        // angles.csv
        FILE * g = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\angles.csv","w");

fprintf(g,"HNCa,%lf\nNCCa,%lf\nHNCo,%lf\na_nc,%lf\ntetra,%lf\ntetid,%lf\n",HNCa/d2r,NCCa/d2r,HNCo
/d2r,a_nc/d2r,tetra/d2r,tetid/d2r);
        fprintf(g,"omN,%lf,%lf,%lf\n",gamma1/d2r,90.0,gamma2/d2r);
        fprintf(g,"omH,%lf,%lf,%lf\n",-90.0,-90.0,90.0);
        fprintf(g,"omHN,%lf,%lf,%lf\n",0.0,90.0,0.0);
        fprintf(g,"sigma,%lf,%lf,%lf\n",sigma[0],sigma[1],sigma[2]);
        fprintf(g,"sigmagly,%lf,%lf,%lf\n",sigmagly[0],sigmagly[1],sigmagly[2]);
        fprintf(g,"chi0,%lf\n",chi0);
        fprintf(g,"chi1,%lf\n",chi1);
        fprintf(g,"chi4,%lf",chi4);
        fclose(g);
        // Qs.csv
        if(kb_max==1)
        {
            g = fopen("C:\\Users\\joell\\Documents\\Nevzorov Lab\\Project2\\Output\\Qs.csv","w");
            for(m=0;m<sz;m++)
                fprintf(g,"%lf,%lf,%lf,%lf,%lf,%lf\n",Q_all->data[6*m],Q_all-
>data[(6*m)+1],Q_all->data[(6*m)+2],Q_all->data[(6*m)+3],Q_all->data[(6*m)+4],Q_all-
>data[(6*m)+5]);
            fclose(g);
        }
        // freqs_targ.csv
        g = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\freqs_targ.csv","w");
        for(m=0;m<sz;m++)

fprintf(g,"%lf,%lf,%lf,%lf\n",sys.target[m][0],sys.target[m][1],sys.target[m][2],sys.target[m][3]
);
        fclose(g);
        // real.csv
        if( (rnd==1) || (rnd==2) || (rnd==3) )
        {
            g = fopen("C:\\Users\\joell\\Documents\\Nevzorov
Lab\\Project2\\Output\\real.csv","w");
            fprintf(g,"#Parmaters:\n");
            fprintf(g,"%d,%d,\n",sz,1);
            for(m=0;m<sz;m++)
                fprintf(g,"%c,",real.restyp[m]);
            fprintf(g,"%c\n",real.restyp[sz]);
            fprintf(g,"%.10lf,%.10lf\n",real.THETA0,real.PHI0);
            for(m=0;m<sz-1;m++)
                fprintf(g,"%.10lf,",real.params[0][m][0]);
            fprintf(g,"\n");
            for(m=0;m<sz-1;m++)
                fprintf(g,"%.10lf,",real.params[0][m][1]);
            fclose(g);
        }
    }
}
```

Appendix C: Python code for CPU version of Roulette

```
import numpy as np
import scipy.signal as ss
import scipy.sparse as sp
import scipy.linalg as la
from copy import copy
from time import time
import threading
import queue
from math import factorial as fact
import os
import sys
spgen = sp.csc_matrix
cos = np.cos
```

```
sin = np.sin
pi = np.pi
d2r = pi/180
i = np.complex64(np.complex(0,1))
hbar=1.055e-34

class config():
    def __init__(self):
        self.v = {
                'ficoor':"./ala.txt",
                'xind':0,
                'dlim':3.0,
                'fisym':"./sym.inp",
                'fiRNG':"./RNG.inp",
                'new':1,
                'firefpath':"./",

                'fopath':"./",
                'foPlow':1,
                'foPlowraw':0,
                'fowid':0,
                'foreport':1,
                'foRNG':1,
                'fosgm':1,
                'foHS':0,
                'foconf':1,
                'shutdown':0,

                'gamS':-2.718E7,
                'gamI':2.675E8,
                'Hhml':[20E3,10E3,5E3],
                'sgmr':0,
                'sgmstd':1,
                'sigs':[2,-2,2,4,1,-3,2,3],

                'L':6,
                'SwrfA':[1,1,1,1,1,1],
                'IwrfA':[1,1,0,0,1,1],

                'temp':1,
                'Ws':[1,8],
                'Wab':3,
                'Wathr':-4,
                'Wap':1E5,

                'runs':1,
                'incr':200,
                'steps':100,

                'numln':[4,5,7],
                'fus':[1.0,1.0,1.0],

                'wrf':58.14E3,
                'W0':500E6,
                'Nm1':256,
                'dnu':0,
                'z':0.5

                }
        self.conf()
        self.sym = np.loadtxt(self.v['fisym'], dtype='int', delimiter=',')
        self.RNG = np.loadtxt(self.v['fiRNG'], dtype='float32', delimiter=',')
        self.readcoords()
        self.qc()

    def conf(self):
        with open("./roulette.conf", "r") as f:
            for line in f:
                ln = line.split()
                if ln[0]!='#':
                    if len(ln)!=3:
```

```
                        sys.exit("QCError: Please make sure all non commented lines in
configuration\
                                  file contain 3 fields: varname (type) value")
                    self.v[ln[0]] = ln[-1]
            f.seek(0)
            self.v['text'] = f.read()


    def readcoords(self):

        self.v['gamS'] = float(self.v['gamS']) # conversion
        self.v['gamI'] = float(self.v['gamI']) # conversion
        with open(self.v['ficoor'],'r') as f:
            f.readline()
            self.restyp=[];self.gamma=[];self.coords=[]
            for line in f:
                buf = line.split()
                self.restyp.append(buf[0])
                if 'H' in buf[0]:
                    self.gamma.append(self.v['gamI'])
                else:
                    self.gamma.append(self.v['gamS'])
                self.coords.append([float(m) for m in buf[-3:]])
        self.coords = np.array(self.coords)
        self.gamma = np.array(self.gamma)
        Nmax,_ = self.coords.shape

        self.v['xind'] = int(self.v['xind']) # conversion
        self.v['dlim'] = float(self.v['dlim']) # conversion
        inds = []
        for m in range(0,Nmax,1):
            r_jk = ((self.coords[self.v['xind'],0]-self.coords[m,0])**2 +
(self.coords[self.v['xind'],1]-self.coords[m,1])**2 + (self.coords[self.v['xind'],2]-
self.coords[m,2])**2)**0.5
            if r_jk<self.v['dlim']:
                inds.append(m)
        self.X = self.coords[inds]
        self.gamma = self.gamma[inds]
        self.N = len(inds)

    def qc(self):
        tick=0
        # Input options
        if os.path.isfile(self.v['ficoor'])==False:
            print("QCError: Structure file %s doesn't exist"%(self.v['ficoor']));tick+=1
        if (self.v['xind']<0) & (self.v['xind']>=self.N):
            print("QCError: Index of x-atom must be between 0-%d"%(self.N-1));tick+=1
        else:
            if self.restyp[self.v['xind']]=='H':
                print("QCError: Index of x-atom refers to an I spin");tick+=1
        if os.path.isfile(self.v['fisym'])==False:
            print("QCError: Symmetry file %s doesn't exist"%(self.v['fisym']));tick+=1
        if os.path.isfile(self.v['fiRNG'])==False:
            print("QCError: Symmetry file %s doesn't exist"%(self.v['fisym']));tick+=1
        self.v['new'] = bool(int(self.v['new'])) # conversion
        if self.v['new']==False:
            if os.path.isdir(self.v['firefpath'])==False:
                print("QCError: Path to refinement files %s is not a
directory"%(self.v['firefpath']));tick+=1

        # Output options
        if os.path.isdir(self.v['fopath'])==False:
            print('QCError: Output path %s is not a directory'%(self.v['fopath']));tick+=1
        self.v['foPlow'] = bool(int(self.v['foPlow'])) # conversion
        self.v['foPlowraw'] = bool(int(self.v['foPlowraw'])) # conversion
        self.v['fowid'] = bool(int(self.v['fowid'])) # conversion
        self.v['foreport'] = bool(int(self.v['foreport'])) # conversion
        self.v['foRNG'] = bool(int(self.v['foRNG'])) # conversion
        self.v['fosgm'] = bool(int(self.v['fosgm'])) # conversion
        self.v['foHS'] = bool(int(self.v['foHS'])) # conversion
        self.v['foconf'] = bool(int(self.v['foconf'])) # conversion
```

```
        self.v['shutdown'] = bool(int(self.v['shutdown'])) # conversion

        # Hamiltonian settings
        self.v['Hhml'] = np.sort([np.float32(m) for m in self.v['Hhml'].split(",")])[::-1] #
conversion
        if len(self.v['Hhml'])!=3:
            print('QCError: Max couplings list Hhml must contain 3 comma separated
floats');tick+=1
        self.v['sgmr'] = bool(int(self.v['sgmr'])) # conversion
        self.v['sgmstd'] = np.float32(self.v['sgmstd']) # conversion
        if self.v['sgmstd']<0:
            print('QCError: sgmstd must be >= 0');tick+=1
        self.v['sigs'] = np.array([float(m) for m in self.v['sigs'].split(",")]) # conversion
        if (self.v['sgmr']==False) & (len(self.v['sigs'])<self.N):
            print("QCError: For sgmr=0, sigs list must contain %d floats");tick+=1

        # PS Architecture
        self.v['L'] = int(self.v['L']) # Conversion
        self.v['SwrfA'] = [np.float32(m) for m in self.v['SwrfA'].split(",")] # conversion
        if len(self.v['SwrfA'])!=self.v['L']:
            print("QCError: S-channel wrf amplitude (SwrfA) must contain %d comma separated
booleans"%(self.v['L']));tick+=1
        self.v['IwrfA'] = [np.float32(m) for m in self.v['IwrfA'].split(",")] # conversion
        if len(self.v['IwrfA'])!=self.v['L']:
            print("QCError: I-channel wrf amplitude (IwrfA) must contain %d comma separated
booleans"%(self.v['L']));tick+=1
        if np.shape(self.RNG)[0]!=14:
            print("QCError: RNG input file must contain 14 rows (Top 7 -> floor vals; Bot 7 ->
ceil vals)");tick+=1
        if np.shape(self.RNG)[1]!=self.v['L']:
            print("QCError: RNG input file must contain %d columns (same as
L)"%(self.v["L"]));tick+=1
        if np.shape(self.sym)[0]!=7:
            print("QCError: sym input file must contain 7 rows");tick+=1
        if np.shape(self.sym)[1]!=self.v['L']:
            print("QCError: sym input file must contain %d columns (same as
L)"%(self.v["L"]));tick+=1

        # Temperature settings
        self.v['temp'] = int(self.v['temp']) # conversion
        if self.v['temp'] not in [0,1]:
            print("QCError: temp can only be 0 (schedule) or 1 (adaptive)");tick+=1
        self.v['Ws'] = [np.float32(m) for m in self.v['Ws'].split(",")] # conversion
        self.v['Wab'] = np.float32(self.v['Wab']) # conversion
        self.v['Wathr'] = np.float32(self.v['Wathr']) # conversion
        self.v['Wap'] = np.float32(self.v['Wap']) # conversion
        if self.v['temp']==0:
            if len(self.v['Ws'])!=2:
                print("QCError: Ws must contain 2 comma separated floats: T_start,T_end");tick+=1
        else:
            if self.v['Wab']<=0:
                print("QCError: Baseline adaptive temperature Wab must be > 0");tick+=1
            if self.v['Wap']<1:
                print("QCError: Adaptive temperature parameter Wap must be > 1");tick+=1

        # MC settings
        self.v['runs'] = int(self.v['runs']) # conversion
        if self.v['runs']<1:
            print("QCError: runs must be > 0");tick+=1
        self.v['incr'] = int(self.v['incr']) # conversion
        if self.v['incr']<1:
            print("QCError: incr must be > 0");tick+=1
        self.v['steps'] = int(self.v['steps']) # conversion
        if self.v['steps']<1:
            print("QCError: steps must be > 0");tick+=1

        # Score settings
        self.v['numln'] = [int(m) for m in self.v['numln'].split(",")] # conversion
        if sum(np.argsort(self.v['numln'])==[0,1,2])!=3:
            print("QCError: Number line cutoffs must be in ascending order");tick+=1
        self.v['fus'] = [float(m) for m in self.v['fus'].split(',')] # conversion
```

```
        # Miscellaneous options
        self.v['wrf'] = np.float32(self.v['wrf']) # conversion
        if self.v['wrf']<=0:
            print("QCError: wrf must be > 0");tick+=1
        self.v['W0'] = np.float32(self.v['W0']) # conversion
        if self.v['W0']<=0:
            print("QCError: Carrier frequency W0 must be > 0");tick+=1
        self.v['Nm1'] = int(self.v['Nm1']) # conversion
        if self.v['Nm1']<=0:
            print("QCError: t1 points Nm1 must be an int between 0 < Nm1 < 1024");tick+=1
        self.v['dnu'] = np.float32(self.v['dnu']) # conversion
        if self.v['dnu']<0:
            print("QCError: Linebroadening parameter dnu must be >= 0");tick+=1
        self.v['z'] = np.float32(self.v['z']) # conversion
        if self.v['z']<0:
            print("QCError: Score penalty coefficient z must be >= 0");tick+=1

        # sym.inp
        if (sum(0==self.sym[4])!=0) | (sum(1==self.sym[4])!=0):
            print("QCError: timing variable(s) assigned Full symmetry (0)\n");tick+=1
        if (sum(0==self.sym[5])!=0) | (sum(1==self.sym[6])!=0):
            print("QCError: rf amplitude variable(s) assigned Full symmetry (0)\n");tick+=1

        if tick>0:
            sys.exit("Please fix %d input error(s)"%(tick))

class scobj():
    def __init__(self):
        self.pm = 0
        self.pmraw = 1E10
        self.wid = 1E10
        self.phase = np.zeros((7, L))

    def copyfrom(self, src, phase=False):
        self.pm = src.pm
        self.pmraw = src.pmraw
        self.wid = src.wid
        if phase==True:
            self.phase = copy(src.phase)

def dagger(A):
    return A.conjugate().transpose()

def operator(spmat, pos, tot):
    dt = spmat.dtype
    if sp.issparse(spmat)==True:
        kron = sp.kron
        eye = sp.eye
    else:
        kron = np.kron
        eye = np.eye

    if pos==0:
        return kron(spmat, eye(2**(tot-1), dtype=dt) )
    elif pos==tot-1:
        return kron(eye(2**(tot-1), dtype=dt), spmat)
    else:
        return kron(kron(sp.eye(2**pos, dtype=dt), spmat), eye(2**(tot-pos-1), dtype=dt) )

def operator2(spmat1, spmat2, pos1, pos2, tot):
    kron = sp.kron if (sp.issparse(spmat1)==True) else np.kron
    ib = pos2-pos1
    mat1 = operator(spmat1, pos1, pos1+ib)
    mat2 = operator(spmat2, 0, tot-pos2)
    return kron(mat1, mat2)

def rotaxis(ang, u):
    R = np.zeros((3,3))
    u = u/np.linalg.norm(u)
    cos = np.cos(ang)
```

```
    sin = np.sin(ang)
    R[0,0] = cos + (u[0]**2)*(1-cos)
    R[1,1] = cos + (u[1]**2)*(1-cos)
    R[2,2] = cos + (u[2]**2)*(1-cos)
    R[0,1] = u[0]*u[1]*(1-cos) - u[2]*sin
    R[1,0] = u[0]*u[1]*(1-cos) + u[2]*sin
    R[0,2] = u[0]*u[2]*(1-cos) + u[1]*sin
    R[2,0] = u[0]*u[2]*(1-cos) - u[1]*sin
    R[1,2] = u[1]*u[2]*(1-cos) - u[0]*sin
    R[2,1] = u[1]*u[2]*(1-cos) + u[0]*sin
    return R

def findrots(coords, coup, axis):
    low = [1E10, -1, ()]
    for m in np.linspace(0,180,1000):
        (a, nmax, amax) = INTAXMat(coords, rot=m, axis=axis)
        dif = abs(coup-amax/2/pi)
        if dif<low[0]:
            low[:2] = [dif, m]
            low[-1] = (a, nmax, amax)
    return low

def INTAXMat(X, rot=0, axis=[1,0,0]):
    Y = X.dot(rotaxis(rot*d2r, axis))
    a = np.zeros((N,N))
    m=0
    while m<N:
        n = m
        while n<N:
            r_mn = ( (Y[m,0]-Y[n,0])**2 + (Y[m,1]-Y[n,1])**2 + (Y[m,2]-Y[n,2])**2 )**0.5
            if r_mn!=0:
                cos_theta = (Y[m,2]-Y[n,2])/r_mn
                a[m,n] = 1E-7*gamma[m]*gamma[n]*hbar*(3*cos_theta**2 - 1) / (r_mn*1E-10)**3
                a[n,m] = a[m,n]
            n+=1
        m+=1
    [a_max, N_max] = [np.max(abs(a[C.v['xind'],:])), np.argmax(abs(a[C.v['xind'],:]))]
    return a, N_max, a_max

def pade(A, q=4):

    tick = 0
    Apr = copy(A)

    # Step 1: If ||A||_1 < 0.5, proceed to step 4
    a1n = np.max(np.sum(abs(Apr), axis=0))
    if a1n>0.5:
        # Step 2: Determine minimum integer m such that ||A||_1 / m < 0.5
        # - find m = 2^n; divide original matrix by m, do n number of matrix multiplication in
step 6
        n=1
        m=2
        while (a1n/m)>0.5:
            n+=1
            m = 2**n
        m = 2**n
        # Step 3: Compute A' = A/m
        Apr /= m
        tick = 1

    # Step 4: compute Nq and Dq
    Nq = np.zeros((sz, sz), dtp)
    Dq = np.zeros((sz, sz), dtp)
    a1 = np.eye(sz, dtype=dtp)
    a2 = np.eye(sz, dtype=dtp)
    for j in range(q+1):
        mul = ( fact(2*q-j)*fact(q) ) / ( fact(2*q)*fact(j)*fact(q-j) )
        if j>0:
            a1 = a1.dot(Apr)
            a2 = a2.dot(-1*Apr)
        Nq += mul*a1
```

```
            Dq += mul*a2

        # Step 5: Compute e^A', which is denoted Rq
        Rq = la.inv(Dq).dot(Nq)

        # Step 6: If steps 2 and 3 were undertaken, compute e^A = (e^A')^n
        if tick==1:
            for _ in range(int(n)):
                Rq = Rq.dot(Rq)

        return Rq

    def cpinv(A):
        R = np.zeros(A.shape, A.dtype)
        Ar = A.real
        Ai = A.imag
        r0 = la.inv(Ar).dot(Ai)
        y00 = la.inv(Ai.dot(r0) + Ar)
        y10 = -1*r0.dot(y00)
        R.real = y00
        R.imag = y10
        return R

    def mostprom(freq, ft, start, end, MAXW):
        stepsz = freq[1]-freq[0]
        MIN = min(freq)
        maxw = int(np.ceil(MAXW/stepsz))

        # This could happen with large scf
        if MIN>start:
            return 0,0,1

        ind11 = int(np.floor((start-MIN)/stepsz))
        ind12 = int(np.ceil((end-MIN)/stepsz))
        inds1,dic1 = ss.find_peaks(ft[ind11:ind12], width=(0, maxw))
        if len(inds1)==0:
            pkh = 1E-10
            pkw = 1E10
            pkf = abs(end-start)/2
        else:
            ind1 = inds1[np.argmax(dic1['prominences'])]+ind11
            pkw = stepsz*ss.peak_widths(ft, [ind1], rel_height=0.5)[0][0]
            pkh = ft[ind1]
            pkf = freq[ind1]

        return pkh, pkw, pkf

    def propg(dev, steps, sx, rho, exD1, exD2):
        g = np.zeros((steps), dtype=dtp)
        exD1i = dagger(exD1)
        exD2i = dagger(exD2)

        hold   = np.empty((sz,sz), dtype=dtp)

        flag = 0
        for n1 in range(steps):
            hold = np.dot(sx, rho)
            g[n1] = np.trace(hold)
            if flag==0:
                hold = np.dot(rho, exD1i)
                rho = np.dot(exD1 , hold)
                flag = 1
            else:
                hold = np.dot(rho, exD2i)
                rho = np.dot(exD2 , hold)
                flag = 0

        return g

    def propgeo(dev, steps, Det, rho, exD1, Q):
        g = np.zeros((steps), dtype=dtp)
```

```
    exD1i = dagger(exD1).astype(dtp)

#        print("Running on device %d (%3.1f%% full)"%(cp.cuda.device.get_device_id(),
(mp.used_bytes()/mp.total_bytes())*100))
    for n1 in range(steps):
        g[n1] = np.trace( Det.dot(rho) )
        rho = exD1.dot(rho).dot(exD1i)

    Q.put(g)

def expmdev(dev, A, q):

    SZ,_ = A.shape

    tick = 0
    Apr = A

    # Step 1: If ||A||_1 < 0.5, proceed to step 4
    a1n = np.max(np.sum(abs(A), axis=0))
    if a1n>0.5:
        # Step 2: Determine minimum integer m such that ||A||_1 / m < 0.5
        # - find m = 2^n; divide original matrix by m, do n number of matrix multiplication in
step 6
        n=1
        m=2
        while (a1n/m)>0.5:
            n+=1
            m = 2**n
        m = 2**n
        # Step 3: Compute A' = A/m
        Apr = (Apr/m).astype(dtp)
        tick = 1

    Ad = np.array(Apr, dtype=dtp)
    hold1 = np.zeros((SZ,SZ), dtype=dtp)
#        print("Running on device %d (%3.1f%% full)"%(cp.cuda.device.get_device_id(),
(mp.used_bytes()/mp.total_bytes())*100))
    for i in range(q+1):
        hold2 = np.eye(SZ, dtype=dtp)
        if i>0:
            for j in range(i):
                hold2 = hold2.dot(Ad)
        hold2 /= fact(i)
        hold1 += hold2

    if tick==1:
        for _ in range(int(n)):
           hold1 = hold1.dot(hold1)



    return hold1

def expmdev2(dev, A, q, Q):
    M_cycle, N_cycle = cur.phase[:2].shape#phase1.shape
    exD = np.eye(sz, dtype=dtp)

    for k in range(N_cycle):
        tick = 0
        Apr = A[k].todense().astype(dtp)

        # Step 1: If ||A||_1 < 0.5, proceed to step 4
        a1n = np.max(np.sum(abs(Apr), axis=0))
        if a1n>0.5:
            # Step 2: Determine minimum integer m such that ||A||_1 / m < 0.5
            # - find m = 2^n; divide original matrix by m, do n number of matrix multiplication
in step 6
            n=1
            m=2
            while (a1n/m)>0.5:
```

```
                n+=1
                m = 2**n
            m = 2**n
            # Step 3: Compute A' = A/m
            Apr /= m
            tick = 1

        Ad = np.array(Apr, dtype=dtp)

        hold1 = np.zeros((sz,sz), dtype=dtp)
        for i in range(q+1):
            hold2 = np.eye(sz, dtype=dtp)
            if i>0:
                for j in range(i):
                    hold2 = hold2.dot(Ad)
            hold2 /= fact(i)
            hold1 += hold2

        if tick==1:
            for _ in range(int(n)):
               hold1 = hold1.dot(hold1)

#             if k==0:
#                 print("     Running on device %d (%3.1f%%
full)"%(cp.cuda.device.get_device_id(), (mp.used_bytes()/mp.total_bytes())*100))

        exD = hold1.dot(exD)

    Q.put(exD)

def pretot():

    sgm = 2*pi*W0*1E-6*sGm

    # SETTING UP AND CALCULATION OF THE INTERACTION MATRICES
    Iz = spgen(0.5*np.array([[1,0],[0,-1]], dtype=dtp))
    Iy = spgen((1/(2*i))*np.array([[0,1],[-1,0]], dtype=dtp))
    Ix = spgen(0.5*np.array([[0,1],[1,0]], dtype=dtp))
    Ip = spgen(np.array([[0,1],[0,0]], dtype=dtp))
    Im = dagger(Ip).tocsc()

    Sx = spgen((sz,sz), dtype=dtp);Sy = spgen((sz,sz), dtype=dtp);Sz = spgen((sz,sz), dtype=dtp)
    Hsgm = spgen((sz,sz), dtype=dtp)
    Ixtot = spgen((sz, sz), dtype=dtp);Iytot = spgen((sz, sz), dtype=dtp);Iztot = spgen((sz, sz),
dtype=dtp)
    Hdiplike = spgen((sz,sz), dtype=dtp);Hdipunlike = spgen((sz,sz), dtype=dtp);Hdiptot =
spgen((sz, sz), dtype=dtp)

    m=0
    while m<N:
        Iyk = operator(Iy, m, N)
        Izk = operator(Iz, m, N)
        Ixk = operator(Ix, m, N)
        if gamma[m]==C.v['gamI']: # Proton
            Ixtot += Ixk
            Iytot += Iyk
            Iztot += Izk
        elif gamma[m]==C.v['gamS']:
            Sx += Ixk
            Sy += Iyk
            Sz += Izk
        Hsgm += sgm[m]*Izk
        n=m+1
        while n<N:
            if gamma[m]==gamma[n]: # Homonuclear
                Hjk = operator2(Iz,Iz,m,n,N)-0.25*(operator2(Ip,Im,m,n,N)+operator2(Im,Ip,m,n,N))
                Hdiplike += a[m,n]*Hjk
            else: # Heteronuclear
                Hjk = operator2(Iz, Iz, m, n, N)
                Hdipunlike += a[m,n]*Hjk
            Hdiptot += a[m,n]*Hjk
```

```
            n+=1
        m+=1

    Sd = operator(Ix, C.v['xind'], N)

    return Hdiptot, Hsgm, Sx, Sy, Ixtot, Iytot, Iztot, Sd

def PHASE(Hdiptot, phase, t, w1):
    Hall = []
    M_cycle, N_cycle = phase.shape
    for m_cycle in range(N_cycle):

        p1 = phase[0,m_cycle]
        p2 = phase[1,m_cycle]
        HrfX = np.cos((pi/2)*p1)*Sx + np.sin((pi/2)*p1)*Sy
        HrfH = np.cos((pi/2)*p2)*Ixtot + np.sin((pi/2)*p2)*Iytot

        H_cycle = (dW[m_cycle]*Iztot) + Hsgm + Hdiptot + (w1[0,m_cycle]*w_rf*HrfX) +
(w1[1,m_cycle]*w_rf*HrfH)
        Hall.append(-i*t90*t[m_cycle]*H_cycle)

    return Hall

def getXD(Hall1, Hall2):
    outq1 = queue.Queue();outq2 = queue.Queue()
    t1 = threading.Thread(target=expmdev2, args=(0, Hall1, 7, outq1))
    t2 = threading.Thread(target=expmdev2, args=(1, Hall2, 7, outq2))
    t1.start();t2.start()
    t1.join();t2.join()
    exDtot1 = outq1.get()
    exDtot2 = outq2.get()
    return exDtot1, exDtot2

def allinone(Hdip, sf, ef):
    Hall1 = PHASE(Hdip, cur.phase[:2], cur.phase[4], cur.phase[5:])
    Hall2 = PHASE(Hdip, cur.phase[2:4], cur.phase[4], cur.phase[5:])

    exDtot1, exDtot2 = getXD(Hall1, Hall2)

    # Normalized initial density matrix
    rho01 = exDtot1.dot(rho00).dot(dagger(exDtot1))
    exDe = exDtot2.dot(exDtot1)
    exDo = exDtot1.dot(exDtot2)

    outq1 = queue.Queue();outq2 = queue.Queue()
    t1 = threading.Thread(target=propgeo, args=(0, Nm1//2, rho_det, rho00, exDe, outq1))
    t2 = threading.Thread(target=propgeo, args=(1, Nm1//2, rho_det, rho01, exDo, outq2))
    t1.start();t2.start()
    t1.join();t2.join()

    Gee = np.zeros((Nm1), dtype=dtp)
    Gee[:Nm1:2] = outq1.get()
    Gee[1:Nm1:2] = outq2.get()

    Nzf = 1025 # zerofill size
    Gee[0] /= 2
    Gee = np.append(Gee, np.zeros((Nzf-Nm1)))

    Nt, = Gee.shape
    dwell = t90*sum(cur.phase[4])
    timedom = np.linspace(0, (Nt-1)*dwell, Nt)
    # Apodization
    G_conv = np.multiply(Gee,np.exp(-timedom*pi*dnu))
    FT = np.fft.fftshift(np.fft.fft(G_conv)).real
    FT *= (1E4*dwell if dcorr==1 else 1)
    FREQ = np.linspace(-1/2/(dwell*scf), 1/2/(dwell*scf), Nzf)
    # NORMALIZATION
#     spac = FREQ[1]-FREQ[0]
#     FT *= 1E4/(spac*sum(((FT[:-1:2])+(FT[1::2]))))
```

```
    if sf=='max': #edit
        sf = min(FREQ)+100
    p, pkw, pf = mostprom(FREQ, FT, sf, ef, 700)
#    if np.argmax(FT)==512:
#        p*=1E-10
    return p, pkw, pf# edit

def run3():
    pm0,pw0,pf0 = allinone(Hdip0, 'max', -1000)
    pm0 = (1E-10 if pm0<0 else pm0)
    pw0 *= abs((amh/2)/pf0)
    pm1,pw1,pf1 = allinone(Hdip1, 'max', -1000)
    pm1 = (1E-10 if pm1<0 else pm1)
    pw1 *= abs((amm/2)/pf1)
    pm2,pw2,pf2 = allinone(Hdip2, 'max', -1000)
    pm2 = (1E-10 if pm2<0 else pm2)
    pw2 *= abs((aml/2)/pf2)

    fac = abs(ratio1 - (pf0/pf1))
    fac += abs(ratio2 - (pf1/pf2))
    fac = 1 - z*fac

    width = (pw0*pw1*pw2)**0.33

    raw = (pm0*pm1*pm2)**0.33

    return -1*fac*raw, raw, width

def AccRej(arg):
    if arg==0:
        # Alternatively phase1 = copy(hold.phase[:2]); phase2 = copy(hold.phase[2:4]); t =
copy(phase[4])
        if (sym[ind1, ind2]==0) | (sym[ind1, ind2]==1):
            cur.phase[ind1%2,     [ind2, -(ind2+1)]] = hold.phase[ind1%2,     [ind2, -(ind2+1)]]
            cur.phase[(ind1%2)+2, [ind2, -(ind2+1)]] = hold.phase[(ind1%2)+2, [ind2, -(ind2+1)]]
        elif (sym[ind1, ind2]==2) | (sym[ind1, ind2]==3):
            cur.phase[ind1, [ind2, -(ind2+1)]] = hold.phase[ind1, [ind2, -(ind2+1)]]
        elif (sym[ind1, ind2]==4) | (sym[ind1, ind2]==5):
            cur.phase[[(ind1%2), (ind1%2)+2], ind2] = hold.phase[[(ind1%2), (ind1%2)+2], ind2]
        else:
            cur.phase[ind1, ind2] = hold.phase[ind1, ind2]
    else:
        # Alternatively hold.phase = np.row_stack((phase1,phase2,t))
        if (sym[ind1, ind2]==0) | (sym[ind1, ind2]==1):
            hold.phase[ind1%2,     [ind2, -(ind2+1)]] = cur.phase[ind1%2,     [ind2, -(ind2+1)]]
            hold.phase[(ind1%2)+2, [ind2, -(ind2+1)]] = cur.phase[(ind1%2)+2, [ind2, -(ind2+1)]]
        elif (sym[ind1, ind2]==2) | (sym[ind1, ind2]==3):
            hold.phase[ind1, [ind2, -(ind2+1)]] = cur.phase[ind1, [ind2, -(ind2+1)]]
        elif (sym[ind1, ind2]==4) | (sym[ind1, ind2]==5):
            hold.phase[[(ind1%2), (ind1%2)+2], ind2] = cur.phase[[(ind1%2), (ind1%2)+2], ind2]
        else:
            hold.phase[ind1, ind2] = cur.phase[ind1, ind2]

def fsps(arg):
    rnd = (np.random.uniform(RNG[ind1,ind2,0], RNG[ind1,ind2,1]) if ind1<5 else (1 if
cur.phase[ind1,ind2]==0 else 0)) # change if w1 continuous
    rnd2 = ( (rnd-2) if (rnd>2) else (4+rnd-2) )
    if (arg==0) | (arg==1): # full (anti)symmetry
        rnd2 = rnd if arg==1 else rnd2
        cur.phase[(ind1%2), [ind2, -(ind2+1)]] = [rnd, rnd2]
        cur.phase[(ind1%2)+2, [ind2, -(ind2+1)]] = [rnd2, rnd]
    elif (arg==2) | (arg==3): # partial (anti)symmetry between mirrored subdwells
        rnd2 = rnd if arg==3 else rnd2
        cur.phase[ind1, [ind2, -(ind2+1)]] = [rnd, rnd2]
    elif (arg==4) | (arg==5): # partial (anti)symmetry between odd and even dwell
        rnd2 = rnd if arg==5 else rnd2
        cur.phase[[(ind1%2),(ind1%2)+2], ind2] = [rnd, rnd2]
    else: # no symmetry
        cur.phase[ind1, ind2] = rnd

def numln(ran):
```

```
    if ran < C.v['numln'][0]:
        ind = np.random.randint(4)
        vtyp = 0
    elif ran < C.v['numln'][1]:
        ind = 4
        vtyp = 1
    else:
        ind = 5+np.random.randint(2)
        vtyp = 2
    return ind, vtyp

def sendmsg():
    import smtplib
    from email.mime.text import MIMEText
    from email.mime.multipart import MIMEMultipart
    from email.mime.base import MIMEBase

    email_sender = 'jlapin@ncsu.edu'
    email_receiver = 'jlapin@ncsu.edu'
    subject = 'Simulation'
    msg = MIMEMultipart()
    msg['From'] = email_sender
    msg['To'] = email_receiver
    msg['Subject'] = subject

    body = 'Runtime = %f'%(time()-START)
    msg.attach(MIMEText(body, 'plain'))

    attachment = open(C.v['fopath']+"report.txt", 'r')
    part = MIMEBase('application', 'octet_stream')
    part.set_payload(attachment.read())
    attachment.close()
    part.add_header('Content-Disposition', "attachment; filename = %s"%('report.txt'))

    msg.attach(part)
    text = msg.as_string()

    connection = smtplib.SMTP('smtp.gmail.com', 587)
    connection.starttls()
    connection.login(email_sender, 'ktckbpspgcldjtfe')
    connection.sendmail(email_sender, email_receiver, text)
    connection.quit()
rep = []
dtp = 'complex64'
C = config()
if C.v['foconf']==True:
    with open(C.v['fopath']+"configfile",'w') as f:
        f.write(C.v['text'])
X = C.X#X[inds]
N = C.N#len(inds)
sz = 2**N
rep.append("N = %d\n"%(N))
print(rep[-1])

# Interaction parameters for NH system
gamma = C.gamma

############################

# PHASES, OFFSETS, AMPLITUDES, AND DURATIONS FOR THE PULSE SEQUENCE
scf = 1.0 # Scaling factor relative to the apparent dwell
L = C.v['L'] # Number of subdwells
dW = np.zeros((L), dtype='float32') # proton offsets

# Score objects
cur = scobj()
hold = scobj()
low = scobj()
raw = scobj()
wSO = scobj()
```

```
# rf amplitude
w_rf = 2*pi*C.v['wrf']
t90 = pi/(2*w_rf)

# Miscellaneous simulation parameters
Nm1 = C.v['Nm1']   # number of data points
W0 = C.v['W0']  # carrier frequency
SGM = C.v['sgmstd']    # sigma coefficient
dnu = np.float32(C.v['dnu'])     # linebroadening, Hz
z = C.v['z'] # ratio penalty coefficient
fus = C.v['fus'] # uphill facilitator
dcorr = 1   # dwell intensity correction

# High, middle, low
[amh,amm,aml] = [C.v['Hhml'][0], C.v['Hhml'][1], C.v['Hhml'][2]]
[difh, roth, (ah, Nmaxh, amh)] = findrots(X, C.v['Hhml'][0], axis=[0,1,0])
rep.append("High Coupling Target: %5d Hz --> Found %5d Hz at %6.2f rotation about [%d %d %d]
axis"%(C.v['Hhml'][0], amh/2/pi, roth, 0, 1, 0))
print(rep[-1])
[difm, rotm, (am, Nmaxm, amm)] = findrots(X, C.v['Hhml'][1], axis=[0,1,0])
rep.append(" Mid Coupling Target: %5d Hz --> Found %5d Hz at %6.2f rotation about [%d %d %d]
axis"%(C.v['Hhml'][1], amm/2/pi, rotm, 0, 1, 0))
print(rep[-1])
[difl, rotl, (al, Nmaxl, aml)] = findrots(X, C.v['Hhml'][2], axis=[0,1,0])
rep.append(" Low Coupling Target: %5d Hz --> Found %5d Hz at %6.2f rotation about [%d %d %d]
axis\n"%(C.v['Hhml'][2], aml/2/pi, rotl, 0, 1, 0))
print(rep[-1])

# Where (or in what ratios) to expect the signals
[amh,amm,aml] = [amh/2/pi, amm/2/pi, aml/2/pi]
ratio1 = amh/amm
ratio2 = amm/aml

# Simulation search restrictions, parameters, etc.
dim1 = L
sym = C.sym[:,:dim1]
RNG = np.zeros((7, dim1, 2)) # Sampling range
RNG[:,:,0] = C.RNG[:7,:dim1]
RNG[:,:,1] = C.RNG[7:,:dim1]

# Refinement or new optimization
new = C.v['new']
if new==False:
    # Grab all csv files in the "Loop/" directory
    files = [m for m in os.listdir(C.v['firefpath']) if m[-3:]=='csv']
    runs = len(files)
else:
    runs = C.v['runs']
    files = []
pms = np.zeros((runs, 4));pms[:,0] = np.arange(runs);allsgm = np.zeros((runs,N))

####################
## Start the MCSA ##
####################
START = time()
for l in range(runs):
    if new==False:
        cur.phase = np.loadtxt(C.v['firefpath']+"%s"%(files[l]), delimiter=",", skiprows=1)
    else:
        files.append("Plow%d.csv"%(l))
        for m in range(5):
            for n in range(L):
                cur.phase[m,n] = np.random.uniform(RNG[m,n%dim1,0], RNG[m,n%dim1,1])
                cur.phase[5,n] = C.v['SwrfA'][n]
                cur.phase[6,n] = C.v['IwrfA'][n]

    # Setup hamiltonians to be used for various sgm, amax
    if C.v['sgmr']==True:
        sGm = np.random.normal(0, SGM, (N,))
    else:
        sGm = SGM*C.v['sigs'][:N]
```

```
    allsgm[l] = sGm
    [a, N_max, a_max] = [ah, Nmaxh, amh]
    Hdip0, Hsgm, Sx, Sy, Ixtot, Iytot, Iztot, Sd = pretot()
    [a, N_max, a_max] = [am, Nmaxm, amm]
    Hdip1, _, _, _, _, _, _, _ = pretot()
    [a, N_max, a_max] = [al, Nmaxl, aml]
    Hdip2, _, _, _, _, _, _, _ = pretot()
    rho_det = Sd.todense()
    rho00 = ((Sd-Ixtot)/(2**(N-1))).todense()

    # Initialize score objects
    cur.pm,cur.pmraw,cur.wid = run3()
    hold.copyfrom(cur, phase=True)
    low.copyfrom(cur, phase=True) # best real score
    wSO.copyfrom(cur, phase=True);wSO.wid = 1E10 # lowest width
    raw.copyfrom(cur, phase=True) # best raw score

    Temp = C.v['temp'] # 0 for schedule; 1 for adaptive
    incr  = C.v['incr']
    steps = C.v['steps']
    WW = (np.linspace(C.v['Ws'][0], C.v['Ws'][1], incr, dtype='float32') if Temp==0 else
C.v['Wab']);W = WW
    scors = np.zeros((incr, 2));hs = np.zeros((7,dim1));mags = np.zeros((2,7,dim1));mags[1]=1E-10
    for m in range(incr): # Temperature loop
        if Temp==0:
            W = WW[m]
        print("m = %d; l = %d (%s)\nW = %f"%(m,l,files[l],W))
        accepth = 0
        acceptl = 0
        reject = 0
        desum = 0

        for n in range(steps): # random steps loop
#             sys.stdout.write("\r%d/%d"%(n,steps))
            # Choose random index (ind1, ind2); Generate random values rnd, rnd2
            ind1,vtyp = numln( np.random.randint(C.v['numln'][-1]) ) # phase or timing
            ind2 = np.random.randint(0, dim1) # which subdwell
            if (ind1<4) & (sum(cur.phase[(ind1%2)+5])!=0):
                cnt=0
                while (cur.phase[(ind1%2)+5,ind2]==0) & (cnt<50):
                    ind2 = np.random.randint(0, dim1)
                    cnt+=1

            # Impose symmetry
            fsps(sym[ind1, ind2]) # function automatically operates on cur
            # cur.phase[5:] = np.round(cur.phase[5:])

            # Score new sequence
            cur.pm,cur.pmraw,cur.wid = run3()
            dE = cur.pm - hold.pm

            # HS statistics
            if (C.v['foHS']==True) & (dE!=0):
                mags[0,ind1,ind2]+=abs(dE);mags[1,ind1,ind2]+=1
            # Facilitate uphill steps
            if dE>0:
                dE*=fus[vtyp]
            # Adaptive temperature variable
            if Temp==1:
                desum+=(dE if dE>0 else 0)
            # Record keeping
            if cur.pm < low.pm: # Lowest score
                low.copyfrom(cur, phase=True)
            if (cur.pm < C.v['Wathr']) & (cur.pmraw > raw.pmraw): # lowest raw score
                raw.copyfrom(cur, phase=True)
            if (cur.pm < C.v['Wathr']) & (cur.wid < wSO.wid): # lowest width
                wSO.copyfrom(cur, phase=True)

            # Update cur and hold states
            if dE<0: # if less -> accept
                hold.copyfrom(cur)
```

```
                AccRej(1)
                acceptl += 1
            elif dE==0: # if equal do a coin flip
                if np.random.rand()>0.5: # accept coin flip
                    hold.copyfrom(cur)
                    AccRej(1)
                    accepth += 1
                    if C.v['foHS']==True:
                        hs[ind1,ind2] += 1
                else: # reject coin flip
                    AccRej(0)
                    reject += 1
            else: # apply metropolis criterion
                if np.exp(-dE*W)>np.random.rand(): # accept metropolis
                    hold.copyfrom(cur)
                    AccRej(1)
                    accepth += 1
                    if C.v['foHS']==True:
                        hs[ind1,ind2] += 1
                else: # reject metropolis
                    AccRej(0)
                    reject += 1

        # End of temp increment
        # print status
        print("   Steps accepted (+|=): %2d"%(accepth))
        print("   Steps accepted (-)  : %2d"%(acceptl))
        print("   Steps rejected      : %2d"%(reject))
        print("   Current score/raw/wid: %7.3f (%6.3f/%4d)"%(hold.pm, hold.pmraw, hold.wid))
        print("   Low score/raw/wid    : %7.3f (%6.3f/%4d)"%(low.pm, low.pmraw, low.wid))
        print("   Global low raw/wid   :         (%6.3f/%4d)\n"%(raw.pmraw, wSO.wid))
        # Save Plow after every increment
        if C.v['foPlow']==True:
            np.savetxt(C.v['fopath']+"%s"%(files[l]), low.phase, delimiter=",",
                   header='%s; sgmstd=%d; w_rf=%.2f; dnu=%.0f; pm/raw/wid=%.3f/%.3f/%d'%(
                           os.path.split(sys.argv[0])[1],SGM, w_rf/2000/pi, dnu, low.pm,
low.pmraw, low.wid))
        scors[m,0] = hold.pm;scors[m,1] = low.pm

        """
        update temperature
        - Logarithm must be tuned to accept a comfortable number of higher
           energy steps, but not so much so that it pops out of deep energy wells
        """
        if Temp==1:
            desum/=steps-acceptl
            W = (C.v['Wab'] if hold.pm>C.v['Wathr'] else -np.log(C.v['Wap']**-1)/desum)

    # End of run
    # Save scores for the run
    pms[l,1] = low.pm;pms[l,2] = low.pmraw;pms[l,3] = low.wid
    # Print out other lows or statistics
    if C.v['foPlowraw']==True: # low raw score parameters
        if raw.pmraw!=low.pmraw:
            np.savetxt(C.v['fopath']+'raw%s'%(files[l]), raw.phase, delimiter=",",
                       header='%s; sgmstd=%d; w_rf=%.2f; dnu=%.0f; pm/raw/wid=%.3f/%.3f/%d'%(
                               os.path.split(sys.argv[0])[1],SGM, w_rf/2000/pi, dnu, raw.pm,
raw.pmraw, raw.wid))
    if C.v['fowid']==True: # low width parameters
        if wSO.wid!=low.wid:
            np.savetxt(C.v['fopath']+"wid%s"%(files[l]), wSO.phase, delimiter=",",
                       header='%s; sgmstd=%d; w_rf=%.2f; dnu=%.0f; pm/raw/wid=%.3f/%.3f/%d'%(
                               os.path.split(sys.argv[0])[1],SGM, w_rf/2000/pi, dnu, wSO.pm,
wSO.pmraw, wSO.wid))
    if C.v['foHS']==True: # Uphill statistics: magnitudes and # of high steps
        np.savetxt(C.v['fopath']+"mags%d.csv"%(l), np.divide(mags[0],mags[1]), delimiter=",",
fmt="%.3f")
        np.savetxt(C.v['fopath']+"hs%d.csv"%(l), hs, delimiter=",", fmt="%d")
# End of program
if C.v['shutdown']==True: # shutdown computer
    os.system("shutdown /s /t 60")
```

```
if C.v['foRNG']==True: # save random number ranges for variables
    np.savetxt(C.v['fopath']+"RNG.txt", np.row_stack((RNG[:,:,0], RNG[:,:,1])), fmt='%.1f',
delimiter=",")
if bool(int(C.v['fosgm']))==True: # save sigmas for reproduceability
    np.savetxt(C.v['fopath']+"sgm.txt", allsgm, fmt='%.3f', delimiter=",")
if bool(int(C.v['foreport']))==True: # save report with scores and misc. info
    rep.append("Total runtime: %.2f"%(time()-START))
    header="\n".join(rep)+"\n"
    for m in files:
        header+=m+" "
    np.savetxt(C.v['fopath']+"report.txt", pms, fmt='%5d %7.2f %7.2f %7.0f', delimiter=" ",
header=header)
#np.savetxt("C:/Users/joell/Documents/Python/Nevzorov/SPIN/scors.csv", scors, fmt='%f',
delimiter=",")
sendmsg() # send email of report
print("\nTotal runtime: %.2f"%(time()-START))
```

Appendix D: Sample configuration (.conf) script for Roulette

Input options
ficoor (str) C:/Users/joell/Documents/Python/Nevzorov/SPINCPU/CA10.txt
xind (int) 2
dlim (float) 2.5
fisym (str) C:/Users/joell/Documents/Python/Nevzorov/SPINCPU/sym.inp
fiRNG (str) C:/Users/joell/Documents/Python/Nevzorov/SPINCPU/RNG.inp
new (bool) 1
firefpath (str)
C:/Users/joell/Documents/Python/Nevzorov/SPINCPU/Loop/refine/
Output options
fopath (str) C:/Users/joell/Documents/Python/Nevzorov/SPINCPU/Loop/
foPlow (bool) 1
foPlowraw (bool) 0
fowid (bool) 1
foreport (bool) 1
foRNG (bool) 1
fosgm (bool) 1
foHS (bool) 0
foconf (bool) 1
shutdown (bool) 0
Hamiltonian settings
gamS (float) 6.72828E7
gamI (float) 2.675E8
Hhml (list(float)) 20E3,10E3,5E3
sgmr (bool) 0
sgmstd (float) 1
sigs (list(float)) 2,-2,2,4,1,-3,2,3,-1,-2
PS architecture
L (int) 6
SwrfA (list(bool)) 1,1,1,1,1,1
IwrfA (list(bool)) 1,1,1,1,1,1
Temperature settings
temp (int) 1

Ws (list(float))	1,8
Wab (float)	6
Wathr (float)	-1.0
Wap (float)	5E6
# MC settings	
runs (int)	5
incr (int)	200
steps (int)	100
# Scoring settings	
numln list(int)	4,5,7
fus (float)	1.0,1.0,0.7
# Miscellaneous options	
wrf (float)	75E3
W0 (float)	300E6
Nm1 (int)	256
dnu (float)	0
z (float)	0.5

Appendix E: Sample RNG.inp input script for Roulette

```
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.8,0.8,0.8,0.8,0.8,0.8
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
4.0,4.0,4.0,4.0,4.0,4.0
4.0,4.0,4.0,4.0,4.0,4.0
4.0,4.0,4.0,4.0,4.0,4.0
4.0,4.0,4.0,4.0,4.0,4.0
4.0,4.0,4.0,4.0,4.0,4.0
1.0,1.0,1.0,1.0,1.0,1.0
1.0,1.0,1.0,1.0,1.0,1.0
```

Appendix F: Sample sym.inp input script for Roulette

```
2,2,2,2,2,2
2,2,2,2,2,2
2,2,2,2,2,2
2,2,2,2,2,2
3,3,3,3,3,3
3,3,3,3,3,3
3,3,3,3,3,3
```