NMR structure calculation using XPLOR-NIH as a modern molecular dynamic's engine for ARIA2 - ARIAXC.

A Thesis to the University of Kent for the degree of

Master's in Science by Research

School of Biosciences

By Luke Richard John Packer

May 2025

140 pages / approx. 39,030 words

Declaration

No part of this thesis has been submitted in support of an application for any degree or other qualification of the University of Kent or any other University or Institution of learning.

Acknowledgments

Here I would like to acknowledge the people that helped me complete and finish this piece of work. I would like to start with my main supervisor in this Master's project Dr. Gary Thompson, without whom I would have been completely lost on many occasions with ideas for work and in many instances a guiding hand in the project itself. Following this I would like to acknowledge my partner, Caroline Harriman (PhD candidate at the University of Kent), for her continued support and unending patience and love throughout this process, without whom I would be truly at my wits end and would not have been able to complete this course.

Next, I would like to thank a couple of people without this project would not have been able to progress. Firstly, Benjamin Bardiaux, and the teams developing CNS and XPLOR-NIH, who all gave me permission to work on their respective programs to further this field of study.

Following this, I would like to give thanks to Dr. Jose Ortega-Roldan and his Laboratory team of 2021 (Dr. Joe, Vicky, Eleanor and Lucas) who all helped give advice and made long days through the extended lockdown due to COVID more bearable. To my family which supported me through this project and could only ever wish the best for me. Finally, I would like to give thanks to my friends with which I had many laughs and memories together throughout my time during this project.

Thank you all.

Abstract

The primary objective of this project was to enhance ARIA2 structure calculation protocol by integrating a more advanced molecular dynamics engine and scripting interface (python), utilising the molecular dynamics program XPLOR. This update aims to improve ARIA2's relevance and applicability to contemporary methods of structural calculation by incorporating the larger variety of structural constraints available in XPLOR such as PREs, SAX data and chemical shift forcefields, .

ARIA2 is an iterative structure calculation program that generates structures based on the most common NMR measurement available, nOes (Nuclear Overhauser Effect), calculating structures and undertaking automatic nOe assignment, using either cartesian or torsion angle, dynamics. However, CNS the molecular dynamics engine that powers ARIA2 is no longer developed and does not have support for the latest molecular dynamics protocols.

A new version ARIA2 ARIA-XC has been developed which uses CNS's parent program XPLOR, which continues to be developed as XPLOR-NIH and contains more modern force fields for working with PREs, Solvent PREs, SAX envelopes and chemical shifts that are not available in CNS. As the scripting interface for XPLOR - NIH is Python2.7 the protocols used by ARIA-XC have been converted to use python as their programming interface. This has further advantages as Python, as opposed to XPLORscipt, is a widely used language when coding programs and is the main scripting language for ARIA.

Side by side comparison between ARIA-XC and ARIA has been carried out as the programs were debugged to ensure that the results from calculations were consistent. This has been carried out using the Werner Tudor domain test set of restraints from ARIA and was supported by the development of the test program RollinStone and a new compassion technique for structure calculation called Structure Injection.

Finally, results from comparisons of the Werner test set show identical structures being calculate with ARIA-XC albeit with the requirement to currently use longer simulated annealing periods.

Abbreviations and definitions

CNS – (Crystallographic and NMR System).

ARIA2 – (Automated Restraint and Iterative Assignment).

XPLOR-NIH – (XPLOR since being develop by NIH).

XPLOR – (A System for X-ray Crystallography and NMR).

ARIAXC - (Automated Restraint and Iterative Assignment XPLOR Calculation/Combined).

IVM – (Internal variable module) Allows XPLOR-NIH is a flexible molecular dynamics package and minimisation between.

XML – (Extensible Markup Language) which file format for storing, transmitting, and reconstructing arbitrary data which cannot perform computing operating serving only data management.

TCL – (is a high-level, general-purpose, interpreted, object-oriented, dynamic programming language, commonly used as a scripting language which is used frequency embedding into C applications.

JSON – (JavaScript Object Notation) is a open standard file format and data enhance using human readable text to store and transmit data object of either pair and arrays.

PDB – (Program Database) is a file format for storing debugging information about a program.

PDF – (portable document format) ensures a complete description of fixed layout text font, vector, graphic and other information, so files cannot be edit or changed.

NOE – (Nuclear Overhauser effect).

Chemical Shifts – change in nuclear magnetic resonance freuacen of a nucleic depending on the electronic environment.

RDC – (Residual Dipolar Coupling) between two spins in molecule solution exhibit partial alignment leading to incomplete averaging of spatially anisotropic dipolar couplings.

PREs – (Paramagnetic Relaxation Enhancements).

VDWs – (Van der Waals).

H-bonds – (Hydrogen Bonds) is dipole-dipole attraction between molecules to a Hydrogen atom.

J-couplings – (nuclear spin-spin coupling) is indirect interaction of the nuclear magnetic moment mediated by the bonding of electrons.

Distance Restraints – used to limit the motion of a system by penalizing the potential when the distance between specified atom pairs exceeds a threshold value.

NMR Observables – Are bonding and interaction which are observables from measure restraints.

Relaxation time – is the Energy gap between the spin-up and spin down states.

FT – (Fourier transforms) is a transform that convert a function into a form that describes the frequencies present in the original function, by mathematical operation.

NOESY - Nuclear Overhauser Effect Spectroscopy.

COSY - Correlated Spectroscopy.

TOCSY – Total Correlation spectroscopy

HSQC – Heteronuclear single quantum coherence spectroscopy

EXSY – (Exchange spectroscopy)

MD – Molecular dynamics.

Ns – (nanoseconds).

Ps – (pecoseconds).

SAXS – (Small-angle X-ray scattering) Is an analytical technique to determine differences in density within a sample with a nanoscale resolution by analysing the elastic scattering angles of X-rays.

CLOUDS – A protocol that relies on precise and abundant interproton distance restraint calculated via relaxation matrix analysis of nOes cross peaks.

MFR – (molecular fragment replacement) This method mines PDB database for homologous peptide fragments on experimental backbone data.

ROSETTA – protein structure prediction short fragments of known protein are assembled by Monte Carlo strategy to yield a native-like protein conformation.

CS-ROSETTA – (Chemical Shift Rosetta) is a robust protocol to exploit mining of protein chemical shift with known corresponding 3D structure *de novo* protein structure generation for chemical shifts.

SPARTA – is a database for empirical prediction of backbone chemical shifts using a combination of backbone. Phi, psi torsion angles and sidechain chi angles for a given protein with known PDB coordinates.

CYANA – (combined assignment and dynamics algorithm for NMR applications).

DYANA – (Dynamics algorithm for NMR applications) based on highly efficient implementation on torsion angle dynamics.

CANDID – (Combined automated NOE assignment and structure determination module) automated assignment o NOESY spectra using DYANA.

GARANT – (General Algorithm for Resonance AssignmeNT) based on mapping peak of predicted from amino acid sequence onto peak observed in multidimensional spectra.

Dictionary – Is a Key-value store of different names where a key is a name which points to a piece of data saved under that.

Variables – Is a value that can change depending on the condition or on information passed into the program.

Lists – Is a sequence of several variables grouped together under a single name.

Classes – is a code template for creating objects, to do a particular action, comprised on collection of variables and functions.

Modules – Is an extension of a min program dedicated to a specific function or goal designed for easy reusability.

Functions – Is a named section of code that performed a specific task.

Placeholders – is a word, character or string used as part of a string template that requires replacement with a variable at runtime.

Strings – Is a data types implemented as an array data of structure of bytes (words) stores a sequence of element typically character.

Booleans – Is a logical data type that have only values true or false.

Floats – is a function that converts values into floating point numbers (e.g. 9.8).

Integers – Are numbers without a fractional component and don't support decimal points.

Force fields – a collection of equations and associated constants designed to reproduce molecular geometry and selected properties of tested structures.

Molecular Dynamics calculation – Is a computer simulation method of analysing the physical movements of atoms and molecules.

Contents

Acknowledgments 3
Abstract 4
Abbreviations and definitions 5
Table of Figures 11
List of Tables 14
Introduction 15
Structure Calculations and Why They Are Important 15
X-Ray Crystallography 15
Electron Microscopy 16
NMR 17
Experiments 18
Magnetisation transfers 22
Introduction to Structure Calculations 25
Outline of the Key Components of an NMR Structure Calculation 25
Structure Calculation programs and their approach to molecular dynamics 27
DIANA 27
DYANA & CYANA 27
XPLOR-NIH 29
CNS 30
CLOUDS 31
MFR 32
CHESHIRE 32
CS-ROSETTA 32
ARIA2 35
Structure Validation 37
Aims for Thesis 38
Methodology 40
XPLOR & CNS 40
Coding 40
ARIA2s interaction with CNS 59
Data representation 59
ARIA2 source code structure 60
Run time project setup 61

Parallelisation and Job Management 63
ARIA2's structure calculation 64
Preparation stage 64
Iterative structure calculation 65
Floating chirality assignment 68
Refinement in explicit solvent 69
Analysis and validation 69
Setup for a Structure Calculation 71
Energy Terms used by ARIA2 72
Simulated annealing 78
The Methods used for Implementing XPLOR-NIH as the molecular dynamic's engine in ARIA2 81
Workspace 81
Conversion Methodology 84
Setting up ARIA2 to use XPLOR as a molecular dynamics engine. 84
Converting the code from CNS scripting language to Python 85
Comparisons between CNS and Python scripting languages, and CNS and XPLOR internal processes 90
Program flow 93
Analysing convergence 94
The structure of the ARIA2 protocol and sequence of conversion 94
Results 100
Results 1 Translation of ARIA2 CNS Molecular Dynamics Calculations to Python 100
Calls to CNS using xplor.command and aria_xplor_command 103
Debugging 105
Detailing Xplor_lib module, a function library 108
Running calculations using XPLOR-NIH's python interpreter 115
Protocols Files and Usage 115
A Look at the preliminary changes to interactions with molecule dynamics engine CNS.py 118
Defining the molecular dynamics engine to ARIAXC 122
Converting parameters into XPLORDict compatible with JSON 122
interfacing with JSON its functions__write_run_json, _update_run_cns_dict, update_chain_data and analysis 124
Access to forcefields via ARIAXC run_cns_patch.py and its patch_run_dict function 124
Results 2 Debugging the ARIAXC calculation protocol 126
Implementation debugging 126

The RollinStone Tool: an ARIA critic 128
Detecting Discrepancies with Rs_differ 134
Debugging ARIAXC with and without RollinStone 138
Debugging insights from RollinStone 139
Results 3 Troubleshooting Errors in Force Fields and Structure Injection 142
Structure Injection 142
Molecular Dynamics Ramping 148
Discussion 152
Why convert ARIA2 to use XPLOR-NIH for calculations ARIA-XC 152
ARIAXC's Chimeric Morphological synthesis 153
Unpicking Erroneous code inside of ARIAXC 154
Convergence and structural integrity 155
Limitations 155
Future Work 156
Conclusion 157
References 158

Table of Figures

Figure 1 NOE / ROSY / NOESY graph. The x axis shows the Noe and the y axis the tumbling weight as a logarithmic value. This shows that the NOE effect cannot be measured at a tumbling weight around 1 where after this it becomes progressively more negative making molecules with a higher with difficult to measure with NOE. But other NOE type experiments possible 21
Figure 2 a (H)CCH-COSY spectrum of the a aliphatic region from the of a unliganded hPTP1EPDZ-2[35]. This spectra see specific for one or two amino acids are boxed. From top to bottom, these are the Ile Cd-Cg cross peaks, valine and isoleucine Cg (methyl) -Cb cross peaks, threonine Cg-Cb cross peaks, alanine Cb-Ca cross peaks, proline Cg-Cd cross peaks, and threonine and serine Ca-Cb cross peaks. 23
Figure 3 CYANA data flow a diagramatic of overview of how the data moves throughout the program. Initially CYANA starts with information about the molecule mainly an amino acid sequence, sequence specific assignments, and measured NOESY cross-peaks positions and volumes. These are passed into the and into a nOe assignment which passes all this information into it structure calculation modelling the molecule based on the information provided and evaluates the nOe assignments. This process occurs iterative and repeating itself multiple times. This is then shown int eh NOE assignment and 3D molecule structure. 29
Figure 4 CS-ROSETTA data flow. Starting with an amino acid sequence can be added to the fragment selection program with its structural database, which uses calculate chemical shift using SPARTA initial fragment candidate which are used in the MFR

fragment selection. From here the fragment candidates are entering into the Rosetta fragment assembly and scored using the SPARTA algorithm using chemical shifts. The models are then rescored and checked for convergence, where if they do will produce a predicted structure. 33
Figure 5 nOe restraints after iteration 1 to 8 (ref) where is assignment parameter P depends on distance from the center of the atom and CN assignment parameter and c_n is number of contributions that restraints are still active in the models number of ambiguous restraint and degenerates. This shows how these parameters are increase (more restrictive restraints) how the models of the molecules structure becomes clearer and more uniform in the ensemble. 36
Figure 6 an example from a modified version of read_data.cns from the ARIA2 protocols directory. This shows examples of how CNS/XPLOR scripting languages use parameters, variables, nOe molecular dynamics statements taking a set number of parameters and how modules can be defined. 44
Figure 7 part of the file sa_hightemp.cns showing evaluate, vector do statements, using variables ($), and parameters (&), as well as the maxwell function for setting random 3d velocities compatible with a particular temperature). 45
Figure 8 A snippet of refine.inp showing a call to protocol/ flag_new.cns using the @@<environment-variable>: <path-and-file_name>(<parameter>,...) syntax. This shows how modules are used with parameter passing in CNS; note the RUN variable is taken from the shell ENVIRONMENT and defines where protocols are located on disk too increase portability and flexibility. 46
Figure 9 Shows the overall layout of CNS and how the different components interact with one another, along with what can be edited [2]. 49
Figure 10. Interactions between the sub-systems in the XPLOR-NIH program. 51
Figure 11 Display a nOe potential square-well function graph of energy against distance in angstroms 54
Figure 12. Shows the overall relationship between the energy of nOe distances restraints and the distance from an atom. 56
Figure 13 The Run time directory tree of ARIA2 (automatically set to run1) and information on what each directory contains, and what each file inside does. (Taken from ref ARIA solution for solid-state NMR)[89] 62
Figure 14 shows through the averaged structures and how they improve and become more similar RMS score becomes lower as the structure calculation ends. 68
Figure 15 shows how as the calculation proceeds through the iterations, and the structure ensemble improves, the distance restraints as they are forcefield are applied and are violated the structure improves. 75
Figure 16 Showing how protein structure ensembles improves with number of ^{1}H-^{1}H restraints for the proteins IgG and Interleukin. 77
Figure 17, show two images on the right shows the geometry location of energy gradient where a minimum energy was found at the red center. The image on the left panel shows this In a heat graph showing energy minimum within 2D space. 79
Figure 18, shows heat maps of clouds of atoms in energy gradiant indicting each point for reference 80
Figure 19 Shows the PyCharm working environment Unix command line, files viewer and editor, and directory tree for selecting and manipulating files. 83
Figure 20 shows a snippet of a downloaded script that help visualise CSV files through PyCharm based on the column. 84

Figure 21 Contents of refine.py 107
Figure 22. Shows a diagram of a snippet of run.json showing the dictionary of data structure. In this example filenames points to file_root which points to hrdc, which is a string. ... 110
Figure 23 The internal structure of a XplorDict 112
Figure 24 Example of a graph from RollinStone showing the detection of an error in the parameter k_unamb 129
Figure 25 An XPLOR /CNS energy table 132
Figure 26 An iteration key 132
Figure 27 example command line use of rs_differ.py 135
Figure 28 comparison files produced by rs_differ displayed using the pyCharm color csv tool. 136
Figure 29 shows two examples from the graphical display from rs_differ.py. This has BATH a variable that is used throughout both programs (ARIAXC and ARIA2) and k_unamb a constant for unambiguous nOe data as the program executes. Showing ARIAXC in the orange dot 137
Figure 30 Lowest energy structure from it0 of the Werner test (b) set after initial manual debugging of ARIAXC and for comparison the same structure from an ARIA2 calculation (A). ... 138
Figure 31 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) after debugging with RollinStone. The 7 lowest energy structures overlaid by Cα backbone atoms are shown. The residues in the structures are coloured with a ramp from 140
Figure 32 Shows Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers] 141
Figure 33 shows diagrammatically how structure injection works. In each row a calculation is carried out and energies calculated which differ between runs and programs [blue and red]. However, at each stage in the calculation where the state of the calculation needs to be compared an identical structure is loaded and energies are calculated. In the case shown even though the energies and structures in the two runs are different, those of the injected structures are the same showing the forcefields have been applied in the same manner. ..... **Error! Bookmark not defined.**
Figure 34 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) during Structure injection. The 7 lowest energy structures overlaid by Cα backbone atoms are shown. 147
Figure 35 Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using of RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers] 147
Figure 36 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) during convergence. The 7 lowest energy structures overlaid by Cα backbone atoms are shown. The results from ARIA2 are on the left and ARIAXC on the right 150
Figure 37 Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using of RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers] 150

List of Tables

Table 1 Scripting interface Difference Comparison. This Tables displays the temperature ramping loop from sa_rc_hightemp for CNS and XPLOR scripting interface syntax on the left with a loop using it while statement. The right displays Python scripting interface writing a loop using a for loop. 43
Table 2 a comparison example of scripts performing rigid-body minimization in the XPLOR and Python scripting languages taken from ARIA2.0: Nuts and Bolts [4]. 48
Table 3 Assigning variables in CNS and python 86
Table 4 defining functions in CNS and python 86
Table 5 calling functions in CNS and python 87
Table 6 A second example of the function call syntax in CNS and python 87
Table 7 Example of an if statement in CNS and python 88
Table 8 Example of a conditional loop in CNS and python 89
Table 9 Logical expression in CNS and python 90
Table 10 Using vector STORES in CNS and XPLOR 91
Table 11. showing a comparison of CNScript (left) and python (right) scripting interface, of the same code from the start of refine in ARIA2. 102
Table 12. Comparison table showing 2 seconds of code from refine.cns/.py with CNS on the left and xplor/python on the right showing use of aria_xplor_command.. 104
Table 13 Another version of the code in table 12, which ignores variable aliasing and could lead to erroneous results 105
Table 14 types of python format options used in the code 105
Table 15 Extra methods in XplorDict beyond those in a standard python dict 113
Table 16 Standalone methods that work with XplorDict 114
Table 17 The major files used in ARIAs molecular dynamics protocols an a summary of the changes made to them, inclding any name changes made to the files in conversion to ARIAXC. In cases were '–' is used this no changes were made to the name of the file. 116
Table 18. RollinStone inputs collected in the parameter args created by the function return from the function get_arg_parser 129
Table 19 Each selection that is selected by RollinStone and the action that is taken for each. 132
Table 20 Examples of information stored per cycle by RollinStone 133
Table 21 RS_differ inputs collected in the parameter args created by the function returned from get_arg_parser 135
Table 22. Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC after using RollinStone debugging tool. 140
Table 23 Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC from structure injection. 147
Table 24 Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC after structure injection and with extended refinement cycles. 148

Introduction

ARIA2 (Ambiguous Restraints Iterative Assignment)[1] is a program for NMR structure calculation and determination, which uses CNS (Crystallography and NMR System)[2,3] as its molecular dynamic's engine. In this thesis, I have developed a version of ARIA2 using XPLOR-NIH[4,5] (XPLOR, since being developed by NIH) instead of CNS as the molecular dynamic's engine[1]. Here, I will introduce biological structure calculation and the NMR (Nuclear Magnetic Resonance) techniques used by ARIA2. I will outline the data needed for ARIA2 structure calculations as well as other biological structural calculations in detail.

Structure Calculations and Why They Are Important

Structure calculations are important for many different reasons. Some of these reasons can be determining an unknown structure or domain of a protein to study its structural information; furthering understanding of intermolecular and intramolecular interactions; ligand protein bonding; and catalytic reactions with enzymes[6]. Knowing the structure of proteins allowed the building of libraries that contain structures (PDB, protein data base), allowing comparison of structures and function[7]. This has propelled drug discovery by determining the binding site shape and the interactions, allowing the development of drugs that bind to the same binding site as biological ligands, which can inhibit or accelerate biological processes[8,9]. It also allows researchers to analyse and classify proteins of unknown activity that have a similar conformation or sequence to another protein or a region, to determine and gain information on its structure[10,11].Thus, we can start understanding a protein before doing experimental work. It also increases our knowledge of how molecules interact with one another, along with protein structures in general. Indeed, the PDB database was used to train AlphaFold[12], the current best performing protein structure prediction program, which appears to be set to revolutionise our approach to structure biology research[13].

X-Ray Crystallography

An important structural biology technique is used to determine the structure of proteins is X-ray crystallography (XRC). This allows you to experimentally determine the atomic structure of a crystalline substance. It works by shooting a beam of X-rays, called the incident beam into a crystal structure of a molecule. The X-rays

diffract in different directions, and the beam scatter at different angles and with different intensities and phases, which is dependent on the structure of the molecule[14]. This information allows you to elucidate a 3D structure from the density of electrons within the crystal sample. From this electron density the mean positions atoms and their chemical bonds within the structure can be determined[14].

Proteins examined by X-ray Crystallography need to be crystallized, which can be a time-consuming process. XRC can be a very powerful technique to determine protein structure, showing precise conformations of protein and active site within the proteins[15]. XRC can analyse molecular structures over a very wide range of molecular masses including membrane proteins and the ribosome as examples[15]. However, NMR by comparison has a molecular mass limit of approximately 30kDa for structure determination[16]. One of the clear differences between NMR and XRC is that NMR structures are determined in solution at room temperature rather than in solid state at a cryogenic temperatures[16].

Electron Microscopy

Another method to view the structure of a molecule is Electron microscopy (EM). This method allows people to view an image of molecules and inorganic materials. EM works by shooting a beam of incident electrons at a sample or specimen, which then diffracts, deflects, or reflects the electrons, which changes the course of the electrons, which is then detected. There are 3 major types of EM microscopy: transmission electron microscopes (TEM), scanning electron microscopes (SEM), and reflection electron microscopy (REM)[17].

Cryo-EM uses a high voltage electron beam produced by a tungsten filament cathode accelerate at the sample using an anode to illuminate a sample, where the beam of incident passes through the sample causes the electron to scatter, creating an image density map[18,19]. Limitations of this method of microscopy have been lessened recently with High resolution electron microscopy being able to measure distances of 3-4 angstroms within biological molecules under favourable conditions[20]. TEM has advantages of XRC as you do not to need to crystallise your samples but can be challenging in its own right with high quality samples flash frozen in water being difficult to prepare[21].

Other disadvantages of Cryo-EM, are, equipment is expensive delicate and is intrinsically sensitive and very and large number of single molecules images to be visualised and processed. Cryo-EM currently and a lower mass limit is 50kDa[20].

NMR

NMR (Nuclear Magnetic Resonance) spectroscopy[22] is a structural biology technique used to determine structural data of proteins, DNA, RNA, sugars, and small molecules in solution or the solid state[23]. It works by measuring signals from atomic nuclei, most notably the isotopes of proton/hydrogen (^{1}H), Carbon (^{13}C), Nitrogen (^{15}N), Phosphorus (^{31}P), and deuterium ($^{2}H/^{2}D$). These isotopes have an intrinsic nuclear magnetic moment, angular momentum, and a magnetic quantum number (spin) I=1/2[24,25,26]. These nuclei, when placed into a constant magnetic field, align with and against a magnetic field, giving two energy levels, alpha and beta (low and high energy). When an electromagnetic field of the correct frequency is applied, this causes transitions which can be detected as a signal/voltage in the coil of a spectrometer's probe. The detected signal can be digitized and processed by using a Fourier-transform[27] to give a spectrum with peaks that are related to chemical environments in the sample[28].

There is a range of NMR experiments that can be used to determine the structure of molecules. These work by probing the covalent structure of the molecule and 3 dimensional distances. Some examples include NOESY (Nuclear Overhauser Effect)[29] for distance determination in the backbone and side chains of proteins and COSY (correlation spectroscopy)[30,31], TOCSY (Total correlation spectroscopy)[29], and HSQC (Heteronuclear single quantum coherence spectroscopy), which are through-bond (heteronuclear correlation)[32] experiments that exploit the covalent structure of molecules.

In through-bond experiments, we can transfer polarisation from one nucleus to another adjacent to it via bonding electrons and collect data on the entire backbone of a protein and its sidechains[32,33]. Other NMR experiments that can be used to measure other physical properties of a molecule, including EXSY (exchange spectroscopy)[26] measuring the presence of chemical exchange or ^{1}H-^{15}N relaxation experiments that can determine the presence of molecular motions at timescales of ps (picoseconds), and ns(nanoseconds).

Experiments

COSY 'Correlation Spectroscopy'

^{1}H-^{1}H COSY experiments can be used to identify spins that are coupled to each other across bonding electrons[34,35]. 2D COSY spectra usually show frequencies for a single isotope, typically ^{1}H, along both axes, with cross-peaks that result from magnetization transfer between two different atoms appearing off the diagonal[31]. Their presence indicates that 2 nuclei are coupled, typically across 3 bonds, which have 2 different chemical shifts[34,36]. An example of a COSY spectrum can be seen in Figure 1.

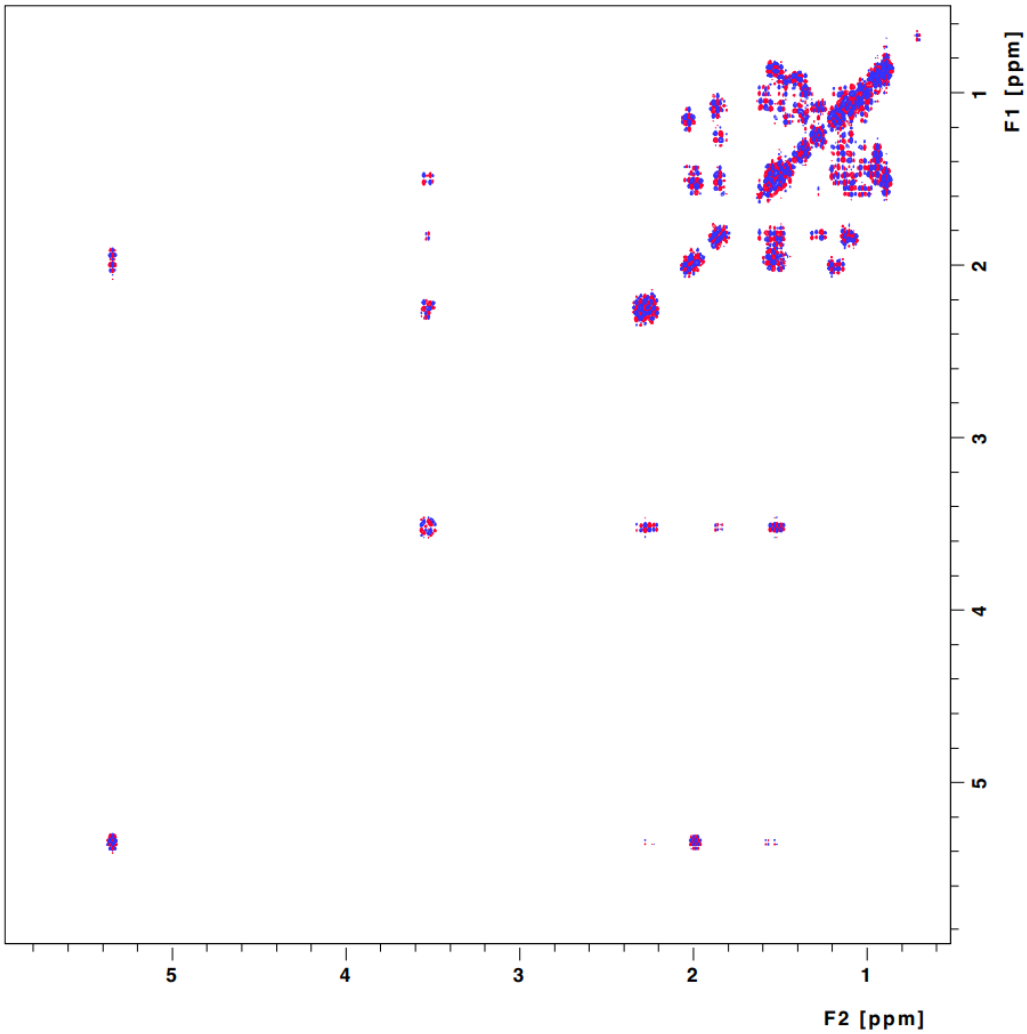


Figure 1 shows a COSY spectrum of cholesterol. Cosy correlation thought 3 bond is typical of H-H, so H-C-C-H where a significant coupling is present

TOCSY 'Total Correlation Spectroscopy'

TOCSY experiments are similar to COSY, where off-diagonal cross-peaks of coupled protons are again observed. However, in this experiment seen in Figure 2, in addition to COSY-type peaks, there are cross-peaks between nuclei that are connected by a chain of couplings across bonds (usually more than 3)[29]. This is useful for identifying large, interconnected networks of atoms. The experiment works by using a repetitive series of pulses during an isotropic mixing period[37].

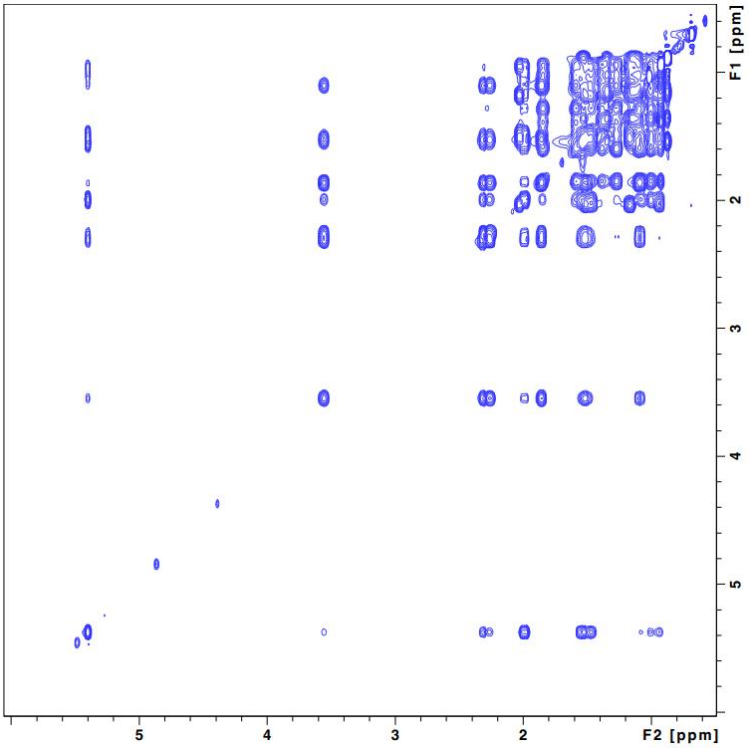


Figure 2 shows a example TOCSY spectrum of Cholesterol. TOCSY correlation though multiple bonds H-H where even small H-H couplings are present, typically across a complete spin system depending on mixing time.

NOESY 'Nuclear Overhauser Effect Spectroscopy'

In NMR, nOe's arise from dipole-dipole interaction due to proximity between active nuclei and give information about the spatial environment when active spins are close (usually less than 5 angstroms)[38]. They also provide information on the relative stereochemistry and atoms in a molecule.

A ^{1}H NOESY experiment transfers nuclear spin polarization from one spin-active nucleus to another via cross-relaxation, hence taking advantage of the nOe effect[39]. In many instances, for small, rapidly tumbling molecules, in the extreme-narrowing limit, the semi-quantitative nature of positive nOe's is useful for structural analysis[1].

This is often used in combination with the measurement of J-coupling constants, which allows for the identification of aromatic ring substitution patterns and aliphatic substituent configuration, and determination of conformational preferences[40]. An issue can occur with the NOESY experiment as nOes are affected by the correlation time, proportional to the inverse rate of molecular tumbling Figure 4 [40]. As the rate of molecular tumbling is determined by the molecule's molecular weight and the solvent viscosity, larger molecular weights and higher viscosities lead to larger correlation times and negative nOes which approach -1.

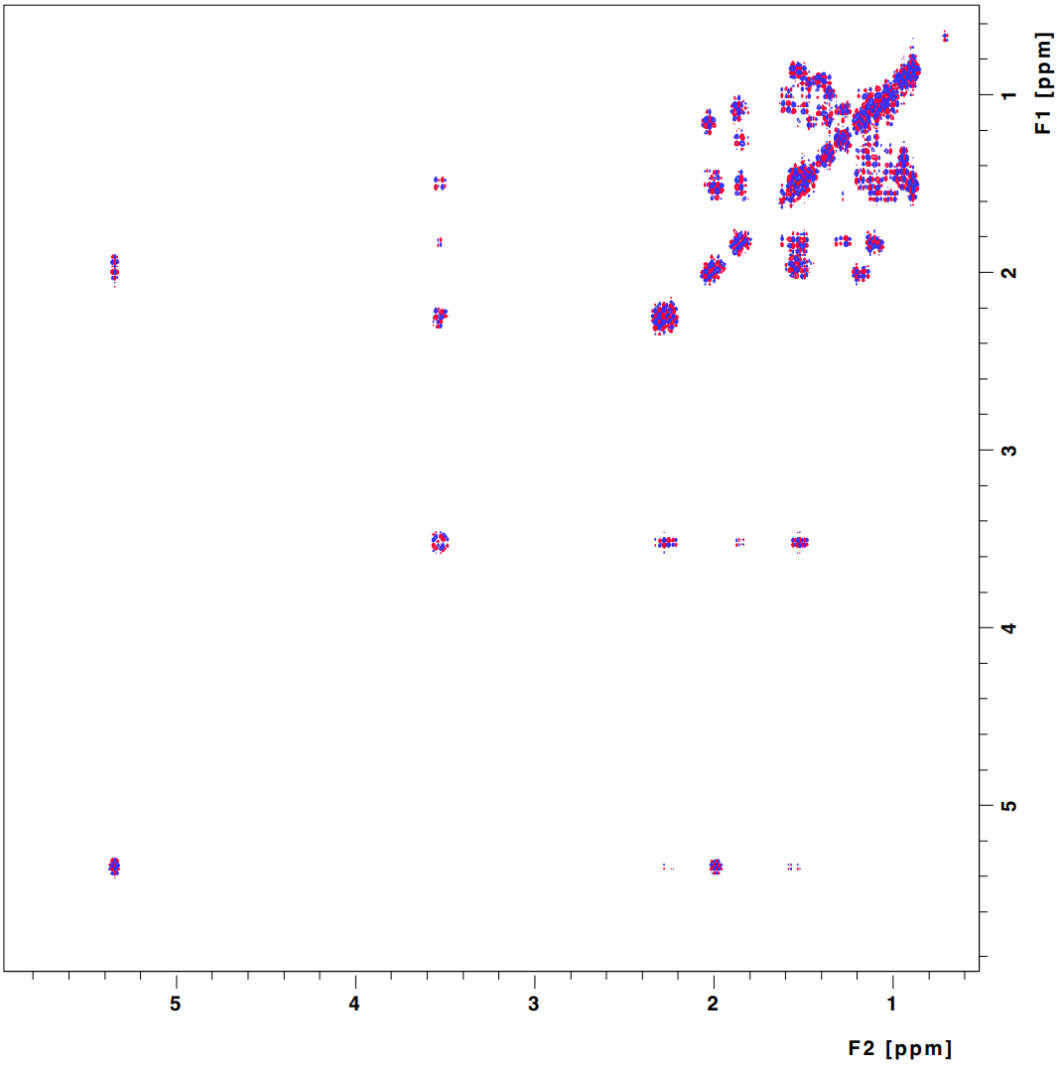


Figure 3 Shows a NOESY spectrum of cholesterol. NOESY connections through space typically between Hs with a distance of less than 6 Å between them.

For medium size molecules (20-30kDa), near the nOe function crossover point, the NOE will become zero, and you will not see any peaks[40]. This can be countered using TROSY (Transverse relaxation optimized spectroscopy nuclear Overhauser

effect), which optimise magnetisation transfer to reduce these losses due to a short T2(transverse relaxational time)[41]. This effect allows the user to study larger proteins with NMR an example of which seen in Figure 3. In addition, NOESY experiments are useful for determining which protons that are close to one another in space, less than six angstroms, even if they are not bonded, which is vital for producing chemical shifts.

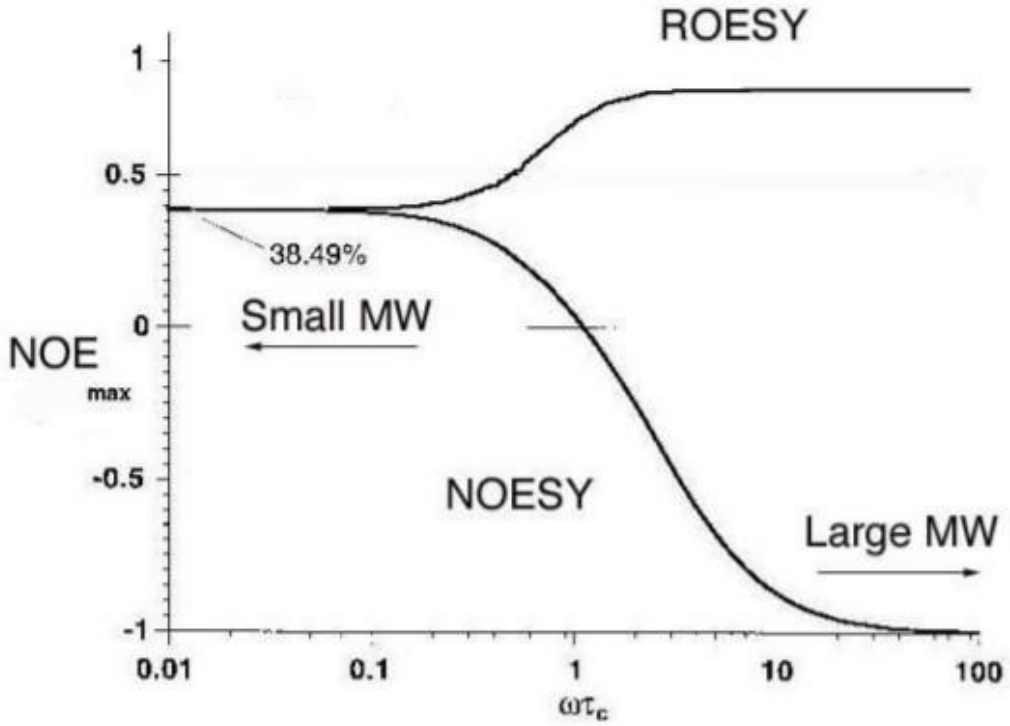


Figure 4 NOE / ROESY / NOESY graph. The x axis shows the Noe and the y axis the tumbling weight as a logarithmic value. This shows that the NOE effect cannot be measured at a tumbling weight around 1, where after this it becomes progressively more negative, making molecules with a higher weight, which is difficult to measure with NOE. But other nOe type experiments are possible

HSQC 'Heteronuclear Single Quantum Coherence Spectroscopy'

HSQC experiments are fundamental experiments for NMR. They are used to measure of NMR correlations between nuclei of different isotopes that are separated by one bond using the $1J_{XH}$ couplings, giving a peak per pair of coupled nuclei[32,42]. The axes of the experiment are the chemical shifts of the two coupled atoms, typically for ^{1}H and ^{13}C or ^{1}H and ^{15}N for proteins. This process occurs by transferring magnetisation between nuclei using the INEPT pulse sequence[43]. The first step starts at a proton which has a greater equilibrium magnetisation, and this step helps create strong signals with good intensities.

An INEPT step then transfers this magnetization to the attached C/N atom. This magnetisation then evolves on the second nucleus (C/N) to measure a chemical shift

and is then transferred back to the first nucleus, via a reverse INEPT step for observation[43]. An extra spin echo pulse can be optionally added to decouple[44] the signal from the second nucleus, collapsing multiplets to a single peak simplifying the spectrum, which along with decoupling during acquisition gives 1 peak per NH or CH pair[45,46]. An example can be seen in Figure 5.

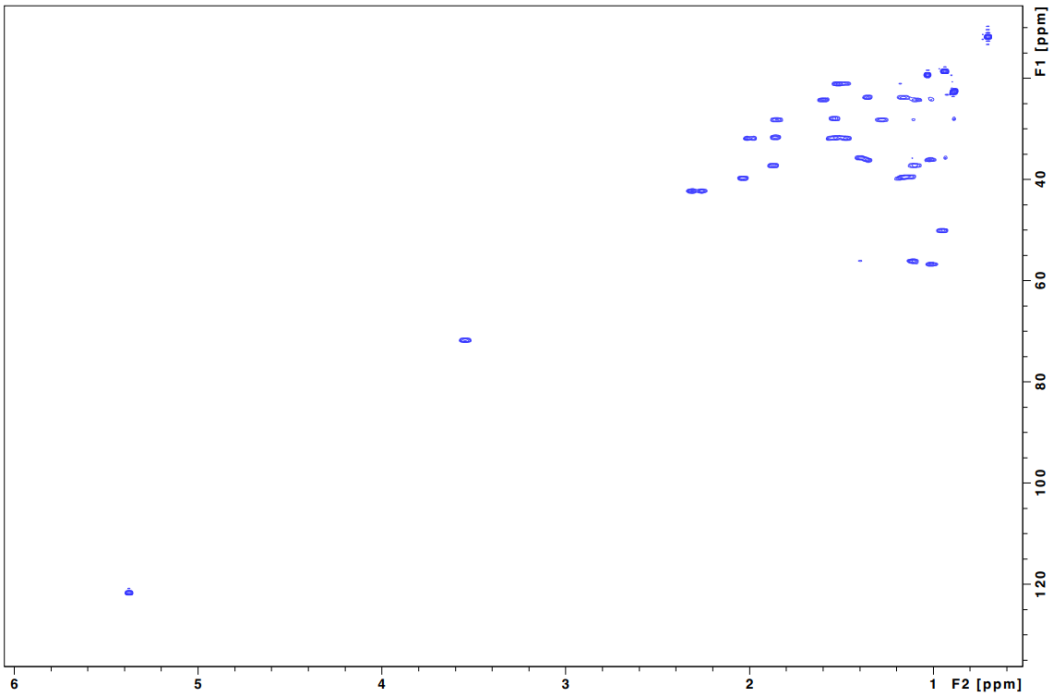


Figure 5 shows a HSQC spectrum for Cholesterol. HSQC typically correlation across one bond between hetetronuclei ie 1H-13C or 1H-15N. This was created by Dr. G Thompson

Magnetisation transfers

The simple experiments discussed above correlate signals from nuclei through bonds and space in a general manner and are applicable to small molecules. However, the following section discusses sequences specific to the structure and couplings available in ^{15}N and ^{13}C labelled macromolecules, specifically proteins.

HCCH

In HCCH COSY and HCCH TOCSY experiments, magnetisation is transferred from the hydrogen nuclei to their attached 13C nuclei. After this, magnetisation is then exchanged with neighbouring 13C nuclei using J-couplings via COSY or TOCSY transfer[47,48]. Finally, magnetisation is transferred back to the hydrogen atoms for detection. Chemical shifts can be measured at each of the excited hydrogen or carbons, giving a variety of experiments to choose from. Examples are (H)CCH and

H(C)CH, which are both 3D, an example can be seen in Figure 6, and the complete 4D HCCH experiments[49].

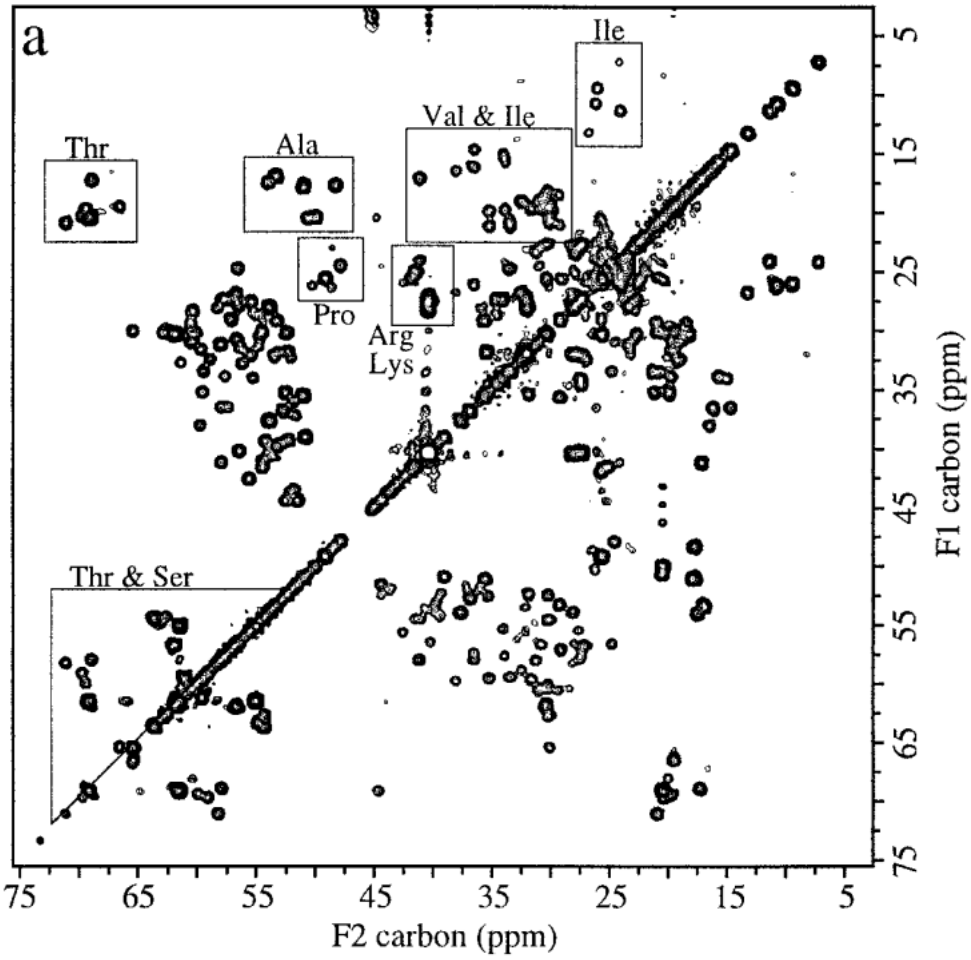


Figure 6 a (H)CCH-COSY spectrum of an aliphatic region of a unliganded hPTP1EPDZ-2[35]. The boxes show specific regions in which the ^{13}C shifts of the common amino acids occur. From top to bottom, these are the Ile Cd-Cg cross peaks, valine and isoleucine Cg (methyl) -Cb cross peaks, threonine Cg-Cb cross peaks, alanine Cb-Ca cross peaks, proline Cg-Cd cross peaks, and threonine and serine Ca-Cb cross peaks.

HNCA

A number of through-bond experiments can be used to assign protein backbone atoms. These typically include, HNCA, HNCACB, HN(CO)CA, HN(CA)CO, HNCO, and finally HN(CO)CACB or (HB)CB(HA)CA(CO)NH which is often called CBCA(CO)NH[41]. These experiments exploit the presence of strong couplings across the peptide bonds, such an $^{1}J_{Ni\text{-}COi\text{-}1}$ to make sequential assignments[41].

The HNCA sequence can be viewed as a set of 4 sequential INEPT type transfers as found in an ^{1}H-^{15}N/^{13}C HSQC with chemical shifts measured at critical points (*)[50]. The transfers are $^{1}H_i$-$^{15}N_i$*, $^{15}N_i$-$^{13}Ca_{i/i-1}$*, $^{13}Ca_{i/i-1}$-$^{15}N_i$, $^{15}N_i$-$^{1}H_i$*, all of which occur via large couplings. In similar way the HN(CO)CA transfers via the pathways $^{1}H_i$-$^{15}N_i$*, $^{15}N_i$-$^{13}CO_{i-1}$, $^{13}CO_{i-1}$-$^{13}Ca^*_{i-1}$ - $^{13}Ca_{i-1}$- $^{13}CO_{i-1}$- $^{13}CO_{i-1}$-$^{15}N_i$, $^{15}N_i$-$^{1}H_i$*[41]. As these two sequences can share a common Ca with an i or i-1 connection, this allows pairs of amides which are i and i-1 to be matched and sequential assignments to be made[41]. Similar magnetisation transfer pathways can be laid out for the other backbone assignments spectra.

Isotopic labelling with ^{13}C and ^{15}N isotopes is required for NMR assignment and structure determination experiments, however, this can be expensive[51]. Isotopic labelling is usually achieved using minimal media, which contains $^{15}NH_4Cl$ and ^{13}C glucose, and expressing protein from a plasmid[51].

Combination experiments

These basic experiments can be combined to measure a variety of data including, nOes, RDCs, the presence of H-bond, J-couplings, etc, which can be used to determine the structure of biological macromolecules.

An example of using the NOESY effect for biomolecular NMR would be a ^{1}H-^{13}C-NOESY-HSQC. In this experiment, an initial NOESY sequence exchanges magnetisation between all hydrogen atoms close in space (<6 angstroms) via the nOe effect[52]. Then, a subsequent HSQC sequence transfers magnetisation back to any covalently bonded ^{13}C nuclei and finally back to ^{1}H for detection[52]. Data can be measured from aliphatic ^{13}C nuclei or aromatic ^{13}C nuclei, or both, using selective or non-selective pulses[52]. An equivalent ^{1}H-^{15}N-NOESY-HSQC can be used using selective ^{15}N pulses that can be used to observe amide-based nOes[52].

[53][53]One issue that arises even with HSQC spectra is that small differences in chemical shift between spectra from different samples can make precise assignment of peaks difficult. This is especially true when comparing to reference spectra from databases such as the BMRB and especially when there are many overlapping proton signals of similar chemical shifts such as in complex mixtures[54]. Acquisition of coupled HSQC spectra can also be advantageous in cases where one-bond ^{1}H-^{13}C coupling constants are of interest, or where decoupling would result in low-quality

spectra due to sample heating (this can be an issue when using long acquisition times or polar solvents in the presence of salts)[54].

Introduction to Structure Calculations

Several NMR methodologies exist for calculating structures based on the above discussed measuring techniques like EM, NMR, and X-ray Crystallography. The following section will be a small survey of available techniques that can be used to calculate molecule structures and putting the rest of this thesis into context, as well as discussing the programs and methodology used in this project.

In this project, ARIA2 is the structure calculation that was updated to use XPLOR-NIH as its molecular dynamics engine, and hence will give a brief outline of the program being discussing other structure calculation programs and algorithms. ARIA2 is an automated structure calculation and nOe (nuclear Overhauser effect) assignment program[1,55]. ARIA2 uses nOes distance restraints that are either ambiguous and unambiguously assigned to calculate a protein structure through an iterative process via simulated annealing by a mixture of torsion angle and Cartesian angle dynamics using CNS for molecular dynamic calculations[2,3]. ARIA2 works by orchestrating and series of CNS calculations with varying parameters, improving restraints which lead to structural convergence. The calculation is repeated to generate multiple structures, which are compared to analyse convergence[1].

Outline of the Key Components of an NMR Structure Calculation

The key components of a structure calculation are the restraints measured and the primary structure and chemical shift of the molecule that you wish to study and determine the structure of. These data can include NOEs, chemical shifts (from peaks on the spectra)[56,57], dihedral bond angles[58], RDCs (Residual Dipolar Couplings)[59], hydrogen bonds, disulfide bonds, VDW forces (van der Waals). The structure of a protein is defined by a series of bonds and atoms in a 3D space. Structures are calculated by restraining a model of a molecule, typically using nOes and dihedral restraints[60].

However, to use these restraints, the chemical shifts of the peak defining them need to be assigned. One of the issues of this process is that some of these peaks may be ambiguous i.e., contain multiple assignments as seen in Figure 7 shows an image of

ambiguous restraints for several scenarios and a structure that has been refined purely based on ambiguous data[6155]. A number of programs, such as ARIA2 and CYANA do automated assignment and structure calculation together, which reduces the burden on the user[1,62,63]. This is because manual assignments needed to calculate 3D structures can take months to do, whereas automated structure calculation can be carried out in a matter of hours via a computer algorithm and often produces a better result[64].

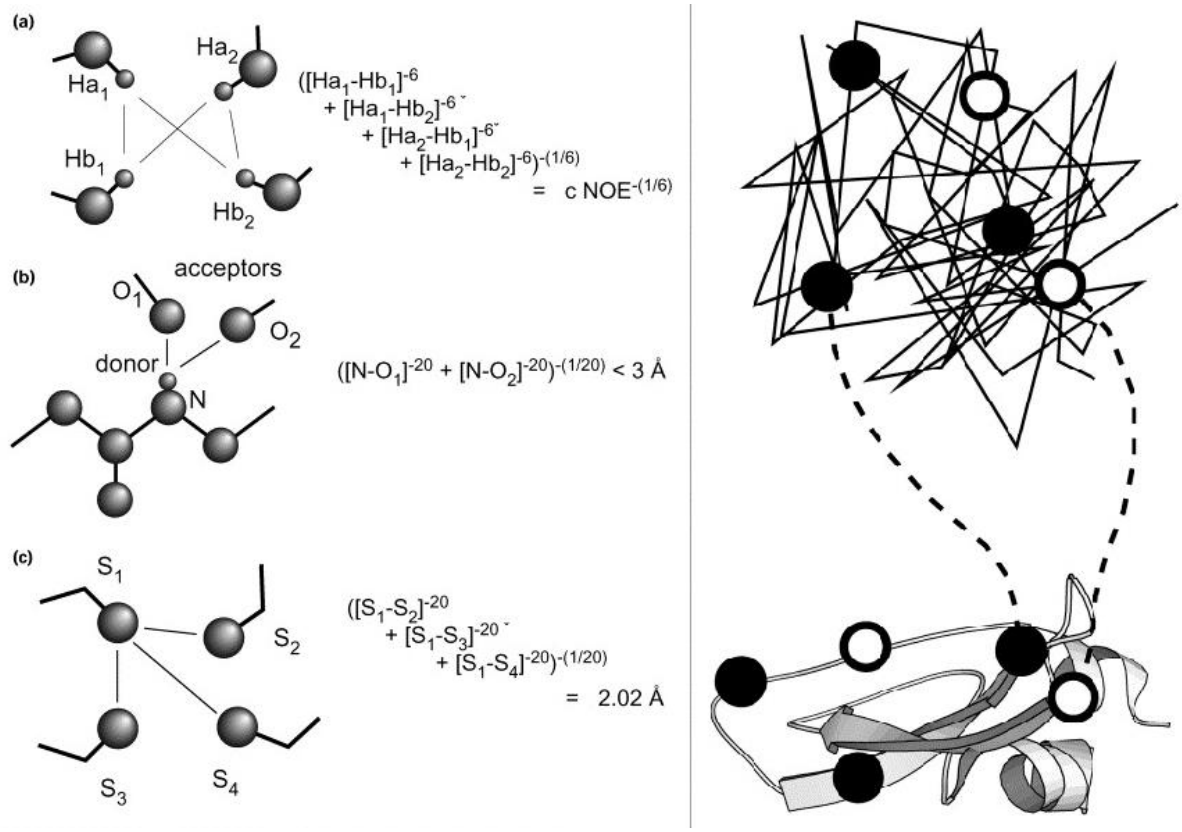


Figure 7 displays 2 images RHS shows 3 examples of ambiguous distance restraints relationships between molecular structure and ambiguous nOe data (square bracketed text denote distances). (a) ambiguous nOe protons with degenerate chemical shifts with 4 contributions, restraint is the sum of contributions to the experimentally observed nOe. (b) protected amide hydrogen can have two possible acceptors. (c) A disulfide bond with unknown connectivity, a larger extent than 6 can be advantageous. LHS shows convergence of a ab initio MDSA method of assignment from a random initial structure. The circle same appearance (white or black) representing protons with same chemical shifts. The original pair of a white and a black proton (representing an implicit assignment) is separated during the calculation and a new pair is formed in the final structure[61].

Structure Calculation programs and their approach to molecular dynamics

To put the ARIA2 protocol in context, we must have a firm understanding of other types of molecular dynamics structure calculation programs and automatic assignments protocols that can be used with NMR data and spectra.

DIANA

DIANA (Dynamics algorithm for NMR application)[65] was used for the initial calculation of protein structures from NMR data, and used distance geometry and the REDAC (redundant dihedral angle constraints) algorithm for computation[66]. The REDAC algorithm reduced the time of assembly of acceptable conformers by a 5-100fold, which is dependent on the complexity of the protein structure and the quality of information provided[66]. However, this approach fell out of use as molecular dynamics based simulated annealing was found to be more efficient and effective[67].

DYANA & CYANA

First DIANA, then DYANA (Dynamics Algorithm for NMR applications)[68] uses and molecular dynamics in torsion angle dynamics, which was fast and efficient, allowing for reduced degrees of bond angle freedom and absence of high frequency bond angle vibrations with simulated annealing[68]. Torsion angle dynamics, compared to the previous method of distance geometry, was faster, reducing the time to complete the calculation. DYANA was combined with the automated nOe assignment and structure determination module CANDID[69]. CANDID uses an iteration approach like ARIA2, using cycles of nOe assignment and structure calculation, which in the combined approach used DYANA's torsion angle dynamics algorithm[69].

The Final automated structure determination protocol for biological macromolecules that uses chemical shift data is the CYANA program (Combined assignment and dynamics algorithm for NMR application)[62]. This program is the successor of the program DYANA and can be used with the GARANT (General Algorithm for Resonance AssignmeNT)[70] and FLYA (automated assignment) algorithms[71]. This algorithm works on the basis of conformational constraints from Monte Carlo simulation and is sometimes similar to ARIA2.

CYANA also uses torsion angle dynamics on a target function, used as the potential energy function, cured with temperature by coupling with the bath (target function) as it is cooled down from its initial high temperature in a simulated annealing protocol[62]. This algorithm is based on a consistent probabilistic model of the nOe assignment process and can replace the vast majority of erroneous data. This is useful due to inaccurate chemical shift assignments, missing NOESY cross-peaks from manual nOe assignment procedures, which can cause a need for a higher range of tolerances when matching chemical shifts and peak positions that are simulated from the established range of acceptance[62]. If this potentially erroneous data can be removed from the structure calculation, it can produce higher-quality structures.

The algorithm has been proven to be quite robust with regard to imperfections of the NOESY peak list and size of chemical shift tolerances, but susceptible to inaccuracies due to a lack of or erroneous resonance assignments, especially for nuclei in NOESY cross-peaks[72]. CYANAs Automated NOESY assignment can be combined with GARANT's or FLYAS automated sequence-specific resonance assignment[73].

In this approach, an ensemble of conformers is calculated from a randomised starting conformer [simulated annealing protocol]. This could also be done with the other algorithm, FLYA, which calculates a 3D protein structure from a set of multidimensional NMR spectra, determined via empirical molecular modelling[73]. CYANA requires information on cross-peak assignments of backbone atoms from chemical shift assignments such as CCPN Assign, NMRView, Sparky, or AUTOPSY. It works as seen in Figure 8, in 3 stages, the first being a de novo[74] structure calculation phase, the second, and then third stages are structure generation by the preceding stages used for additional input for the determination of chemical shift assignments, for competing automation of this program good NOESY spectra is required[72].

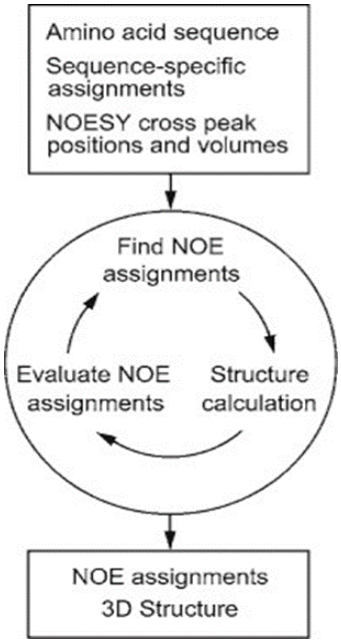


Figure 8 CYANA data flow diagram of an overview of how the data moves throughout the program. Initially, CYANA starts with information about the molecule, mainly an amino acid sequence, sequence-specific assignments, and measured NOESY cross-peaks positions and volumes. These are passed into an nOe assignment, which passes all its output into a structure calculation. This process occurs iteratively and repeats itself multiple times. This results in an NOE assignment and 3D molecular structure[75,76].

XPLOR-NIH

XPLOR-NIH is a structure determination program that builds on the X-PLOR program, including additional tools developed at the NIH, and is still in active development by Schweitzer and Clore[4]. It is a good choice to use as a molecular dynamics engine for a number of reasons. As X-PLOR is the predecessor of CNS, it has many of the same methods and inputs as CNS, with some slight changes in the syntax, and supports many of the same forcefields[77].

More importantly, XPLOR-NIH is more modern, maintained, supports ensemble calculations for systems in multiple conformers, and has more sophisticated forcefields, including PREs (Paramagnetic Relaxation Enhancement) data, CamShift, and SAXS (Small Angle X-ray scattering)[78]. These can be used to extend and augment ARIA2's structure calculation protocol. In addition, XPLOR-NIH's code has changed from using FLECS, a dialect of FORTRAN77, to a more flexible and dynamic C++/FORTRAN combination[4,77]. XPLOR-NIH, much like CNS, has procedures and task files to carry out specific functions; these also assist with modularity[4]. XPLOR-NIH can also be easily set up for parallelization on multiple processors, much like CNS, allowing for faster calculations.

Furthermore, XPLOR-NIH has an interface for Python, a GPSL (general purpose scripting language). Either Python 2.7, or Python is supported, depending on the version of XPLOR-NIH used. Python is a much more sophisticated language to use to drive the XPLOR-NIH molecular dynamics engine than its native language, allowing for complex tasks, and more maintainable and sophisticated libraries.

Interpreted GPSLs, such as Python, are not computationally efficient for numerically intensive tasks such as molecular dynamics calculations[4]. Thus, an interface between the GPSLs that XPLOR-NIH uses (Tcl, Python) and the efficient compiled calculation functions is necessary. XPLOR-NIH uses SWIG (the software interface generator) to semi-automatically generate the code needed to call the C and C++ functions present in the classic X-PLOR interface from its Python & Tcl interpreters[4]. The implementation also provides more sophisticated procedures to simplify calculations from within the X-PLOR interface system in the same fashion as CNS, written with Python. The GPSL interface is currently under active development. Both Python and TCL interfaces have been implemented to maximize the number of external programs and tools that can be utilised with XPLOR-NIH.

X-PLOR, being the predecessor to CNS, and XPLOR-NIH, being an additional implementation from the original XPLOR they share the same methods, file formats, and calculations for processing data as CNS for nOes, chemical shifts, dihedrals, RDC, h-Bonds, J-Couplings, VDWs, PREs, CSA, and GMC potentials, with just some small differences in syntax at some places to activate the protocols and calculations in comparison to CNS[4,5].

CNS

When ARIA2 was developed, CNS was chosen as a good molecular dynamics engine as it has advanced features for XRC and NMR, and structure calculation. These included, at the time, having fast Cartesian and torsion angle dynamics implementations and pertinent forcefields for nOes, H-bonds, ambiguous and unambiguous distance restraints, J-couplings, and dihedrals[2]. CNS also has its own CNS language, which contains compiled facilities for saving and setting parameters in variables and manipulating variables during molecular dynamics calculations, which is based on XPLOR.

One of the advantages and disadvantages of CNS is that it has its own scripting language tailored to structure calculations. The CNS language is implemented as an interpreter written in FORTRAN and provides symbols, control flow, io (input/output), functions/procedures, modules, and algorithms[2]. The CNS language has hard-wired functions for efficient processing of computing-intensive tasks in Unix-based operating systems, which allowed for a flexible multi-level hierarchical approach for the most commonly used algorithms in macromolecular structure determination[2].

The CNS language works with 2 different elements: symbols and parameters. Parameters are untyped data elements of arbitrary length that contain a collection of CNS commands or a collection of strings. Symbols are typed variables, containing strings, floats, integers, or Boolean/logical variables[2]. CNS also supports parallelisation, so it can be run on clusters to speed up structure calculation. The CNS code base also contains library modules for common structural biology parameters and tasks. The CNS command interface is almost identical to X-PLOR with minor differences and extensions (modules)[2].

CLOUDS

Another method to structure calculation programs is an assignment free methods used by CLOUDS, a recent assignment free protocol that takes a minimalistic approach. This is an ability of assignment-free structure determination[79,80]. To work well, this protocol requires precise and abundant interproton distance restraints calculated from a relaxation matrix analysis of sets of experimental NOE spectroscopy cross-peaks[79]. Where a cloud of unassigned, unconnected H atoms is further condensed into a structured proton distribution (clouds) via a molecular dynamics simulated-annealing scheme in which the internuclear distance and VDW's (van der Waals) repulsive term are the only active restraints[79].

This was adapted from the MIDGE protocol[39]. The proton density (*foc*) is generated by combining many different combinations of these "clouds," where each one is calculated from a different trajectory[79]. The covalent structure of the protein is fitted to the *foc* via three different programs, the back finder BAF, the sidechain finder (SIF), and then EMBED, which optimizes the global structure[81].

MFR

Residual dipolar couplings can provide methods that determine the backbone structure of a protein without a need for assignment, and in some cases, the measurement of nOes[82]. The first example of this program was MFR (molecular fragment replacement)[83], which searched for fragments of seven contiguous amino acid residues that fit the measured residual dipolar couplings[80].

With this approach, an initial structure was built from overlapping fragments by MFR, from census values of the torsion angles of non-terminal residues[80]. Furthermore, the presence of disorder loops can lead to uncertainties in the translational position, between rigid elements defined by MFR, which require distance information (NOEs) to resolve[83]. RDCs also require a compatible alignment medium to make measurements, and MFR requires a large number of RDCs to be measured rather than looking at a select fragment from its database[84].

CHESHIRE

Chemical shift-based structure determination is possible as chemical shifts are highly sensitive to their local structural environment. However, they are mostly used to study conformational changes on ligand binding[85]. The CHESHIRE (CHEmical SHIft REfinement) algorithm was the first program to generate near-atomic resolution structures from chemical shifts[86].

The program uses the secondary chemical shift to predict the secondary structure at 3 and 9 residue fragments[86,87]. These fragments are then used to generate a library of trail structures based on a database of fragments with known chemical shifts and structures[75]. Finally, these fragments are assembled, and good quality structures are selected using a scoring function based on chemical shifts and a standard molecular dynamics forcefield[74].

CS-ROSETTA

Another structure determination program that uses chemical shift data is CS-ROSETTA (chemical shift ROSETTA)[88]system for chemical shift-based protein structure prediction using the ROSETTA method[89,90]. CS-ROSETTA is an automated structure generation program that uses empirical chemical shift data gathered from proteins with known 3D structures using the SPARTA program[91]. As with the program

CHESHIRE, CS-ROSETTA uses 3 and 9-residue fragments which are selected from a fragment database using SPARTA based on their agreement with observed C_{alpha}, C_{beta}, Co, NH, H_{alpha}, and N shifts.

CS-ROSETTA uses Monte Carlo methods from PDB ROSETTA to calculate theoretical protein-like structures (decoys) using these fragments[88]. The structures are then scored based on how well they agree with the observed chemical shifts, again using SPARTA[91]. This program can also use structural restraints derived from other NMR experiments (NOEs, RDCs, etc) and/or the structures of related proteins to help model challenging targets with high accuracy[88]. It should be noted, however, that the sidechain orientations in the resulting structures are entirely based on results from ROSETTA and are not directly restrained by NMR data[88].

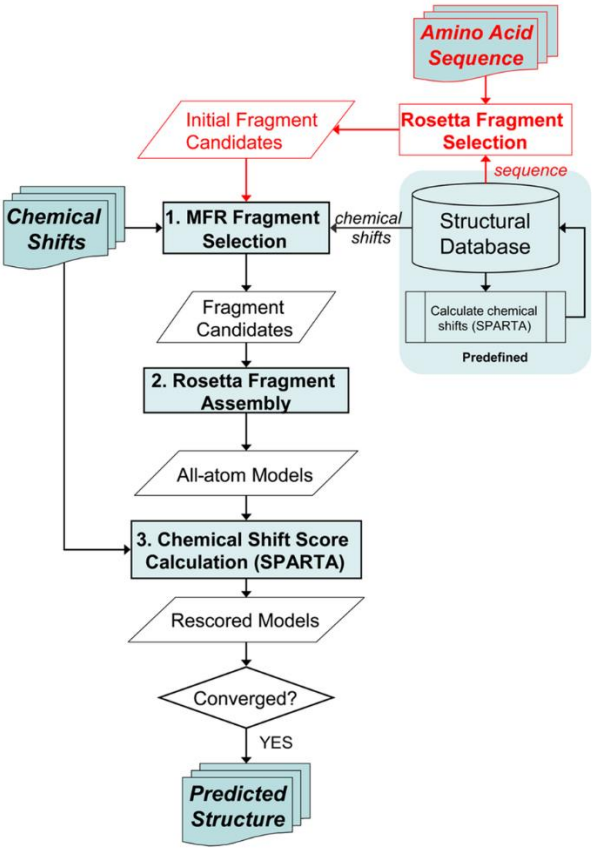


Figure 9 CS-ROSETTA data flow. Starting with an amino acid sequence a fragment selection program selects 3 and 9 residue fragments from a structural database, using calculated chemical shifts from SPARTA to select candidates. The fragment seelcted are used in the Rosetta structure calculation and the resulting structures are scored using the SPARTA algorithm using chemical shifts. The models are for convergence and if they do converge a predicted structure results[92].

Later versions of CS-ROSETTA can use 2 algorithms: Abrelax (an initio relax)[88], which is described above, or RASREC (resolution-adapted structural recombination)[93] using torsion angles, NMR chemical shift, RDCs, and nOes as

restraints[93]. When using these dynamics algorithms, it is also possible to use automatically assigned nOes using AutoNOE (automatic NOESY assignment)[94] which assigns 2d, 3d, and 4d NOESY cross-peaks, to guide the conformational search in the nOe network, which is largely inspired by the ARIA2 protocol[94].

The Abrelax (*ab initio relaxation*) algorithm will start the initialisation of structure generation by creating an instance of the protocol for the idealisation fragment sampling loop. Closure of the loop and structure refinement, and apply to invoke each protocol using the instance created during the initialisation step[88]. It is a standard protocol that uses the amino acid sequence and NMR chemical shift guided segment, and assembles them using a fragment sampler to model near-native protein structures[88,91].

For both algorithms, as seen in Figure 9 structure generation is initialised by a fragment sampling loop that selects 3 and 9-residue fragments from a database that match the observed chemical shift values. In both protocols, many structures (10000+) are then calculated using these fragments and the Rosetta protocols and forcefields to produce near-native protein structures[88]. In the basic protocol, the best structure is then selected by its match to the observed chemical shifts using Sparta for back calculation. In the RasRec protocol, this is supplemented by recombination of molecular fragments that match the chemical shifts in an iterative procedure, followed by further filtering by chemical shifts[88,91].

Fragment sampling is carried out in the low-resolution mode in four distinct stages, where in each stage, fragment insertion moves are performed on the pose[89]. This involves selecting a candidate segment from the fragment library at a given position and adjusting the torsion angles for the target protein chain to match the angles from the segment[95].

During a RASREC calculation, each pose is evaluated with a stage-specific score term and weights[95]. Low-energy candidates at each stage are retained for further sampling. Abrelax protocol provides tuneable parameters whose weights can be modified during the iteration cycles, and phases can be skipped if deemed necessary[95,84]. Setting up the structure calculation is supported by automated setup scripts that initialise method parameters to standard values[96].

RASREC is an iterative Abrelax protocol that exploits the feature of an intermediate structure to build a self-consistent core of a protein that resembles the native fold[91]. To complete this, RASREC employs several optimization strategies, embedding them into distinct phases of exploration, feature reuse, and structure refinement[91]. RASREC calculation can be run in parallel with 200-2000 processes at the same time using the MPI framework, but this is computationally expensive as this is a very large number usually run on a cluster.

RASREC uses this for multiple purposes, using a workload manager to distribute the structure calculation across computer clusters during multiple stages of resampling and refinement[93]. The main difference between Abrelax and RASREC, even though they are structurally similar, is that RASREC calculates batches of decoy structures to reach convergence, while Abrelax refines structures and measures convergence, annotating structures calculated by RASREC with information on their agreement with data[91].

Structures produced by RACRES are notably higher quality than those of Abrelax models, and it converges for structures of higher complexity and molecular mass. Also, structures calculated by RASREC are of better quality than those from Abrelax, as indicated by a better energy funnel, suggesting that optimization strategies implemented in RASREC exhibited great promise in finding near-native structures rapidly[91].

ARIA2

ARIA2 is a program that speeds up and automates NOE and chemical shift assignment[40] and structure calculation using ambiguous and unambiguous distance restraints in an iterative structure calculation scheme[97,98,99]. It does this by using simulated annealing in conjunction with a mixture of torsion angle and Cartesian angle dynamics[100]. The simulated annealing method used by ARIA2 is an algorithm in which a model of an atomic structure is heated and slowly cooled as several target functions are applied[1]. The ARIA2 protocol uses a series of iterations in which it does simulated annealing calculations on a structure. After each iteration of simulated annealing, nOe restraints are selected and analysed for violations[101].

The analysis of violations becomes stricter at each round, leading to an ensemble of structures with good convergence if the restraints are correct[102]. 2 parameters control the assignment, *p,* the assignment parameter which controls the distance from the centre of an atom which restrained atoms can be at, to be part of an active nOe, and c_n, the number of restraints per peak as seen in Figure 10[64].

ARIA2 uses CNS as a molecular dynamics and structure calculation engine for the structure determination protocol, which includes refinement in explicit solvent, either water, deuterated water, or deuterium[103].

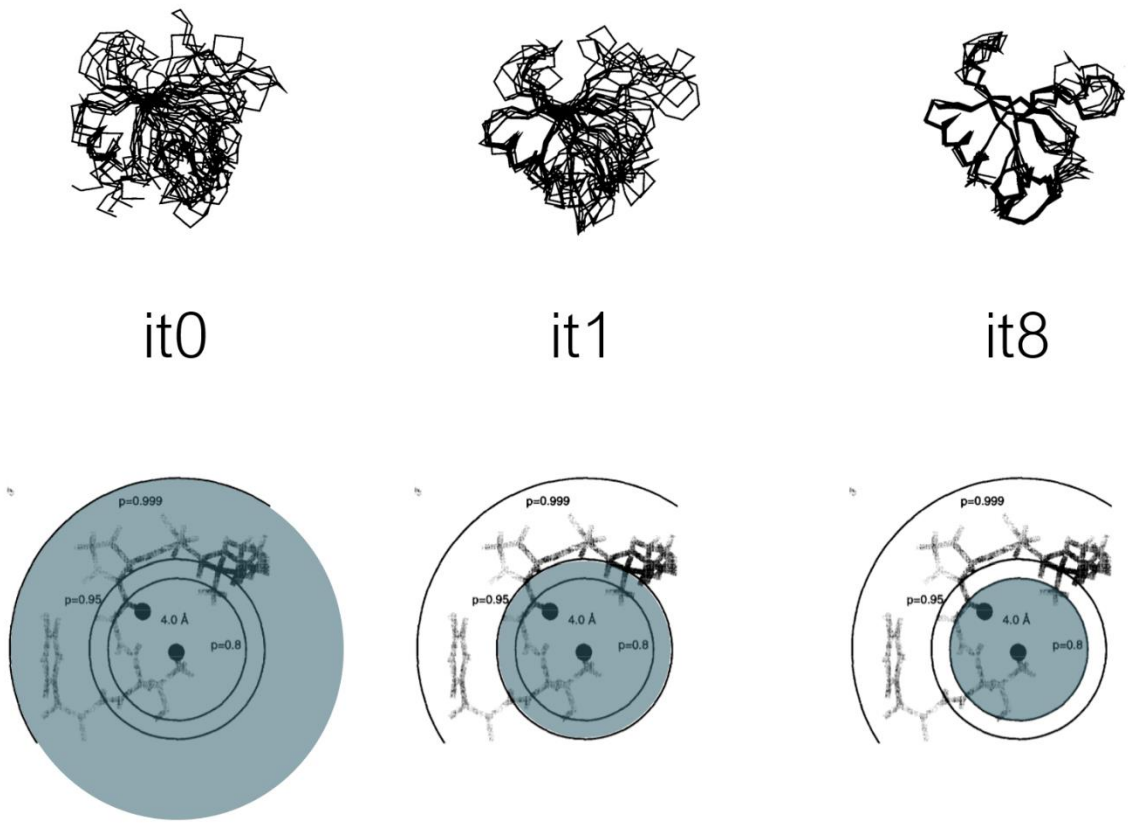


Figure 10 nOe restraints after iteration 1 to 8 (ref), where the assignment parameter P depends on the distance from the centre of the atom and c_n parameter controls the number of contributions that are active for each nOe restraint. This shows how these parameters are increased (more restrictive restraints) and how the models of the molecule's structure becomes clearer and more uniform in the ensemble[104].

ARIA2 uses project, input, and output files that use Extensible Markup Language (XML), which facilitates data validation and interchange. ARIA2 defines XML formats to describe molecular systems, chemical shifts, nOe cross-peaks, and molecular sequences. Additionally, ARIA2 offers options to get information and restraints directly from a CCPN project[1]. ARIA2 can also export the results of calculations to

the PDB, BRMB, and CCPN analysis[1]. Exporting to CCPN analysis allows for the assessment of violations directly with the acquired NMR data used for the structure calculation[1].

ARIA2 has a graphical interface that allows the user to modify all relevant project parameters, including the iteration protocol, the simulated annealing schedule, the restraining potentials, and the names and locations of data files[105]. ARIA2 is a program and a set of software libraries written in the object-oriented programming language Python.

ARIA2 uses several internal routines and external programs to evaluate the quality of the final set of structures, including WHAT IF, PROCHECK, and PROSA II[105].

ARIA2 is now used in a new method involving ARIA2 for the iterative nOe assignment and YASARA (Yet Another Scientific Artificial Reality Application)[106,107] for the structure calculation, the resulting program is called YARIA. YARIA integration uses several important structure validation checks, with the closeness to the deposited structure is similar to those calculated with ARIA[94]. With YARIA protocols building on YASARA's knowledge-based force fields, from further improvements in the quality of NMR structure calculation protocols[108].

Structure Validation

Structure validation is an important part of the structure calculation, as errors in input data can lead to invalid structures. ARIA2 includes structure validation in the structure calculation process, as it provides several approaches and external programs.

Validation of structures is not trivial and there are several packages and programs that carry out validation. The Completeness score of a structure and its restraint set describes how much of the available restraint data was observed. The second approach for structure validation is to analyse how consistent a structure is compared to known structures.

ARIA uses programs like WHAT IF, PROCHECK, and PROSA II to evaluate the quality of both the final set of structure and the solvent refined ensemble, and every program has a separate report which are stored with the structures[105].

WHAT IF[109] provides a packing quality and bump-scores (Van der Waals clashes). A database of protein structures is incorporated, which can be queried for Z-scores, which demonstrates the normality of bond/length angles and planarity distribution within the structure[109].

PROCHECK[110] is a suite of programs that checks the stereochemistry of and protein structure, including Ramachandran statistics. It gives an assignment of the overall quality of the structure as compared to a well-refined structure of the same resolution by X-ray, and it highlights regions that may need further examination[110].

PROSA II[111] again provides Z scores using knowledge-based potential derived from high-resolution crystal structures, which assess how well a structure fits its sequence and quantifies the match between the target structure and the target sequence compared to alternative folds from the PDB[111].

The completeness method is used to determine the credibility of structures is the completeness score of the restraint set that gives insight into the local reliability of each structure[112]. If the completeness of the restraints for a region of the protein is zero, this means it is not restrained. If it is one, it includes in all restraints[112]. A completeness significantly less than one indicates an unreliable region of a structure. Some reasons for low completeness are that the cross-peaks / chemical shifts are missing data set, because of chemical shift degeneracy or broadening due to relaxation effects[112]. Alternatively, cross-peaks may be rejected by violation analysis during the calculation because of errors in assignment[105].

There are several other validation methods, including ANSUUR, CING, QUEEN, and MolProbity. These validation methods are not used within CNS, XPLOR-NIH, or ARIA2 and so are not as relevant to the project, and hence they will not be covered.

Aims for Thesis

ARIA2 is an essential program for NMR protein structure determination, yet its reliance on CNS limits its access to modern forcefields and computational techniques. To enhance its utility, this work aimed to develop a modified version of ARIA2 that retains its core functionalities but replaces CNS with XPLOR-NIH as the molecular dynamics engine.

Additionally, replacing the molecular dynamics engine could unlock new opportunities for NMR structure calculation, including improved integration with contemporary methodologies. Specifically, the modified ARIA2 would incorporate CamShift, enabling chemical shift-based structure calculations in datasets with sparse nOes. Furthermore, it could support the analysis of structures using a combination of nOe and SAXS data, an approach currently absent in the CNS implementation.

Beyond methodological enhancements, XPLOR-NIH offers several computational advantages that could be incorporated into ARIA2 functionality:

- Paramagnetic Relaxation Enhancement (PRE) module based on a modified Solomon-Bloembergen equation, improving refinement accuracy.
- Advanced torsion angle potentials, leading to more reliable and faster structure calculation.
- PASD (Probabilistic Assignment of Sparse Distance restraints) module, enhancing NOE assignment reliability.
- Flexible Python scripting, allowing for adaptable workflows and automated structure calculations.

The Python integration support several new aspects to the program:

- better integration with other software via python libraries suichas the use of NEF for IO.
- Support for solid-state NMR, particularly in relation to polymorphism in protein structures.
- Cloud computing compatibility, facilitating large-scale NMR structure determinations.
- Potential AI-driven NOE assignment, streamlining data analysis and improving structure accuracy.

By modernising ARIA2 with XPLOR-NIH, the aim is to significantly broaden its computational capabilities, enhance accessibility, and functionality (including new paths for integration with the latest NMR techniques).

Methodology

In this chapter I will discuss both CNS and XPLOR-NIH in terms of molecular dynamics engines in detail in relevance to the project in state it exists, how they use nOes and distance restraints, certain sources code and scripting languages are both employed to get an understanding of how this program both stack up next to one another and if any challenges may arise when implementing XPLOR-NIH.

Following this I will discuss ARIA as a structure calculation and its approach to molecular dynamics and using the CNS as a molecular dynamics engine and its project set up, the focus of its protocols files and uses of each, as well as understanding and locating points of interest.

This will be followed the Methodologies employed by me in the project, the setup I used to run and edit the programs, the running the ARIA calculation as well and installing he structure calculations, determining the order of the calculation which will be executed in and how best the order in which to approach the implementation of XPLOR-NIH. How the initial methodology I used when implementing ARIAXC and how this would change at different stages of the implementation.

XPLOR & CNS

Both CNS and XPLOR have remarkably similar syntax and philosophies, as CNS was developed as a modern replacement for XPLOR by the authors of XPLOR. XPLOR-NIH is an extended version of the original XPLOR system with additional forcefields and a new python interface for running calculations. This allows both programs to be discussed at the same. This section will discuss how the programs are organised and move on to how both programs have their molecular dynamic facilities interfaces with ARIA2 and its automatic assignment of nOes from NOESY spectra.

Coding

In this section, I discuss CNS and XPLOR languages in relevance to ARIA2 as a molecular dynamics engine, talking about the source code and its scripting language capabilities.

Source code:

The source code of both XPLOR and CNS have related code bases with CNS being a derivative of XPLOR. CNS is written in FORTRAN77 and is compatible with UNIX-based operating systems but uses additional common extensions to FORTRAN so it can have dimension statements on common blocks and have structured loop statements. They can both also use *while* statements for looping. It also has a pre-processor that can be used with functions to make a highly modular set of subroutines and functions. In both programs structure calculation data resides in separate common blocks, which XPLOR-NIH has dynamic memory allocation can be achieved through custom FORTRAN code.

XPLOR, on the other hand, was originally written in FLECS a dialect of FORTRAN77[4]. XPLOR-NIH is an improved version of XPLOR which amongst other improvements has been translated to standard FORTRAN to remove the FLECS pre-processing stage and improve debugging. More recent additions to XPLOR-NIH have used C++ as it is more expressive and extensible than FORTRAN which lacks support for object-oriented programming techniques, and has poor character memory and memory management tools, making it cumbersome to develop new features like potential energy terms. It also makes use of C++ template libraries and borrows heavily from the standard C++ expression and container types, like lists, and includes new functionality including linear algebra component such as Matrix, and Vector classes.

This new C++ framework mirrors that in XPLORs FORTRAN code base but at the same time doesn't impose on performance. An example of the new C++ facilities is the C++ selection type which stores the selection string and calls XPLOR's atom selection routines maintaining fidelity with the XPLOR at selection syntax. Some of the new C++ features are available in the original XPLOR language however the preferred mode of access is via the scripting interfaces [python and tcl; *vide infra*].

Scripting languages

XPLOR and CNS have inbuilt custom languages which are domain-specific for structure calculation and contain features that deal specifically with the minutiae of structure calculation algorithms used with X-ray crystallography, NMR, and molecular

dynamics. This can make both CNS and XPLOR hard to learn and apply when coding without dedicated training and practice. This is in comparison to general purpose scripting languages (GPSLs) such as TCL, python and LUA that are made and used for many different purposes. These GPSLs are highly refined and were developed over multiple generations of languages and after millions of user years of scripting. They therefore have a wealth of material and support for training developers in their use.

By contrast the XPLOR and CNS scripting languages (*XPLOR-script* from here on in) are archaic idiosyncratic and thus prioritise structures and routines such as those for managing and storing coordinates in 3D space, including vectors, weights, and specific details of utility for molecular dynamics. Therefor the function-specific languages in XPLOR and CNS have uniquely difficult and unusual syntaxes, as most scripting languages have developed significantly since XPLOR and CNS was release as CNS is an XPLOR clone. Both languages are now archaic, as most scripting languages have developed significantly since XPLOR was released as CNS is an XPLOR clone.

The CNS scripting language

This CNS scripting language and interface has much in common with XPLOR. It also has several improvements including the availability of modules and procedures which improve code modularity, an HTML interface, and symbolic data structure manipulation. However, in general, its can be treated as an almost source code-compatible extension of the XPLOR scripting language. In Table 1 you can see scripting interfaces of CNS, XPLOR and Python doing a loop statement. In this example you can see both CNS and XPLOR used `while,` statements to undergo a loop. Python (on the right) uses a loop using a *for* statement. The `while` statement works as while a condition is met the loop repeats itself, in this case while `$kbon`, is lower than 10 as does calculation to the variable during the loop increasing its number. The Python `for` loops is less complicated to write as it is easier to guarantee loop termination.

Table 1 Scripting interface Difference Comparison. This Tables displays the temperature ramping loop from sa_rc_hightemp for CNS and XPLOR scripting interface syntax on the left with a loop using it while statement. The right displays Python scripting interface writing a loop using a for loop.

CNS + XPLOR syntax	Python

`evaluate(kbon=10)`
 	
`while ($kbon < 10) loop stage1`
`  evaluate ($kbon = min(10, $kbon * sqrt(1.25)))`
`  evaluate ($kang = $kbon/100)`
`end loop stage1`
	`kbon_target = 10`
	
	`n = int(floor(log(kbon_target/kbon_start)/log(sqrt(1.25))))`
	
	`for loop in range(n):`
	`  kbon = 0.005 * sqrt(1.25)**float(loop)`
	`  kang = (kbon/100.0)`

CNS & XPLOR Scripting Language Details

The XPLOR scripting language 'XPLOR-script' has elements similar to several other scripting languages such as Python and bash including structured control loops and symbol substitution. When writing code with this language it uses two data elements symbols and parameters which store and retrieves data. Symbols are typed variables, including numbers, characters strings (of restricted length, as both scripting languages allocates the same size block of memory for every string/variable), and logical variables. In Figure 11, Variables are accessed via a name denoted by a dollar sign (`$`) as can be seen below example where variables are being set, in the first case to another variable or in the second example to a Boolean constant (true).

```
evaluate ($n_rest = $data.nrestraints)

evaluate($jcouplings_on = true)
```

Figure 11 Shows can CNS scripting language access variables using a $ from CNS.py. Setting a variable n_rest to be what was contained inside of library data.nrestraints. followed by a setting a secondary variable to jcouplings_on to true.

Secondly in CNS there are parameters which are untyped elements of arbitrary length which may contain arbitrary collections of CNS commands, numbers, strings, or symbols, which are accessed on by a name proceeded by an ampersand (`&`). Both symbols and parameters may contain single data elements, or they may represent compounds of data structures or arbitrary complexity, examples of these from the file `read_data.cns` can be seen below showing one if statement which determines if `data.rdc1_choice` is equal to the string `VANGLE`, then undergoes some code. In the second part is during a potential harmonics statement setting coefficients based on a number of parameters using the ampersand seen in Figure 12.

```
! A comment to the end of the line starts with an ! mark
module (Iteration; Data; Toppar)              ! definition of a module in a separate file
                                              ! includes variables passed to module in brackets

display start read_data.cns                   ! display data to the output stream, in this case the
                                              ! file is defined by the string: read_data.cns
if ($DEBUG eq true) then                      ! start of an if control flow block which doesn't run if the
                                              ! variable debug isn't true
     set echo on message on end               ! body of if block sets the control variables message
                                              ! and echo to on (true)
end if                                        ! end of the if block

evaluate ($n_rest = $data.nrestraints)        ! evaluate the variable n_rest to have the same value as
                                              ! data.nrestraints

noe                                           ! custom syntax to define noe restrain parameters
   reset                                      ! this includes resetting the noe restraints
   nrestraints = $n_rest                      ! assigning the number of restraints to $n_rest
   ceiling 1000                               ! setting the ceiling attribute of the noe term to 1000 for
                                              ! following restraints, not there is no need for an = sign!
end                                           ! the end of the block

! ... more code not shown...

if (&data.unamb.forsit le & iteration) then   ! another if block note use of attribute rather
                                              ! than variables
   noe class dist @@NEWIT:unambig.tbl end     ! import noe restraints into the class dist
                                              ! @@ inserts the contents of a file as text each time
                                              ! the loop is run
                                              ! NEWIT: is interpreted as the value of the unix
                                              ! environment variable NEWIT which is prepended as
                                              ! text to the filename
                                              ! unambig.tbl is the name of the file (
end if
```

Figure 12 an example from a modified version of read_data.cns from the ARIA2 protocols directory. This shows examples of how CNS/XPLOR scripting languages use parameters, variables, nOe molecular dynamics statements taking a set number of parameters and how modules can be defined.

Additionally, a hierarchy of data structures can be denoted using a period (.) Thus information can be stored in compound symbols and parameters such as `SaProtocol.carinit_t`, were `carinit_t`, is a child under the `SaProtocol`, parent. Figure 13, shows further examples of the XPLOR-script syntax including evaluating statements including variables, for example `ncycle` is set to 10 by the `evaluate` function and *step* is used to update itself by incrementation. In this example these statements are followed by a series of vector statements (a novel and never repeated feature of the XPLOR-script language) which applies random velocities to all atoms using the maxwell function and the target temperature

&SaProtocol.carinit_t, using the & for parameters from the call to the module *sa_l_hightemp*. This example is from *sa_l_hightemp.cns*. This example includes the start of the module file and shows the initial uses of data parameters being passed in from an outer module or scope (*nstep, SaProtocol & Data*) using the module statement. How the module *sa_l_hightemp* would be called is described in modules and procedures below.

```
! module sa_l_hightemp.cns
! cartesian dynamics high temperature phase

module(nstep; SaProtocol; Data)                    ! engaes module calls run.cns in brackets

evaluate ($ncycle=10)                              ! evaluates variable ncycle to integer 10
evaluate ($nstep1=int(&nstep / $ncycle )+1 )       ! sets nstep1 variable dependant on nstep from
                                                   ! run.cns using & symbol and ncycle variable using
                                                   ! the £ symbol

do (vx = maxwell(&SaProtocol.carinit_t)) (all)     ! velocity maxwell function on all atoms take variable
do (vy = maxwell(&SaProtocol.carinit_t)) (all)     ! from SaProtocol.carinit_t inside of run.cns
do (vz = maxwell(&SaProtocol.carinit_t)) (all)     ! in all direction in 3D space using & symbol.

evaluate ($ini_noe = 10.0)                         ! sets variable using $ symbol  inti_noe to 10.0
evaluate ($fin_noe = 50.0)                         ! sets variables using $ symbol fin_noe to 50.0
evaluate ($ini_amb =  1.0)                         ! sets variable using $ symbol k_noe to 1.0
evaluate ($k_noe = $ini_noe)                       ! sets variable using $ symbol k_noe to be what ini_noe
                                                   ! to 10.0
evaluate ($noe_fac = ($fin_noe/$ini_noe)^(1/$ncycle)) ! sets noe_fac calculation with other variables
```

Figure 13 part of the file sa_hightemp.cns showing evaluate, vector do statements, using variables ($), and parameters (&), as well as the maxwell function for setting random 3d velocities compatible with a particular temperature).

Modules and procedures

A CNS script is built from a script and one or more modules [files that behave as separate functions] that exist as separate files on disc with each containing a collection of CNS commands that carry out the required task. As well as the script text modules also have syntax to define the parameters passed into the modules each of which has its own namespace. This is in contrast to XPLOR where there is an inclusion-based use of external code using the @ symbol filled by a filename. This inserts the text of the second file into the first and implies a _flat_ name space. The structure of CNS script files is similar to other languages such as FORTRAN or C++, which define a set of input parameters to functions that are passed into them when they are invoked. However, it should be noted that CNS modules have no

return values. Modules allow CNS functions to be developed independently and reused increasing the clarity of the implementation of the underlying algorithms.

The parameter passed into a module inherit the scope of the calling task file or module, and as such behave like analogous local variables. Examples of how a module is called is shown in Figure 14, which shows part of *refinement.inp* calling *flags_new.cns* module inside of the protocol directory inside the protocols directory of a run. The flags module sets up variables controlling which functionalities of the protocols in *refine.inp* are used.

```
!set the energy flags
@@RUN:protocols/flags_new.cns(Iteration=$Iteration; Data=$Data)
```

Figure 14 A snippet of refine.inp showing a call to protocol/ flag_new.cns using the @@<environment-variable>: <path-and-file_name>(<parameter>,...) syntax. This shows how modules are used with parameter passing in CNS; note the RUN variable is taken from the shell ENVIRONMENT and defines where protocols are located on disk too increase portability and flexibility.

Library modules

The CNS code base includes library modules that store static information called library modules. Most of these are for x-ray crystallography, but they also include random coil chemical shifts and a molecular parameters database which contains multi-dimensional probability distributions, of preferred sidechain rotamers in protein and nucleic acids[2, 3].

Task files

Task files consist of CNS language statements and module invocations [3]. These files provide a range of predefined CNS's protocols for either XRC and/or NMR structure. They are not used by ARIA2 which uses its own graphical user interface and XML project file. These were not employed when implementing XPLOR-NIH and hence will not be covered in any more detail.

Contrasting XPLORs scripting language with CNS

XPLOR-NIH continues to implement classic scripting language of XPLOR (XSL). As discussed, is remarkably similar to CNS's only differing in detail, for example it doesn't contain modules only direct text inclusion using the @@ and @ syntax. Along with using C++ for its latest components and forcefields, XPLOR-NIH interfaces to two GPSLs: Python, and TCL using the swig library[4,5]. Python version 2 is supported by XPLOR-NIH versions less than 3.0 and python 3 is provided by

XPLOR versions greater than 3.0 (XPLOR 3.6 supports Python 3.9.9). The most important of these GPSLs for this project is Python 2, as ARIA2 already uses this in its implementation, aside from the parts of the package used for molecular dynamics calculations which use the CNS language. The XPLOR-NIH Python package has an extensible toolbox for developing further functionality and has a precompiled library of functions wrapping (most of) XPLORs molecular dynamics commands. The use of python in XPLOR-NIH provides a modern environment including flexible and well-designed loop and control flow statements. This allows XPLOR-NIH's python scripts to interact with external libraries while at the same time calling low levels functions in the XPLOR language using the `xplor.command` function and accessing the values of XSL variables. The flexibility of the XPLOR interface allows advanced techniques to be used including:

1. Building XPLOR forcefields in pure python.
2. Using prebuilt functions to build and create topology and parameter and structure definitions.
3. Complex temperature ramping protocols across a group of structures during ensemble molecular dynamics calculations.
4. Simple object orientated manipulation of forcefields and their parameters.

XPOR-NIH also provides an interactive mode [started using pyXplor], or REPL (read evaluate print loop) where it is possible to directly interact with the state of the program from both the Python language and the XSL. XPLOR-NIH's Python interpreter pyXplor is invoked with Python script files but a legacy interface is also provided by the `xplor.command`.

Other examples of library functions include:

1. The `protocol.iniStruct` function which loads atom definition and covalent coordinate definitions from a protein structure file (PSF).
2. Access to global settings such as random number seeds and error logging levels *via* the built-in `simWorld` object.
3. Access to arrays of atomic properties can be obtained via methods of simulation objects in the built-in XPLOR module for easy manipulation of atomic properties can be obtained via `AtomSel` objects. The atom selection language is identical to that in the XPLOR interface.

4. Structures are easily read and written via the PDBTool class [5].

To give a concrete example of the differences between the XSL interface and the Python interface Table 1 shows how dynamics are accessed from the two interfaces with the Python version using the IVM (Internal Variable machine) implementation of torsion angle molecular dynamics.

Table 2 a comparison example of scripts performing rigid-body minimization in the XPLOR and Python scripting languages taken from ARIA2.0: Nuts and Bolts [4].

XPLOR	Python	Description
Dynamics internal	From ivm import IVM	Load IVM module
itype = powell	Min = IVM ()	Create an IVM object
depred=0.0001	min.setStepTypepet(powell)	Choose minimisation
nstep=1000	min.setDEprep(0.0001)	Prediction of initial energy drop
etol=0.0000001	min.setNumStep(1000)	Number of minimisation step energy
nprint=50	min.sETolerance(1e-7)	tolerance for exit condition
group=(resid 2:249)	min.setPrintInterval(50)	How often to print energy values
group=(resid 301:385)	min.GroupList([AtomSel(‘resid 2:249’),	Atom selections to group
end	AtomSel(‘resid 301:385’)])	into rigid bodies
	min.run()	Perform minimisation

CNS packages overall lay out and how they interact with CNS’s sub systems

For both XPLOR and CNS there are several components of the program that communicate with each other. Figure 15 shows a diagrammatic overview of CNS. All the systems can be controlled by the user. The HTML interface is not used by ARIA2 which operates scripts, which can call modules and procedures. Scripts are all written in CNS’s scripting language and communicate via the interpreter with the low-level functions in CNS’s Fortran source code. Calls to functions and modules can be made directly or indirectly by the user to give maximum control and flexibility.

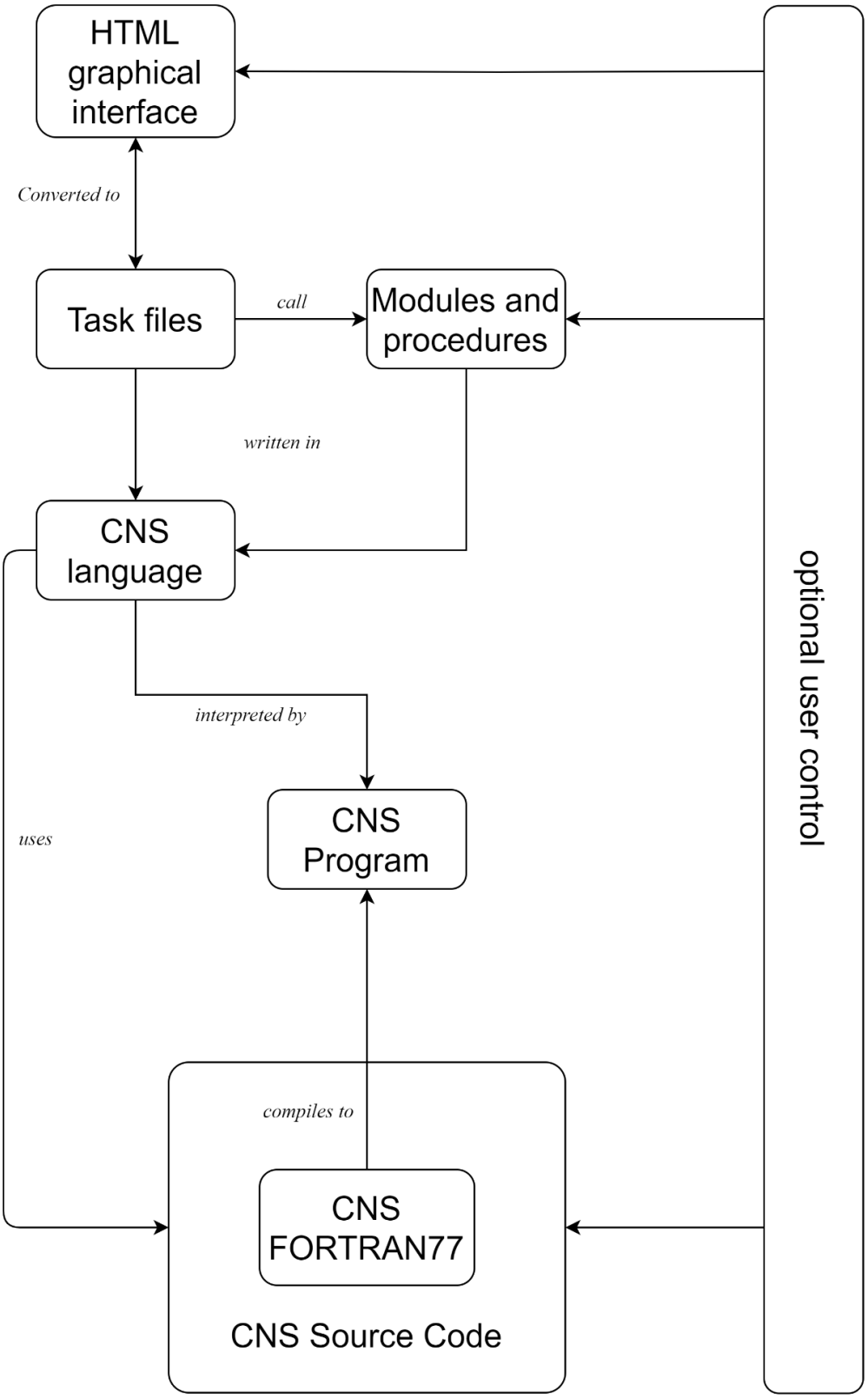


Figure 15 Shows the overall layout of CNS and how the different components interact with one another [2].

XPLOR-NIH on the other hand contains 6 separate sub-systems that communicate with one another to providing several ways to use the program at via both a high-and

lower-level interfaces that are increasingly harder to use but more powerful in the order TCL/Python /> XSL> C++> FORTRAN all of which interact relatively seamlessly. The interactions between the sub systems are connected by direct function calls (XSL, C++ and FORTRAN) or SWIG wrappers (TCL/Python to C++/FORTRAN). SWIG wrappers are procedurally generated and allow an interface directly to XPLOR molecular dynamics code from high level languages.

The GPSL SWIG wrappers mostly contain functions and parameter definitions and are used in much the same ways as the previously discussed CNS scripts to specify parameters, initialise molecular dynamics & minimisation calculations and analyse results. However, it should be noted that the python libraries being written in an objected-orientated language do use OOP features examples including the selection of atoms and querying their attributes. In Figure 16 below you can see a simplified organisation of XPLOR-NIH's interpreters and interactions. The XPLOR-NIH libraries have a clear organisation and provide a direct route from the GPSLs (python and TCL) to raw XSL commands via `xplor.comand()`. However, most interactions are via higher-level libraries which provide better-designed and simpler layers, as a library on top of the raw XPLORcommand and swig interfaces. These higher-level interfaces allow the user to communicate quickly and efficiently with XPLOR-NIH, are time-efficient to use, and provide a range of options to the user. The XPLOR-NIH package is also integrated with the VMD molecular graphic package allowing users to visualise molecular dynamics simulations carried out by the program in real-time or by reading trajectories.

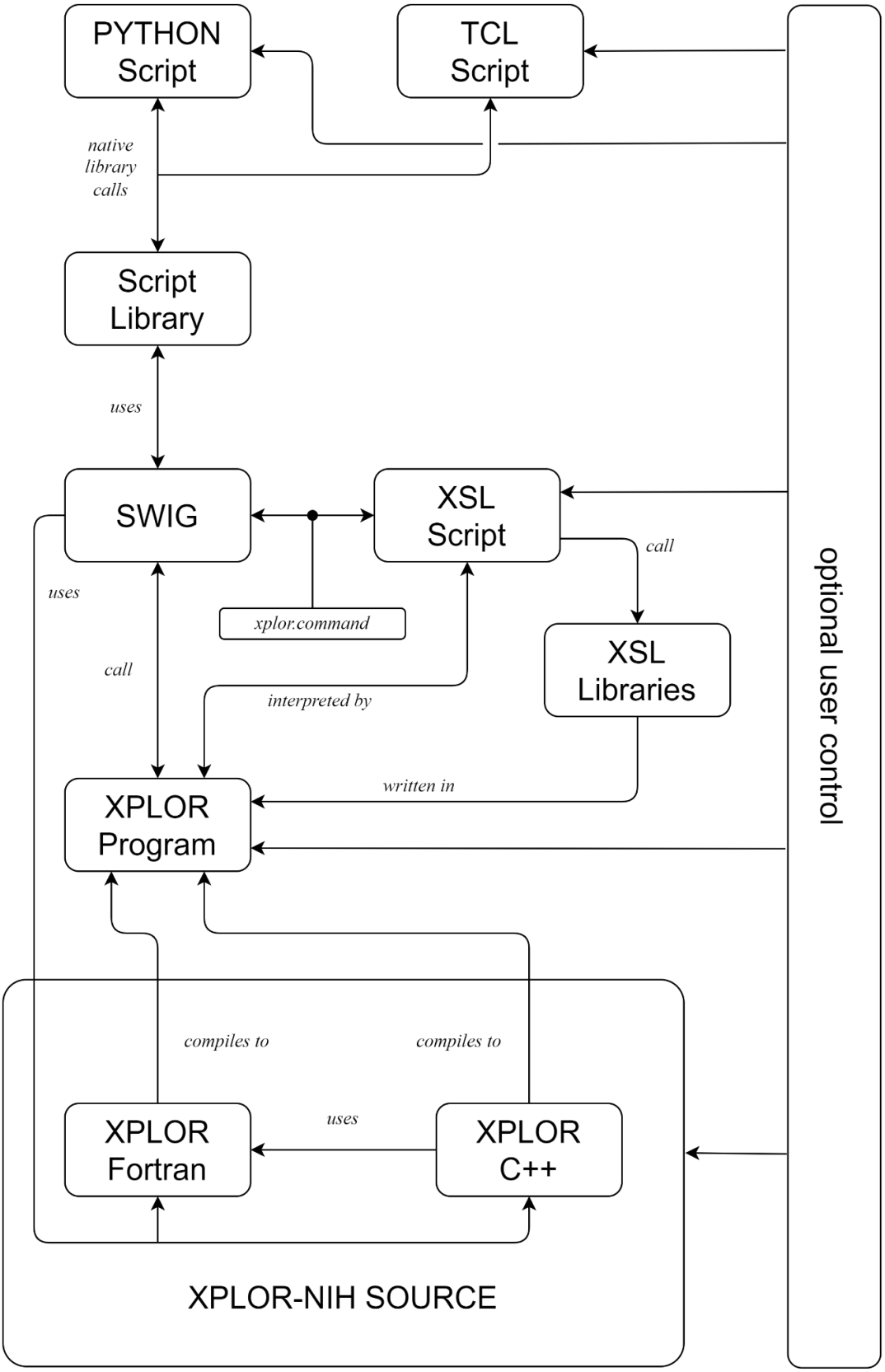


Figure 16. Interactions between the sub-systems in the XPLOR-NIH program. The user can interface with several options currently including the XPLOR-NIH source code, TCL script, Python, XSL.

Figure 16 displays the layout and interaction inside of XPLOR-NIH which are considerably more complex than within CNS but has the advantage of the GPSLs which are more modern, and flexible than the very specialised XSL scripting languages. This allows for more flexible choices when utilising the programs capabilities, in the case of this project allowing for the coding in the more powerful and syntactically consistent and lean python scripting language which also has far more support and very large numbers of users [python has more than 7 million unique windows downloads in 2021-2022] and vast amounts of training information [google search: python programming for beginners produced 6,000 results in 2023].

Parallelisation

Parallelisation is a computation technique that allows for multiple physical or hyper threaded processors to be used for different or related proposes simultaneously within the same or different computers. CNS implements parallelisation using a single program multiple data (SPMD) model, with the parallel virtual machine (PVM) parallel programming environment being used to provide portability across computing platforms. Parallelisation of the program is carried out at 2 levels, through processors and groups.

Parallelization in XPLOR-NIH can be achieved by the `StructureLoop` class. This causes actual parallel execution by spawning subprocesses by specifying the -parallel and -machine flags in the pyXplor command line which active process monitoring of the primary-parent process and child-worker processes and shutdown of parent and child processes.

ARIA2 uses its own parallelisation model directly without using the CNS parallelisation constructs via calls to external csh scripts on the local machine or via ssh password-less sessions on remote machines. This model is simpler than CNS's inbuilt parallel programming model and is suitable for NMR structure calculations which are embarrassingly parallel as they don't require state to be transferred by inter process communication between the calculations as they are independent.

XPLOR-NIH's native structure calculation protocols (scripts in input and Marvin/PASD) use independent calculation methods closely related to those used in ARIA2 using torsion and/or cartesian angle dynamics. During these calculations as with ARIA2 structures start with randomised initial coordinates and velocities. This randomisation ensures a complete sampling of the potential energy surface so leads to a better probability of finding the minimal global minimum and helps to identify the structural space that is consistent with NMR data. XPLOR-NIH also has the capabilities to carry out ensemble-based calculations where restraints are satisfied as an average over multiple dynamics trajectories in parallel.

Molecular Dynamics for Structure Calculation

This section discusses the molecular dynamics and minimisation facilities provided by CNS and XPLOR-NIH and will focus on how ARIA2 employs them and highlight significant differences.

ARIA2 currently employs several restraint types to carry out molecular structure calculations. The main forcefield used is the nOe function which implements distance restraints. This can also supplement with restraints from H-bonds, J-couplings, residual dipolar couplings, and dihedral angles. Structure calculation carries out energy minimisation through simulated annealing using a mixture of torsion angle and cartesian angle dynamics and newton-based energy minimisation. ARIA2 as discussed above uses the CNS molecular dynamics engine to power the structure determination program. XPLOR-NIH provides several additional forcefields for structure calculation using several observables including secondary chemical shifts (including $^{13}C\alpha$, $^{13}C\beta$, HN, N, HA, CO) heteronuclear T1/T2 values, solvent PREs, spin labels (PREs), SAX and WAXS scattering curves, chemical shift anisotropy values and cryo-EM envelopes.

The nOe forcefield

Distance restraints from nOes are based on either harmonic functions, quadratic square-well functions, or quadratic asymptotic functions inside both CNS and XPLOR [2,4,5]. The form of the function is

$$\Delta = \begin{cases} R - (d + d_{plus} - d_{off}) & d + d_{plus} - d_{off} < R \\ 0 & d - d_{minus} < R < d + d_{plus} - d_{off} \\ (d - d_{minus} - R) & R < d - d_{minus} \end{cases}$$

Equation 1. Quadratic square-well function.

In Equation 1 d_{plus} and d_{minus} are described by the `ASSIGn` statement in CNS as seen below.

```
assign <selection> <selection> d dminus dplus
```

were d_{off} is specified by the `SQCOnstant` statement and the `R` defined as the `AVERage` distance between selected sets of atoms (either 6Å, 3Å or a of sum of both). This allows the area in which the lowest energy can be found to be larger as it creates a square well (bottom) of energy that would fit this allowing for a larger energy range and increasing range of nOes potential to fit easier. An example of such a graph can be seen in Figure 17 which displays a plot nOes potential energies against distance in angstroms to graphical representation of a square-well function.

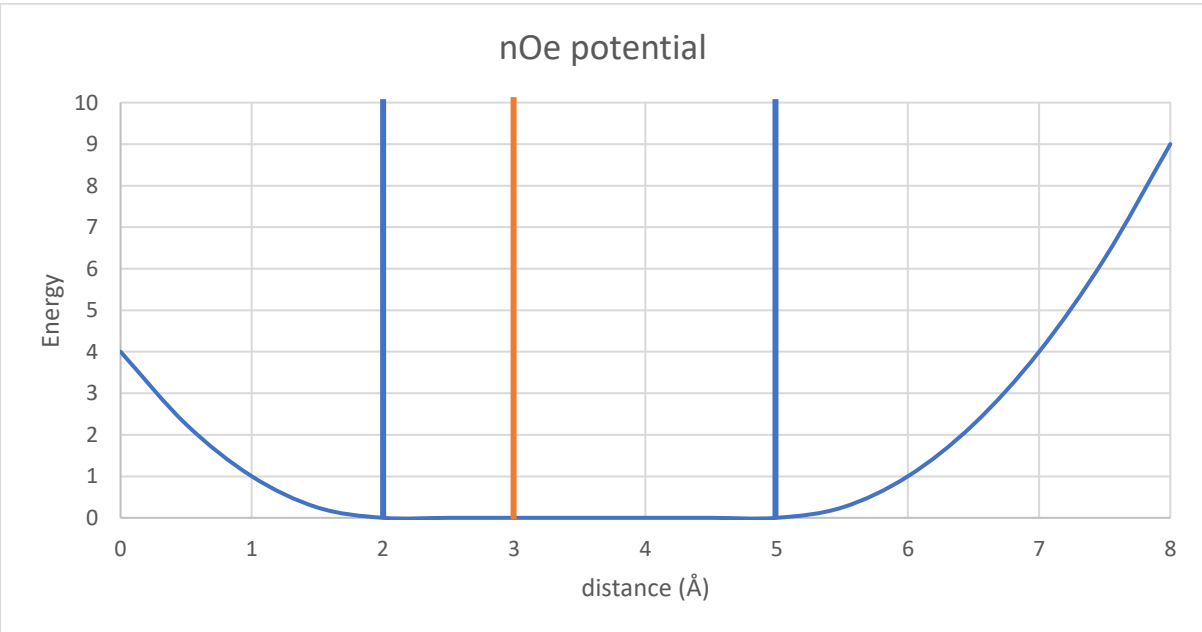


Figure 17 Display a nOe soft square-well potential function showing a graph of energy against distance in angstroms. The optimum energy is at 3Å, with energies increasing rapidly outside the well at 2Å and 5Å.

In Figure 17, as a square-well function is a more open-ended bottom to graph, with a normal log well harmonic function would less area for the nOe to find the correct distance and hence lowest energy. This allows for more error within the calculation and will produce better structures. In Figure 17, shows this by medium 3 Å, producing the optimum distance and best energy with from 2 Å to 5 Å, allowing for the same result in a normal log well harmonic function this would be a point at 3 Å and be extremely difficult to reach causing large energies within the calculation as there are thousand or ten of thousands of nOes measurement for a single molecule/protein. Potentially causing the structure calculation to fail.

In the case of prochiral centres with unknown stereospecific assignments or ambiguous nOe assignments sum averaging is used (Equation 2).

$$\overline{\overline{D}} = \sum_{n=1}^{N} (r_n^{-6})^{-1/6}$$

Equation 2. Ambiguous nOe assignments sum averaging.

where the distance is summed across the ambiguous restraints prevents the dominance of violated terms in a combined energy function. In Equation 2 $\overline{\overline{D}}$ is the effective distance N is the total number of distances and r_n is the value of each distance. Ambiguity of restraints occurs because of chemical shift degeneracy either due to peak overlaps or because the system is oligomeric. Figure 18 below shows how the energy of the restraint changes as the distance between the atoms change. It should be noted that short distances can dominate a set of restraints as the restraint energy typically goes up as r^n ($n \geq 2$) beyond the optimum of the nOe potential. Sum averaging or ambiguous distance emphasises the closest ambiguous restraints and are used to avoid one unsatisfied restraints dominating the potential.

Ambiguous Distance Restraints (ADRs)

-Ambiguous NOEs are those for which more than one assignmen is possible

$$V_{total} = \sum_{a=1}^{N_\delta} V_a$$

- **The volumes (intensities) of these can be treated as sums of possible contributions…..**

$$V_{calc} \approx \sum_{a=1}^{N_\delta} d_a^{-6}$$

- **….. and can be approximated with a 6th power law**

$$\overline{D} \equiv \left(\sum_{a=1}^{N_\delta} d_a^{-6} \right)^{-1/6}$$

- **Ambiguous distance restraint: an effective or summed "distance" between more than two points**

-Ambiguous NOEs can be used in iterative procedures for simultaneous structure calculation and NOE assignment

Figure 18. The overall relationship between the energy of nOe distances restraints and the ambiguous distance from multiple atom pairs.

CNS and XPLOR also allow for direct refinement against nOe intensities using either the full-relaxation matrix method or quasi-relaxation matrix method which iterates between energies based on distances and back calculated nOe intensities from a relaxation matrix until convergence has been achieved[2]. These techniques avoid under estimation of distances due to the use of the ISP (isolated spin pair) approximation which ignores spin diffusion across several nuclei connected by short distance.

The stereo related ^{1}H-chemical shifts and nOes from prochiral protons and methyl groups are automatically handled by floating chirality assignment in ARIA2 without the need for making a prior stereo assignment.

Chemical shifts

Chemical shift restraint includes terms calculating shifts from ring currents, magnetic susceptibility, and electric field effects. An example of a chemical shift forcefield is Camshift which uses an empirical model which is mined from the chemical shifts and structures in the PDB and BMRB databases.

Other restraints [RDCs & Relaxation]

Residual dipolar couplings (RDCs) and heteronuclear T1/T2 restraints provide long-range structural information as they provide information on the orientation of bond vectors (typically H-N or H-C) to an external axis system. In the case of RDCs the alignment is to the alignment tensor imposed by an alignment medium (typically a liquid crystal). In the case of relaxation differences in T1/T2 relaxation are caused by the anisotropic radial diffusion of the molecule. This can provide structural information for the observed bond vectors when relaxation due to internal molecular dynamics doesn't cause additional relaxation. For these terms the external system is represented by an artificial tetra-atomic molecule consisting of four atoms, representing the x, y, and z axes of the tensor and its origin o. As the orientation of the axis system is generally not completely defines before calculation starts it is allowed to float during the simulated-annealing calculations.

Non bonded interactions

While non-bonded interactions are not a structural constraint, their control during structure calculation is particularly important. Non-bonded interaction may be modelled as simplified repulsive (REPEL) or full attractive and repulsive Van der Waals potentials (VdW). As the starting points for the NMR structure calculation and refinement protocol structures are randomised extended strands corresponding to each molecular entity (polypeptide chain or oligonucleotide acid strand) or pre-folded structures that require refining. The VdW potential terms must be reduced during calculation to allow molecular angles to be removed and for parts of the molecular structure to pass through each other on their way to the global minimum energy structure.

ARIA2, XPLOR, and CNS have similar basic protocols for structure calculation. The first section of the protocol consists of reading the various data structures, including the sequence of the molecule (parameters and topology), which are followed by the

initialisation of the 3D structures with random conformations, often starting from random coordinates. A constant high temperature cartesian or torsion angle annealing stage is the first calculation carried out. This is followed by a slow-cooling stage with either torsion angle or cartesian dynamics. Finally, an (optional) additional cartesian cooling stage and minimisation stage follow. Restraint and molecular potentials are ramped during the annealing with separate terms being applied for different classes of distance restraints including hydrogen bonds, ambiguous distance restraints, unambiguous distance restraints and VdW interactions. The overall process is called simulated annealing. Multiple calculations [typically 20+] are performed by starting the simulated annealing calculation with different randomly selected initial atomic velocities and structures to explore conformational space and avoid the final structure being biased by the molecules starting geometry.

Once annealing is completed analysis of deviation and violations of experimental and covalent geometry restraints is carried out and written to the header sections of the coordinate files corresponding to the particular calculations. The acceptability of the trail is tested and analysed using energies or restraint violations to determine a subset of different structures. The major difference between ARIA2's protocols and XPLOR/CNS's original protocols is that ARIA2 carries out several rounds of calculations with assignment and selects restraints to transfer between rounds by whether they are violated or not, while varying selection parameters to reduce the restraint space. XPLOR/CNSs original protocols required manual assignment; the latest protocol included with XPLOR-NIH: Marvin (aka PASD) also carry out automated assignment.

PDB I/O & other tools

Both the XPLOR and CNS structure calculation engines also have many general-purpose tools to help generate and manipulate molecular structures which because of their lineage are almost identical. These tools are general purpose and can deal with proteins, nucleic acid, carbohydrates, and other chemical structures. The coordinates statement can be used to copy atoms and geometry, fit one set of atoms to the geometry of another set of atoms, orient rotate and translate sets of atoms and calculate RMSDs / radii of gyration. Finally, the statements with the same syntax can be used to read and write PDB structure coordinate files in XPLOR/CNS, however, in

XPLOR-NIH the reading and creating of coordinate files by the coordinate function is superseded by the much more effective python PDBTools class.

ARIA2s interaction with CNS

ARIA2 needs to integrate with the CNS program to carry out molecular dynamics. This section describes how ARIA2 interfaces with CNS which also serves as a basis for how the XPLOR implementation was designed. Here I will describe ARIA2 and how it is structured, and will it be possible to accomplish the theory I suggested of implementing XPLOR-NIH as a molecular dynamics engine.

ARIA2 is a program that speeds up and automates the nOe assignment process using ambiguous and unambiguous distance restraints using an iterative structure calculation scheme. It does this by simulated annealing in conjunction with nOe assignment. The nOe assignment is an integral step in determining the 3D structure of biomolecular structures from NMR data. The nOe peaks carry information about internuclear distances which are derived from the cross-relaxation rate between two atoms that are measured in a NOESY experiment. Other complementary information can also be used with the nOe-based approach as discussed in the introduction. As NOESY restraints serve as the primary source of structural information ARIA2 has several modules dedicated to analysing cross-peak intensities / volumes quantitatively:

1. The isolated spin-pair approximation
2. Full matrix relaxation analysis
3. The log-normal forcefield

Data representation

All input data used directly by ARIA2 is in the XML (eXtensible Markup Language)[100] format, which defines portable and human-readable text format for information exchange and document validation. This streamlines the data validation process and represents the experimental information as explicitly as possible. An important set of data that are read in this format are Ambiguous Distance Restraints (ADRs). These connect measured peak volumes/intensities, possible assignments, and upper and lower bounds. Other data read in XML formats include chemical shifts for each peak,

assignments, the proteins sequence, and flags and parameters from the project file and the project file itself.

ARIA2 source code structure

ARIA2 uses Python2 as its implementation language as an interface between CNS and its own internal data structures. The ARIA2 program is divided into 2 hierarchies of files on disk *cns/* and *src/* which contain its major components.

`cns/src` -	patches for cns1.2 that allow ARIA2 to run.
`cns/protocol` -	cns protocols used by ARIA2 to carry out structure calculations and analysis.
`cns/toppar` -	topology and parameters files defining amino acid structure and force forcefield parameters.
`src/py` -	ARIA2's Python source code
`src/csh` -	csh utility scripts (such as check host)

The *py/aria* directory has 3 subdirectories and a series of Python source code files that define how ARIA2 works, reads data, and analyses it. Examples of source code files inside the *py* folder include.

- *NOESYSpectrum.py* which defines classes and routines for reading and managing NOESY spectra and peaks.
- *Assignment.py* contains class and functions that deal with the assignment of ambiguous and unambiguous nOes that are stored in instances of *NOESYSpectrum* class.

An example of how the classes interact and are divided into logical units is the class *NOESYSpectrum* is bundled with *AssignmentXMLPicker* class which reads in assignment data from XML files, using methods to read and set atom cross-peaks, etc. This ensures only the correct nOe information is read and stored in the class instance. The first file that needs to be changed when porting AIRA2 to use XPLOR-NIH was *cns.py*. This was one of the most important files for the project as it defines the interaction between the CNS program and its algorithms for molecular

dynamics and ARIA2. The class that defines these interactions is CNS which is a subclass of *StructureEngine* and defines a general interface to structure calculation engines. In this role it:

1. Organises which directories are to be used for temporary files and working directories.
2. Where and when to copy files into the working directories.
3. Where to place the results of the structure calculation.
4. Interaction with function in `xmultils`, to read in data and setting for ARIA2 itself and settings used by CNS.
5. Arranges the protocol file/classes/modules for the projects to be copied over into the working directory of the structure calculation.
6. Contains and organises a set of scripts that files are used to analyse the models and energies of the molecules during the calculation. Examples are *energy.inp*, *minimize.inp*, *rmsd.inp*, and *noe_violations.inp*. These files also organise information about the molecule that is required for running the structure calculation such as *ambig.tbl* and *unambig.tbl* that contain ambiguous and unambiguous restraints. It also initialises peak selections for calculation by setting several structures including *DATA_AMBIGUOUS*, *DATA_UNAMBIGUOUS*, and *DATA_HBONDS*.

Other files in the *py/* hierarchy provide several services such as defining molecular structure (Chains, Residues, and Atoms), defining the ARIA2's GUI (graphics-user-interface) defining data analysis and reports, etc. However, all of these use the interface in the class *CNS* when interacting with the structure calculation engine so compartmentalising and modularizing the program and limiting the number of files that needs to be modified to change the structure calculation engine.

Run time project setup

During the setup phase ARIA2 write a set of subdirectories which are used during the calculation and contain all information required regarding the structure calculation, data used, and analysis of models. This directory structure can be seen in the Figure 19.

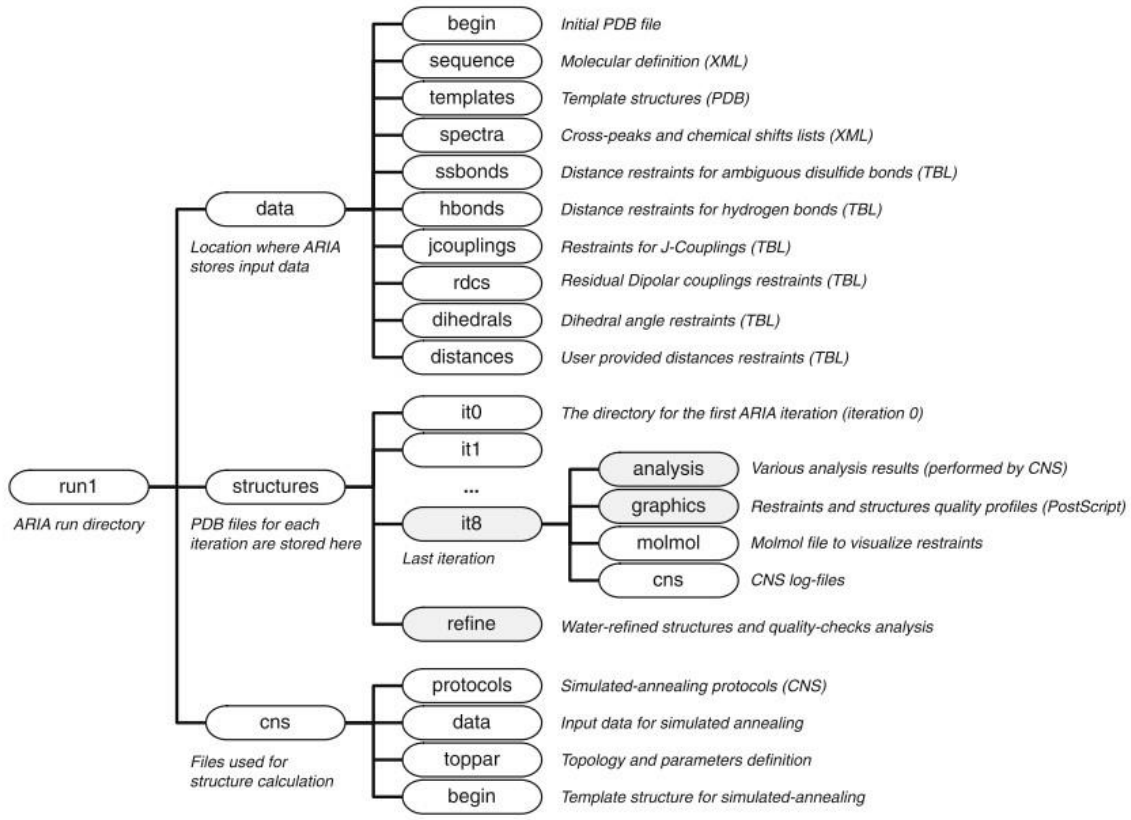


Figure 19 The run time directory tree of ARIA2 (for run1) and information on what each directory contains, and what each file does. The three main directories are: Data - relevant information for this structure calculation, Structures- output and analysis of structures for each iteration including log files and cns – parameters and protocols used by CNS for simulated annealing. [89]

As the setup of the program occurs it creates working subdirectories for the program to save information about each structure calculation iteration (it0-it8 & refine). These directories contain data for each structure calculated and the restraints used (pdb files restraints and floating stereochemistry assignments) which are created on the fly while the calculation is running and can be accessed at runtime. Figure 19, shows a directory tree of the run time set up of the program. The `data` directory contains information about the molecular input data (initial structure, molecular definition, restraints etc). The `structures` directory contains directories (`it0-it`) that contain the output from each calculation. The outputs include using the tudor domain HRDC as an example

1. `.pdb` files that describes the structures `<structure-name>_<num>.pdb` where `structure_name` is the name of the molecule (for example HRDC) and `num` is the structure number
2. `.float` files that describe the stereo specific assignments for each relevant residue and group of atoms

3. The nOe restraint files *ambig*.tbl and *unamig.tbl*
4. the analysis of nOe restraints `noe_restraints.<category>` where category is one of: `ambiguous, merged, unambiguous, assignments, rms (variations)` and `violations`
5. a report for the iteration (`report`)
6. and best fitted *fitted_<structure_name>.pdb* and average structures *structure_name* is the name of the molecule (for example HRDC)
7. output logs for the cns structure calculations in the `cns` directory

The CNS directory contains 4 sub directories. The most important contents of these are

1. *toppar* which contains the topology and parameters available to use during structure calculations copied from the main ARIA2 distribution
2. data which contains data used by XPLOR principally the molecular sequence as a *.pdb* file
3. `protocols` which contain all the protocol files from the main distribution used for structure calculation which are copied from the main ARIA2 distribution
4. begin which contains the initial data for the structure calculation including
    a. the protein structure file *<structure_name>.psf* where `structure_name` is the name of the molecule (for example HRDC)
    b. the template structure pdb file *<structure_name>_template.pdb* where *structure_name* is the name of the molecule (for example HRDC)
    c. a table file listing stereoisomeric methyls *methyls.tbl*
    d. a script to setup a list of atoms that can be swapped during floating stereospecific assignment *setup_swap_list.tbl*

Parallelisation and Job Management

As structure calculation is resource-intensive ARIA2 provides process-based parallelisation. This can be configured using the project file by defining the number of processes that should be used and the hostnames to use (which should have password less access). Alternatively, parallel computation can also be setup in the GUI interface, which acts as a wrapper around the project file. ARIA2 also acts as a job manager and scheduler and will pick up and complete a calculation if it is interrupted part way through.

ARIA2's structure calculation

ARIA2 structure calculation can be separated into several stages.

1. Preparation / setup
2. Iterative structure calculation and nOe assignment
3. Structure refinement in explicit solvent
4. Analysis and validation of the resulting structure

It should be noted that stage 1 is run as a separate command `aria2 - s <project-file>` and stages 2 – 4 are run by `aria2 <project-file>`.

Preparation stage

The preparation stage involves filtering data, creating the molecular topology, and initial nOe assignment of all possible assignments for each peak within chemical shift tolerances. These assignments are called the root assignment and are stored in the *spectrum.xml* file. Assignment and filtering of data is the process in which ARIA2 creates basic chemical shift assignments (the root assignment) and checks them for consistency. This is carried out in three stages.

1. Creations of unique assignments consisting of one atom and one chemical shift.
2. Groups of equivalent atoms are assigned to exactly one chemical shift including, floating chirality assignment.
3. Chemical shift data is filtered and cleaned including removing peaks that are on the diagonal which is completed by comparing the shift on the dimension and the chemical shift tolerances. Peaks with incomplete assignments and negative peaks are also removed at this stage.

There can also be user input at the preparation stage to specify an initial template structure. It must be noted however, if the initial temperatures of the structure are set higher, it may result in more violated tolerances, and thus must be reduced to a small value as they will now be at a larger angle and distances at higher temperatures as atoms has more kinetic energy at higher temperatures.

The second process during the preparation step is creating the molecular topology. This uses the CNS script `generate.inp` which creates a molecular topology file (MTF) that gives information on the molecule's name, the chains, and atoms present,

the charge of each atom, the mass of each atom, each atoms covalent connectivity and the atoms taking part in structural constraint terms of the forcefield (angles, dihedral angles etc). This is followed by the CNS script *generate_template.inp* which constructs molecules in extended conformations from partially randomised atom coordinates, carries out molecular dynamics on the structures to achieve acceptable covalent geometry randomises the proteins phi (ϕ) and psi angles (ψ), and writes the initial coordinates to PDB files.

In the initial nOe assignment ARIA2 uses the chemical shift list to derive possible assignments for every cross-peak present in the input peak lists,

Chemical shifts are assigned to cross-peak whose frequency window matches their chemical shift, and the frequency windows are defined by isotope; for example, ^{1}H shift windows are typically tighter than for ^{15}N as ^{1}H spectra have smaller sweep-widths and better resolution. Frequency window sizes are an important parameter that needs to be chosen carefully. Narrow windows lead to potentially incomplete seed assignment, while overly generous window sizes can produce a highly degenerate initial assignment that can cause severe convergence problems. Completeness of chemical shifts should typically not fall below 90% to achieve reasonable convergence for structured regions. Convergence checks are discussed in further detail later.

ARIA2 provides two places where manual assignment can be made. Firstly, assignments contained in the original peak list are passed through to the spectrum XML files during data conversion and take priority over automated assignments. Secondly, the user can manipulate the spectrum XML file to add a manual assignment. These manual assignments enter the seed assignment list and are not modified during calculation.

Iterative structure calculation

The iterative structure calculation also contains several stages of its own involving.

1. Analysis of the structure ensemble
2. Calibration of nOe intensity/volume vs distance
3. Detection and removal of inconsistencies by violation analysis
4. Reduction of assignment possibilities

5. Calculations of new restraint sets
6. Structure calculation and floating chirality assignment.

When the structure calculation starts there is an initial analysis of the structure ensemble from the previously calculated structure. This is done by ARIA2 to ensure a reliable estimate excluding outliers. Issues arise in the minimisation algorithm as it cannot converge the structure in every instance. To overcome this ARIA2 determines between converging and non-converging structures by total energy, by selecting a set of structure with the lowest energies and creating ensemble averages based on these.

Calibration in ARIA2 occurs from observed volumes using a *calibration* factor α calculated at the start of each round oof calculation. The *calibration* factor uses the ratio of average experimental $V^{\exp}$ to theoretical volumes V^{th} as an estimate for the constant.

$$\alpha = \frac{\sum_i V_i^{\exp}}{\sum_i V_i^{th}}$$

A different approach is used on the first round of calculations where V^{th} values have not been calculated. In this case if an initial structure ensemble is not provided calibration uses d_{avg}, the average distance of spin pair causing nOes in the initial random ensemble.

$$\alpha_0 = n^{-1} \sum_{i=1}^{n} \frac{V_i^{exp}}{d_{avg}^{-6}}$$

During the calibration calculation cross-peaks with a theoretical volume less than $V_c = d_c^{-6}$ are considered as outliers and excluded from calibration. d_c is a user-defined parameter but is by default set to 6.0 Å (typically the maximum distance at which nOes can be observed). Every spectrum is calibrated separately. Additionally, theoretical volumes can be calculated using either isolated spin-pair approximation or a full relaxation matrix analysis. For the relaxation matrix approach using test systems significantly improved quality indices for Ramachandran coverage, packing quality, and reduction of nOe violations are found.

The Detection of inconsistent restraint after each round of calculation by violation analysis is required to achieve convergence, detecting wrong assignments, remove noise peaks. ARIA2 does this based on the hypothesis of structural consistency. This is done to assess whether the restraints follow general trends imposed on the structure by the entire data set, by comparing distance bounds in one structure with corresponding distances found in the rest of the ensemble. If the restraint doesn't follow this trend and is outside of the bounds by more than the violation tolerance (0.5 as default, it will be rejected. If a restraint is rejected is controlled by the following equation:

$$f_i = S^{-1} \sum_{j=1}^{S} \left[\theta\left(L_i - d_i^j + t\right) + \theta\left(d_i^j - U_i - t\right)\right]$$

Where L_i and U_i denote lower and upper bounds of the i-th restraint, and d_i^j shows distance found in the j-th structure; θ is the Heaviside function. These are used to calculate the fraction f_i of structures violating restraint i. The calibration factor α is manly applied to the set of non-violated restraints.

Cross-peaks in ARIA2 are assigned in an indirect fashion by removing improbable assignments. The r^{-6} averaging used ensures assignments with large and unreasonable distances contributes a small portion of the calculated nOe energy E_{NOE}. ARIA2 calculates how much each assignment contributes using a normalised partial volume,

$$\omega_i = \frac{V_i}{\sum_1^{n_c} V_i}.$$

To reduce the number of contributions for each assignment over the calculation rounds ARIA2 reduces the number of restraints by using only the largest contributions m, which meet the requirement of the function

$$\sum_1^m \omega_i \geq \omega_c.$$

Where ω_c is a user-defined *ambiguity cutoff* which is set to one during the first iteration of the calculation and is typically reduced over rounds of calculation so its is 0.8 by the last round of iteration, so gradually reducing the number of unambiguous assignments.

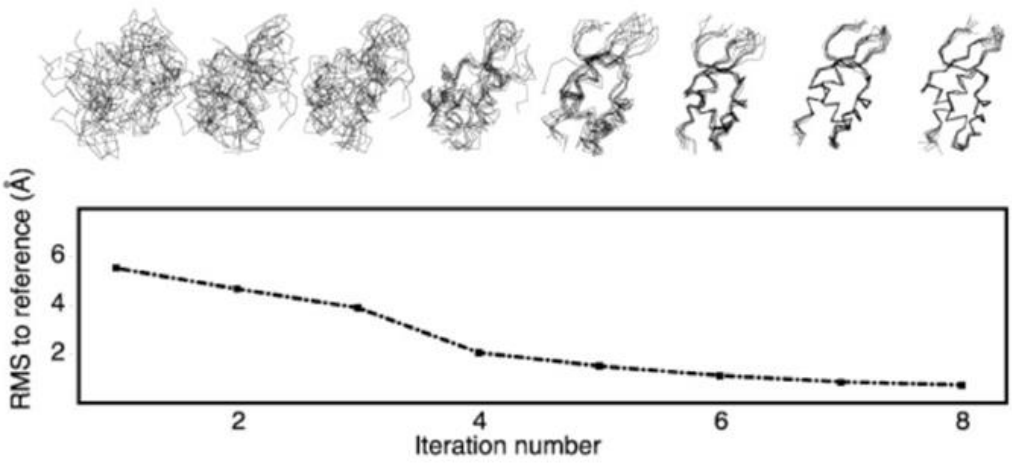


Figure 20 shows an overlay of structures during the steps in an iterative structure calculation and how the RMSD to a known reference structure decreases at each stage[113].

This typically (with reasonable input data) leads to a convergence of the structures over 8 iterations, where the violation restraints are consistently removed leading to a converged set of structures that consistent with each other and the input data. The removed restraints are listed in the output files in the analysis of each round of structure calculation and in the results. Figure 20, shows how a single structure which is randomised at the start of the calculation with a high RMSD, as the structure calculation progresses throughout the iterations, the structure above becomes more converged and defined and the RMSD values start to drop. This shows how the structure of a molecule will become more order the more iterations a calculation goes through (optimum roughly 8, as negligible difference after that).

Floating chirality assignment

ARIA2 uses floating stereospecific chemical shift assignment. For example, in a protein the 2 methylene protons in a geminal pair or the methyl protein of the isopropyl group of valine or leucine can have different shifts depending on their spatial orientation but there is often no direct information to assign which shift belongs with which atom/methyl. ARIA2 deals with this lack of information by testing both alternatives during structure calculation and then selecting the more energetically viable options repeatedly during each iteration. The results of these calculations are being written to a file alongside the current `.pdb` file with the `.float` extension at the end of each iteration.

Refinement in explicit solvent

Refinement in with a more realistic forcefield with solvent is usually the final stage in a calculation. During this stage the structure is refined in a bath of explicit solvent molecules (TIP3-water, or DMSO), which generally improves structure quality.

In this short calculation a full molecular dynamic force fields are turned used, including electrostatic and Lennard-Jones potentials. The parameters used in solvent refinement are consistent with the force field used for the rest of the structure calculation and validation and do not perturb the result away from the results in the previous round or effect the validation results in a negative manner.

Analysis and validation

Analysis of the final structure ensemble (typically the *n* lowest energy structures) involves multiple processes including:

1. Analysis of ordered regions
2. Determination of the lowest energy structure
3. Alignment and overlay of structures using ordered regions.
4. Writing of MOLMOL files for molecular visualisation
5. Analysis and writing of violation lists.
6. Quality checks using Whatif, Procheck, and Prosa II (if installed)
7. Calculation and writing of nOe contact maps.

At the end of the structure calculation round (it1-it8), ARIA2 produces multiple output files for each structure including a coordinate `.pdb` file a `.float` floating assignment file and a *.out* file which contains the output from CNS. Additional files ARIA2 produces include a several report files that support analysis of structures including.

1. `noe_restraints.merged` Is a file that displays all merged restraints
2. *noe_restraints.xml* and *.pickle* contains a list of nOe based distance restraints that are stored in an XML formal and/or the pickle python binary format
3. `noe_restraint.assignment` contains listings of fully, partially, or unassigned peaks

4. `noe_restraints.unambig`, and `noe_restraints.ambig` contain the distance violations for both unambiguous and ambiguous restraints respectively
5. `noe_restraints.violations` contains upper bound violation and contains all the information from the unambiguous and ambiguous restraint files in one place

These report files also contain additional information on the molecule including restrained bond, the average distances in for restraints in the ensemble, cross-peak references and summaries of analyses of the restraints lists and structure ensemble.

At the end of the structure calculation ARIA2 produces output files for each structure throughout each iteration in *.pdb* and *.out* files contain information on the structure calculation and protein database files. ARIA2 in addition produces a series of report files on top of the aforementioned files, which analyse specific to nOe restraints categories and overall summaries. These files deal with the analysis of nOe restraints including unambiguous and ambiguous restraints, violations, assignments, merged restraints, and report files (.xml / .pickle and report summary). The merged XML and pickle files report on all merged restraints, in machine readable formats that can be used by external programs for further analysis.

There are in addition to nOe restraints other types of analysis that can be carried out on the structures produced during the structure calculation. ARIA2 provides restraints files for the MOLMOL graphics package during the final iteration of the structure calculation distance with lower and upper bounds being written into the directory */molmol*. These assist in the identification of erroneous restraint by visualising distances from ARIA2 restraints and assignments. These are present in CNS as a series of scripts that calculate restraints energies ensemble RMSD for the average best structure in each iteration of the structure calculation which is stored in the *analysis* subdirectory (note this doesn't occur for Python binary data).Other methods ARIA2 provides to ensure the structure produced is good enough are the quality checks provided by the , 3rd party programs WHAT IF, PROCHECK, and

PROSA II. These check the final set of structures in the last iteration and the solvent refined ensemble against heuristics (Z scores, phi, psi and chi1 angles etc) from several sets of high-quality structures drawn from the PDB. Each of these will produce a separate report file in *quality_checks.** which are stored in subdirectories for each ensemble.

Additionally, after each stage of calculation, ARIA2 does convergence checks, checking how well the systems restraints are after the molecular dynamics calculation. The number of steps of calculation that need to be carried out to achieve convergence are mostly determined by the system size and the number of restraints. The default number of steps is configured for a system of 150 residues, but for larger molecules or poorer restraint sets, ARIA2 may need a larger number of steps to make sure the structure has converged. The first step to determine a good convergence is checking the total energy of the system which for a protein of about 100 residues gives average energies of 100kcal mol^{-6}, with a variation of 10% being normal.

For structures with larger average energies or large energy variations this suggests that the calculation is not converged, which can be checked by looking at the conformation variance of the ensemble: the RMSD scores. This can be due to poor data quality, number of dynamic steps or the over generous size of chemical shift tolerances during root chemical shift list assignment.

Low average energy, large energy variation: this suggests a broad unpronounced energy minimum with a small number of restraints. This may also occur with an overly tight violation tolerance with a high number of restraint rejections with not enough data being present to cause to convergence. With highly ambiguous, noisy data requiring a more relaxed violation threshold is required in the first iteration of structure calculation.

Setup for a Structure Calculation

The ARIA2 class that controls and runs the molecular dynamics and structure calculation is CNS which is contained in the *cns.py* Before the structure calculation starts this:

1. Creates the project result directory.

2. Copies files from the *cns/protocol* directory to the project directory
3. It starts with *generate.inp*, then *generate_template.inp*, then moving to the refine script *refine.inp* and scripts called by it.

But this will be covered in more detail later in the thesis.

Energy Terms used by ARIA2

While nOe distance restraints are the main restraints used in ARIA2/CNS structure calculation several other restraints may also be used. This Chapter of the thesis will discuss how ARIA2 deals with different terms, will utilise the molecular dynamics engine.

H-bonds

Hydrogen bonds are incorporated by ARIA2 as two distance restraints one between the hydrogen donor and acceptor and the second distance between acceptor and the hydrogen atom in the bond. These restraints are applied early in each round of calculation and left on and are defined in a separate hydrogen bond restraint file such as *data/hbonds/hbonds.tbl* after being added to the project file.

J-coupling

The Karplus equation defines the empirical relationship between a 3-bond J-coupling $^{3}J_{HH}$ and its related dihedral angle. The calculated J-coupling can be directly refined against the observed dihedral angle throughout the calculation using a file such as `data/dihedrals/3J.tbl` after being added to the project file.

Residual dipolar couplings (RDCs)

ARIA2 has two approaches to dealing with RDC data as restraints. The first is the SANI forcefield which directly uses the user specified RDC values. The rhombicity and magnitude of the alignment tensor are used to directly refine bond orientations against an alignment tensor added to the structure calculations as a pseudo molecule OO-XX-YY-ZZ. The second uses the VEAN forcefield which applies angular restraints between pairs of bond vectors (N-H, C-H, etc) as a list of admissible angles. The VEAN angular restraints must be precalculated with a separate program but do not require an alignment tensor and have a smoother and much less degenerate energy surface and so are typically used during the hot phase of a calculation whereas SANI restraints are used during the cool steps.

Disulfide bridges

If the connectivity of the disulfide bridges is known, then the relevant bond is added to the molecular topology of the protein. However, during torsion angle dynamics (TAD), disulfide bridges cannot be treated as fixed distances and are modelled as standard harmonic bonds as the correct functioning of the TAD engine requires there to be no fixed bonds in loops. ARIA2 automatically detects the disulfide bonds from the molecular topology during TAD and then reincorporates them as distance restraints with a slowly increasing weight being applied to the potential throughout the calculation. Covalent connectivity's introduced by patches that are unknown to ARIA2/CNS are often treated as ambiguous distance restraint derived from nOe measurements for the same reason.

Dihedral angle restraints

ARIA2 supports the use of dihedral restraints from programs such as TALOS and CSI which predict the likely values of phi-psi main-chain dihedral angles given a set of chemical shift assignments (typically H, N, Hα, Cα, Cβ, and CO). The predictions are incorporated as dihedral angle restraints using a flat-bottom harmonic potential.

nOes

nOes and hence NOESY experiments serve as the primary source of molecular information for the structure calculation. ARIA2 has two methods of analysing information from nOe cross-peaks, the isolated spin-pair approximation (ISA), or full relaxation matrix back-calculation.

The isolated spin-pair approximation method is particularly well defined for data with short mixing times but fails in dense networks of spins at long mixing times.

It assumes the nOe is caused by a dipolar interaction of 2 groups of spins and is defined by

$$V_{ij} = \alpha\, d_{ij}^{-6}$$

where V_{ij} is the volume of the peak observed and d_{ij} is the distance calculated between the two contributing spins i and j. The scaling coefficient α, depends on the properties of the system being investigated along with the setup of the experiment

used which cannot be measured directly and which hence is estimated during the structure calculation.

$$V_{ij} = \alpha\, d_{ij}^{-6}$$

The nOe dipolar interaction for groups of magnetically equivalent spins such as pairs of methyls is calculated by using a discrete slow jump model suing the equations.

$$V_{ij} = \alpha\, n_I\, n_J\, \hat{d}_{ij}^{-6} \quad \text{where} \quad \hat{d}_{IJ}^{-6} := \frac{1}{n_I\, n_J} \sum_{I \times J} d_{ij}^{-6}$$

i and j to denote two groups of spins $of\ size\ n_I$ and n_j which the nOes are between. The calculation assumes r^{-6} averaging over pairwise contributions, with the effective distance being $\hat{d}_{IJ}$ and α has the same meaning.

The relaxation matrix analysis involves making a complete calculation of the nOes for two spins i and j placed in relation to a further set of spatially close spins t which can enhance transfer of coherence ('spin diffusion'). It allows for the use of longer mixing times and produces results with better correlations between the observed and actual distances within a structure

Distance Restraints

ARIA2 employs its variations of MDSA (Molecular dynamics simulation) protocol to find the global energy minimum of an objective function using molecular dynamics force fields from CNS. Using experimental data form in the form of conformational restraints expressed via energy potentials. Distance restraint from manual assignment can be made a part of the calculation but automated assignments is supported as well. ARIA2 uses a flat-bottom harmonic-wall potential with zero energy between the distance bounds and linear asymptotes for the nOe function and $1/r^{-6}$ sum averaging[104]. This allows for large distance violations that may occur due to the automated assignment procedures.

Ambiguous distance restraints

Distances I nOes are derived from cross-peak volumes to formulate restraints for structure calculation using the relation of from eq (the isolated spin pair approximation).

$$D = (\alpha^{-1} V)^{-\frac{1}{6}}$$

Which changes the volume V into a target distance D using a measured coefficient α. The isolated spin pair approximation models of nOe volumes do not model all the mechanisms which affect relaxation rates leading to inaccuracies. This limitation is solved in ARIA2 by adding upper and lower bounds for the restraint L and U. In the case of ARIA2 L & U are calculated as :

$$L = \max(0, D - \delta),$$

$$U = D + \delta$$

where D is the estimated distance from an isolated spin pair or full relaxation matrix calculation. The polynomial δ is used to calculate the error margin as $\delta = 0.12\ D^2$. This is to restrain the distance bound to an appropriate range rather than a unique value, reducing systematic bias to potentially erroneous restraint distances.

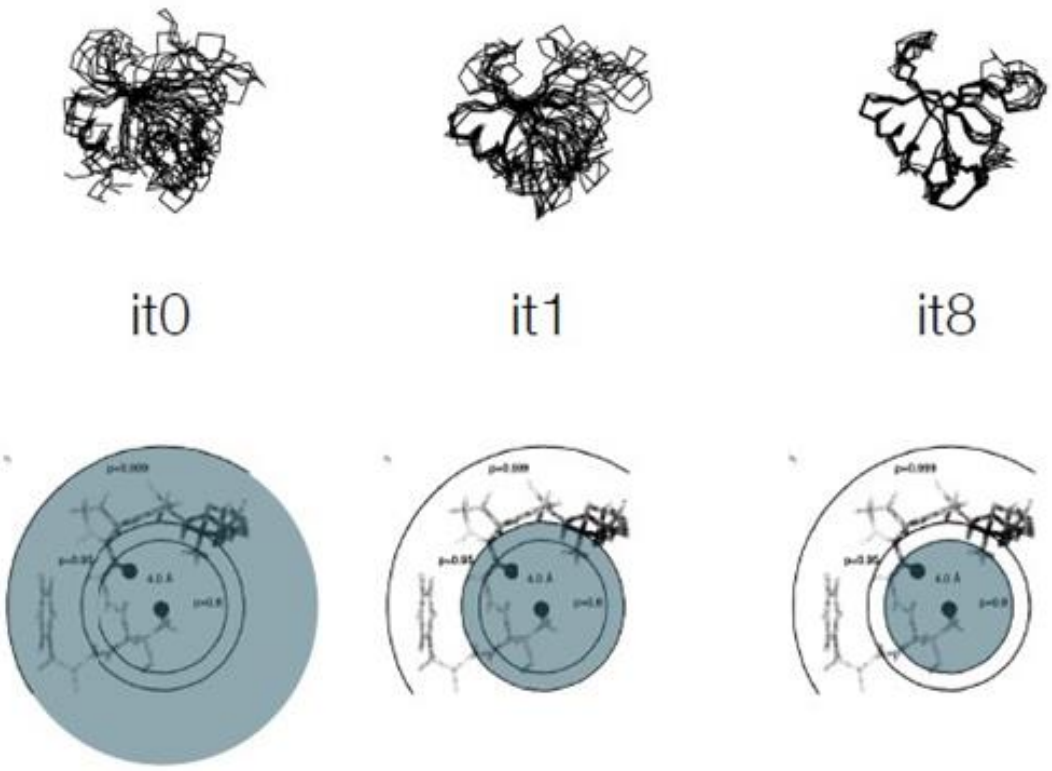


Figure 21 shows how as the calculation proceeds through the iterations, and the structure ensemble improves and converges, against the distance restraints as the forcefield are applied and, reduced therefore previously acceptable restraints are violated the structure improves.

The effect or refining the distance restraints during an ARIA2 calculation distance restraints can be seen in Figure 21; at each iteration of the calculation as bad restraints are removed the convergence of the structure ensemble improves, as well as removing bad distance restraints ARIA2 also reduces the distance it searches for distances restraints, that are satisfied. This is controlled by the value p which is changed throughout the calculation and is reduced in later rounds as seen in Figure 21.

As ARIA2 mainly uses on short distance nOe restraints and is the most numerous of the restraints applied to the structure calculation. Observable ^{1}H-^{1}H distances / nOes are so numerous for smaller proteins (< 25kDa) that structure can be calculated just based off these restraints alone, the more numerous they are the better the quality and higher fidelity the structure is. This can be seen in Figure 22, showing structures of the interleukin 1β molecule which has a length of 152 residues. The structure on the left uses 536 measured ^{1}H-^{1}H distance restraints and has an RMSD of 2.0 Å, and the structure on the right uses 2780 ^{1}H-^{1}H distance restraints and has an RMSD value of 0.4 Å. The Interleukin structure on the left with less restraint and a higher RMSD is less ordered and has much more variation within in structure than the structure on the right with more restraints and a lower RMSD score. This can also be seen in the top of the figure showing the structure of IgG antibody calculated with a gradually increasing number of ^{1}H-^{1}H distance restraints which shows how the quality of the structures produced by ARIA2 increases with the number of restraints.

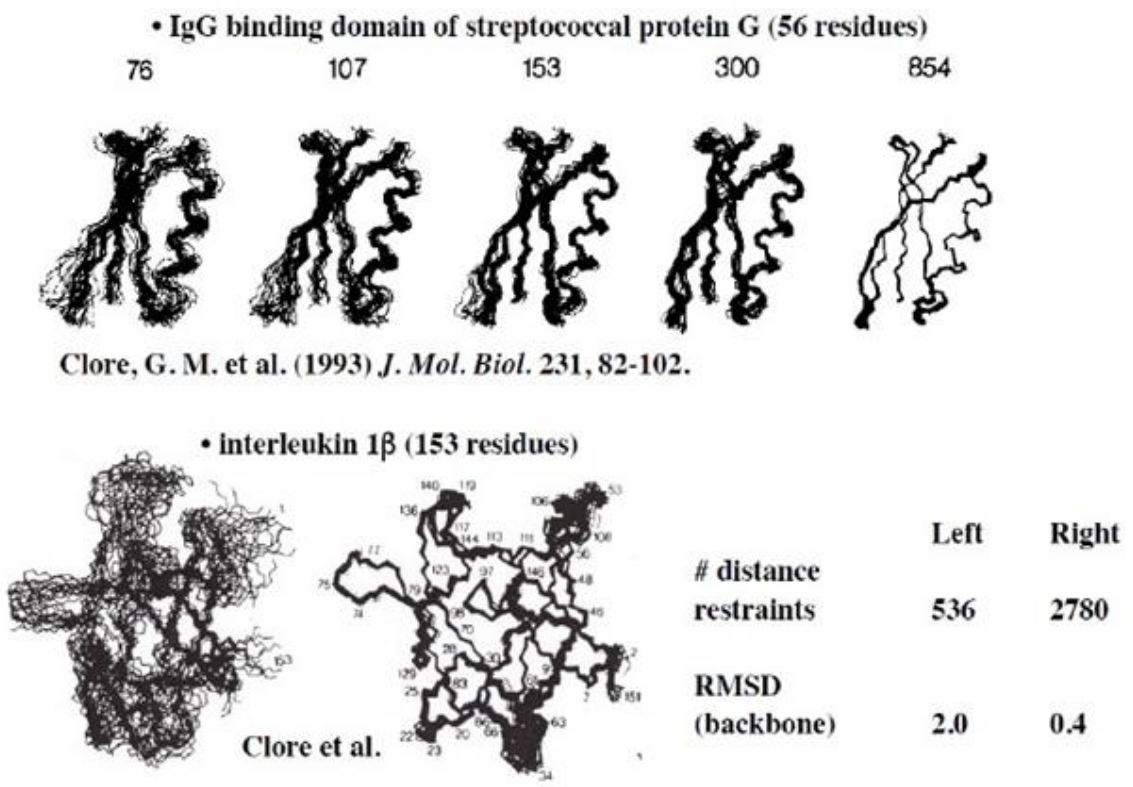


Figure 22 Showing how protein structure ensembles improves with number of ^{1}H-^{1}H restraints for the proteins IgG and Interleukin, showing that as more distance restraints are used a, more defined the structure will be produced , with a lower the RMSD.

Chemical shift-based assignment

ARIA2 uses the chemical shift list from the sequential and sidechain resonance assignments to derive possible assignments, and in fact all possible assignments are made as part of ARIA2's 'root' assignment. The peak position is defined by its frequency coordinates in each dimension of the spectrum. To make up for imprecision in the chemical shift measurements, which could be due to systematic experimental errors, and uncertainty of cross-peaks coordinates, ARIA2 employs chemical shift tolerances ($\boldsymbol{\sigma}_{1/2}$) allow for inaccuracies in peak measurements. Values of $\boldsymbol{\sigma}_{1/2}$ are assigned on a per axis and per spectrum basis on input spectra.

Completeness of assignment also plays are large role in structure convergence and accuracy. Incomplete chemical shift assignments tend to lead to structures with poor convergence (RMSD) and less structural information. The completeness of restraint sets shows the local reliability of each structure. An atom wise completeness is defined by the ratio of the number of observed to expected restraints involving the atom[92]. Indeed, if completeness is zero if there are no restraints at all and the structure would be random with completely uninformative atom positions. If all

structurally possible constraints could be measured completeness would be 1 and a structure with the maximum possible resolution from the available data would be achieved, this is rarely if ever possible. It can be found that completeness values greater than 1 can be calculated and observed. However, this indicates some possible flaw in the data with too many restraints being measured including some with wrong resonance assignments. Low completeness values can occur for several reasons

1. due to missing cross-peaks because of relaxation especially exchange or relaxation due to slow tumbling
2. peaks hidden by artefacts obscuring the data such as Ha peaks under the water line or peaks from buffer components
3. incomplete chemical shift lists
4. rejection of peaks in violation analysis due to miscalibration

Floating Chirality assignment

ARIA2 uses an approach called floating chirality to deal with prochiral centres such as methylene proton pairs and the isopropyl substituents of Valine and Leucine which are difficult to chemical shifts assign stereo-specifically experimentally by NMR. During calculation stereoscopically related substituent groups are tested in both orientation during the structure calculation, and the most energetically favourable is then used for the next step of the molecular dynamics calculation.

Simulated annealing

Typically, two methods are used to find the global minimum energy function of the structural restraints and covalent geometry during structure calculation, energy minimisation and molecular dynamics. Energy minimisation can only walk down an energy surface and cannot get out of local minimum as shown in Figure 23.

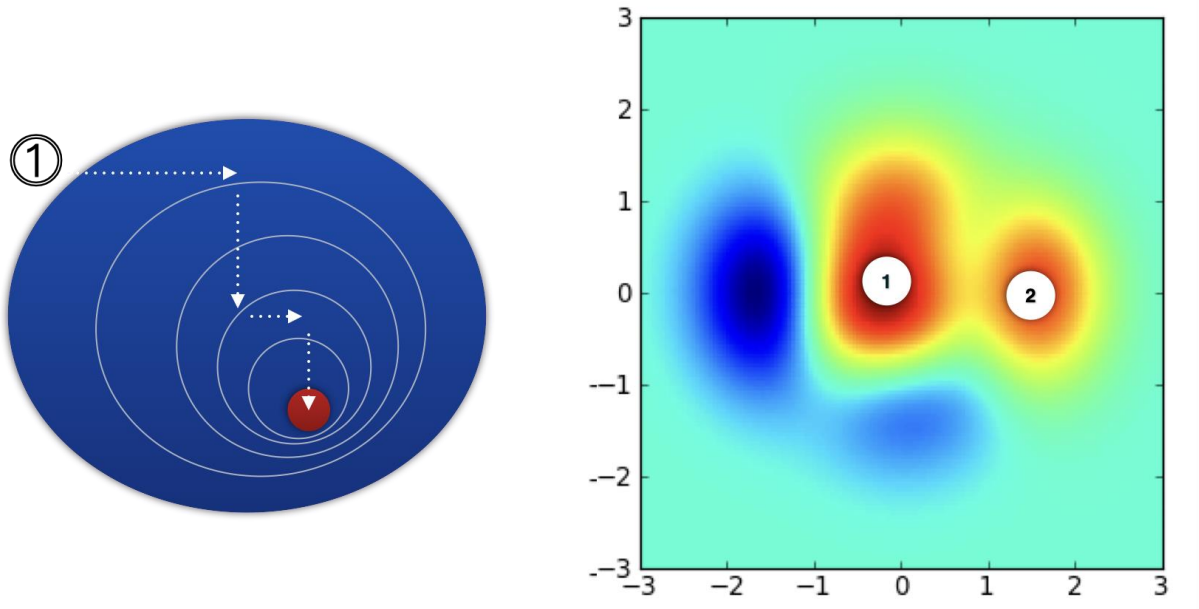


Figure 23, shows two images visually representing the processes of minimisation (left) and simulated annealing (right). On the left the starting point is shown as 1, the energy minimum is at the red dot and minimisation occurs along x and y gradiesntas shown by the arrows .On the right two high energy starting positions are shown by 1 and 2 and the global minimum is at -2 x and 0 y (blue)

In Figure 24 the left panel shows gradient descent from position 1 to the low energy position represented by the red dot as a series of gradient descents. Right panel shows two local minima A & B (red orange) and a global minimum C (blue) on an energy surface. Gradient descent minimisation from ① would lead to the local minimum A and, gradient descent from ② would lead to the point B which is again only a local minimum. This is as opposed to simulated annealing methods which allow for random walks across the energy surface allowing jumps between minima to obtain a final global minimum for energy in some calculations. One such calculation is shown in Figure 24 where the calculation starts outside a local minimum repeated heating of a molecule will give the system enough energy to overcome its energy restraints and sometimes walk into the most energy desirable arrangement. However, as shown sometimes simulated annealing doesn't result in a result in the global minimum ③ and it's therefore essential that multiple structure calculations are carried out to identify a global minimum.

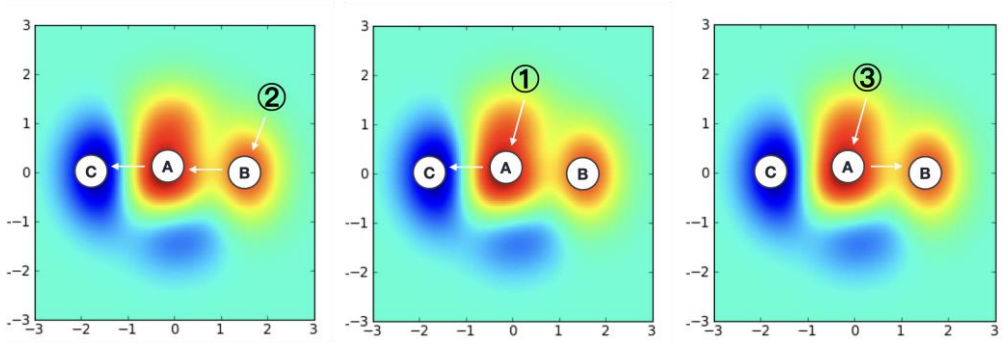


Figure 24, shows the same energy map as figure 22 and several outcomes of simulated annealing start from different postions 1,2 and 3. In each case it is clear how the simulated annealing protocol can move the calulation state between minima often but not always resulting in a movement to (c) the global minimum.

ARIA2 uses the simulated annealing approach for structure calculation using the addition of random velocities to the molecule atoms to simulate heating in a Molecular Dynamics Simulated Annealing (MDSA) protocol using 'frictional' cooling / heating. ARIA2 can use either *cartesian dynamics (CD)or torsion angle dynamics (TAD). TAD has the benefit of reducing the calculation time and allows for the MDSA temperature to be much higher (1000 as opposed to 100s of K) so increasing the speed of convergence. Molecular structures obtained with TAD* D generally provide better local geometries due to the greater spatial sampling. The MDSA protocol in ARIA2 uses 3 phases of molecular dynamics simulation:

1. an initial high-temperature search phase, often with weaker restraints and reduce VdW interactions using TAD
2. a cooling phase where the temperature slowly decreases, during which restraints are slowly added, using a mixture of both TAD and CD
3. a final quench to 0K using molecular minimisation

The length of the cooling stage determines the slope of the bath temperature which provides frictional cooling. In ARIA2 this plays a vital role in convergence in the structure calculation for ambiguous data and extended cooling has been shown to aid convergence[92]. With minimization protocol-based primary on separate scaling of different energy terms that have relatively low force constant with other restraints being available during the structure calculation. Finally, the number of structures in the calculation and the selection of the best structures plays an important role in the calculation and typically only best 30% of structure by energy terms, will be used to

select restraints for the next iteration of calculations and to recalibrate the nOe distances.

The Methods used for Implementing XPLOR-NIH as the molecular dynamic's engine in ARIA2

The following section will discuss,

1. the environment and tools used while converting the ARIACNS code to the XPLOR-NIH implementation ARIAXC.
2. How XPLOR-NIH was implemented as the molecular dynamic's engine for ARIA2's structure calculation.
3. The syntax of the XPLOR language and its relation to Python.
4. The example W*erner* project provided by ARIA2 contains the structure calculation of a Tudor domain.
5. The files that had to be modified to accomplish the conversion.
6. The debugging checks made to compare the code and molecular dynamics results of ARIA2 against ARIAXC.

Workspace

Virtual environment setup

The work was conducted on a windows 10 system, with the window subsystem for Linux 2 WSL2 installed using an Ubuntu based image. The virtual machine allowed for the download, compilation, and execution of ARIA2, as ARIA2 is only supported Unix-based systems. Within WSL2 the following software was installed and used the Linux standard development tools [Fortran, gcc, gnu make] ARIA2, CNS, and XPLOR-NIH. Once downloaded ARIA's *Werner* example protein structure calculation was run to completion with its default configuration to provide a baseline calculation that the port to XPLOR-NIH must complete and to provide a reference of the structures and data layout used by ARIA-CNS. This configuration provided a system where I could manipulate the program's code properly, allowing for the editing, reading, and display of information from ARIA and all the molecular dynamics programs (CNS and XPLOR-NIH) using command line Linux programs while at the same time editing the program using windows a window based IDE (integrated development environment; PyCharm) and editor (sublime text).

Server repository GitHub

Code created during the development of the port was stored in a local git repository for version control. During the development of the program, the repository was regularly synchronised with a copy stored in a private GitHub repository for collaboration and data/source code backup. This allowed small incremental changes to be made which were then pushed to the server. This meant that project could be reverted to the original if required and rolled back to a specific commit if I the changes applied did not give suitable results. This also workflow also provided a suitable disaster recovery plan if required. This approach also allowed for interactive work with my mentor/supervisor including use of remote co-coding using the Code With Me feature of PyCharm.

GitHub is a flexible repository as it can manage the addition of extra files or directories that were relevant throughout the project, and on project completion standard provides be a good place to allow people to download finished project. My final GitHub repositories contained the entirety of the ARIAXC directory tree, and a copy of the original ARIA2 installation, with minor edits for debugging statements (with display statements files start to end of files) for comparison against the original ARIA at the end of the project and the source code of the RollinStone ARIA2 monitoring tool.

PyCharm

Originally, I planned to use command line programs to edit files setting up and running ARIA, CNS, and XPLOR-NIH, but this proved to be a slow and ineffective method for programming on such a large-scale project. This was due to the need to edit and navigate multiple files simultaneously, the need to compare code between older and newer files, and complications of jumping from output of shell commands such as *more*, *less,* and *grep* to the files being edited. The search for a better tool led to the program PyCharm, a multiple-purpose coding tool, which allows me to work with code in its editor but also through its in built Unix terminal. It also comes with multiple tile-able editor windows and directory tree which allows me to easily view and select the files to edit seen in Figure 25. It is also a particularly excellent choice for editing the ARIAXC files as it has an inbuilt intelligent assistance program that understands the syntax of the Python and other mainstream program languages and the scripts been used. For much of the project the languages used were python bash

and csh which are well supported by PyCharm, unfortunately as the XPLOR and CNS scripting languages are niche languages they weren't supported by PyCharm. Other non-code files including JSON, XML, and CSV files were also supported by PyCharm via additional plugins provided by the PyCharm community. These plugins helped visualise and speed development by highlighting the correct syntax for the files and colouring columns and line in otherwise difficult to visualise files, as seen in Figure 26.

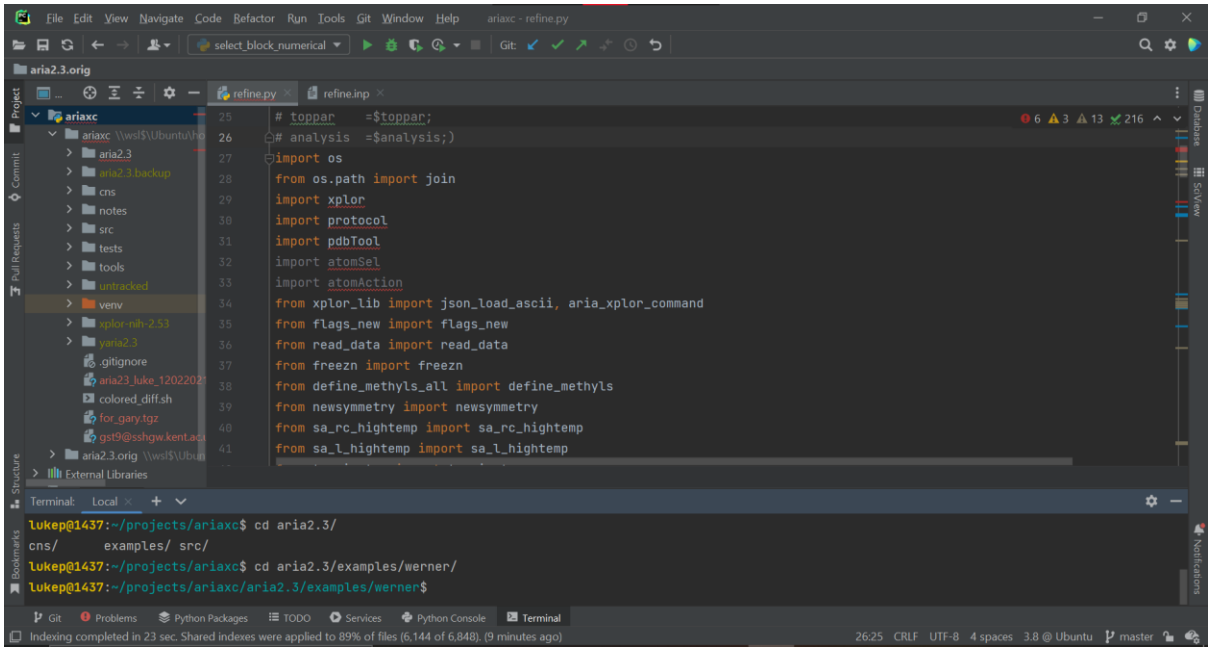
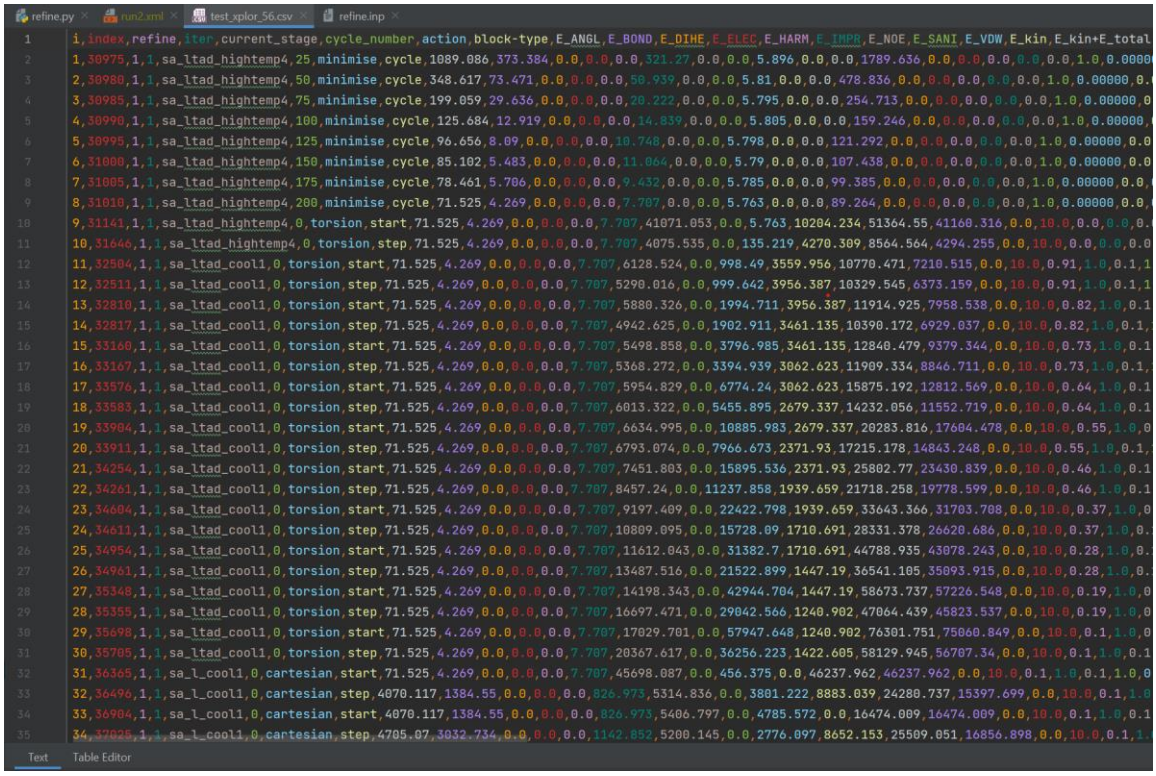


Figure 25 Shows the PyCharm working environment Unix command line, files viewer and editor, and directory tree for selecting and manipulating files.



refine.py | run2.xml | test_xplor_56.csv | refine.inp

```
i,index,refine,iter,current_stage,cycle_number,action,block-type,E_ANGL,E_BOND,E_DIHE,E_ELEC,E_HARM,E_IMPR,E_NOE,E_SANI,E_VDW,E_kin,E_kin+E_total
1,30975,1,1,sa_ltad_hightemp4,25,minimise,cycle,1089.086,373.384,0.0,0.0,0.0,321.27,0.0,0.0,5.896,0.0,0.0,1789.636,0.0,0.0,0.0,0.0,0.0,1.0,0.0000
2,30980,1,1,sa_ltad_hightemp4,50,minimise,cycle,348.617,73.471,0.0,0.0,0.0,50.939,0.0,0.0,5.81,0.0,0.0,478.836,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0.
3,30985,1,1,sa_ltad_hightemp4,75,minimise,cycle,199.059,29.636,0.0,0.0,0.0,20.222,0.0,0.0,5.795,0.0,0.0,254.713,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0
4,30990,1,1,sa_ltad_hightemp4,100,minimise,cycle,125.684,12.919,0.0,0.0,0.0,14.839,0.0,0.0,5.805,0.0,0.0,159.246,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,
5,30995,1,1,sa_ltad_hightemp4,125,minimise,cycle,96.656,8.09,0.0,0.0,0.0,10.748,0.0,0.0,5.798,0.0,0.0,121.292,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0.0
6,31000,1,1,sa_ltad_hightemp4,150,minimise,cycle,85.102,5.483,0.0,0.0,0.0,11.064,0.0,0.0,5.79,0.0,0.0,107.438,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0.0
7,31005,1,1,sa_ltad_hightemp4,175,minimise,cycle,78.461,5.706,0.0,0.0,0.0,9.432,0.0,0.0,5.785,0.0,0.0,99.385,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0.0,
8,31010,1,1,sa_ltad_hightemp4,200,minimise,cycle,71.525,4.269,0.0,0.0,0.0,7.707,0.0,0.0,5.763,0.0,0.0,89.264,0.0,0.0,0.0,0.0,0.0,1.0,0.00000,0.0,
9,31141,1,1,sa_ltad_hightemp4,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,41071.053,0.0,5.763,10204.234,51364.55,41160.316,0.0,10.0,0.0,0.0,0.0,
10,31646,1,1,sa_ltad_hightemp4,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,4075.535,0.0,135.219,4270.309,8564.564,4294.255,0.0,10.0,0.0,0.0,0.0,0.0
11,32504,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,6128.524,0.0,998.49,3559.956,10770.471,7210.515,0.0,10.0,0.91,1.0,0.1,1
12,32511,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,5290.016,0.0,999.642,3956.387,10329.545,6373.159,0.0,10.0,0.91,1.0,0.1,1
13,32810,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,5880.326,0.0,1994.711,3956.387,11914.925,7958.538,0.0,10.0,0.82,1.0,0.1
14,32817,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,4942.625,0.0,1902.911,3461.135,10390.172,6929.037,0.0,10.0,0.82,1.0,0.1,
15,33160,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,5498.858,0.0,3796.985,3461.135,12840.479,9379.344,0.0,10.0,0.73,1.0,0.1
16,33167,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,5368.272,0.0,3394.939,3062.623,11909.334,8846.711,0.0,10.0,0.73,1.0,0.1,
17,33576,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,5954.829,0.0,6774.24,3062.623,15875.192,12812.569,0.0,10.0,0.64,1.0,0.1
18,33583,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,6013.322,0.0,5455.895,2679.337,14232.056,11552.719,0.0,10.0,0.64,1.0,0.1
19,33984,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,6634.995,0.0,10885.983,2679.337,20283.816,17604.478,0.0,10.0,0.55,1.0,0
20,33911,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,6793.074,0.0,7966.673,2371.93,17215.178,14843.248,0.0,10.0,0.55,1.0,0.1,
21,34254,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,7451.803,0.0,15895.536,2371.93,25802.77,23430.839,0.0,10.0,0.46,1.0,0.1
22,34261,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,8457.24,0.0,11237.858,1939.659,21718.258,19778.599,0.0,10.0,0.46,1.0,0.1
23,34604,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,9197.409,0.0,22422.798,1939.659,33643.366,31703.708,0.0,10.0,0.37,1.0,0
24,34611,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,10809.095,0.0,15728.09,1710.691,28331.378,26620.686,0.0,10.0,0.37,1.0,0.
25,34954,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,11612.043,0.0,31382.7,1710.691,44788.935,43078.243,0.0,10.0,0.28,1.0,0.
26,34961,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,13487.516,0.0,21522.899,1447.19,36541.105,35093.915,0.0,10.0,0.28,1.0,0.
27,35348,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,14198.343,0.0,42944.704,1447.19,58673.737,57226.548,0.0,10.0,0.19,1.0,0
28,35355,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,16697.471,0.0,29042.566,1240.902,47064.439,45823.537,0.0,10.0,0.19,1.0,0
29,35698,1,1,sa_ltad_cool1,0,torsion,start,71.525,4.269,0.0,0.0,0.0,7.707,17029.701,0.0,57947.648,1240.902,76301.751,75060.849,0.0,10.0,0.1,1.0,0
30,35705,1,1,sa_ltad_cool1,0,torsion,step,71.525,4.269,0.0,0.0,0.0,7.707,20367.617,0.0,36256.223,1422.605,58129.945,56707.34,0.0,10.0,0.1,1.0,0.1
31,36365,1,1,sa_l_cool1,0,cartesian,start,71.525,4.269,0.0,0.0,0.0,7.707,45698.087,0.0,456.375,0.0,46237.962,46237.962,0.0,10.0,0.1,1.0,0.1,1.0,0
32,36496,1,1,sa_l_cool1,0,cartesian,step,4070.117,1384.55,0.0,0.0,0.0,826.973,5314.836,0.0,3801.222,8883.039,24280.737,15397.699,0.0,10.0,0.1,1.0
33,36904,1,1,sa_l_cool1,0,cartesian,start,4070.117,1384.55,0.0,0.0,0.0,826.973,5406.797,0.0,4785.572,0.0,16474.009,16474.009,0.0,10.0,0.1,1.0,0.1
34,37025,1,1,sa_l_cool1,0,cartesian,step,4705.07,3032.734,0.0,0.0,0.0,1142.852,5200.145,0.0,2776.097,8652.153,25509.051,16856.898,0.0,10.0,0.1,1.
```

Text | Table Editor

Figure 26 shows a snippet of a pycharm plugin that displays CSV files using colour to helping identify which which values go together.

Conversion Methodology

Setting up ARIA2 to use XPLOR as a molecular dynamics engine.

Once ARIA2 was installed and the W*erner* test example was run to completion and validated the original ARIA2 source tree was replicated to a new PyCharm project.

Analysis of ARIA's source code, including running the program with debugging statements, showed that several major components of the structure calculation program required modification. They were as follows:

1. `cns.py`, interfaces between ARIA2s internal structures and CNS. It controls CNS's molecular dynamics calculations by creating templates using CNS' scripting language [`run.cns`], these templates have placeholders which are replaced by values from the ARIA2 projects data storerun.cns [e.g toppar.par_nonbonded]
2. it also initialises and monitors structure calculations via the scripts `refine.csh` and run.cns. Finally, it is responsible for setting up the temporary directory structure used refinement and carrying out calculations either directly or via the ARIA2 job manage using the methods

a. *create_psf_file*
b. *generate_template_pdb*
c. *go*

3. The files `protocols/generate.inp` and `protocols/generate_template.inp` set up the structures of the protein [the protein structure file, or PSF] and the template structure used for initial molecular dynamics before the structure calculation starts.

4. Structure calculation is carried out by a set of modules in the protocols directory that run the ARIA2 structure calculation [`protocols/sa_l_reduced.cns, protocols/sa_ls_cool2.cns etc`]. These are all called by the file `protocols/refine.inp` and have parameters set by values in `run.cns` and are run by `refine`.csh.

Converting the code from CNS scripting language to Python

Once PyCharm and an Ubuntu virtual machine was installed ARIA2 was run with the example data from `examples/werner/read-to-use` using XPLOR_NIH as the molecular dynamic's engine by changing `data.whichMD`. Following this at least four sets of changes needed to be made to the CNS code in the protocols directory to allow it to be used with XPLOR-NIH.

1. Identify a method to read the parameters produced by ARIA2 into the python runtime in XPLOR-NIH
2. The logic which was going to be carried out by python rather than XPLOR needed to be identified and the cns code replaced.
3. Cns scripting language code that needed to be wrapped in python `xplor.command(str)` needed to be identified
4. Differences between the XPLOR and CNS syntax for a number of operations needed to be identified and corrected

This required an understand of how the CNS scripting language was being used to power the molecular dynamics calculations and how the language compared to python. This was achieved by examination of the code and by analysis of the CNS syntax manual and examination of idiomatic python code. Examples of differences. In syntax are shown below

Assigning variables

In CNS values are set using the `evaluate` statement and variables are prepended by a dollar sign in python assignment is carried out using the = sign and variables are plain names as shown in

CNS	Python
`evaluate($test=1)`	`test=1`

Table 3

Table 3 Assigning variables in CNS and python

CNS	Python
`evaluate($test=1)`	`test=1`

Modules in CNS and functions in python

CNS defines functions as a construct called a module which is equivalent to the body of a function in python. Below is an example of defining a function in CNS and python

Table 4

CNS	python

`! in a separate file such a refine.cns`
 	
`module(nstep; SaProtocol)`
 	
`!... body of module`
`evaluate($test=1)`
 	
`! no return statement`
	`# in a file called refine_example.py`
	
	`def refine(nstep, sa_protocol):`
	
	`  # body of function indented 4 characters`
	`  test=1`
	
	`  return test`

Table 4 defining functions in CNS and python

CNS	python

`! in a separate file such a refine.cns`
 	
`module(nstep; SaProtocol)`
 	
`!... body of module`
`evaluate($test=1)`
 	
`! no return statement`
	`# in a file called refine_example.py`
	
	`def refine(nstep, sa_protocol):`
	
	`  # body of function indented 4 characters`
	`  test=1`
	
	`  return test`

Things to note that are different are

1. The module in CNS is defined by the filename so the module in refine.cns is called refine
2. There can be only on module per file in CNS
3. In python there can be multiple function in a file (which is also called a module but is completely different
4. The start of a function is defined by the `def` keyword in python
5. The code inside a CNS module is defined by the whole file after the module statement. Whereas in python the code of the function follows the a : and new line and is consistently indented (typically by 4 spaces)
6. The parmeters passed into a CNS module are defined by the keyword `module` and parameters are listed in brackets after this. Whereas in python the parameters passed to a function in brackets following the keyword `def` and the name function in brackets
7. CNS modules have no return values whereas python has return values

Calling functions in CNS and python is also has vastly different syntax in CNS and python Table 5.

Table 5 calling functions in CNS and python

CNS
`@@<option-envionment-variable>:<filename>(<parameters>)`
python
`from <python-module> import <func-name>` `<func-name>(parameters)`

And examples would be Table 5.

Table 6 A second example of the function call syntax in CNS and python

CNS
evaluate($nStepValue=1) *evaluate($saProtocolValue="test")* *@@RUN:protocols/refine(nStep=$nStepValue;saProtcol=$saProtocolValue;)*
python

`from refine_example import refine`
 	
`n_step_value =1`
`sa_protocol_value = "test"`
`refine(n_step=n_step_value, sa_protocol=sa_protocol_value) # parameter names are often`
`# optional`

Major differences and points to note are

1. A function is CNS is called by importing the file text using the symbol @@
2. In CNS the name of the function or module is defined by its filename. The filename comes in two parts an optional environment variable and a path
3. Function parameters in CNS and python both have brackets and can name parameters (this is optional python and require in CNS)
4. Parameter passed in CNS end with a; denoting the end of the parameter. Parameters passed in in python are separated by commas
5. In python external functions have to be imported before us, typically at the top of the file

If statements

If statement in CNS are mostly the same as in python but requiring a closing `'end if'` statement

Table 7.

CNS	python
`if ($Data.rdc3_firstIt > $iteration) then`	**`if data.rdc3_firstit > iteration:`**
`evaluate ($Data.rdc3_hot = 0.0)`	`data.rdc3_hot = 0.0`

`if ($Data.rdc3_firstIt > $iteration) then`
 	
`evaluate ($Data.rdc3_hot = 0.0)`
 	
`end if`
	`if data.rdc3_firstit > iteration:`
	
	`data.rdc3_hot = 0.0`
	`finished=True`

`evaluate($finished=yes)`	**`finished=True`**

Table 7 Example of an if statement in CNS and python

It should also be noted

1. CNS logical statements in an `'if'` statement have to be surrounded by brackets and followed by an then. Whereas in python conditional statements are closed by a :

2. In CNS the body of the if statement is ended by an `end if` statement. Whereas in python the body of an if statement follows a new line and is consistently indented (typically by 4 spaces) and terminates at outdent of 4 spaces

Conditional loops

Loop statements are carried out with a *while* statement which repeats while a logical statement is true in CNS but generally in python these can be replaced with a *for* statement which iterates over a series of values as seen in

CNS

`evaluate($parcount=1)`
`while ($parcount le 5) loop pars`
`  display $parcount`
`  evaluate ($parcount = $parcount + 1)`
`end loop pars`

python
`for paracount in range(1, 6):` `  print(parcount)`

Table 8. Python does have a while statement, but it is little use and is considered to be dangerous!

Table 8 Example of a conditional loop in CNS and python

CNS

`evaluate($parcount=1)`
`while ($parcount le 5) loop pars`
`  display $parcount`
`  evaluate ($parcount = $parcount + 1)`
`end loop pars`

python
`for paracount in range(1, 6):` `  print(parcount)`

Differences to note include

1. CNS uses while loops and a lot of maths. Python uses a rather elegant iterator syntax
2. CNS defines a while loop between while *<logical-expression>* loop *<label>* and end loop *<label>.* Python defines its loop by a consistently indented block (usually of 4 characters) proceeded by a colon.

3. Logical statement in CNS have to be surrounded by brackets, pythons don't

Comparisons between CNS and Python scripting languages, and CNS and XPLOR internal processes

logical statements in CNS are different to those in python as shown in

Table 9.

Logic	CNS	python
equals	eq / =	==
not equal	ne / #	!=
greater than	gt />	>
less than	lt / <	<
greater than equal	ge / >=	>=
less than equal	le / =	<=
less than equal	le / =	<=

Table 9 Logical expression in CNS and python

The CNS/XPLOR STORE system

To keep track of atom selections CNS and XPLOR define data structures called stores. In each program there are 10 stores called STORE0 to STORE9 which are accessed by so called vector statements. Each store consists of a list of floats each of which are indexed by the id of an atom in the protein structure file and can take any value. Stores are written updated and displayed in CNSscript and XPLORscript using vector statements (in XPLOR they are actually proceeded by the key word vector, but this was dropped in CNS). It should be noted that stores can be used in all places that atom selections are used

Some examples are useful to understand the system are shown in

Setting STORE1 to all 0s
do (store1 = 0) (all)
Setting STORE1 to 1 for all CA atoms
do (store1=1) ((name CA))
using STORE1 as a selection
(STORE1)

Table 10

Setting STORE1 to all 0s
do (store1 = 0) (all)
Setting STORE1 to 1 for all CA atoms

do (store1=1) ((name CA))
using STORE1 as a selection
(STORE1)

Table 10 Using vector STORES in CNS and XPLOR

CNS specific constructs

Some structures are specific to CNS (and by analogy XPLOR). For example, many statements to control the molecular dynamics engine have their own block structure which begins with a name (e.g. noe) and ends with an end statement. Examples are shown below for

1. Scaling the nOe potential.

```
noe scale pack $k_pack_hot end
```

Scale the energies of the nOes in the pack class to *$k_pack_hot.*

2. Setting up parameters on the nOe potentials.

```
noe
  ceiling 100
  potential ambi lognormal
  potential dist lognormal
  potential hbond lognormal
end
```

Set the maximum energy for each nOe to 100 kCal mol^{-1} and set the potential types for class *ambi* and *dist* to the log normal potential. Note each nOe restraint can be added to one class which is used to change parameters for a group of nOes together.

3. Display is used to print output to screen.

```
display Debug set to True
```

Will print the message “Debug set to True” to screen and is somewhat equivalent to pythons print statement and as well as displaying text can display variables. Note display does not have an end statement.

4. Calculate and display the energy of the calculation.

```
energy end
```

Calculates the energy at a particular point.

```
energy print end
```

Prints the energy of the active potentials in the form.

```
-------------------------------------------------------------------------------
| E(kin)+E(total)=16983.384       E(kin)=0.000         temperature=0.000      |
| Etotal =16983.384  grad(E)=21.439     E(BOND)=195.717    E(ANGL)=251.161    |
| E(IMPR)=55.200     E(VDW )=5836.746   E(HARM)=0.000      E(NOE )=10644.559  |
| E(SANI)=0.000                                                               |
-------------------------------------------------------------------------------
```

5. The *aria* manipulates the state of aria specific functions including

    a. Analysing nOes
    b. Flipping labels of prochiral nOes as part of the floating assignment mechanism
    c. Calibrating nOe distance restraints

6. The question mark statement

```
$?
```

Prints information about the current status of the structure calculation including the

symbol table as shown below.

```
Current Symbol Table
 $KBOLTZ=  0.198719E-02 (real)
 $PI=   3.14159     (real)
 $TIMFAC=  0.488882E-01 (real)
 $NAME="lukepacker" (string)
 $SYSTEM="darwin/x86_64" (string)
 …
```

This along with `display` statements ended up being a powerful debugging tool when programming ARIAXC giving information on how ARIA2/CNS set themselves up when they started a run, their program flow, and the order in which the files and modules were run and access run within the structure calculation. This information was key to understanding the organisation of ARIA2 and the best method of debugging the revised code in programming ARIAXC.

Program flow

The flow of the protocol in ARIA2 is as follows.

1. generate.inp - iteration 0 only
2. generate_template.inp - iteration 0 only
3. refine.inp
4. new_flags.inp
5. read_data.inp
6. freezn.inp
7. define_methyls_all.inp
8. sa_l_randomchain.inp
9. newsymmetry.inp
10. cartesian torsion
    torsiontop.inp
11. sa_l_reduced.inp sa_l_reduced.inp
12. sa_rc_hightemp.inp sa_ltad_hightemp4.inp

13. sa_l_hightemp.inp sa_ltad_cool1.inp
14. sa_l_cool1.inp
15. sa_ls_cool2.inp
16. swap.inp
17. print_coorheader_auto.inp or print coorheader.inp

Note the indentation shows the call graph for the refinement. It can clearly be seen that most of the calculation is run by *refine.inp* which to sets up parameters and variables for the other modules/files to conduct specific tasks and controls the logic of the refinement. Additionally, the control flow is different for torsion angle dynamics and cartesian dynamics which are controlled by the switch `data.whichMD` and some steps are only conducted on the first iteration of the calculation.

Analysing convergence

At the end of each calculation after the code was modified PyMol was used to view the resulting PDB structures. These were analysed to see if they were converged by aligning the structural ensemble on the lowest energy structure (energies are provided in the header of each PDB file calculated by ARIA2) and calculating an all atom RMSD. A well converged ensemble typically has a Ca RMSD of ordered regions ~<= 1.0.

The structure of the ARIA2 protocol and sequence of conversion

The section gives a very formal discussion of how ARIA2 sets up its structure calculation, and is discussed in logical order of ARIA2 protocols and follows the logical order of conversion into ARIAXC.

Generate Files

The *generate.inp* [*generate.py* in ARIAXC] script is the start of the ARIA2 calculation which processes and handles:

1. Creating the topology of the molecule for the structure calculations from the sequence.
2. Applying patches for non-canonical molecular structure [disulphides, histadines, etc]
3. Writes a protein structure file (psf)
4. Creates and saves methyl tables in a file called `RUN:begin/methyls.tbl.` by:
    a. Sets the display to the methyl table file (`RUN:begin/methyls.tbl`)
    b. Running `define_methyl_ini` setting them in the toplogy
    c. Running the `define_methyls_all`
5. Setting up swapping of other prochairal groups and aromatics by .
    a. Setting the display to the `setup_swap_list.tbl`
    b. Running the `setup_swap_init` script

When developing generate.py this file I wrote it in python using the XPLOR-NIH command line utility `seq2psf` function, that takes sequence files and turn it to a *psf* file. However, this uses different forcefield (topology and parameter files). This made the *psf* incompatible with forcefields used by ARIA2. Therefore, instead the original CNS code was translated and called through `xplor.command`.

The generation of `methyls.tbl` was however, translated to use another mechanism with `methyls.tbl` which contains a series of calls to define mehtyls via the atom indices being replaced by a JSON `methyls.json` containing a list of methyl atom indices instead.

```
Methyls.tbl

do (store1 = 0) (all)
do (store1 = 1) (id 9)
do (store1 = 1) (id 15)
do (store1 = 1) (id 36)
…
```

`generate_template.inp` is the second of the generate files which are run to initialise ARIA2. It carries out the following tasks:

1. Calls run.cns as a module to load parameters (spectrum, filenames, spectra, data, iterations, saprotocol, refine, relax, toppar, analysis)
2. Calls module `flag_new.cns` to set default energy flag values
3. Reads the <molecule-name>.psf *psf* from inside *the aria project directory cns/begin directory*
4. Randomizes the y/z coordinates of the atoms in the molecules molecule along a line
5. Setup energy term and dynamics potential terms and regularize the structure via molecular dynamics to give the template good covalent geometry
6. Write the template structure out to a pdb file

In ARIA-XC implementation the logicical statements in the file was replaced by logic written in python with direct calls to XPLOR script being reserved for calls to set forcefield parameters and calls to the CNS minimisation routines '`dynamics verlet`' and '`minimize powell'` via the `xplor.command` function.

cns.py

The file cns.py is the component of ARIA2 that controls communication between ARIA2 and molecular dynamics engine used CNS. The actual communication is controlled by the class CNS which is a subclass of StructureEngine (a general interface to structure calculation engines).

refine

The file `refine.inp` is the file that drives a structure calculation. However, apart from loading the current structure, topology, parameters, and coordinates and writing the final coordinates, it is mostly concerned with housekeeping (setting flags etc) for the structure calculation with molecular dynamics statement carried out by modules/functions it calls. It also carries out the final energy minimisation after simulated annealing before saving the PDB file for the current structure.

flags_new

The file/module `flags_new`, determines the state a series of flags via *if* statements at the start of the calculation. The file can also enable and disable components of the calculations energy function and potential based on flags in the project.

read_data

The *read_data* module load restraint data used during the calculation of the molecules structure. These include the restraints lists for hydrogen bonds, nOes including ambiguous and unambiguous nOe data, hydrogen bonds, dihedrials, J-couplings, residual dipolar coupling, bond length and angles, and RDCs. Depending on the data read potential terms are enable and parameters set.

freezn

The `freezn` file is required for refinement of proteins bound to Zn and as such was not a target of the port to ARIAXC.

define_methyl_all

The script `define_methyls_all`, functions to store all methyls and other equivalent groups into store 1 for later use during floating chirality assignment. As well as the methyls this includes the HE2/HE3 and HD2/HD3 atoms of phenyl alanine's and tyrosine.

newsymmetry

This module sets up calculation for symmetric dimers and was not used by ARIAXC (outside the scope of the port).

sa_l_randomchain

The `sa_l_randomchain`, randomises a molecular structure by random torsion rotations around a proteins N-Cα and Cα-CO bonds so randomising the structure before calculation. The process is controlled by the value of the CNS/XPLOR random number seed $seed.

newsymmetry

This file sets calculation of symmetric oligomers, setting distance symmetry restraints, packing restraints, and non-crystallographic symmetry restraints. It was not used in ARIA-XC as these features are not currently supported.

sa_l_reduced

This file is only used if cartesian dynamics is enabled using a flag. During initial cartesian calculations, a reduced number of atoms are used to reduce the computational burden. This reduction is carried out by increasing the size of certain side chain atoms for each protein residue type and ignoring the rest. The ids of the active atoms are stored into STORE1 by this module.

sa_rc_hightemp

This module runs the high-temperature phase for the Cartesian molecular dynamics simulated annealing for symmetric multimers and was not used.

torsiontop

This file sets up CNS for torsion angle dynamics calculations including the breaking of cyclic sets of bonds of required.

Sa_l_hightemp / Sa_ltad_hightemp4

These modules runs the high-temperature phase for the Cartesian /Torsion molecular dynamics (decided by whether `Data.whichMD` equals cartesian) simulated annealing for single chains. The start by setting the mass of all atoms to 100 to improve computational efficiency by reducing high frequency vibrations so allowing an increase in the length of the molecular dynamics time step. An nOe statement is called to set parameters for high temperature dynamics calculations using parameters from run.cns. Initial force constants and weights for other potential terms are also set at this point (noe, bond, angle harmonic vdw etc). The structure is then minimised under the reduced potential defined by an igroup restraints interaction call which only uses the atoms defined in STORE1 but maintains all angle, dihedral and improper restraints. This is followed by a number of rounds of high temperature dynamics (`length nstep, timestep SaProtocol.timestep, temperature SaPRotocol.carinit_t/ SaPRotocol.tadinit_t` respectively for cartesian and tosrions dynamcis) during which VdW and reduced atom interactions are ramped up. Swapping of floating chirality assigned atoms is carried out during this phase by calls to swap. Checks are made on the real temperature during this phase and the timestep is reduced on each cycle if it's is too far away from the bath temperature to avoid the calculation exploding.

It should be noted some bond weights and parameters are different between cartesian and torsion angle dynamics as calculation is carried out at a much higher temperature in torsion angle.

sa_ltad_cool1 / Sa_l_cool1

These carry out an initial ramped cooling either using torsion dynamics or (`sa_ltad_cool1`) or cartesian dynamics (`sa_l_cool1`) and take parameter. During the cooling stages the weights for a number of restraints are ramped up (SANI RDCs, VEAN RDCs, unambiguous nOes ambiguous nOes, dihedrals etc are ramped up). Floating chirality swapping is again carried out using calls to the swap module. All atom interactions are on. Temperatures run from `SaProtocol.tadinit_t` to `SaProtocol.final2_t` [torsion dynamics] and `SaProtocol.carinit_t - &SaProtocol.final2_t` [cartesian dynamics]

sa_ls_cool2

This carries out the final cooling stage from `Saprotocol.final1_t` to `SaProtocol.final2_t` (note the apparent error on the current version of ARIA2). Dynamics are carried out in cartesian space in all cases, presumably as this gives a better ensemble of structures. All restraint eights are now 1.

swap

This module carries out floating assignment of stereo specifically assigned groups. The module swap.cns systematically and randomly swaps the prochiral protons in selected methyl, aromatic and methyl groups. After each swap it recalculates the nOe energy and selects the state with the lowest emery and retains it using the aria flip command. Flips are controlled by the value of q which is defined for each group of atoms and stored in STORE9.

The value of q on each round is set as

```
! reduce q randomly

if ($swap > 0) then
   vector do (q = q * ran()) (attr q > -0.5)
end if
```

which reduces it through the calculation so annealing the assignments.

logn_auto_weight

This file sets up the log normal nOe function and was not used isn ARIA-XC.

print_coorheader

This module adds the REMARKS header at the top of each calculated file which includes total and all forcefield component energies. A list of rms deviations for all force field terms and a summary of the number of restraints violated.

It has the form

```
REMARK FILENAME=hrdc_15.pdbc/a/e/w/run5/s/it8/
REMARK ==============================================================
REMARK            overall,bonds,angles,improper,dihe,vdw,elec,noe,cdih,coup,sani,vean
REMARK energies: 1168.84, 32.7061, 145.405, 20.2435, 581.713, 176.419, 0, 212.353, 0, 0, 0, 0
REMARK ==============================================================
REMARK            bonds,angles,impropers,dihe,noe,cdih,coup,sani,vean
REMARK rms-dev.: 4.618948E-03,0.586644,0.408112,43.248,5.077534E-02,0,0, 0, 0
REMARK ==============================================================
REMARK               noe,cdih,coup,sani,vean
REMARK               >0.5,>5,>1,>0,>5
REMARK violations.: 1, 0, 0, 0, 0
REMARK ==============================================================
REMARK DATE:2021-10-12  03:24:37.900471       created by user: garythompson
REMARK VERSION:2.51
ATOM      1  HA  MET     1      25.476  12.197   5.530  1.00  0.00
…
```

print_coorheader_auto.cns

This module prints the remarks header for calculations using the log harmonic nOe potential and was not used by ARIAXC.

refine_water refine_dmso generate_water generate_dmso

These modules carry out refinement in a box of water or dmso after it8 and were not used by ARIA-XC which limited its port to achieving a converged structure in it8.

Results

Results 1 Translation of ARIA2 CNS Molecular Dynamics Calculations to Python

This chapter describes how the script interface between ARIA2, and CNS was converted to use python and calls to the XPLORscript scripting language via XPLOR-NIH's `xplor.command()` interface. As a preface to this an overview of the

use of the XPLOR language is given following which some basic infrastructure and utility functions are described (module xplor_lib). The chapter then follows the logical order of how interface functions and scripts are called during one cycle of an ARIA2 calculation.

The original approach to modifying the code to run with python was to translate the code until a complete calculation step ran to completion. At this point little attention was paid to whether the calculation converged, though analysis showed that it did not.

Overview

Modification of the code to python was generally carried out in the order in which the protocol files are called during a calculation. This helped to keep the original scripting code in CNS as similar and close as possible to the new Python XPLOR-NIH GPSL. In

Table 11is a side-by-side example showing how python structures were kept close to

ARIA2/CNS	ARIAXC/Python

`evaluate ($iteration = $Data.iteration)`
`evaluate ($count = $Data.count)`
 	
`if ($Data.rdc1_firstIt > $iteration) then`
`  evaluate ($Data.rdc1_hot = 0.0)`
`  evaluate ($Data.rdc1_cool1 = 0.0)`
`  evaluate ($Data.rdc1_cool2 = 0.0)`
`end if`
 	
`if ($Data.rdc2_firstIt > $iteration) then`
 	
`  evaluate ($Data.rdc2_hot = 0.0)`
`  evaluate ($Data.rdc2_cool1 = 0.0)`
`  evaluate ($Data.rdc2_cool2 = 0.0)`
`end if`
	`iteration = json_data['data']['iteration']`
	`count = json_data['data']['count']`
	
	`if json_data['data']['rdc1_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc1_hot'] = 0.0`
	`  json_data['data']['rdc1_cool1'] = 0.0`
	`  json_data['data']['rdc1_cool2'] = 0.0`
	
	`if json_data['data']['rdc2_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc2_hot'] = 0.0`
	`  json_data['data']['rdc2_cool1'] = 0.0`
	`  json_data['data']['rdc2_cool2'] = 0.0`

those in CNSscript in part of the flow control in refine.py.

ARIA2/CNS	ARIAXC/Python

`evaluate ($iteration = $Data.iteration)`
`evaluate ($count = $Data.count)`
 	
`if ($Data.rdc1_firstIt > $iteration) then`
`  evaluate ($Data.rdc1_hot = 0.0)`
`  evaluate ($Data.rdc1_cool1 = 0.0)`
`  evaluate ($Data.rdc1_cool2 = 0.0)`
`end if`
 	
`if ($Data.rdc2_firstIt > $iteration) then`
 	
`  evaluate ($Data.rdc2_hot = 0.0)`
`  evaluate ($Data.rdc2_cool1 = 0.0)`
`  evaluate ($Data.rdc2_cool2 = 0.0)`
`end if`
	`iteration = json_data['data']['iteration']`
	`count = json_data['data']['count']`
	
	`if json_data['data']['rdc1_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc1_hot'] = 0.0`
	`  json_data['data']['rdc1_cool1'] = 0.0`
	`  json_data['data']['rdc1_cool2'] = 0.0`
	
	`if json_data['data']['rdc2_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc2_hot'] = 0.0`
	`  json_data['data']['rdc2_cool1'] = 0.0`
	`  json_data['data']['rdc2_cool2'] = 0.0`

Table 11. showing a comparison of CNScript (left) and the XPLOR python (right) scripting interface, for the same code from the start of the refine.cns file in ARIA and refine.py file in ARIAXC.

As seen in

Table 11. CNS scripting syntax is quite similar to that of python.

ARIA2/CNS	ARIAXC/Python

`evaluate ($iteration = $Data.iteration)`
`evaluate ($count = $Data.count)`
 	
`if ($Data.rdc1_firstIt > $iteration) then`
`  evaluate ($Data.rdc1_hot = 0.0)`
`  evaluate ($Data.rdc1_cool1 = 0.0)`
`  evaluate ($Data.rdc1_cool2 = 0.0)`
`end if`
 	
`if ($Data.rdc2_firstIt > $iteration) then`
 	
`  evaluate ($Data.rdc2_hot = 0.0)`
`  evaluate ($Data.rdc2_cool1 = 0.0)`
`  evaluate ($Data.rdc2_cool2 = 0.0)`
`end if`
	`iteration = json_data['data']['iteration']`
	`count = json_data['data']['count']`
	
	`if json_data['data']['rdc1_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc1_hot'] = 0.0`
	`  json_data['data']['rdc1_cool1'] = 0.0`
	`  json_data['data']['rdc1_cool2'] = 0.0`
	
	`if json_data['data']['rdc2_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc2_hot'] = 0.0`
	`  json_data['data']['rdc2_cool1'] = 0.0`
	`  json_data['data']['rdc2_cool2'] = 0.0`

Note the CNS scripting language can use hierarchical labels of the form

<name1>.<name2>.<name-3> to reference variables. For example in

ARIA2/CNS	ARIAXC/Python

`evaluate ($iteration = $Data.iteration)`
`evaluate ($count = $Data.count)`
 	
`if ($Data.rdc1_firstIt > $iteration) then`
`  evaluate ($Data.rdc1_hot = 0.0)`
`  evaluate ($Data.rdc1_cool1 = 0.0)`
`  evaluate ($Data.rdc1_cool2 = 0.0)`
`end if`
 	
`if ($Data.rdc2_firstIt > $iteration) then`
 	
`  evaluate ($Data.rdc2_hot = 0.0)`
`  evaluate ($Data.rdc2_cool1 = 0.0)`
`  evaluate ($Data.rdc2_cool2 = 0.0)`
`end if`
	`iteration = json_data['data']['iteration']`
	`count = json_data['data']['count']`
	
	`if json_data['data']['rdc1_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc1_hot'] = 0.0`
	`  json_data['data']['rdc1_cool1'] = 0.0`
	`  json_data['data']['rdc1_cool2'] = 0.0`
	
	`if json_data['data']['rdc2_firstIt'] >`
	`json_data['data']['iteration']:`
	`  json_data['data']['rdc2_hot'] = 0.0`
	`  json_data['data']['rdc2_cool1'] = 0.0`
	`  json_data['data']['rdc2_cool2'] = 0.0`

Table 11 *$Data.rdc2_cool1* has the value 2.0 and name- 1 is Data and name-2 is *rdc2_cool1*. The most notable change in pythons is that data values are stored inside a data dictionary [XplorDict; discussed in results] in ARIAXC / python which replaced generated code and hierarchical variable names and was used to communicate variables from ARIA2.

To guarantee initially that ARIAXC worked exactly the same as ARIA and get the code to work as fast as possible I implemented these files with the XPLOR scripting language for MD and with python for all other scripting purposes. As stated, before CNS and XPLOR scripting languages are very similar syntax-wise and would reduce any unknown variation between ARIAXC and ARIA as at this point did not have the facilities to view the MD completely. XPLOR scripts appear as either a string with commands or as entire modules/files and can be seen as still denoted with the *.cns* or *.inp* inside of the *cns/protocols* directory. An example of this can be seen in with the module *swap.cnsrefine.cns/.py* and in

Table 11, showing how closely CNS and XPLOR scripting languages are to one

ARIA2/CNS	ARIAXC/Python
`evaluate ($iteration = $Data.iteration)` `evaluate ($count = $Data.count)`	`iteration = json_data['data']['iteration']` `count = json_data['data']['count']`

`if ($Data.rdc1_firstIt > $iteration) then`
`  evaluate ($Data.rdc1_hot = 0.0)`
`  evaluate ($Data.rdc1_cool1 = 0.0)`
`  evaluate ($Data.rdc1_cool2 = 0.0)`
`end if`
	`if json_data['data']['rdc1_firstIt'] > json_data['data']['iteration']:`
	`  json_data['data']['rdc1_hot'] = 0.0`
	`  json_data['data']['rdc1_cool1'] = 0.0`
	`  json_data['data']['rdc1_cool2'] = 0.0`

`if ($Data.rdc2_firstIt > $iteration) then`
`  evaluate ($Data.rdc2_hot = 0.0)`
`  evaluate ($Data.rdc2_cool1 = 0.0)`
`  evaluate ($Data.rdc2_cool2 = 0.0)`
`end if`
	`if json_data['data']['rdc2_firstIt'] > json_data['data']['iteration']:`
	`  json_data['data']['rdc2_hot'] = 0.0`
	`  json_data['data']['rdc2_cool1'] = 0.0`
	`  json_data['data']['rdc2_cool2'] = 0.0`

another but also how similar the MD commands are.

Calls to CNS using xplor.command and aria_xplor_command

Much of the code translation was orientated towards getting the transfer of parameters between ARIA2 and XPLOR-NIH correct and assuring that the logical control flow of the program was controlled using Python. This meant that much of the calculation depended on using calls to *xplor.command* () a Python function provided by XPLOR-NIH to run a Python string as an Xplor command. An example of this is shown in

Table 12(in this case *xplor.command* () is wrapped in an additional wrapper function

CNS scripting language	XPLOR-NIH Python

`evaluate ($k_pack_cool1 = $Data.packing_cool1)`
`noe scale pack $k_pack_cool1 end`
	`aria_xplor_command(`
	`'''`
	`  evaluate($k_pack_cool1=${k_pack_cool1}f)`
	`  noe scale pack $k_pack_cool1 end`
	`''', {'k_pack_cool1' : k_pack_cool1}`
	`)`

aria_xplor_command which adds debugging output and allows a dictionary of variable names and values to be used)

Table 12. Comparison table showing 2 examples of code from refine.cns/.py with CNS on the left and xplor/python on the right showing use of aria_xplor_command.

CNS scripting language	XPLOR-NIH Python

`evaluate ($k_pack_cool1 = $Data.packing_cool1)`
`noe scale pack $k_pack_cool1 end`
	`aria_xplor_command(`
	`'''`
	`  evaluate($k_pack_cool1=${k_pack_cool1}f)`
	`  noe scale pack $k_pack_cool1 end`
	`''', {'k_pack_cool1' : k_pack_cool1}`
	`)`

It should be noted that it was possible to use the version shown in

CNS scripting language	XPLOR-NIH Python

`evaluate ($k_pack_cool1 = $Data.packing_cool1)`
`noe scale pack $k_pack_cool1 end`
	`aria_xplor_command(`
	`'''`
	`  evaluate($k_pack_cool1=${k_pack_cool1}f)`
	`  noe scale pack $k_pack_cool1 end`
	`''', {'k_pack_cool1' : k_pack_cool1}`
	`)`

Table 12 but this doesn't set the CNS variable *$k_pack_cool1*, which could be used at a later point in the program (note that unlike python the namespace in CNS is flat so all routines can see all variables)

Table 13 Another version of the code in table 12, which ignores variable aliasing and could lead to erroneous results

CNS scripting language	XPLOR scripting language

`evaluate ($k_pack_cool1 = $Data.packing_cool1)`
 	
`noe scale pack $k_pack_cool1 end`
	`aria_xplor_command(`
	`'''`
	`  noe scale pack ${k_pack_cool1}f end`
	`''', {'k_pack_cool1' : k_pack_cool1})`

It should be noted that variables are passed into aria_xplor_command as a dictionary of named values which are converted into strings using the python string format routine. These variable string conversions require codes to identify the type of the variable as shown in Table 13.

Table 14 types of python format options used in the code

Variable type	Code x in {variable}x
String	s
Float	f
Integer	i
Boolean	b

The types of python format options used in the code are shown in Table 14

Debugging

It was not possible to execute CNS and XPLOR-NIH under a debugger, so wide use was made of print and echo statements for debugging. Additional tools such as *aria_xplor_command()* and `rollinstone` command were implemented to improve the quality of the debugging information and improve the speed of the code turnaround when dealing with Rscript and CNSscript code .

The Werner test system

All work on the structure calculation files and modules was tested using the Werner structure calculation dataset provided by ARIA2 in `examples/werner`. This calculation was carried out with a stock implementation ARIA2/CNS with no changes made to the project file and data from the `ready-to-use` directory. This is a tudor fold protein with well know high quality nOe restraints which has been deposited in the PDB and as such was a good model to develop ARIAXC with.

Isolating a single calculation from ARIA2 for analysis and debugging

As a single ARIA2 structure calculation can take several hours and requires 8 iterations of calculation typically across at least 10-20 structures a more streamlined approach was required to debugging as waiting for a full structure calculation would take too long. The best method for debugging while programming the majority of the structure calculation starting from *refine*, would be to get almost immediate feedback, to tell whether there was a syntax error or other errors within the code. To achieve this, we used the files in the ARIA2 *tmp/* folder in which structures are stored during each round before completion, to conduct single rounds of calculation. Specifically, we only calculated it0 of 8 with 20 structures and analysed run_cns_20 (structures are calculated in revere numerical order so run_cns_20 is the first calculation completed. To understand how this is achieved, we must examine the directory structure used by ARIA2 during calculation and identify the directory from which a single calculation is executed in the case of one of the W*erner* example runs this was in the directory

`werner/temp/aria_temp.tmp4WXlMO1632952670`

(4WXlMO1632952670 is a random number generated by ARIA2 each time it starts a calculation) which contains one directory for each of the 20 calculations named run_cns_20, run_cns_19 etc. Each of these then contains a csh script *refine.csh.* Each *`refine.csh`* is a csh script that conducts a single calculation and maybe modified and executed independently. Each one locally manages where the information used in the calculation will be stored for the structure calculation. The contents of refine.csh are shown in Figure 27. The information in the file includes its location, its local project path, the path to its XPLOR-NIH executable, were the individual run.cns parameters file will be stored, XPLOR-NIH/ARIA2 working

directory, and finally command line to call the XPLOR-NIH executable pyXplor to run the structure calculation writing which is written to out to *refine.out*.

```
#! /bin/csh
# SGE/PBS/SLURM/LFS facility
#$ -N run_cns_1
#$ -S /bin/csh
#PBS -N run_cns_1
#PBS -S /bin/csh
#BSUB -J run_cns_1
#BSUB -L /bin/csh
#SBATCH -J run_cns_1
#SBATCH --ignore-pbs

## results will be stored here
setenv NEWIT /Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/run2/structures/it8

## project path
setenv RUN /Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/run2/cns.

##Python path
setenv PYTHONPATH /Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/run2/cns/protocols

## individual run.cns is stored here
setenv RUN_CNS
/Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/temp/aria_temp.tmp4WX1MO1632952670/run_cns_1.

## CNS working directory
cd
/Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/temp/aria_temp.tmp4WX1MO1632952670/run_cns_1

## solves some NFS sync problems
cat /Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/run2/cns/protocols/refine.py > /dev/null

## command line
/Users/garythompson/programs/xplor-nih/2.51/bin/pyXplor <
/Users/garythompson/Dropbox/git/ariaxc/ariaxc/aria2.3/examples/werner/run2/cns/protocols/refine.py >! refine.out
touch done
```

Figure 27 Contents of refine.py with instructions for ARIA2 with locations of files and run time operations for the program to run in shell script. This shows path to Python, stores for the structure calculation program, location of Molecular Dynamics engine (CNS working directory) and run time temporary files.

Note a number of interesting features can be seen.

1. calculations are setup to run on a cluster using PBS
2. there are some work arounds to avoid problems with NFS (copying `refine.py` to `/dev/null`)
3. the file done is created to signify a completed calculation
4. the result of the calculation are stored in *`refine.out`* in the temporary directory

This single calculation can be run independently from ARIA2 in the shell by typing the command.

`csh refine.csh`

which allows faster turn around during debugging

To make the process of debugging even faster during debugging in the template for *`refine.csh`* *(in `aria/src/cns.py`) t*he last line of the file was changed to

```
…/pyXplor <  …/cns/protocols/refine.py
```

so output was directed to the terminal rather than `refine.out`

Detailing Xplor_lib module, a function library

XPLOR contains a number of utility and library functions written during the project required to interface python to XPLORscript and to ease the code conversion and debugging, they are detailed below.

decorator trace_module

The first function inside of `xplor_lib` is a decorator function `trace_module`. This wraps a python function, and prints `START <function-name>` when it is called and *END* <function-name> when it returns. Its source code is:

```
def trace_module(function_call):
    def wrapTheFunction(*args, **kwargs):
        print('START %s'%function_call.__name__)
        result = function_call(*args, **kwargs)
        print('END %s'%function_call.__name__)
        return result
    return wrapTheFunction
```

This wrapper's function was designed for debugging XPLOR modules defined in Python and was used by the RollinStore debugging tool. An example of its use from sa_l_cool1.py is show below.

```
@trace_module
def sa_l_cool1(saprot, data, toppar, debug):

    if debug:
        xplor.command('set echo on message on end')

    ncycle = ((saprot['carinit_t']- saprot['final1_t'])/float(saprot['tempstep']))
    nstep = int(saprot['cool1_steps']/float(ncycle))

    …
```

A call to sa_l_cool1 would print:

```
START sa_l_cool1
  … output from sa_l_cool1
END sa_l_cool1
```

Functionality of aria_xplor_command

This command is designed to run an XPLORscript command interface by passing a python string template containing the XPLORscript commands and also a list of values to be used in the template using the python %(<variable-name>)x syntax and dictionary of name value pairs. The function prints a dummy `'EVALUATE: symbol set to <value> (<type>)'` string which can be captured by RollinStone for debugging to follow assignment of variables and mimics the string produced by CNSscript on variable evaluation.

```
def aria_xplor_command(template, params = {}):
    template = '''
    set message all echo on end
    %s''' % template
    for param, value in params.items():
        print('EVALUATE: symbol $%s set to    %s     (%s)' % (
              param.upper(),
              str(value),
              value.__class__.__name)
             )
    xplor.command(template % params)
```

This was designed to keep the code as similar to the ARIA2 CNSscript code as possible. Once ARIAXC is completely updated to use the XPLOR-NIH python scripting commands for the molecular dynamics and setup this command should no longer be needed.

Defining a Dictionary with class XplorDict

XplorDict is a dictionary class used to store values from ARIA2 read from `run.json` in a hierarchical manner. A diarammatic example of an `XplorDict` is shown below in Figure 28.

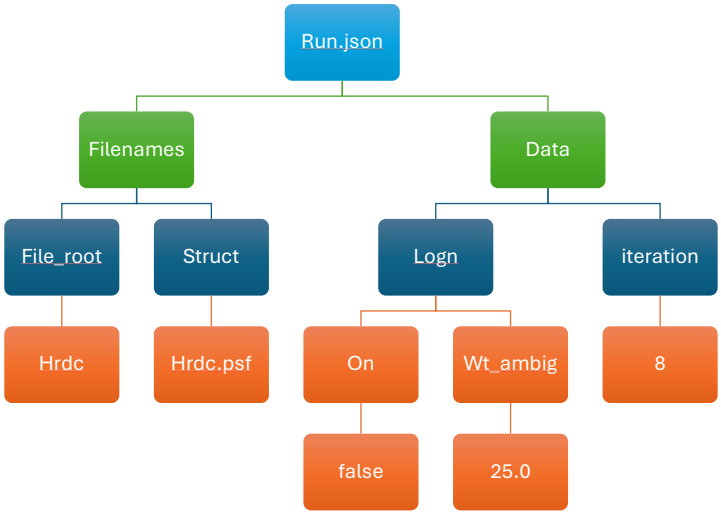


Figure 28. A diagram of a snippet of run.json showing the dictionary of data structure. In this example filenames points to file_root which points to hrdc, which is a string. The equivalent dictionary structure is shown in figure 28.

As an example, (Figure 28. A diagram of a snippet of run.json showing the dictionary of data structure. In this example filenames points to file_root which points to hrdc, which is a string.) is the CNS parameter named `data.Logn.Wt_ambig` is set to 25.0. In Python the `XplorDict` instance is created using the `XplorDict()` constructor and modified using the python array syntax.

```
xplor_dict =XplorDict()

xplor_dict['data']['Logn']['Wt_ambig'] = 25.0
```

this create a dictionary named x inside xplor_dict which contains another XplorDict dictionary y, which then contains the value z which is set to 10 and which can be reread by the python interface. At the same time this will lead to creation of a variable x.y.z inside XPLOR with the same value as what z is set to (10).

Overall XplorDicts characteristic features are as follows.

1. it can contain other dictionaries or values
2. it knows the name of the key that contains it in the parent
3. it has a reference to the parent

4. can get its own path or the path to a variable in it by walking the parent links (which must be no longer than 19 letters including commas, dots and underscores)
5. it knows the type of non-dictionary values in it which must be compatible with the XPLORscript types: int bool sting or float
6. setting a non-dictionary value sets its XPLORscript value via *xplor.command('evaluate $variable=<variable-value>)*
7. upon reading a non dictionary value it reads its XPLORscript value using *xplor.command('', '<variable-value-name>)*
8. it can check types of values coming back from XPLOR for compatibility
9. it understands the XPLORscript alternatives for Boolean True and False YES / NO and ON / OFF

Note in the future once use of direct access to XPLORscript values via *xplor.command()* is no longer used it is expected that coordination of values between Python and XPLORscript will no longer be needed. Note currently the stored values of non-dictionary values are not used as all values are read from and stored in XPLORscipts engine. A diagram of how an XPLORDict is stored is shown below Figure 29 The internal structure of a XplorDict, with the route used to access the variable filenames.struct highlighted.

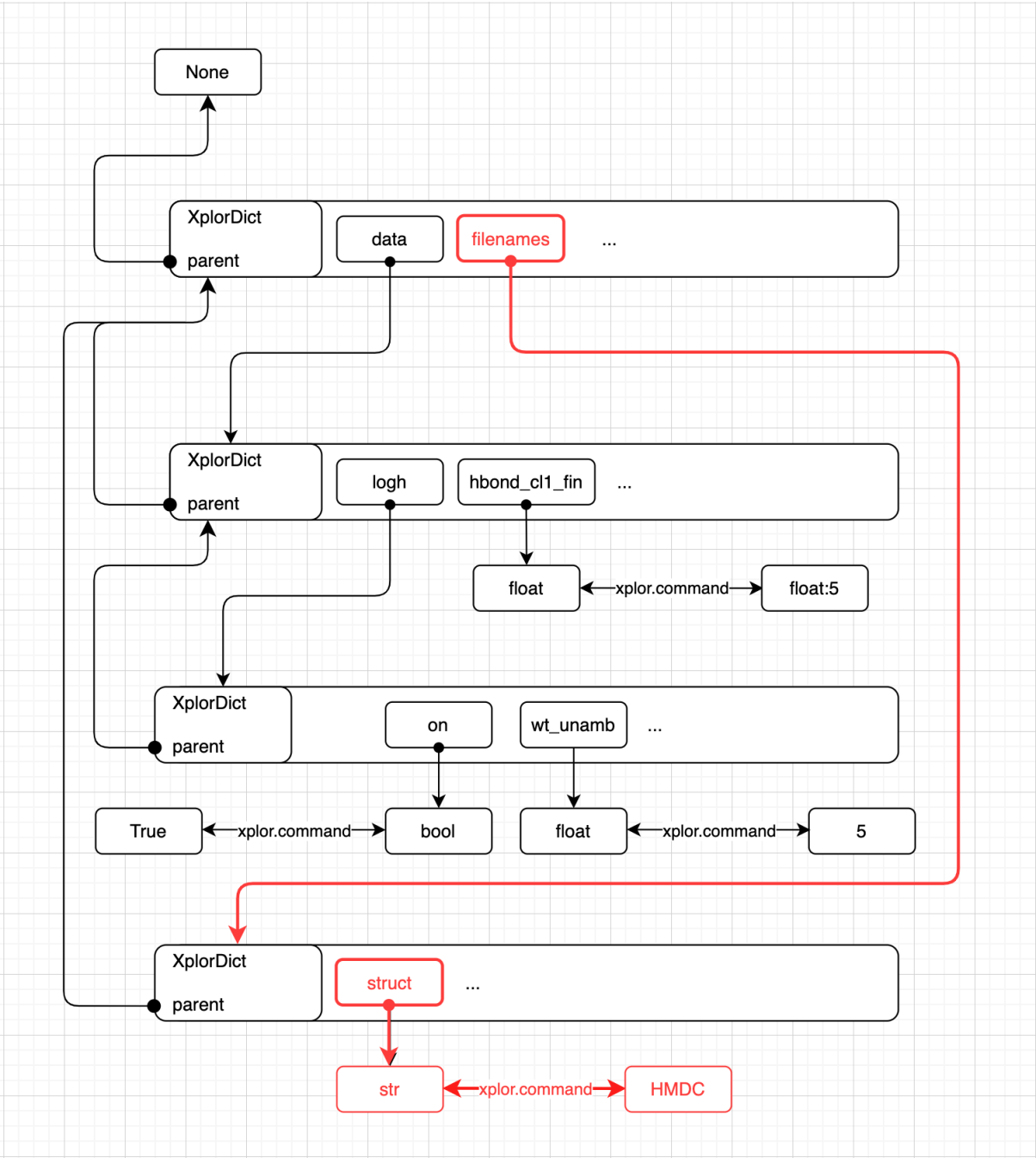


Figure 29 The internal structure of a XplorDict, and the interactions between named values and the hierarchy. Two examples can be followed. In red `filenames.struct` points to the value `HMDC` which is a value that can be accessed via the dictionary and an xplor command. Secondly in black `data.logh.on` points to a `boolean` value again accessible via the dictionary or an xplor command which in this case is set to `on` or `True`

The XplorDict module also contained some methods beyond the functions provided by a normal python dictionary, which functions are detailed in the Table 15 Extra methods in XplorDict beyond those in a standard python dict.

Table 15 Extra methods in XplorDict beyond those in a standard python dict

method	documentation
dump_xplor()	print all current state of XPLOR, all variables, dictionaries and types of objects using the command '*$?*' This is a static method
`items(self)`	All values in the current XploDict as a tuple(key,value)
`parents(self)`	the parent XplorDict dictionaries of the current XplorDict
`path(self)`	Return the path to the current dictionary as dot separated string
`raw_path(self)`	the same as above except the parent dictionary names are returned as a list of strings
`name(self)`	Returns the name of the dictionary
`__str__(self)`	Print a string representation of the XplorDict

XplorDict also has some standalone functions which can act on an XplorDict globally. These can be seen in

get_keys(xplor_dict)	Returns the keys stored in the XplorDict and all its children
ensure_keys(xplor_dict, keys, factory=XplorDict)	For each key in the list keys ensure it exists in the XplorDict if it doesn't create the key in the dict using the factory to make a new instance
from_keys(xplor_dict, keys)	For each key in *xplor_dict add its value toa results list which is then returned*
dict_to_xplor_dict(dict)	Convert an ordinary python dict to an XplorDict

Table 16. With their function names and signatures, and the purpose of them.

get_keys(xplor_dict)	Returns the keys stored in the XplorDict and all its children
ensure_keys(xplor_dict, keys, factory=XplorDict)	For each key in the list keys ensure it exists in the XplorDict if it doesn't create the key in the dict using the factory to make a new instance
from_keys(xplor_dict, keys)	For each key in *xplor_dict add its value toa results list which is then returned*

dict_to_xplor_dict(dict)	Convert an ordinary python dict to an XplorDict

Table 16 Standalone methods that work with XplorDict

A communication tool Json_load_ascii

The function `xplor_lib. json_load_ascii`, was built to load the json dictionary written by cns.py to communicate with the XPLOR-NIH python. This is a hierarchical dictionary of dictionaries and keys using `json.load` which loads a dictionary which is converted to an XplorDict using `xplor_lib. dict_to_xplor_dict`. The load function specifically takes a converter to ensure json Unicode is converted to 8-bit ascii as XPLORscript and python2.7 can only use ascii. Currently, the routine also calls the `xplor_lib. patch_run_dict` to add several missing parameters defined in the template used to create `run.cns` in ARIA2.

A Store of Stores Store_list.py

Store lists STORE1-STORE9 were a difficult part of the program to deal with when implementing ARIA2 using XPLOR-NIH as they are not supported by the XPLOR-NIH python interface. Stores are a place where information about molecular structure are stored, specifically information about atoms and selections of atoms in a molecular structure. Each store entry within a particular STORE (1-9) is writeable and readable and holds a float values for every atom in the defined structure, with one float per atom in atom id order. These are predominately used in the ARIA2/CNS when storing selections. An example of their used would be defining up methyls and storing selections of methyls and other atoms used during float stereochemical assignment (STORE1these are used by `swap.cns` and setup by `setup_swap_ini.cns`. Problems occurred it became in its present implementation XPLOR -NIH doesn't have python access to the stores. The solution to this problem was definition of custom python list class `store_lists.py` which provides access to the stores as a native python list. The constructor for `Store_list` takes a single store number defining the store to write and provides read and write access in atom id order, the lists are of fixed lengths (number of atoms in the structure. StoreList was generously provided by Dr. Gary Thompson.

The StoreList class is initialised with `__init__` initiated with a `store_number` and its values are those that are currently inside store in XPLOR-NIH. As an example, the follow code does the following for a 6 atom molecule with store 1 initialises to have all values equal to 0.0 by XPLOR-NIH. An example of its usage is as follows:

```
import xplor
from store_list import StoreList
xplor.command('vector do STORE1 0.0')
store_1=StoreList(1)
print(store_1)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
for i in range(4):
    store_1[i] = 2.0
store_1[4]=6.0
print(store_1)
[2.0, 2.0, 2.0, 2.0, 6.0, 0.0]
```

StoreList behaves like an ordinary mutable python list of fixed length, and which can only store floats. The only other extra function provided by StoreList is the `dump_store` function displays and "dumps" all of the stores in log files when running the program, allowing you to see all values inside of a store. StoreList was provided by Dr Gary S Thompson.

Running calculations using XPLOR-NIH's python interpreter

XPLOR-NIH python scripts were run in refine.sh by calling pyXplor the XPLOR-NIH python interpreter.

Protocols Files and Usage

Table 17 list all the files used in the ARIA2/CNS structure calculation protocol, their equivalent python files (or – if they weren't ported to XPLOR-NIH) and an overview of their function.

Table 17 The major files used in ARIAs molecular dynamics protocols an a summary of the changes made to them, inclding any name changes made to the files in conversion to ARIAXC. In cases were '–' is used this no changes were made to the name of the file.

ARIA File name	ARIAXC Filename	overview
define_methyls_all.cns	Define_methyls_all.py	Function is to define all methyl groups into store 1
generate_dmso.cns	-	This generates a solvent box of dmso molecules around the structure for solvent refinement
print_coorheader.cns	print_coorheader.py	Print header information onto an output PDB file including energies and restraint violations
sa_l_hightemp.cns	sa_l_hightemp.py	High temperature cartesian dynamics
sa_ltad_cool1.cns	sa_ltad_cool1.py	First stage of torsion angle dynamics cooling
swap.cns	swap.cns	Systematically swap randomly selected methylene protons, accepts the conformation with lowest nOe energies, this can be used in conjunction with floating assignment using the ARIA FLIP command
define_methyls_ini.cns	define_methyls_ini.py	This file intialises methyls tables defining all methyls and all aromatic groups including STORE1.
generate_water.cns	-	This generates a solvent box of TIP3 water molecules around the structure for solvent refinement
print_coorheader_auto.cns	print_coorheader_auto.py	Doe the same as print_coorheader buu for when a log harmonic nOe function is used
sa_l_randomchain.cns	sa_l_randomchain.py	Randomises a structure by random rotations around the N-CA and CA-C bonds
sa_ltad_hightemp4.cns	sa_ltad_hightemp4.py	high temperature phase of the torsion angle dynamics
torsiontop.cns	torsinotop.py	Setup the tosion angle dynamics engine topology
flags_new.cns	flags_new.py	Set a series Booleans in run.cns/json controlling various aspects of the structure calculation
logn_auto_weight.cns	-	Sets up the log harmonic nOe function
read_data.cns	read_data.py	reads restraint data into the calculation

sa_l_reduced.cns	sa_l_reduced.py	Sets up a reduced representation of the structure for cartesian dynamics
sa_rc_hightemp.cns	sa_rc_hightemp.py	High temperature cartesian dynamics
write_symnoe.cns	-	writing of symmetry related nOes
freezn.cns	-	setup of calculations with bound ions
newsymmetry.cns	-	setup of calculations with symmetric oligomers
sa_l_cool1.cns	sa_l_cool1.py	Run the initial cooling phase for Cartesian and Torsion angle dynamics from and minimisation of nOe potentials. Slowly reducing the energy within the calculation and finding global minimal for the potential energy statements.
sa_ls_cool2.cns	sa_ls_cool2.py	This si the 2nd and final cooling phase in the calculation for both Cartesian and torsion angle dynamics Brining the structure further down into the global minimum for the energy potentials, using minimisation and energy statement
setup_swap_init.cns	septup_swap_init.py	Initially selects all methyls, isopropanol, and guadino into store 1 under different triggers. Then defines all floating groups; for isopropyl ad hetero atoms
write_symnoe2.cns	-	Is the updated version of the *write_symnoe2* which deal with updated forcefields and symmetry statements for CNS.
generate.inp	generate.py	This files is the first of 2 files that setup the structure calculation in the working directory. Findig the location of data setting the types of bonds and structural information that will be used , as well as writing structural PSF file of the molecule.
Generate_template.inp	Generate_template.py	This is the 2nd of the generate files that sets up the templates for that data will be written into as well as

		rinning patches for atoms like Zn^+, minimisation and dynmaics statements.
Refine.inp	Refine.py	Runnings once the generate files are complete the entire structure calculation, organising the order and parameters of element of the structure calculation before initialising each part of the program.
Refine_OLD.inp	Refine_OLD.inp	Is a old and now deprecated code that used to run the structure calculation and is now not in use
Refine_dmso.inp	Refine_dmso.inp	Carry out a final refinement in a box of DMSO
refine_water.inp	-	Carry out a final refinement in a box of water

Unused CNS files

A number of CNS file were not modified during the port and were replaced by stub files so that if their function were called an error will be raised These files are indicated in Table 17 by a – in the ARIAXC Filename column. A stub was so that it was clear which feature of the CNS implementation of ARIA were not ported (such as the log normal potential interface, metal ion support and symmetric multiuser support have is currently not available in XPLOR port). These features were ignored as they are not a core part of making the XPLOR-NIH implementation work and in some cases do not have the correct FORTRAN support code in XPLOR-NIH (e.g. the Log Normal potential).

A Look at the preliminary changes to interactions with molecule dynamics engine CNS.py

Before discussing the implementation of the CNS protocol files to python, it is important to discuss *cns.py*. This is used by ARIA to setup data structure for the entire structure calculation from the project file This file was edited multiple times throughout the overall process as errors were found and corrected. The edits that were made are described below along with why they were required and what they accomplished the discussion runs in the order of the change in the file and not in the order of the changes were made.

A section that modifies and translates the names of aria variables to XPLOR variables is the first change that you will see of me inside of *cns.py*. This section contains a python dictionary of name translations. These used the dictionary called *JSON_TRANSLATION*. JSON_TRANSLATION is needed because the version of XPLOR used has a technical limitation that variable names are limited to no more than 19 characters long (which includes periods), and many of the current CNS variables are much longer (for reasons of legibility and documentation). To solve this, JSON_TRANSLATION, was used to convert ARIA/CNS variable names throughout the XPLOR port to ensure name consistency. The original names and new names were stored as strings and tuples of strings respectively as show below.

```
JSON_TRANSLATIONS = {
    'counter' : ('data', 'count'),
    'iteration' : ('data', 'iteration'),
    'initial_pdb' : ('data', 'initial_pdb'),
    'randomize_mainchain' : ('data', 'rdmize_mchain'),
    'use_template' : ('data', 'use_template'),
    'file_root' : ('filenames', 'fileroot'),
```

In the example above of JSON_TRANLSATIONS is a dictionary of dictionaries, that is used to help converts data from CNS/ARIA2 names to XPLOR-NIH names. For example, `data.randomize_mainchain` is over 19 character long and will not work in XPLOR-NIH and was translated to `data.rdmize_mchain`, which is 18 character long and useable in XPLOR-NIH.

`JSON_TRANSLATIONS`, was also used to convert strings to the hierarchical keys used in `run.json`. An example can been seen above, were the dictionary has a keys `'data'` *and* `'filenames'`. The values; `'count'`, `'iteration'`, `'initial_pdb'`, `'rdmize_mchain'` *and* `'use_template'` are all stored under data whereas `'fileroot'` is stored under filenames. All values are either float, integer, string or boolean values. This conversion was carried out for all items originally saved in `run.cns` which now currently reside in `run.json.`

`Protocols_core` is the next component of the code that required modifying. This tuple contains a list of file names (as strings) that need to be copied over to the

working directory of the program from the built-in protocols distributed with ARIA. This was modified to replace CNS filenames with Python filenames. As seen in the below text this has been mostly kept the same with just edits to <filenames> give them Python extensions, for example. *Torsiontop.cns* became *torsiontop.py* or *refine.inp* became *refine.py*.

CNS implementation

```
PROTOCOLS_CORE = ('boxtyp20.pdb', 'define_methyls_all.cns',
                    'define_methyls_ini.cns', 'dmso.pdb',
                    'flags_new.cns', 'generate.inp',
                    'generate_dmso.cns', 'generate_template.inp',
                    'generate_water.cns', 'print_coorheader.cns',
                    'read_data.cns', 'refine.inp',
                    'refine_dmso.inp', 'refine_water.inp',
                    'sa_l_cool1.cns', 'sa_l_hightemp.cns',
                    'sa_l_randomchain.cns', 'sa_l_reduced.cns',
                    'sa_ls_cool2.cns', 'sa_ltad_cool1.cns',
                    'sa_ltad_hightemp4.cns', 'setup_swap_init.cns',
                    'swap.cns', 'torsiontop.cns', 'xplortodiana3.inp',
                    'write_symnoe.cns', 'write_symnoe2.cns', 'newsymmetry.cns',
                   'sa_rc_hightemp.cns', 'freezn.cns', 'logn_auto_weight.cns',
                    'print_coorheader_auto.cns')
```

XPLOR implementation

```
PROTOCOLS_CORE = ('boxtyp20.pdb', 'define_methyls_all.py',
                    'define_methyls_ini.py', 'dmso.pdb',
                    'flags_new.py', 'generate.py',
                    'generate_dmso.cns', 'generate_template.py',
                    'generate_water.cns', 'print_coorheader.py',
                    'read_data.py', 'refine.py',
                    'refine_dmso.inp', 'refine_water.inp',
                    'sa_l_cool1.py', 'sa_l_hightemp.py',
                    'sa_l_randomchain.cns', 'sa_l_reduced.py,
                    'sa_ls_cool2.py', 'sa_ltad_cool1.py',
                    'sa_ltad_hightemp4.py', 'setup_swap_init.py',
                    'swap.py, 'torsiontop.py', 'test_ener.py', 'xplortodiana3.inp',
                    'write_symnoe.cns', 'write_symnoe2.cns', 'newsymmetry.py',
                    'sa_rc_hightemp.py', 'freezn.py', 'logn_auto_weight.cns',
                    'print_coorheader_auto.py', 'xplor_lib.py', 'store_list.py',
                    'run_cns_patch.py'
```

The template string *CSH_SCRIPT_REFINE* also needed modification. This string is a template for the file *csh_refine.csh* which ARIA2 writes to the working directory to setup and run a calculation using CNS. This required updating to deal with Python by adding PYTHONPATH environment variable to point to *data/protocols* directory where refinement scripts are stored. Additionally, during debugging the output from each calculation was output to the terminal by commenting out the output file for the run and adding a warning to each run of cns.py that displays.

'WARNING [Project]: CSH_SCRIPT_REFINE modified to directly output to terminal'

The final code in the file is shown below with changes / additions are annotated red.

```
CSH_SCRIPT_REFINE = '''\
#! %(sge_job_shell)s
# SGE/PBS/SLURM/LFS facility
#$ -N %(sge_job_name)s
#$ -S %(sge_job_shell)s
#PBS -N %(sge_job_name)s
#PBS -S %(sge_job_shell)s
#BSUB -J %(sge_job_name)s
#BSUB -L %(sge_job_shell)s
#SBATCH -J %(sge_job_name)s
#SBATCH --ignore-pbs

## results will be stored here
setenv NEWIT %(iteration_path)s

## project path
setenv RUN %(cns_resource_path)s

##Python path
setenv PYTHONPATH %(python_path)s

## individual run.cns is stored here
setenv RUN_CNS %(cns_working_dir)s

## CNS working directory
cd %(cns_working_dir)s

## solves some NFS sync problems

cat %(cns_input_file)s > /dev/null

## command line
%(cns_executable)s < %(cns_input_file)s #>! %(cns_output_file)s

touch done
'''


print('WARNING [Project]: CSH_SCRIPT_REFINE modified to directly output to
terminal')
```

The template string `CSH_SCRIPT_REFINE_CONDOR` for the condor batch system also required modifying in the same way with addition of a PYTHONPATH environment variable and changes to output.

The variable `run.cns` which contains the path to the file defining where data for a CNS calculation is read from during a calculation (value `'run.cns'`) and was

replaced by *run.json* as parameters were imported from a JSON file rather than being generated from the execution of a CNS or XPLOR script.

Defining the molecular dynamics engine to ARIAXC

For the program to run the user also has to define the path to the xplor_exceutable. This was achieved by changing *structure_generation local_executable* parameter in the project file (e.g. run1.xml) to point to pyXplor file for each calculation as shown below.

```
<structure_generation engine="cns">
  <cns local_executable="/Users/garythompson/programs/xplor-nih/2.51/bin/pyXplor"
```

Converting parameters into XPLORDict compatible with JSON

The function *_translate_to_json_dict* was written help with conversion of parameters from ARIA2 internal dictionaries to the format used by the *run.json* file and XplorDict data storage in XPLOR-NIH. It takes a target dictionary to write to an entry name and a value. It then uses *JSON_TRANSLATION_DICT* to look up the new destination of the value as a tuple of names and then uses the internal function *iterate_tuples_or_values* to iterate over these and create new sub dictionaries or key value pairs in the target dictionary. The code for the method is shown below

```
def _translate_to_json_dict(self, dict, entry, value):
    def iterate_tuples_or_values(targets):

        result = []
        elements = [elem for elem in targets if not isinstance(elem, (tuple, list))]
        if len(elements) != 0:
            result.append(tuple(elements))
        else:
            tuples = [elem for elem in targets if isinstance(elem, (tuple, list))]
            result.extend(tuples)

        for elem in result:
            yield elem

    for translation in iterate_tuples_or_values(JSON_TRANSLATIONS[entry]):
        if len(translation) == 2:
            container_name, value_name = translation
            dict[container_name][value_name] = value
        elif len(translation) == 3:
            container_name_0, container_name_1, value_name = translation
            if not container_name_1 in dict[container_name_0]:
                dict[container_name_0][container_name_1] = OrderedDict()
            dict[container_name_0][container_name_1][value_name] = value
```

```
        else:
            raise Exception('unexpected translation entry got ' + `translation`)
```

interfacing with JSON its functions__write_run_json, _update_run_cns_dict, update_chain_data and analysis

The original function inside of *cns.py* which wrote the CNS variables inside of *run.cns* was called `_write_run_cns`. This was replaced by `_write_run_json`. This method was based on `_write_run_cns` but was heavily modified so as to setup the dictionaries of dictionaries used to transmit parameters to XPLOR-NIH using JSON. The methods `_update_run_cns_dict`, `update_chain_data` were also heavily modified to conform to the new dictionary of dictionaries format used. Parameters were written to `run.json` using the `json.dump()` function. The change of name to `_write_run_json` from `write_run_cns` required a small change to the analysis method to use the new write function name.

Access to forcefields via ARIAXC run_cns_patch.py and its patch_run_dict function

The function `patch_run_dict` is called by `json_load_ascii` to patch the XplorDict runtime parameter dictionary with extra parameters that are not provided by ARIA but have defaults at runtime. This is a stopgap till a better mechanism is designed. The contents of the function are:

```
def patch_run_dict(default_cns_dict):

    file_root = default_cns_dict['filenames']['file_root']

    #TODO reference most likely pulls a reference structure find it
    #default_cns_dict['filenames']['reference'] = reference
    default_cns_dict['filenames']['template'] = file_root + '_template.pdb'

    default_cns_dict['filenames']['struct_2'] = ''
    default_cns_dict['filenames']['struct_3'] = ''
    default_cns_dict['filenames']['struct_4'] = ''
    default_cns_dict['filenames']['struct_5'] = ''

    default_cns_dict['toppar']['pdb_or_seq'] = 'PDB'

    default_cns_dict['toppar']['par_nonbond'] = "PROLSQ"

    if default_cns_dict['toppar']['prot_par_1'] == 'parallhdg5.3softbig.pro':
        default_cns_dict['toppar']['softfield'] = True
    else:
        default_cns_dict['toppar']['softfield'] = False

    default_cns_dict['toppar']['prot_par_3'] = ''
    default_cns_dict['toppar']['prot_par_4'] = ''
    default_cns_dict['toppar']['prot_par_5'] = ''
```

```
if default_cns_dict['data']['rdc1_on'] == False:
    default_cns_dict['data']['rdc_1_choice'] = False

if default_cns_dict['data']['rdc2_on'] == False:
    default_cns_dict['data']['rdc2_choice'] = False

if default_cns_dict['data']['rdc3_on'] == False:
    default_cns_dict['data']['rdc3_choice'] = False

if default_cns_dict['data']['rdc4_on'] == False:
    default_cns_dict['data']['rdc4_choice'] = False

if default_cns_dict['data']['rdc5_on'] == False:
    default_cns_dict['data']['rdc5_choice'] = False

default_cns_dict['saprot']['fbeta'] = 20
default_cns_dict['saprot']['mass'] = 100
default_cns_dict['saprot']['tempstep'] = 50

default_cns_dict['toppar']['ss_amb_unamb'] = 'unambiguous'

default_cns_dict['toppar']['xplortodiana'] = False

default_cns_dict['data']['flags']['bonds'] = True
default_cns_dict['data']['flags']['angle'] = True
default_cns_dict['data']['flags']['impro'] = True
default_cns_dict['data']['flags']['dihed'] = True
default_cns_dict['data']['flags']['vdw'] = True
default_cns_dict['data']['flags']['elec'] = False

default_cns_dict['data']['flags']['noe'] = True

default_cns_dict['data']['flags']['coup'] = False
default_cns_dict['data']['flags']['vean'] = False
default_cns_dict['data']['flags']['sani'] = False
default_cns_dict['data']['flags']['dani'] = False

default_cns_dict['data']['flags']['plan'] = False
default_cns_dict['data']['flags']['ncs'] = False

if default_cns_dict['toppar']['par_nonbond'] == 'PROLSQ':
    default_cns_dict['toppar']['repel_radius'] = 1.0
    default_cns_dict['toppar']['repel_rcons'] = 20
    default_cns_dict['toppar']['repel_rexpo'] = 4
    default_cns_dict['toppar']['repel_irexp'] = 1

elif default_cns_dict['toppar']['par_nonbond'] == 'PARMALLH6':
    default_cns_dict['toppar']['repel_radius'] = 0.8
    default_cns_dict['toppar']['repel_rcons'] = 5.0
    default_cns_dict['toppar']['repel_rexpo'] = 2
    default_cns_dict['toppar']['repel_irexp'] = 2

elif default_cns_dict['toppar']['par_nonbond'] == 'OPLSX':
    default_cns_dict['toppar']['repel_radius'] = 0.0

else:# now the standard PARALLHDG parameters
    default_cns_dict['toppar']['repel_radius'] = 0.78
    default_cns_dict['toppar']['repel_rcons'] = 5.0
    default_cns_dict['toppar']['repel_rexpo'] = 2
    default_cns_dict['toppar']['repel_irexp'] = 2

default_cns_dict['jcouplings_on'] = ''
print('patching')
return default_cns_dict
```

Results 2 Debugging the ARIAXC calculation protocol

This chapter of the thesis will discuss how the ARIAXC program was initially debugged while it was being ported process and the techniques used to make the process as efficient as possible. This will be followed by an analysis of why this method became ineffective and how this was solved by developing another program called RollinStone. This discussion will include how RollinStone works and how it has a larger scope then just debugging and allows for detailed comparisons and analyses of dynamics calculations that were not previously possible. It will also include an explanation of how RollinStone works internally along with the available command line parameters and options. Finally, an example analysis will demonstrate how the RollinStone suite allowed analysis and display of runs of ARIAXC and ARIA2 with identical data was used to help debugging the port of ARIA2 to ARIAXC.

Implementation debugging

Initially, when running the ARIAXC structure calculation, the output of ARIA2 and ARIAXC for the same molecule and parameters were stored in individual `refine.out` files and compared manually. This made debugging slow, and the causes of failure to converge or crashes were not always immediately available from these output logs. To increase the speed of debugging a single refinement file was [code break 1 below showing contents of `refine.csh` driver script] was saved and the redirection of output to disk was removed [Code break 2 contents of the `refine.csh` *driver script to output to screen*] thus allowing for all output to be directly printed on the terminal. As debugging a complete calculation involves at least 7 structure calculations and a minimum of 8 iterations for each structure, lasting minutes and producing large amounts of output, the quantity of data to analyse was far too large. It was therefore decided to initially just debug one round of calculation [it0 for structure 1] which was achieved by initialising the calculation of a complete ensemble and manually stopping it and then manually running a single calculation, using the `refine.csh` command, in the calculations working directory.

```
## command line
%(executable)s <
```

```
/home/lukep/projects/ariaxc/aria2.3/examples/werner/run1/cns/protocols/refine.inp
>! refine.out
```

Code break 1

```
## command line
%(executable)s <
/home/lukep/projects/ariaxc/aria2.3/examples/werner/run1/cns/protocols/refine.inp #
>! refine.out
```

Code break 2

This allowed for debugging “on the fly”, at the command line (standard out) for quick analysis of new code and testing its functionality with print statements, error messages, and exit commands. Initial attempts at debugging the new code by the addition of a print statement allowed several errors to be found, including differences in the number of some loops that were performing, python syntax errors, and errors in the states of variables during a calculation this process was still slow which lead to the development of the *RollinStone* program described later in this chapter.

Later in the project, once basic debugging was complete debugging was sped up by carrying out complete calculations in parallel using the ARIA job manager. To achieve this project XML was edited to create parallel runs as shown code below, dramatically reducing runtimes.

```
<job_manager default_command="csh -f">
<host enabled="yes"
    command="csh -f"
    executable="/home/lukep/projects/ariaxc/xplor-nih-2.53/bin/pyXplor"
  n_cpu="8" use_absolute_path="yes"/>
  </job_manager>
```

This Job manager enables the host to control certain elements of the structure calculation, including the executable used for calculation [in the case of ARIAXC pyXplor], and the number of CPUs used `[n_cpu]`. During the conversion from ARIA2 to ARIAXC, a point was reached where the code would be completed without errors, including output to log files using these techniques. However, the structure calculations were not converged suggesting that some of the parameters controlling the calculation were not set correctly during the calculation even though the calculations competed.

The RollinStone Tool: an ARIA critic

RollinStone was created after the initial port to use Python/XPLOR-NIH was completed. Since when a structure calculation runs correctly there are many parameters that need to be correct for it to succeed. There was therefore a need to be able to monitor these and output the equivalent parameters and outputs (e.g. energies) in both ARIAXC and ARIA2 to see if they are consistent.

This wasn't reasonable with the use of print statements and uninterpreted log files (which are enormous) [ARIA runs 8 iterations with 20 structures each ~≥26,000 lines per file for each structure produced in each iteration a total ~ 1.4M lines of text to be analyses for a complete structure calculation.

Therefore, the RollinStone program suite was developed, based on two programs, *`rollinstone.py`* and *`rs_differ.py`*, and modifications to the scripts in protocols [the test harness]. These were created to harvest interesting components of this and analyse it for efficient debugging. The design of the programs allows the data to be displayed to be filtered by parameter, step range, action, run and iteration to make the process more informative, and comprehensive, and reduce the mental load and time required to analyse the output data.

The basic procedures that RollinStone carried out were

1. *`rollinstone.py`* parses CNS and XPLOR log files to gather
    a. Structure number
    b. Iteration number
    c. Currently module/subroutine
    d. Dynamics step number [a minimisation was treated as 1 dynamics step]
    e. parameters natively present in the log file (e.g. energies)
2. Save the collected data tables to CSV files
3. *`rs_differ.py`* compares outputs from 2 runs [either CNS or XPLOR-NIH] by
    a. Taking in 2 CSV table files from rollinstone and reading them as Pandas dataframes

b. Aligning the outputs by structure number, iteration number, and dynamics step number [using a Pandas join]
c. Finding the parts that were different between the files
d. Either placing these differences into another *"comparison file"*
e. Or saving the differences and overall changes of parameters in PDF files as graphs or tables

RollinStone tool is scripted in Python3.10 rather than the 2.7 syntax used in ARIAXC implementation. This allowed the use of modern constructs such as Python new *match* statement syntax to be used, which allowed this program to build more quickly and succinctly. An example of a comparison graph that shows an error detected in the parameter is shown in Figure 30.

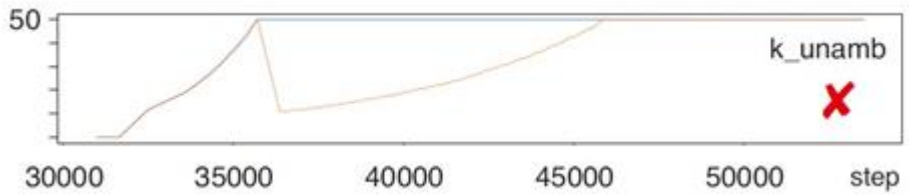


Figure 30 Example of a graph from RollinStone showing the detection of an error in the parameter k_unamb. Blue line ARIA2 and the dotted red line ARIAXC.

RollinStone internals and CLI

RollinStone is a relatively simple program and effective program that was impactful by sorting through vast quantities of data identifying errors and that allows for quick debugging and analysis of CNS and XPLOR-NIH dynamics trajectories inside versions of ARIA [ARIA2 /ARIAXC]

The program `rollinstone.py` takes a series of arguments that control how the program is executed and allows for flexibility in its use. The arguments used by the program are listed in the Table 18.

Table 18. RollinStone inputs collected in the parameter args created by the function return from the function get_arg_parser

Command line	Parameter Name	Function
root	PATH	Root directory of the structure calculation (e.g aria2.3/)

-r	structure_id	ID of the structure being calculates (e.g. 1. default all)
-i	iteration	The iteration to analyse (e.g. it1, default is all)
-t	test	Run the program test suites
-o	out	Where to save the run, Default standard out
-f	format	Format to save output data (default tabulation, other alternatives are csv (comma separated values) or json)
-e	energy_only	Only display energy in the output (default false)
-m	max_line	Only process up to this line number in the input (default all)
--show	show	A pair of integers defining the range of lines to display
-v	verbose	Print verbose information if root is not given, if structure file or iteration number to output for the user to add / correct in the command line

RollinStone uses the command line commands to help locate the correct directories and files that should be analyses and is designed so it can be used on any system, as the layout may vary on the users' machine. An example of the program in use in use can be seen in the code below use of RollinStone program suite to collect data from dynamics calculations. where it is run twice to produce outputs to be compared using outputs from ARIAXC and ARIA2.

```
# ARIAXC
Python3.10 rollin_stone.py root
/home/lukep/projects/ariaxc/aria2.3/examples/werner/temp2 -r run_cns_1 -i 1 -o
comparison_files/rs_xplor -f csv

# ARIA
Python3.10 rollin_stone.py root
/home/lukep/projects/ariaxc/aria2.3.backup/examples/werner/temp -r run_cns_1 -i 1 -
o comparison_files/rs_cns -f csv
```

In this case, these commands will write out a table CSV file called `rs_xplor.csv` in the `tools/rollinstone/comparison_files/` directory for the ARIAXC command, and a file `rs_cns.csv` in the same directory or the ARIA command.

These files can be used with the `rs_differ.py` program to analyses any differences between the 2 structure calculations.

The structure of *rollinstone.py* consists of several main classes that run the program *`LineIterator`*, *`FileParser`*, and *`TableFormatter`*.

The internal logic and structures of *`rollinstone.py`* are as follows.

Entry to the program occurs via the function *`main()`*, which in order

1. parses command line arguments created by the function `get_arg_parser()` which returns `args`.
2. Instantiate a *`FileParser(args)`* using the `args` and calls `parse_files` on it to parse the input files which are stored in the *`FileParser table`* property.
3. Creates a *`TableFormatter(out, args.format)`* which takes a output file or stdout and the *`FileParsers`* *table* and writes an output table.

The *`FileParser`* class carries out two tasks

1. It Iterates over the directories of an ARIA calculation defined by *`root`* and finds its log files (from the arguments at the start of the program *`-r`* and *`-i`*).
2. Parse the discovered XPLOR and CNS log files into a table [dict] of data, *`self.table`*, indexed by the structure number and the iteration number.

The entering point of the parse is the methods *`parse_files()`* and parameters passed to the class *`FileParsers`* initialiser should contain the *root* of the ARIA calculations data,structures provided by the command line argument root and the flags that control the data parsed into the program. The *`parser`* then parses the data from each iteration using the method *`parse_iteration(dir_path)`* which takes a template for the paths of an iteration directory to parse in the form of a glob expression, e.g:

`aria2.3/examples/werner/run2/temp/*/run_cns_*`

This then reads the lines in each files and calls *`_parse_file_lines(lines)`* with the lines. *`_parse_file_lines`* then uses an iterator class call *`LineIterator`* to count lines, stop iteration if the maximum line count has been reached, print the

current line to screen if verbose mode is engaged and return a line at which point the line is split into fields and empty lines are discarded.

The main component of the `_parse_file_lines` is a large match statement that matches each line in the input to a particular structure in the file and passes it on to a particular parser Table 19.

Table 19 Each selection that is selected by RollinStone and the action that is taken for each

structure	Action
File/functions start / end	Add file to `file_stack`
Symbol evaluation i.e. x=1.2	Call `_process_symbols()`
Minimisation cycle	Call `_process_symbols()`
Dynamics step	Call `_process_symbols()`
Initial torsion dynamics setup	Call `_process_symbols()`
Initial cartesian dynamics setup	Call `_process_symbols()`
Cartesian dynamics start	Call `_process_symbols()`

The `_process_symbols()` methods make use of a utility function called `grab_lines_with_bar()` to grab complete tables of XPLOR/CNS energies delimited with bar symbols [|], for example like the one shown in Figure 31.

```
--------------- rms fluctations for last       50 steps -----------------------
| E(kin)+E(total)=17.722          E(kin)=0.000         temperature=0.000      |
| Etotal =17.722     grad(E)=0.822      E(BOND)=0.771      E(ANGL)=0.895      |
| E(IMPR)=0.483      E(VDW )=0.006      E(HARM)=0.000      E(NOE )=17.110     |
| E(SANI)=0.000                                                               |
-------------------------------------------------------------------------------
```

Figure 31 An XPLOR /CNS energy table from command display ener statement, that print to output all energy terms at the point it is called.

Each function call then gathers the information gathered into the value to be stored and an iteration key, in Figure 32 which defines the exact stage of the calculation that has been reached.

```
class IterationKey(NamedTuple):
    index: int
    refine: int
    iter: int
    current_stage: str
    cycle_number: int
    action: str
    block_type: str
```

Figure 32 An the class iteration key, that bring back the tuple for each parameter a integer, or a string.

The fields of the *IterationKey* are *index*, the current *IterationKey* number, *refine* the index of the structure calculation that the information comes from, *iter* the current ARIA iteration number (1-8), *stage* the CNS/XPLOR script that is being called [e.g. *sa_ltad_hightemp4.cns* etc] *cycle number* the current cycle number of a minimisation or dynamics calculation, *action* the action being taken which changes the state of the molecule which is one of *cartesian, minimise and torsion* and finally a block type which can be one of *start, step or cycle* which defines which part of a minimisation or dynamics step is taking place. With each key a full set of the variables that define the programs state are save [constants, energies and internal state are save as the data for the key]. Examples of data that are saves are shown in Table 20.

Table 20 Examples of information stored per cycle by RollinStone

Value type	Example
K_BON	constant defining the weight of bond energies
E_NOE	Total energy of the nOe restraint terms
Temperature	The target temperature of the temperature bath

The final part or RollinStone the *TableFormatter* class encapsulates the output of the data in the read lines from *parse_file_lines(),* in a table which can be output to json csv files or pretty printed output using the tabulate library each line contains the contents of an *IterationKey* and the information on the programs states [Energies, constants and internal parameters such as the temperature bath]. The data saved at each cycle are:

E_ANGL E_BOND E_DIHE E_IMPR E_NOE E_VDW E_kin E_kin+E_total Etotalamb_fac asy asy_add atom_id bath bor_fac_1 bor_fac_2 bor_fac_3 bor_fac_4 bor_fac_5 calcsteps cen_fac_1 cen_fac_2 cen_fac_3 cen_fac_4 cen_fac_5 count countcycle critical cyclesteps debug di_counter fin_con fin_flt fin_noe flt_fac grad_E haveplan hbond_fac high_steps i_cool ini_con ini_flt isoc_counter isop_counter iteration k_amb k_bor_1 k_bor_2 k_bor_3 k_bor_4 k_bor_5 k_cen_1 k_cen_2 k_cen_3 k_cen_4 k_cen_5 k_flt k_hbond k_noe k_pack_cool1 k_pack_cool2 k_sani_1 k_sani_2 k_sani_3 k_sani_4 k_sani_5 k_unamb k_vdw k_vdwfact masy masy_add n_atom_id n_rest name1 name2 ncycle new_save nreassign nstep phi planfile psfcount psi randomfilename reassign res resid sani_fac_1 sani_fac_2 sani_fac_3 sani_fac_4 sani_fac_5 seed seg ssdd_counter ssde_counter ssee_counter sssd_counter ssse_counter ssss_counter

structurefile swap temperature templatefile tempstep time timestep unamb_fac

A number of utility functions were also written to help facilitate the execution of the program and their facilities have been listed below,

- *run_tests()*, a function that uses the *pytest* function to run unit tests of the program and then exits the program. Iif anything test fail it will display an error message. This function is called when the *-t* is passed on the command line.
- *exit_error(error_msg)*, called when non recoverable errors are encountered and prints out the error message to standard output and exits the program.

Detecting Discrepancies with Rs_differ

The rs*_differ.py program* (a Rollin Stone DIFFerence finder and graphical display) consists of two components. The program works in 2 parts

1. firstly, it takes information from *RollinStone* output (*rs_cns* and *rs_xplor* CSV files) and finds the differences between them.
2. Secondly the data read in the first section can be displayed as either a PDF with a series of graphs showing differences for each item, or a table either as a CSV file or a tabulate formatted table.

The entry point to the *rs_differ* program is the `main()` function which has three major components the function *get_arg_parser()* function, *FileChecker(args)* class, and the *XplorCnsGraph(args)* class. The first of these as in the *rs_differ* program is the argument parser, which as in the *rollinstone* program allows a number of flags to be set on the command line. A list of the parameters and options can be seen in Table 21.

Table 21 RS_differ inputs collected in the parameter args created by the function returned from get_arg_parser

option	Argument name	Function of argument
-x	xplor_csv	The name of the XPLOR csv file to compare
-c	cns_csv	The ID of the CNS csv file to compare
-o	out	Where to save the output of rs_differ, the default standard out
-f	format	The format to save tabulated data in, the default is csv, alternatives are jsonor tabulate
-g	Graph out	The name of file to save the the comparison graphs in, default= comparison_graphs
--graph-format	Graph format	type of file to save graphsas , default = pdf, alternatives = png, gif, raw, jpeg
-t	test	Run the programs unit tests rather than the program

An example of using `rs_differ.py` can be seen in Figure 33. Which show how the the arguments are used.

```
## Rs_differ

python3.10 rs_differ.py -x comparison_files/rs_xplor.csv -c comparison_files.rs_cns.csv -o
xplor_cns_comp -f csv -g comparison_graphs -gf pdf
```

Figure 33 example command line use of rs_differ.py . This starts by invoking python3.10 with the script file name rs_differ.py, using the -x switch which precedes the xplor log file name and the -c switch which precedes the CNS log file name, the -o switch denotes the output file name and the -f switch the format of the output file, finally the -g and -f switches define the e graph filename and format required.

In this example python3.10 is used to run the script `-x` and *-c* define the XPLOR and CNS table files to be compared. The option `-o xplor_cns_comp` sets where to save the text output, and the option `-f csv` defines the format of the saved information as a CSV file and sets the file extension of the file. For the graph outputs the `-g` option defines the name of the graph file if you want graphical displays and the `–graph-format` sets the format of the graphs and their file extension for graph save types. NOTE graph formats other than PDF will create many image files in the output directory as they don't define the idea of a page).

Internally the first part to rs_differ.py is the class `FileChecker(args)` working through the function `file_diff_checker()`. This takes the ARIA2 and ARIAXC from RollinStone and compares them. Initially the data from each csv file is read into a pandas data frame they are merged using the pandas merge function with

`how='outer'` and the join variable being the index `i` of each file. The merged table file will have items from `xplor_csv` labelled `x_` and those from `cns_csv` labelled `y_`. The result is then written out to the output defined by the `-o` option with the form from the `-f` option. An example of a comparison file can be seen in 9.

Figure 34 comparison files produced by rs_differ displayed using the pyCharm color csv tool. This program shows the variables, energies(E), constants (k), and location in the structure calculation (index, iteration, refine current _stage). This file display's ARIAXC data with the prefix _x denoted, and ARIA2 prefixed with _y.

Figure 34. shows the differences between the item_x being from ARIAXC in the first image, and item_y from the original ARIA2 code in the second image.

The second part of `rs_differ.py` is the class `XplorCnsGraph(args)` the main method of this class is `display_graphs()`. This takes the previously made table file (whether saved or not) and creates a series of graphs that compare values from related columns such as `E_BOND_x` and `E_BOND_y`. Each comparison is placed in a

single graph which is autoscaled and the _x (ARIAXC) and _y (ARIA2) values are presented with different colours [orange dotted and blue solid]. Figure 35 shows an example of the output produced; the x scale is the step number of the calculation [*i*].

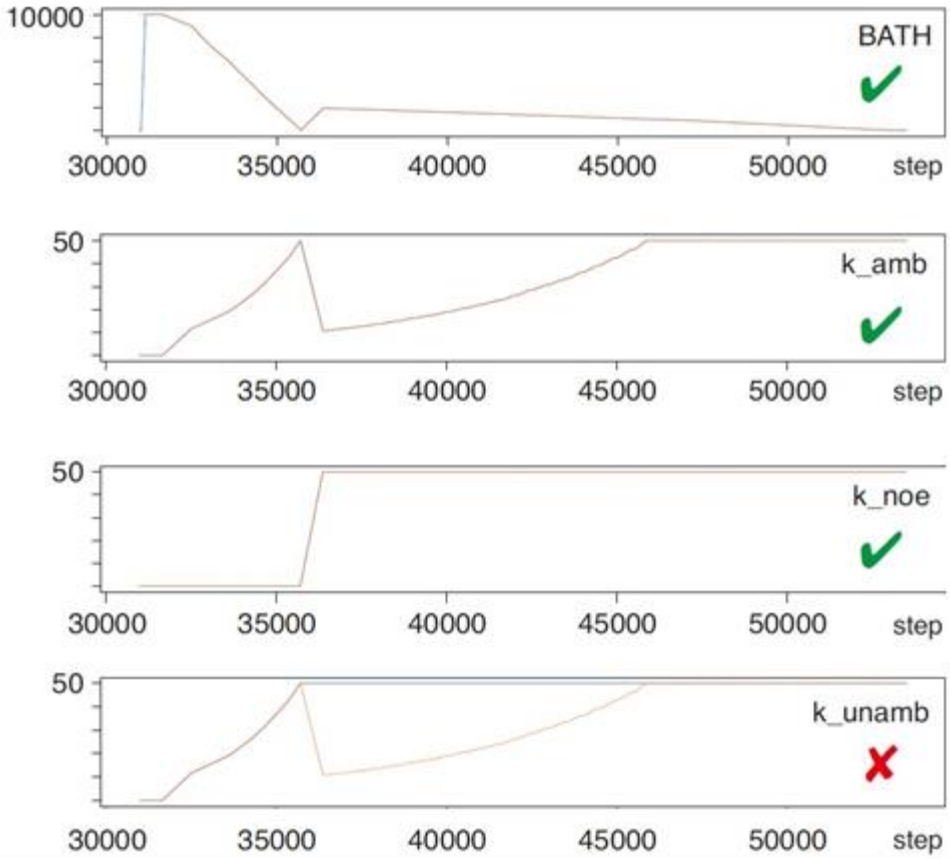


Figure 35 shows two examples from the graphical display from rs_differ.py. This shows the BATH variable that is used throughout both programs to control temperature (ARIAXC red dotted line and ARIA2 blue solid line) and k_unamb a constant for scaling unambiguous nOe energies as the program executes. Graphs with ticks have the same output from both programs, while inconsistent data outputs for the two programs are indicated by a cross.

The example in Figure 35 shows 4 variables `BATH` (target temperature of the temperature bath), `k_amb` (weight for ambiguous nOe data), `k_noe` (weight of all nOes), and `k_unamb` (weight for unambiguous nOe data) it can be observed that in this case the value of `k_unamb` incorrectly plateaus just after step 35000 due to a bug, which was later corrected. To do this the value of step [*i*] was cross-referenced against the output files to find files involved using START and END columns.

There are also some other small support functions in the program that function in the same fashion as with `rollinstone.py:main()`, `run_test()`, `exit_error(msg)`

Debugging ARIAXC with and without RollinStone

The previous section of this chapter discussed how and why the RollinStone program was created and its intended use to allow for debugging of the new ARIAXC program by comparison of its state and output to the original ARIA2 program. This made debugging relatively quick, especially considering the large scale of the debugging output of the ARIAXC program and appears to have led to the removal of almost all the major bugs within the code.

Before RollinStone was written many bugs were identified by manual analysis of the output, but even then, the structures calculated from the Werner Tudor test proteins data were not converged and had extended structures such as those shown in Figure 36 being produced at it0, which shows no collapse of the structure. It was found that this was due to errors in the nOe restraint handling code which failed to apply the nOe restraints. For comparison an equivalent structure from ARIA2 at this point of the calculation is shown. It should be noted that as the calculation was only carried out to it0 at this stage of the calculation ARIA does not produce any refinement statistics.

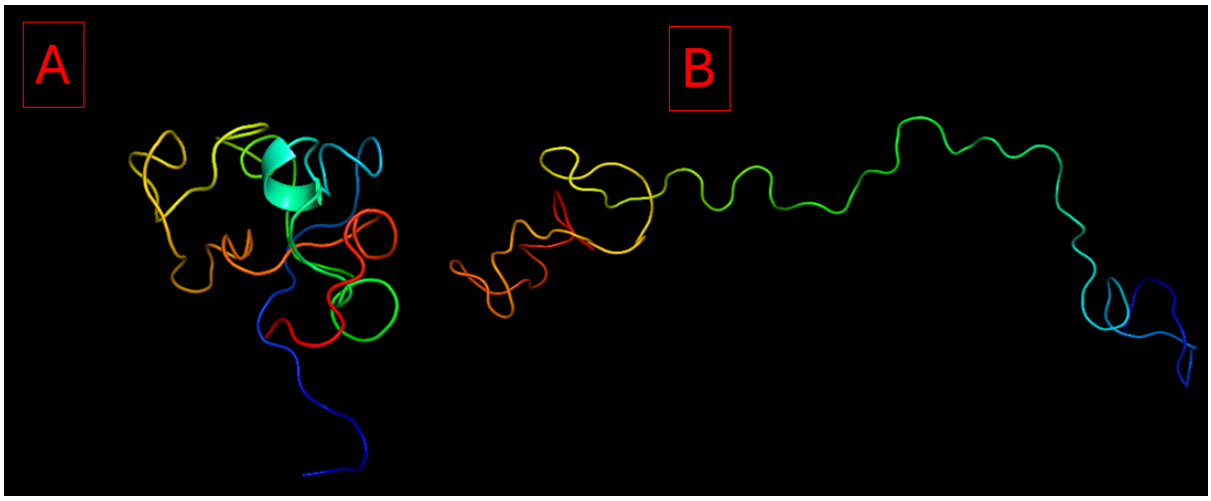


Figure 36 Lowest energy structure from it0 of the Werner test (A). from an ARIA2 calculation (B) after initial manual debugging of ARIAXC.The collapes of the structure and some alpha-helices are visible at this stage in ARIA2, but the helix formations has not occurred for ARIAXC and the structure is extended. It is clear that ARIAXC at this stage is not dealing with restraints as intended and needs further analysis of the programming.

However, even after correcting the nOe weight errors, the resulting structures using the Werner test were not well converged and had a disordered structure. It was therefore decided that a more systematic approach to debugging the program was required. This led to the development of the RollinStone program.

Debugging insights from RollinStone

The implementation of RollinStone allowed the identification of a number of obvious and more subtle issues within the structure calculation, as discussed above, that would not have been easily identified otherwise. These were correct as a series of improvements and corrections over several months each of which was saved to the git repository used for code management as commit. Specific errors that were removed included

1. The wrong constant *mrswi_hot* was being used instead of *asy_hot* to set the force constants for nOe distance restraints in `sa_ltad_hightemp4.py`. The high-temperature structural search phase.
2. In `flags_new.py` before energy calculations were made and the forcefield components in use were setup the command, *flags exclude * end* which switches off all energy terms was not being called. This leads to extra unexpected energy terms being enabled.
3. In the patching process run_cns_patch.py the forefield parameter *toppar.par_nonbond* was not being set to *PROLSQ*, leading to the wrong forefield being used.
4. In `sa_ltad_cool1.py, sa_l_cool1.py, sa_l_hightemp.py, sa_ls_cool2.py, sa_ltad_cool1.py` and `sa_ltsa_hightemp.py` in a number of places integer division rather than float divisions were being made with integer values
   *k_vdwfact = (fin_con/ini_con)**(1/ncycle)*
   replaced with
   *k_vdwfact = (fin_con/ini_con)**(1/float(ncycle))*
5. In *sa_l_cool1.py and sa_ls_cool2.py* noe energies and forces were not scaled correctly

Once the bugs in ARIAXC have been corrected, the structure calculation was repeated, and analysis of the refinement statistics was repeated.

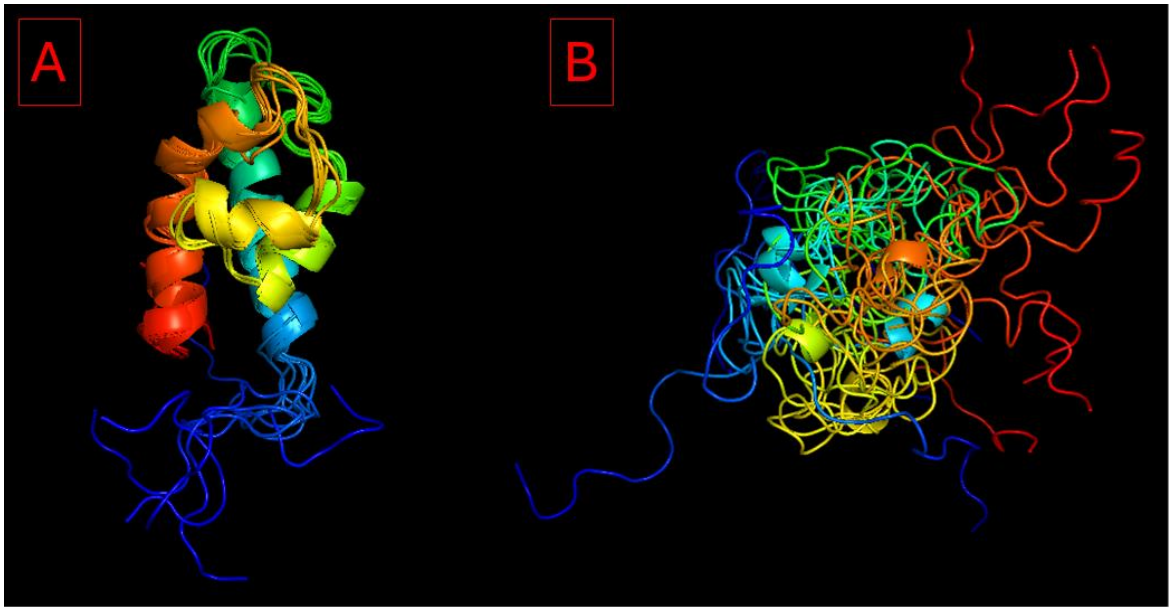


Figure 37 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) after debugging with RollinStone. The 7 lowest energy structures overlaid by Cα backbone atoms are shown. The residues in the structures are coloured blue to red. ARIA2 shows a highly similar and structured proteins with 3 alpha-helices forming in the middle werner with some disorder regions at the end of the protein. The ARIAXC structure at this point is highly disordered and not showing any signs of convergence.

After debugging with RollinStone Table 22 shows that ARIAXC has better structure ensembles, Figure 37, lower energies than after the manual debugging, there were fewer nOe violations and the Cα RMSD had improved. This is in not the case when comparing ARIA2 to ARAIXC post RollinStone, with much higher energy besides overall energy, with Significant difference between there restraint violations. However, the number of residues with poor Ramachandran had increased (favoured 30->20, outliers 27-> 38) which is not unexpected for a constrained structure with nOe restraints that are still not correct as seen in Figure 38, showing Ramachandran violations are still largely inflated when compared to ARIA2 structure ensemble.

Table 22. Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC after using RollinStone debugging tool.

Energies (kCal mol^{-1}) averages in brackets	ARIA2	ARIAXC post RollinStone	ARIAXC pre RollinStone
overall	1023.99	1491.21	2915.83
bond	26.7971	47.0351	64.2595
angles	102.593	192.598	254.677
impropers	15.8122	27.6716	55.1623
dihedrals	552.259	0	615.377
vdw	158.187	230.759	385.916

noe	168.34	993.141	1540.44
No. Restraint Violations			
noe	0	22	39
Rmsd Å			
Cα	0.5755	8.591667	9.591667
Ramachandran Violations◆			
favoured	57 [64.04%]	20 [22.47%]	30 [33.71%]
outliers	9 [10.11%]	38 [42.70%]	27 [30.34%]
Z-score	-7.07 ± 0.62	-9.22 ± 0.34	-9.06 ± 0.40
* calculated on the the Cα atoms of residues 15-86, which encompasses all regions of secondary structure in the ARIA2 structure ◆ molprobity, lowest energy structure			

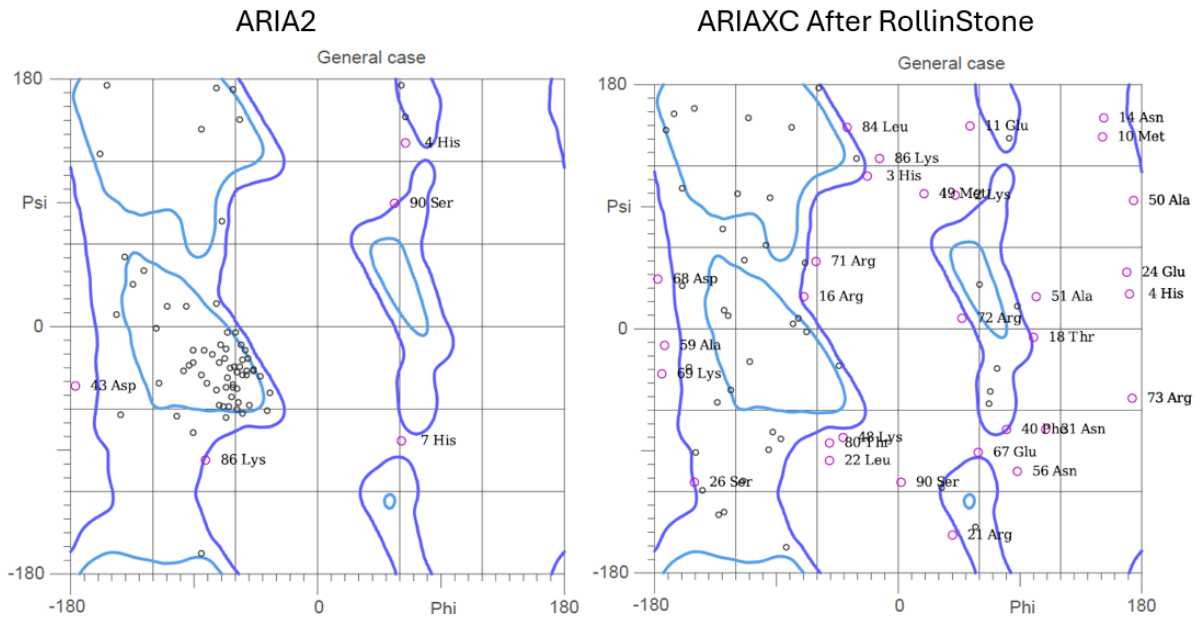


Figure 38 Shows Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers]

At this point, there were no obvious bugs within ARIAXC pertaining to contents or variables during calculation as all constants, and variables in use in ARIAXC could be shown to have the same values as ARIA2 through the use of RollinStone. However, at the same time the structures in the Werner test were still not converged, and still have very large differences between ARIA2 and ARIAXC especially in regards to restraint and Ramachdran violations. This led to the hypothesis that there must still be a systemic problem with one of the molecular dynamics terms, but not its parameters and lead to a new approach debugging approach which was described as Structure Injection.

Results 3 Troubleshooting Errors in Force Fields and Structure Injection

Structure Injection

Once it was determined that there was no error in the constants or variables used during each calculation or the number of loops carried by using RollinStone to compare ARIAXC to ARIA2, the obvious source of error left was a difference in the energies or derivatives arising from the potential terms used during the calculation. However, comparing structure calculation between even very similar programs is a problem using energies from potential terms. This is because each structure calculation in ARIAXC / ARIA2 even when starting with the same coordinates and random number seed is unique as the order of numeric operations will be subtly different leading to differences in floating point values that drive otherwise equivalent molecular dynamics calculations to distinctly different results. To solve this problem a novel method was devised by Dr. Gary Thompson called 'Structure Injection'. In this approach a structure calculation was run using ARIA2 or ARIAXC and before each dynamics or minimisation step coordinates were saved to a temporary file and replaced with a test structure. Once the test structure coordinates were read an energy calculation was made, the energies were displayed and then the original coordinates from the trajectory were reloaded (figure 38)

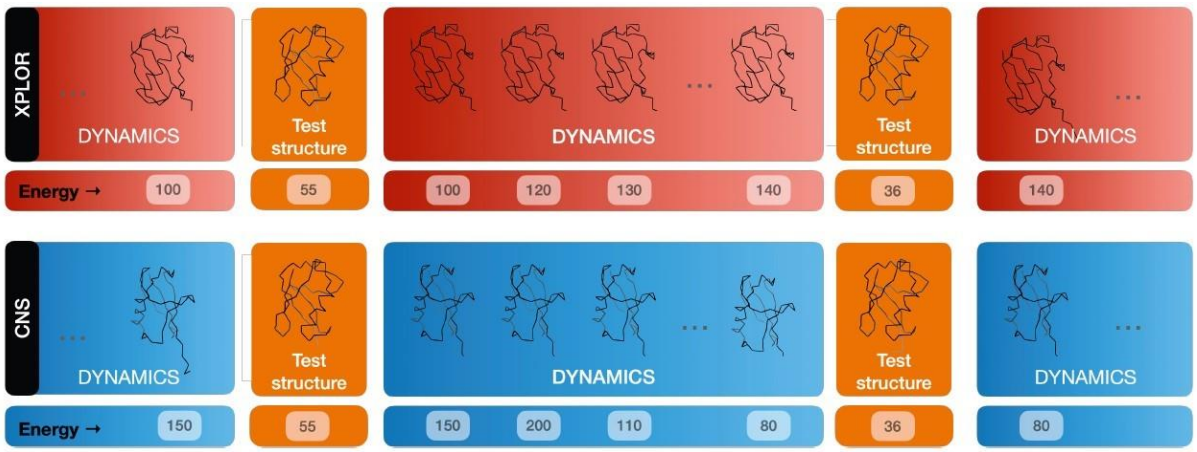


Figure 39 shows diagrammatically how structure injection works. In each row a calculation is carried out and energies calculated which differ between runs and programs [blue and red] except for the injected structures which being the same have the same energies.

The *test_ener* a function was created to help support with this method, and is located in *xplor_lib* in ARIAXC and as a separate module test_ener.cns in ARIA2. The test_ener function is shown in Table 23 A and B, comparing ARIA2 using the CNS scripting language, and ARIAXC written in python.

Tables 23A and B. compare the functions test_ener functions in ARIA2 and ARIAXC.

A. ARIA2 test_ener.cns

```
module(remark;count)
display START test_ener.cns

display local remark &remark
home/luke/git/ariaxc/ariaxc/test.pdb")
evaluate($save_path = "/tmp/save_cns_" + encode(&count) + ".pdb")
write coordinates sele= (all) output=$save_path end
coor @@$test_path
energy end

if ($exist_elec = FALSE) then
   display elec not defined
   evaluate ($elec=0.0)
end if

if ($exist_noe = FALSE) then
   display noe not defined
   evaluate ($noe=0.0)
end if

if ($exist_cdih = FALSE) then
    display cdih not defined
    evaluate ($cdih=0.0)
end if
if ($exist_coup = FALSE) then
    display coup not defined
    evaluate ($coup=0.0)
end if
if ($exist_sani = FALSE) then
    display sani not defined
    evaluate ($sani=0.0)
end if
if ($exist_vean = FALSE)  then
   display vean not defined
   evaluate ($vean=0.0)
end if

display test energies header [ cns | &remark ]: overall, bonds, angles, improper, dihe, vdw, elec, noe, cdih, coup, sani,
vean
display test energies values [ cns | &remark ]: $ener, $bond, $angl, $impr, $dihe, $vdw, $elec, $noe, $cdih, $coup, $sani,
$vean

display debug dihe
print threshold=0.0 dihe
display debug dihe end
display debug noe
print threshold=0.0 noe
display debug noe end
display debug stop

coor @@$save_path
ener end
display END test_ener.cns
```

B. ARIAXC test_ener.py

```
import xplor
def test_ener(remark, count):
   xplor.command("""
       display START test_ener.cns
       evaluate($test_path = home/luke/git/ariaxc/ariaxc/test.pdb")
       evaluate($save_path = "/home/luke/git/ariaxc/ariaxc/save_%(count)d.pdb")
       write coordinates sele= (all) output=$save_path end
       coor @$test_path
       energy end

       if ($exist_elec = FALSE) then
          display elec not defined
          evaluate ($elec=0.0)
       end if

       if ($exist_noe = FALSE) then
          display noe not defined
          evaluate ($noe=0.0)
       end if

       if ($exist_cdih = FALSE) then
           display cdih not defined
           evaluate ($cdih=0.0)
       end if
       if ($exist_coup = FALSE) then
           display coup not defined
           evaluate ($coup=0.0)
       end if
       if ($exist_sani = FALSE) then
           display sani not defined
           evaluate ($sani=0.0)
       end if
       if ($exist_vean = FALSE)  then
          display vean not defined
          evaluate ($vean=0.0)
       end if

       display test energies header [xplor | %(remark)s]: overall, bonds, angles, improper, dihe, vdw, elec, noe, cdih,
coup, sani, vean
       display test energies values [xplor | %(remark)s]: $ener, $bond, $angl, $impr, $dihe, $vdw, $elec, $noe, $cdih,
$coup, $sani, $vean

       display debug dihe
       print threshold=0.0 dihe
       display debug dihe end
       display debug noe
       print threshold=0.0 noe
       display debug noe end
       display debug stop

       coor @@$save_path

       ener end
       display END test_ener.cns
   """ % {'remark': remark, 'count': count})
```

In both versions the control flow is as follows

1. create a temporary coordinate file with the current coordinates in it

2. save the current coordinates to the file

3. read the coordinates of the test structure (the werner final structure lowest energy conformer) from disk

4. recalculate energies

5. display overall energies and forcefield component energies (bond, noe vean etc)

6. read the coordinates of the save state back from disk

The `test_ener` functions were used only before a minimisation or dymanics statements in the structure calculations, the example below shows test_ener being called in `sa_ltad_hightemp4.py` just before a `minimisation` step.

```
test_ener("sa_ltad_hightemp4 minimize (iteration= %(iteration)i structure=
%(count)i)" % {'iteration':data['iteration'], 'count': data['count']},
data['count'])
xplor.command("minimize powell nstep=200 drop=10.  nprint=25 end”)
```

Analysis of the resulting energies showed that the nOe values were not being calculated correctly and this was tracked down to an error in calculating nOe restraint terms caused by an error in the translation of the floating assignment swap calculations to Python. This can be seen in Figure 40 how structure injection is used to pick u errors where the values of paraeters are correct which will show a correct result in RollinStone but have been input incorrectly into a potential term. IN hthis case the CNS reference panel C are the energies from injection sutrtcure into the CNS calculation which is the ground truth. In panel B there is a error with the wrong parameters being provided for the nOe forcefield in the `sa_ltad_cool1` cooling loop where a parameter to the nOe forcefield was swapped to the wrong parameter. Panel A show the same calculation after the error with the nOe term was corrected but still contains an error in the parameters to the dihedral forcefield. Figure 40 also shows how error can be tracked to forcefield components and the relevant cooling loop using structure injection.

			A. xplor good noe params bad dihe params							diffs xplor good noe params bad dihe params						
			overall	bonds	angles	improper	dihe	vdw	noe	overall	bonds	angles	improper	dihe	vdw	noe
sa_tad_hightemp4	minimize	cycle	137.83	19.1399	100.398	18.0781	815.658	0.213602	0	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_hightemp4	torsion	1	6346.36	19.1399	100.398	18.0781	815.658	0.213602	6208.53	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	2	6864.72	19.1399	100.398	18.0781	815.658	0.177835	6726.93	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	3	7355.95	19.1399	100.398	18.0781	815.658	0.354868	7217.98	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	4	7831.13	19.1399	100.398	18.0781	815.658	0.708043	7692.81	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	5	8287.45	19.1399	100.398	18.0781	815.658	1.41279	8148.42	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	6	8617.48	19.1399	100.398	18.0781	815.658	2.81889	8477.05	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	7	8836.25	19.1399	100.398	18.0781	815.658	5.62441	8693.01	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	8	8886.88	19.1399	100.398	18.0781	815.658	11.2222	8738.04	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	9	8698.43	19.1399	100.398	18.0781	815.658	22.3911	8538.42	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	10	8135.97	19.1399	100.398	18.0781	815.658	44.6761	7953.68	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	11	7197.55	19.1399	100.398	18.0781	815.658	89.1405	6970.79	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00

			B. xplor incorrect noe params bad dihe params							diffs xplor incorrect noe params bad dihe params						
			overall	bonds	angles	improper	dihe	vdw	noe	overall	bonds	angles	improper	dihe	vdw	noe
sa_tad_hightemp4	minimize	cycle	137.83	19.1399	100.398	18.0781	815.658	0.213602	0	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_hightemp4	torsion	1	6346.36	19.1399	100.398	18.0781	815.658	0.213602	6208.53	-42.50	0.00	0.00	0.00	-42.50	0.00	0.00
sa_tad_cool1	torsion	2	6990.98	19.1399	100.398	18.0781	815.658	0.177835	6853.18	83.75	0.00	0.00	0.00	-42.50	0.00	126.25
sa_tad_cool1	torsion	3	7510.52	19.1399	100.398	18.0781	815.658	0.354868	7372.54	112.06	0.00	0.00	0.00	-42.50	0.00	154.56
sa_tad_cool1	torsion	4	8023.09	19.1399	100.398	18.0781	815.658	0.708043	7884.76	149.45	0.00	0.00	0.00	-42.50	0.00	191.95
sa_tad_cool1	torsion	5	8499.06	19.1399	100.398	18.0781	815.658	1.41279	8360.04	169.12	0.00	0.00	0.00	-42.50	0.00	211.62
sa_tad_cool1	torsion	6	8865.1	19.1399	100.398	18.0781	815.658	2.81889	8724.67	205.12	0.00	0.00	0.00	-42.50	0.00	247.62
sa_tad_cool1	torsion	7	8992.67	19.1399	100.398	18.0781	815.658	5.62441	8849.43	113.92	0.00	0.00	0.00	-42.50	0.00	156.42
sa_tad_cool1	torsion	8	9001.98	19.1399	100.398	18.0781	815.658	11.2222	8853.14	72.60	0.00	0.00	0.00	-42.50	0.00	115.10
sa_tad_cool1	torsion	9	8763.04	19.1399	100.398	18.0781	815.658	22.3911	8603.03	22.11	0.00	0.00	0.00	-42.50	0.00	64.61
sa_tad_cool1	torsion	10	8241.42	19.1399	100.398	18.0781	815.658	44.6761	8059.13	62.95	0.00	0.00	0.00	-42.50	0.00	105.45
sa_tad_cool1	torsion	11	7258.7	19.1399	100.398	18.0781	815.658	89.1405	7031.94	18.65	0.00	0.00	0.00	-42.50	0.00	61.15

			C. cns reference						
			overall	bonds	angles	improper	dihe	vdw	noe
sa_tad_hightemp4	minimize	cycle	137.83	19.1399	100.398	18.0781	858.158	0.213602	0
sa_tad_hightemp4	torsion	1	6346.36	19.1399	100.398	18.0781	858.158	0.213602	6208.53
sa_tad_cool1	torsion	2	6864.72	19.1399	100.398	18.0781	858.158	0.177859	6726.93
sa_tad_cool1	torsion	3	7355.95	19.1399	100.398	18.0781	858.158	0.354875	7217.98
sa_tad_cool1	torsion	4	7831.13	19.1399	100.398	18.0781	858.158	0.708068	7692.81
sa_tad_cool1	torsion	5	8287.45	19.1399	100.398	18.0781	858.158	1.41278	8148.42
sa_tad_cool1	torsion	6	8617.48	19.1399	100.398	18.0781	858.158	2.81887	8477.05
sa_tad_cool1	torsion	7	8836.25	19.1399	100.398	18.0781	858.158	5.62438	8693.01
sa_tad_cool1	torsion	8	8886.88	19.1399	100.398	18.0781	858.158	11.2222	8738.04
sa_tad_cool1	torsion	9	8698.43	19.1399	100.398	18.0781	858.158	22.3911	8538.42
sa_tad_cool1	torsion	10	8135.97	19.1399	100.398	18.0781	858.158	44.6761	7953.68
sa_tad_cool1	torsion	11	7197.55	19.1399	100.398	18.0781	858.158	89.1405	6970.79

Figure 40. Shows a series of 3 tables A, B, and C of energies for different restraints, against a file and type of molecular dynamics either torion or minimisation and the loop it occurred in. A showing ARIAXC with good nOe parameters but poor Dihedral parameters. B showing ARIAXC incorrect nOe parameters and poor dihedral parameters C being a ARIA2 original reference, showing how structure injection fixed nOe but not dihedral restraint a issue RollinStone could not solve. With Green showing correct energy and red showing an incorrect energy.

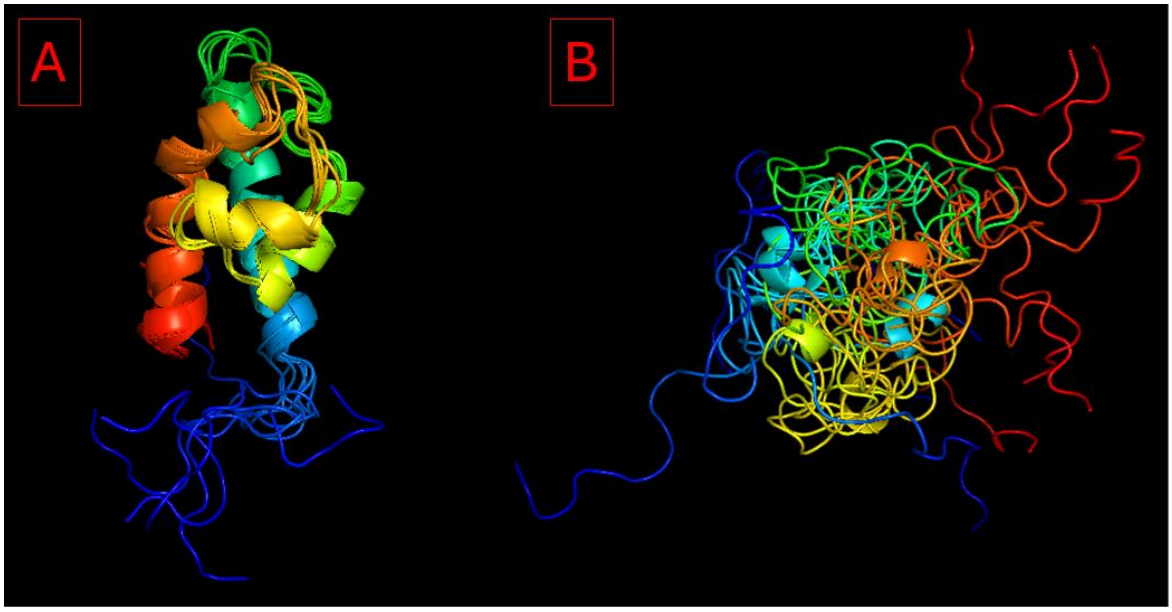


Figure 41 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) after erros were fixed using Structure Injection. The 7 lowest energy structures overlaid by Cα backbone atoms are shown.

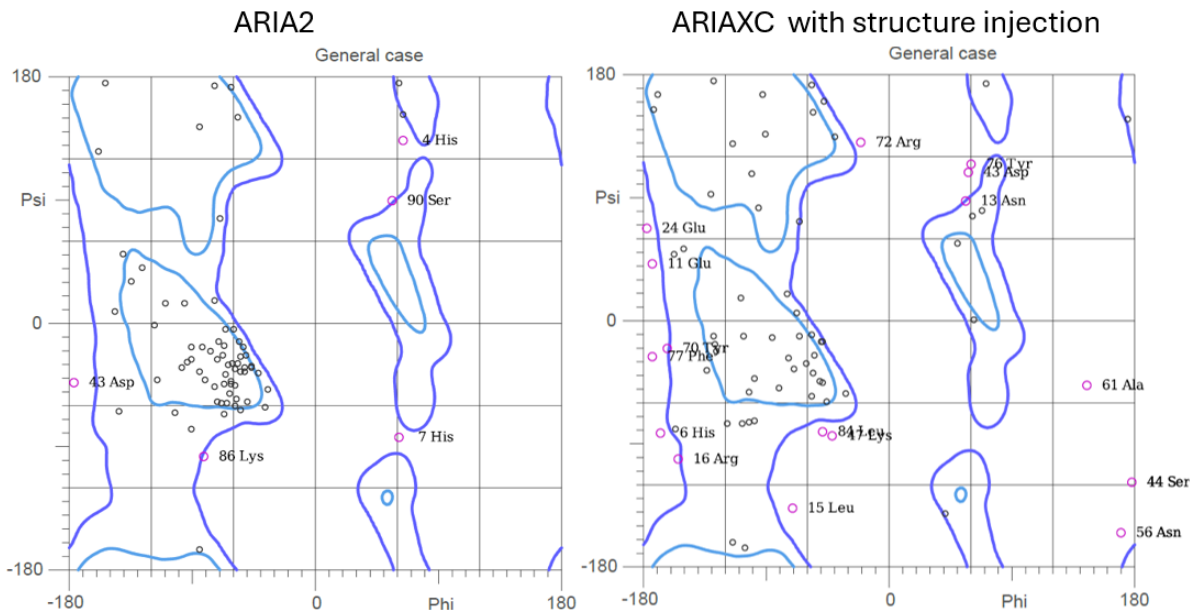


Figure 42 Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using of RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers]

Table 24 Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC from structure injection.

Energies (kCal mol^{-1}) averages in brackets	ARIA2	ARIAXC
overall	1023.99	3159.3
bond	26.7971	129.481

angles	102.593	131.249
impropers	15.8122	42.0696
dihedrals	552.259	596.668
vdw	158.187	495.907
noe	168.34	1580.98
No. Restraint Violations		
noe	0	36
Rmsd Å		
Cα	0.5755	0.418833???
Ramachandran Violations		
favoured	57 [64.04%]	32 [25.96%]
outliers	9 [10.11%]	21 [23.60%]
Z-score	-7.07 ± 0.62	-8.72 ± 0.46
* calculated on theCα atoms of residues 15-86, which encompasses all regions of secondary structure in the ARIA2 structure ◆ molprobity, lowest energy structure		

This led to further analysis of what could be preventing convergence. At this point all that remained was an error in the MD calculations and it was hypothesised that the standard Torsion dynamics engine in XPLOR-NIH as opposed to the newer IVM module was faulty or not as effective as the version in CNS which predated it.

Molecular Dynamics Ramping

To test if this was the case we decided to run the calculations for a considerably longer dynamics cycles (10x longer) to see if this would allow convergence (which would indicate that the energies and forcefields were correct). This was achieved by changing the *md_parameters* in the project file as below.

```
<md_parameters … " steps_high="100000" … steps_cool1="50000" steps_cool2="40000"/>
```

The results of these calculations are shown below in Table 25, Figure 43 and Figure 44.

Table 25 Refinement energy and restraint statistics for the best 7 structures from it8 of the Werner example set calculated using ARIA2 and ARIAXC after structure injection and with extended refinement cycles.

Energies (kCal mol^{-1}) averages in brackets	ARIA2	ARIAXC
overall	1023.99	1168.84
bond	26.7971	32.7061
angles	102.593	145.405
impropers	15.8122	20.2435

dihedrals	552.259	581.713
vdw	158.187	176.419
noe	168.34	212.353
No. Restraint Violations		
noe	0	1
Rmsd Å		
Cα	0.5755	0.418833
Ramachandran Violations		
favoured	57 [64.04%]	62 [69.66%]
outliers	9 [10.11%]	10 [11.24%]
Z-score	-7.07 ± 0.62	-5.95 ± 0.71
* calculated on theCα atoms of residues 15-86, which encompasses all regions of secondary structure in the ARIA2 structure ◆ molprobity, lowest energy structure		

This shows that energies from ARIAXC have decreased dramatically and while higher are within ~10% of those produced by ARIA2. The number of violations for nOes has also reduced to 1 which is a major improvement upon the previous calculation of 22 violations.

However, the largest difference is in the value of the Cα RMSD which dropped from 8.59 Å to 0.418 Å which is lower than the values produced by ARIA2. This result is further confirmed with the improved Ramachandran analysis with phi/psi pairs clustered in alpha region and with equivalent statistics for the number of residues in the favoured 69.66% (66/89) and in the allowed and favoured regions 80.9% (72/89) seen in ARIA2. Furthermore, the number of Ramachandran outliers was the same as ARIA2 with a most of the outliers being in disordered regions [1-14, 86-89].

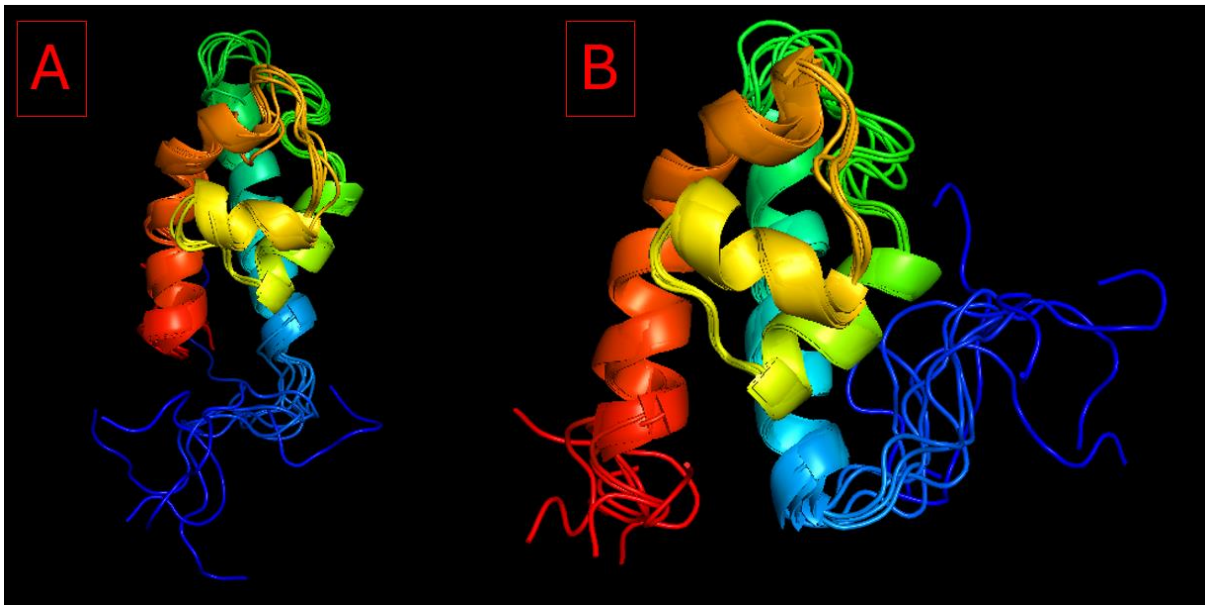


Figure 43 The best 7 structures of it8 of the Werner test set produced by ARIA2 (A) & ARIAXC (B) using an extended slow cooling phase. The 7 lowest energy structures overlaid by Cα backbone atoms are shown. ARIA2 and AIRAXC aremuch more similar here after using RollinStone, and structure injection with alph-ahelices occurring in similar regions, with similar disordered regions at the end of both proteins.

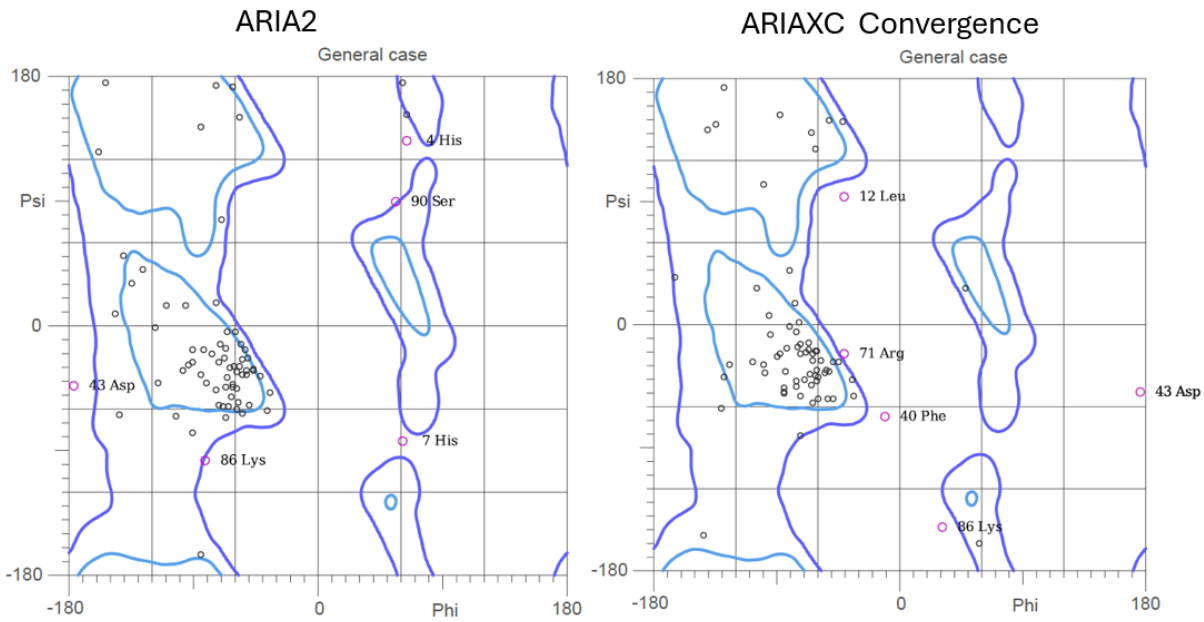


Figure 44 Ramachandran plot of general cases of ARIA2 (left) against ARIAXC after using debugging using of RollinStone [light blue regions are favoured, dark blue regions allowed, and outside of the dark blue regions are outliers]

Overall, from these results it can be seen that the initial implementation of ARIA-XC which ran to completion did not produce any recognisable structures, with the calculated structures having very high energies and RMSDs and Ramachandran geometry analysis consistent with an unfolded protein with up to 60% of residues

being in disallowed regions of Ramachandran space. However, with bugs removed manually and through the use of RollinStone and Structure Inject along with the use of longer simulated annealing steps converged structures with parameters almost identical to the ARIA2 test set were achieved, showing that the underling protocol was being carried out correctly.

The slow of convergence of the calculations suggests that while the fundamentals are complete there may be some form of bug or error in the way the XPLOR molecular dynamics was setup or in its implementation and while the project had no further time to run the completion conversion of ARIA2 to ARIA-XC is close to hand.

Discussion

Why convert ARIA2 to use XPLOR-NIH for calculations ARIA-XC

Protein and small molecule structures can be determined from NMR data, and while there are many observables available from NMR spectra there is one main observable that is used to determine the structure of molecules and proteins. This is the nOe, which measures inter proton distance and can tell us about the de novo global fold determination of polymer chains. nOe restraints can be combined with other restraints that can be measured from the signals produced from NMR and related techniques such as Chemical Shifts, RDCs, Solvent PREs, PREs and SAX envelopes and provide a potent method for calculation structures of proteins in solution. However, for the ARIA2 program many of these forcefields and restraints are not available as they are not supported by CNS, the molecular dynamics engine of ARIA2. This thesis has described how we converted ARIA2 to ARIA-XC, to a version of ARIA2 that uses XPLOR-NIH for molecular dynamics calculations which supports all of these restraints.

While NMR has many limitations, it also has many advantages when comparing structure determination to other methods like X-ray Crystallography. X-ray Crystallography is the most popular approach to determining the structure of proteins and other biomolecules, as it can produce more accurate structures and is often a faster and less costly preparation stage to view its structures. The advantages of NMR are it can view structures in solution, and it can report on dynamics and excited state structures with small populations. These benefits are especially important in flexible domains as they let us see how conformation changes and reactions occur in the proteins native environment, which is extremely important to determine how proteins function and they bind with other molecules.

The computation aspect of NMR spectroscopy brings many different approaches to determining the structures of bio molecules. The most abundant and hence valuable of these restraints and bonds come from the nOe effect of which many thousands can be measured for proteins and RNA / DNA with masses up to |25kDa. ARIA2 and CYANA are the two main programs used for calculation of NMR structure and separate themselves from other calculation programs by carrying out iterative simulated annealing on an ensemble of structures with most conformational restraints coming from nOes. Besides CYANA & ARIA2 the other approaches to NMR structure calculation are programs such as Chemical shift-ROSETTA (CS-ROSETTA), Cheshire, which calculates structures from chemical shifts.

ARIAXC's Chimeric Morphological synthesis

The path from running ARIA2 with CNS as the dynamics engine to ARIA-XC with XPLOR-NIH as the dynamics engine was achieved in several stages. Initially, changes were made until a complete structure calculation succeeded with XPLOR-NIH as the molecular dynamics engine ignoring whether the resulting structures were correct. The main changes made were

1. Changes to the internal structures of ARIA2 to save structure calculation data in a format readable by the XPLOR-NIH python implementation using JSON files, specifically *`run.json (cns.py)`*.
2. Changes to allow ARIA2 to start XPLOR-NIH as a calculation engine and call python code `(cns.py)`
3. Implementation of methods to read structure calculation data in XPLOR-NIH from *`run.json`* which included building a library of functions, *`XplorLib`* including, the XplorDict class to hold configuration information. This included name mapping as XPLOR-NIH has a limit of 19 characters on variable names.
4. Conversion of the protocols used by ARIA2 to calculate structured to ones using the XPLOR-NIHs python interpreter.
5. Implementation of access to the XPLOR-NIH vector STORES 1-9 using the ctypes library

Not unexpectedly debugging took considerable amounts of time in the project as ARIA2 and XPLOR/CNs are large and complicated programs. There were three stages to bugging ARIAXC as it was developed

Unpicking Erroneous code inside of ARIAXC

The first stage was the initial translation from CNS scripting interface to Python, where the program was modified to run until completion of iteration 0 which was run independently using the `refine.csh` command. Debugging was achieved, using print statement to monitor variables, until no syntax errors were showing up. This was a sufficient method of debugging at this stage as it was quick and gave almost immediate feedback on syntax errors in the Python code.

The second stage of debugging ARIAXC involved complete structure calculations across multiple iterations. However, this soon became inadequate as complete structure calculations with 9 iterations (it0-it8) and multiple structures became increasingly difficult with the huge amount of text output to look through and locate errors in.

This led later to the development of the RollinStone program as a test harness designed to reveal the many hidden details of the ARIA2 / XPLOR-NIH / CNS runtimes. This was the most efficient and sensible way to debug the program and proved to be highly successful. Having the ability to analyze these programs at this level of detail with easy comparisons of program state was crucial to debugging the structure calculation. This was proven by RollinStone's ability to debug dynamics loops, temperature ramping, and cooling where many bugs were found in the program, this level and accuracy of debugging would not be possible without it.

After removing all possible bugs within ARIAXC with RollinStone it was still found that resulting structures did not converge. A more analytical approach was required as it was problematic to compare XPLOR and CNS calculations as they used different source codes and therefore dynamics trajectories from them are not directly comparable. A novel method created by Dr. Gary Thompson was to check that both implementations produced the same energies when calculating energies with the same test structure 'structure injection' which was temporarily loaded during structure. This approach would be crucial to find errors in the implemented

algorithms particularly the swapping of prochiral methyl groups for stereo specific assignment.

Convergence and structural integrity

The challenge to find convergence from the original structure was that it was found that even with all energies and forcefields correctly ramped throughout the calculation the calculations still did not converge. It was therefore posited that there may be a fault in classic XPLOR torsion dynamics calculation and an increased number of dynamics steps was used to see if this would aid convergence. This was found to be the case showing that while there were still problems the converted protocol worked and could calculate structures with structural parameters that were consistent with ARIA2s output. During the debugging phase number of nOe violations energies and Ramachandran violations and distributions were used to monitor the quality of the resulting structures.

Limitations

In the project, we encountered many limitations largest being the length of a master project didn't finally allow me to see what stopped ARIAXC from converging only that it needed more cycles of temperature ramping and cool steps. Checking how each XPLOR/CNS handles their nOe potentials, and determining whether it would be beneficial to start incorporating the IVM module from XPLOR-NIH into the calculation would be a good next step. Another limitation of work was that we didn't compare the results of calculation with ARIAXC compared to ARIA2 on de novo protein structures or get to use the more novel forcefields available for calculations provide by XPLOR-NIH.

Initially, this project was conceived as a hybrid computational and laboratory-based study, with the intention of comparing measured protein structures from another laboratory. However, the project was undertaken during the COVID-19 pandemic and subsequently, the project evolved into a purely computational approach using ARIA2's example protein, Tudor domain, to test the validity of the structure calculation.

Future Work

Several options for future development can occur to make this program fully operational. The first is solving convergence, as we know XPLOR-NIH dynamics calculations behave differently from CNS. This will most likely involve using the XPLOR-NIH IVM dynamics module which is more modern and will allow for much faster structure generation, and better-quality structure to be created.

The second possibility is making the ARIA2 code more integrated with more XPLOR-NIHs facilities and adding additional functions that were not ported back into ARIA-XC making it a more effective program by surfacing all the methods available in ARIA2. An example is getting the structure calculation to run with other restraints and patches such as refinement of structure in water, addition of cis prolines, metal ion binding, multimeric structures and disulphide bonds.

Another would be the addition of modern forcefields for the ARIAXC to allow for more advanced technologies provided with XPLOR-NIH and more modern literature on the structure refinement tools, and protocols, or the inclusion of more modern forcefields which may lead to a better iterative produce to structure refinement. These would involve restraints from SAX curves, PREs, solvent PREs and chemical shifts, allowing for a much wider range observables and precise information about the resulting structures.

Conclusion

This thesis described the ARIAXC project which aimed to update the structure calculation program ARIA2 to use the more modern and capable dynamics engine of XPLOR-NIH rather than CNS. ARIA2 is a structure calculation tool that has diverse usage in NMR-based structure calculation strategies especially those based on nOes. However, the access to the powerful forcefields in XPLOR-NIH especially those related to PREs solvent PREs, and SAX as examples would enrich the toolset of those calculating structures using NMR.

Several major steps were taken in this project. The first was implementing the XPLOR-NIH as the molecular dynamic's engine rather than CNS, writing new support libraries, and translating the scripting interface into Python. Once the basic conversion was complete debugging ARIAXC became an important problem, and the RollinStone output critic program was written to access and compare constants, variables, and energies throughout all stages of a structure calculation in vanilla ARIA2 and ARIAXC. Once this debugging process was completed further debugging was undertaken using a strategy called ‘structure injection’ which allowed a very accurate comparison of energies between CNS and XPLOR-NIH and the discovery of further bugs. Final testing was carried out using complete structure calculation comparing ARIAXC against ARIA2, comparing convergence of the final stage models of ARIAXC & ARIA2 using quality metrics including Ramachandran plots.

Further work is still required as ARIAXC calculations don’t converge as quickly or produce the same quality of model as with the original ARIA2, scripting is currently in python2.8 and XPLOR-NIH has moved onto python-3, the Python scripts still call out to a significant amount of XPLOR scripts and water-refinement, metal ions, and symmetric multimers have not been tested.

Overall, the project succeeded, and complete structure calculations can be carried out with ARIAXC, and it opens pathways to more effective, efficient, and refined structure calculations.

References

1. Rieping, W. *et al.* ARIA2: Automated NOE assignment and data integration in NMR structure calculation. **23**, 381–382 (2007).

2. Brunger, A. T. *et al.* Crystallography & NMR System: A New Software Suite for Macromolecular Structure Determination. *Acta Cryst* **54**, 905–921 (1998).

3. Brunger, A. T. Version 1.2 of the Crystallography and NMR system. (2007) doi:10.1038/nprot.2007.406.

4. Schwieters, C. D., Kuszewski, J. J., Tjandra, N. & Clore, G. M. The Xplor-NIH NMR molecular structure determination package. *Journal of Magnetic Resonance* **160**, 65–73 (2003).

5. Schwieters, C. D., Kuszewski, J. J. & Clore, G. M. Using Xplor-NIH for NMR molecular structure determination. (2005) doi:10.1016/j.pnmrs.2005.10.001.

6. Güntert, P. Structure calculation of biological macromolecules from NMR data. *Q Rev Biophys* **31**, 145–237 (1998).

7. Bernstein, F. C. *et al.* The protein data bank: A computer-based archival file for macromolecular structures. *J Mol Biol* **112**, 535–542 (1977).

8. Harner, M. J., Frank, A. O. & Fesik, S. W. Fragment-based drug discovery using NMR spectroscopy. *J Biomol NMR* **56**, 65–75 (2013).

9. Horst, R., Farley, K. A., Kormos, B. L. & Withka, J. M. NMR spectroscopy: the swiss army knife of drug discovery. *J Biomol NMR* **74**, 509–519 (2020).

10. Chou, P. Y. & Fasman, G. D. Prediction of Protein Conformation. *Biochemistry* **13**, 222–245 (1974).

11. Jespersen, M. C., Peters, B., Nielsen, M. & Marcatili, P. BepiPred-2.0: improving sequence-based B-cell epitope prediction using conformational epitopes. *Nucleic Acids Res* **45**, W24–W29 (2017).

12. Varadi, M. *et al.* AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic Acids Res* **50**, D439–D444 (2022).

13. Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature 2021 596:7873* **596**, 583–589 (2021).

14. Davis, A. M., Teague, S. J. & Kleywegt, G. J. Application and limitations of x-ray crystallographic data in structure-based ligand and drug design. *Angewandte Chemie - International Edition* **42**, 2718–2736 (2003).

15. Thompson, A. L. & Watkin, D. J. X-ray crystallography and chirality: understanding the limitations. *Tetrahedron Asymmetry* **20**, 712–717 (2009).

16. Rupp, B. & Krishnan, V. V. Macromolecular Structure Determination: Comparison of X-ray Crystallography and NMR Spectroscopy The molecular structure of afamin View project Crystallogenesis View project Macromolecular Structure Determination: Comparison of

X-ray Crystallography and NMR Spectroscopy. (2012) doi:10.1002/9780470015902.a0002716.pub2.

17. Goodhew, P. J., Humphreys, J. & Humphreys, J. Electron Microscopy and Analysis. *Electron Microscopy and Analysis* (2000) doi:10.1201/9781482289343.

18. Bai, X. chen, McMullan, G. & Scheres, S. H. W. How cryo-EM is revolutionizing structural biology. *Trends Biochem Sci* **40**, 49–57 (2015).

19. Kucukelbir, A., Sigworth, F. J. & Tagare, H. D. Quantifying the local resolution of cryo-EM density maps. *Nature Methods 2013 11:1* **11**, 63–65 (2013).

20. Liu, Y., Huynh, D. T. & Yeates, T. O. A 3.8 Å resolution cryo-EM structure of a small protein bound to an imaging scaffold. *Nature Communications 2019 10:1* **10**, 1–7 (2019).

21. Harris, J. R. *et al.* Comparison of the decameric structure of peroxiredoxin-II by transmission electron microscopy and X-ray crystallography. *Biochimica et Biophysica Acta (BBA) - Protein Structure and Molecular Enzymology* **1547**, 221–234 (2001).

22. Bloch, F. Nuclear Induction. *Physical Review* **70**, 460 (1946).

23. Reif, B., Ashbrook, S. E., Emsley, L. & Hong, M. Solid-state NMR spectroscopy. *Nature Reviews Methods Primers 2021 1:1* **1**, 1–23 (2021).

24. Freeman, R. & Whiffen, D. H. Determination of the relative signs of proton spin coupling constants by double irradiation. *Mol Phys* **4**, 321–325 (1961).

25. Bystrov, V. F. Spin—spin coupling and the conformational states of peptide systems. *Prog Nucl Magn Reson Spectrosc* **10**, 41–82 (1976).

26. Williamson, M. P. Chapter 3 Applications of the NOE in Molecular Biology. *Annu Rep NMR Spectrosc* **65**, 77–109 (2009).

27. Ernst, R. R. & Anderson, W. A. Application of Fourier Transform Spectroscopy to Magnetic Resonance. *Review of Scientific Instruments* **37**, 93 (2004).

28. Wishart, D. S., Sykes, B. D. & Richards, F. M. The chemical shift index: a fast and simple method for the assignment of protein secondary structure through NMR spectroscopy. *Biochemistry* **31**, 1647–1651 (1992).

29. Mumenthaler, C., Güntert, P., Braun, W. & Wüthrich, K. Automated combined assignment of NOESY spectra and three-dimensional protein structure determination. *Journal of Biomolecular NMR 1997 10:4* **10**, 351–362 (1997).

30. Marion, D. & Wüthrich, K. APPLICATION OF PHASE SENSITIVE TWO-DIMENSIONAL CORRELATED SPECTROSCOPY (COSY) FOR MEASUREMENTS OF 1 H - 1 H SPIN-SPIN COUPLING CONSTANTS IN PROTEINS . 114–121 (1995) doi:10.1142/9789812795830_0013.

31. Bowden, G. J., Heseltine, T. & Prandolini, M. J. Double quantum and cross correlation peaks in COSY NMR spectra. *Chem Phys Lett* **233**, 639–643 (1995).

32. Clore, G. M., Gronenborn, A. M., Clore, G. M. & Gronenborn, A. M. APPLICATIONS OF THREE-AND FOUR-DIMENSIONAL HETERONUCLEAR NMR SPECTROSCOPY TO PROTEIN STRUCTURE DETERMINATION. *Prooress in NMR Spectroscopy* **23**, 43–92 (1991).

33. Overhauser, A. W. Polarization of Nuclei in Metals. *Physical Review* **92**, 411 (1953).

34. Aguilar, J. A., Colbourne, A. A., Cassani, J., Nilsson, M. & Morris, G. A. Decoupling Two-Dimensional NMR Spectroscopy in Both Dimensions: Pure Shift NOESY and COSY. *Angewandte Chemie International Edition* **51**, 6460–6463 (2012).

35. Féraud, B., Govaerts, B., Verleysen, M. & de Tullio, P. Statistical treatment of 2D NMR COSY spectra in metabolomics: data preparation, clustering-based evaluation of the Metabolomic Informative Content and comparison with 1H-NMR. *Metabolomics* **11**, 1756–1768 (2015).

36. Xi, Y., Deropp, J. S., Viant, M. R., Woodruff, D. L. & Yu, P. Automated screening for metabolites in complex mixtures using 2D COSY NMR spectroscopy. doi:10.1007/s11306-006-0036-0.

37. Bingol, K., Zhang, F., Bruschweiler-Li, L. & Brüschweiler, R. Quantitative analysis of metabolic mixtures by two-dimensional 13C constant-time TOCSY NMR spectroscopy. *Anal Chem* **85**, 6414–6420 (2013).

38. Neuhaus, D. Nuclear Overhauser Effect. *Encyclopedia of Magnetic Resonance* (2011) doi:10.1002/9780470034590.EMRSTM0350.PUB2.

39. Madrid, M., Llinás, E. & Llinás, M. Model-independent refinement of interproton distances generated from 1H NMR overhauser intensities. *Journal of Magnetic Resonance (1969)* **93**, 329–346 (1991).

40. Nilges, M. & O'Donoghue, S. I. Ambiguous NOEs and automated NOE assignment. *Prog Nucl Magn Reson Spectrosc* **32**, 107–139 (1998).

41. Sattler, M., Schleucher, J. & Griesinger, C. Heteronuclear multidimensional NMR experiments for the structure determination of proteins in solution employing pulsed field gradients.

42. Bodenhausen, G. & Ruben, D. J. Natural abundance nitrogen-15 NMR by enhanced heteronuclear spectroscopy. *Chem Phys Lett* **69**, 185–189 (1980).

43. Morris, G. A. Sensitivity enhancement in nitrogen-15 NMR: polarization transfer using the INEPT pulse sequence. *J Am Chem Soc* **102**, 428–429 (2002).

44. Piantini, U., Sørensen, O. W. & Ernst, R. R. Multiple Quantum Filters for Elucidating NMR Coupling Networks. *J Am Chem Soc* **104**, 6800–6801 (1982).

45. Städeli, W., Bigler, P. & Von Philipsborn, W. 15N,1H coupling constants in pyridines and pyrimidines An application of the INEPT pulse sequence. *Organic Magnetic Resonance* **16**, 170–172 (1981).

46. Glemarec, C., Remaud, G. & Chattopadhyaya, J. Distinction of 15N resonances based on differences in small 15N, 1H coupling constants. An application of 1H-decoupled INEPT experiments. *Magnetic Resonance in Chemistry* **26**, 307–310 (1988).

47. Simon, B., Zanie, K. & Sattler, M. A TROSY relayed HCCH-COSY experiment for correlating adenine H2/H8 resonances in uniformly 13C-labeled RNA molecules. *Journal of Biomolecular NMR 2001 20:2* **20**, 173–176 (2001).

48. Kay, L. E., Xu, G. Y., Singer, A. U., Muhandiram, D. R. & Formankay, J. D. A Gradient-Enhanced HCCH-TOCSY Experiment for Recording Side-Chain 1H and 13C Correlations in H2O Samples of Proteins. *J Magn Reson B* **101**, 333–337 (1993).

49. Gehring, K. & Ekiel, I. H(C)CH-COSY and (H)CCH-COSY Experiments for13C-Labeled Proteins in H2O Solution. *Journal of Magnetic Resonance* **135**, 185–193 (1998).

50. Skrynnikov, N. R., Dahlquist, F. W. & Kay, L. E. Reconstructing NMR spectra of 'invisible' excited protein states using HSQC and HMQC experiments. *J Am Chem Soc* **124**, 12352–12360 (2002).

51. Verardi, R., Traaseth, N. J., Masterson, L. R., Vostrikov, V. V. & Veglia, G. Isotope Labeling for Solution and Solid-State NMR Spectroscopy of Membrane Proteins. *Adv Exp Med Biol* **992**, 35 (2012).

52. Diercks, T., Coles, M. & Kessler, H. An efficient strategy for assignment of cross-peaks in 3D heteronuclear NOESY experiments. *Journal of Biomolecular NMR 1999 15:2* **15**, 177–180 (1999).

53. Bingol, K., Bruschweiler-Li, L., Li, D. W. & Bruüschweiler, R. Customized metabolomics database for the analysis of NMR 1H-1H TOCSY and 13C-1H HSQC-TOCSY spectra of complex mixtures. *Anal Chem* **86**, 5494–5501 (2014).

54. Martin, G. E. & Williams, A. J. Applications of 1H-15N long-range heteronuclear shift correlation and 15N NMR in Alkaloid Chemistry. *Annu Rep NMR Spectrosc* **84**, 1–76 (2015).

55. Nilges, M. Calculation of protein structures with ambiguous distance restraints. Automated assignment of ambiguous NOE crosspeaks and disulphide connectivities. *J Mol Biol* **245**, 645–660 (1995).

56. Wishart, D. S. *et al.* 1H, 13C and 15N chemical shift referencing in biomolecular NMR. *Journal of Biomolecular NMR 1995 6:2* **6**, 135–140 (1995).

57. Nerli, S., McShan, A. C. & Sgourakis, N. G. Chemical shift-based methods in NMR structure determination. *Prog Nucl Magn Reson Spectrosc* **106–107**, 1 (2018).

58. Arthanari, H. *et al.* Assessment of the molecular dynamics structure of DNA in solution based on calculated and observed NMR NOESY volumes and dihedral angles from scalar coupling constants. *Biopolymers* **68**, 3–15 (2003).

59. Prestegard, J. H., Bougault, C. M. & Kishore, A. I. Residual Dipolar Couplings in Structure Determination of Biomolecules. *Chem Rev* **104**, 3519–3540 (2004).

60. Garrett, D. S., Powers, R., Gronenborn, A. M. & Clore, G. M. A Common Sense Approach to Peak Picking in Two-, Three-, and Four-Dimensional Spectra Using Automatic Computer Analysis of Contour Diagrams. *Robert Powers Publications* (1991).

61. Nilges, M. Ambiguous distance data in the calculation of NMR structures. *Fold Des* **2**, S53–S57 (1997).

62. Güntert, P. Automated NMR structure calculation with CYANA. *Methods Mol Biol* **278**, 353–378 (2004).

63. Nilges, M. Calculation of Protein Structures with Ambiguous Distance Restraints. Automated Assignment of Ambiguous NOE Crosspeaks and Disulphide Connectivities. *J Mol Biol* **245**, 645–660 (1995).

64. Linge, J. P., O'Donoghue, S. I. & Nilges, M. Automated Assignment of Ambiguous Nuclear Overhauser Effects with ARIA. *Methods Enzymol* **339**, 71–90 (2001).

65. Güntert, P., Braun, W. & Wüthrich, K. Efficient computation of three-dimensional protein structures in solution from nuclear magnetic resonance data using the program DIANA and the supporting programs CALIBA, HABAS and GLOMSA. *J Mol Biol* **217**, 517–530 (1991).

66. Güntert, P. & Wüthrich, K. Improved efficiency of protein structure calculations from NMR data using the program DIANA with redundant dihedral angle constraints. 448–457 (1995) doi:10.1142/9789812795830_0038.

67. Berndt, K. D., Güntert, P. & Wüthrich, K. Conformational sampling by NMR solution structures calculated with the program DIANA evaluated by comparison with long-time molecular dynamics calculations in explicit water. *Proteins: Structure, Function, and Bioinformatics* **24**, 304–313 (1995).

68. Güntert, P., Mumenthaler, C. & Wüthrich, K. Torsion angle dynamics for NMR structure calculation with the new program Dyana. *J Mol Biol* **273**, 283–298 (1997).

69. Herrmann, T., Güntert, P. & Wüthrich, K. Protein NMR Structure Determination with Automated NOE Assignment Using the New Software CANDID and the Torsion Angle Dynamics Algorithm DYANA. *J Mol Biol* **319**, 209–227 (2002).

70. Bartels, C., Güntert, P., Billeter, M. & Wüthrich, K. GARANT-a general algorithm for resonance assignment of multidimensional nuclear magnetic resonance spectra. *Journal of Computational Chemistry* **18**, 139–149 (1998).

71. Schmidt, E. & Güntert, P. A new algorithm for reliable and general NMR resonance assignment. *J Am Chem Soc* **134**, 12817–12829 (2012).

72. Buchner, L. & Güntert, P. Systematic evaluation of combined automated NOE assignment and structure calculation with CYANA. *J Biomol NMR* **62**, 81–95 (2015).

73. Güntert, P. & Buchner, L. Combined automated NOE assignment and structure calculation with CYANA. *J Biomol NMR* **62**, 453–471 (2015).

74. Bowers, P. M., Strauss, C. E. M. & Baker, D. De novo protein structure determination using sparse NMR data. *Journal of Biomolecular NMR 2000 18:4* **18**, 311–318 (2000).

75. Güntert, P. Automated NMR structure calculation with CYANA. *Methods Mol Biol* **278**, 353–378 (2004).

76. Güntert, P. & Buchner, L. Combined automated NOE assignment and structure calculation with CYANA. *J Biomol NMR* **62**, 453–471 (2015).

77. Güntert, P. Automated protein structure determination from NMR data. *Advances in Biomedical Spectroscopy* **3**, 338–365 (2011).

78. Schwieters, C. D., Bermejo, G. A. & Clore, G. M. Xplor-NIH for molecular structure determination from NMR and other data sources. *Protein Science* **27**, 26–40 (2018).

79. Grishaev, A. & Llinás, M. CLOUDS, a protocol for deriving a molecular proton density via NMR. *Proceedings of the National Academy of Sciences* **99**, 6707–6712 (2002).

80. Güntert, P. Automated structure determination from NMR spectra. *European Biophysics Journal* **38**, 129–143 (2009).

81. Grishaev, A. & Llinás, M. Protein structure elucidation from NMR proton densities. *Proceedings of the National Academy of Sciences* **99**, 6713–6718 (2002).

82. Prestegard, J. H., Mayer, K. L., Valafar, H. & Benison, G. C. Determination of Protein Backbone Structures from Residual Dipolar Couplings. *Methods Enzymol* **394**, 175–209 (2005).

83. Frank Delaglio, Georg Kontaxis, and & Bax*, A. Protein Structure Determination Using Molecular Fragment Replacement and NMR Dipolar Couplings. *J Am Chem Soc* **122**, 2142–2143 (2000).

84. Kontaxis, G., Delaglio, F. & Bax, A. Molecular Fragment Replacement Approach to Protein Structure Determination by Chemical Shift and Dipolar Homology Database Mining. *Methods Enzymol* **394**, 42–78 (2005).

85. Güntert, P. Automated structure determination from NMR spectra. *European Biophysics Journal* **38**, 129–143 (2009).

86. Cavalli, A., Salvatella, X., Dobson, C. M. & Vendruscolo, M. Protein structure determination from NMR chemical shifts. *Proceedings of the National Academy of Sciences* **104**, 9615–9620 (2007).

87. Nerli, S., McShan, A. C. & Sgourakis, N. G. Chemical shift-based methods in NMR structure determination. *Prog Nucl Magn Reson Spectrosc* **106–107**, 1 (2018).

88. Shen, Y. *et al.* Consistent blind protein structure generation from NMR chemical shift data. *Proceedings of the National Academy of Sciences* **105**, 4685–4690 (2008).

89. Leaver-Fay, A. *et al.* ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Methods Enzymol* **487**, 545–574 (2011).

90. Alford, R. F. *et al.* The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J Chem Theory Comput* **13**, 3031–3048 (2017).

91. Nerli, S. & Sgourakis, N. G. CS-ROSETTA. *Methods Enzymol* **614**, 321 (2019).

92. Nerli, S. & Sgourakis, N. G. CS-ROSETTA. *Methods Enzymol* **614**, 321 (2019).

93. Lange, O. F. & Baker, D. Resolution-adapted recombination of structural features significantly improves sampling in restraint-guided structure calculation. *Proteins* **80**, 884–895 (2012).

94. Lange, O. F. Automatic NOESY assignment in CS-RASREC-Rosetta. *J Biomol NMR* **59**, 147–159 (2014).

95. Bradley, P., Misura, K. M. S. & Baker, D. Toward high-resolution de novo structure prediction for small proteins. *Science* **309**, 1868–1871 (2005).

96. Raman, S. *et al.* Accurate automated protein NMR structure determination using unassigned NOESY data. *J Am Chem Soc* **132**, 202–207 (2010).

97. Nilges, M. Ambiguous distance data in the calculation of NMR structures. *Fold Des* **2**, S53–S57 (1997).

98. Linge, J. P., O'Donoghue, S. I. & Nilges, M. Automated assignment of ambiguous nuclear overhauser effects with ARIA. in *Methods in Enzymology* vol. 339 71–90 (Academic Press, 2001).

99. Fossi, M. *et al.* Influence of chemical shift tolerances on NMR structure calculations using ARIA protocols for assigning NOE data. *J Biomol NMR* **31**, 21–34 (2005).

100. Stein, E. G., Rice, L. M. & Brünger, A. T. Torsion-Angle Molecular Dynamics as a New Efficient Tool for NMR Structure Calculation. *Journal of Magnetic Resonance* **124**, 154–164 (1997).

101. Bardiaux, B., Malliavin, T. & Nilges, M. ARIA for solution and solid-state NMR. *Methods in Molecular Biology* **831**, 453–483 (2012).

102. Mareuil, F., Malliavin, T. E., Nilges, M. & Bardiaux, B. Improved reliability, accuracy and quality in automated NMR structure calculation with ARIA. *J Biomol NMR* **62**, 425–438 (2015).

103. Linge, J. P., Williams, M. A., Spronk, C. A. E. M., Bonvin, A. M. J. J. & Nilges, M. Refinement of protein structures in explicit solvent. *Proteins: Structure, Function and Genetics* **50**, 496–506 (2003).

104. Nilges, M. & O'Donoghue, S. I. Ambiguous NOEs and automated NOE assignment. *Prog Nucl Magn Reson Spectrosc* **32**, 107–139 (1998).

105. Habeck, M., Rieping, W., Linge, J. P. & Nilges, M. NOE assignment with ARIA 2.0: the nuts and bolts. *Methods Mol Biol* **278**, 379–402 (2004).

106. Krieger, E. & Vriend, G. YASARA View - molecular graphics for all devices - from smartphones to workstations. *Bioinformatics* **30**, 2981–2982 (2014).

107. Krieger, E. *et al.* Improving physical realism, stereochemistry, and side-chain accuracy in homology modeling: Four approaches that performed well in CASP8. *Proteins: Structure, Function and Bioinformatics* vol. 77 114–122 Preprint at https://doi.org/10.1002/prot.22570 (2009).

108. Spronk, C. A. E. M. *et al.* Structure and dynamics of helicobacter pylori nickel-chaperone hypa: An integrated approach using nmr spectroscopy, functional assays and computational tools. *Journal of Biological Inorganic Chemistry* **23**, 1309–1330 (2018).

109. Vriend, G. WHAT IF: A molecular modeling and drug design program. *J Mol Graph* **8**, 52–56 (1990).

110. Laskowski, R. A., MacArthur, M. W., Moss, D. S. & Thornton, J. M. PROCHECK: a program to check the stereochemical quality of protein structures. *J Appl Crystallogr* **26**, 283–291 (1993).

111. Sippl, M. J. Recognition of errors in three-dimensional structures of proteins. *Proteins: Structure, Function, and Bioinformatics* **17**, 355–362 (1993).

112. Doreleijers, J. F., Raves, M. L., Rullmann, T. & Kaptein, R. Completeness of NOEs in protein structures: A statistical analysis of NMR data. *J Biomol NMR* **14**, 123–132 (1999).

113. Linge, J. P., O'Donoghue, S. I. & Nilges, M. Automated assignment of ambiguous nuclear overhauser effects with ARIA. in *Methods in Enzymology* vol. 339 71–90 (Academic Press, 2001).