relax

Version 1.3.7

A program for NMR relaxation data analysis

January 10, 2011

Contents

List of Figures

Abbreviations

AIC Akaike's Information Criteria

AICc small sample size corrected AIC

BIC Bayesian Information Criteria

$C(\tau)$ correlation function

χ^2 chi-squared function

CSA chemical shift anisotropy

$\mathfrak{D}$ the set of diffusion tensor parameters

$\mathfrak{D}_{\parallel}$ the eigenvalue of the spheroid diffusion tensor corresponding to the unique axis of the tensor

$\mathfrak{D}_{\perp}$ the eigenvalue of the spheroid diffusion tensor corresponding to the two axes perpendicular to the unique axis

$\mathfrak{D}_a$ the anisotropic component of the Brownian rotational diffusion tensor

$\mathfrak{D}_{iso}$ the isotropic component of the Brownian rotational diffusion tensor

$\mathfrak{D}_r$ the rhombic component of the Brownian rotational diffusion tensor

$\mathfrak{D}_{ratio}$ the ratio of $\mathfrak{D}_{\parallel}$ to $\mathfrak{D}_{\perp}$

$\mathfrak{D}_x$ the eigenvalue of the Brownian rotational diffusion tensor in which the corresponding eigenvector defines the x-axis of the tensor

$\mathfrak{D}_y$ the eigenvalue of the Brownian rotational diffusion tensor in which the corresponding eigenvector defines the y-axis of the tensor

$\mathfrak{D}_z$ the eigenvalue of the Brownian rotational diffusion tensor in which the corresponding eigenvector defines the z-axis of the tensor

ϵ_i elimination value

$J(\omega)$ spectral density function

NOE nuclear Overhauser effect

pdf probability distribution function

r bond length

R_1 spin-lattice relaxation rate

R_2 spin-spin relaxation rate

R_{ex} chemical exchange relaxation rate

S^2, S_f^2, **and** S_s^2 model-free generalised order parameters

τ_e, τ_f, **and** τ_s model-free effective internal correlation times

τ_m global rotational correlation time

Chapter 1

Introduction

The program relax is designed for the study of the dynamics of proteins or other macromolecules though the analysis of NMR relaxation data. It is a community driven project created by NMR spectroscopists for NMR spectroscopists. It supports exponential curve fitting for the calculation of the R_1 and R_2 relaxation rates, calculation of the NOE, reduced spectral density mapping, and the Lipari and Szabo model-free analysis.

The aim of relax is to provide a seamless and extremely flexible environment able to accept input in any format produced by other NMR software, able to faultlessly create input files, control, and read output from various programs including Modelfree and Dasha, output results in many formats, and visualise the data by controlling programs such as Grace, OpenDX, MOLMOL, and PyMOL. All data analysis tools from optimisation to model selection to Monte Carlo simulations are inbuilt into relax. Therefore the use of additional programs is optional.

The flexibility of relax arises from the choice of either relax's scripting capabilities or its Python prompt interface. Extremely complex scripts can be created from simple building blocks to fully automate data analysis. A number of sample scripts have been provided to help understand script construction. In addition, any of Python's powerful features or functions can be incorporated as the script is executed as an arbitrary Python source file within relax's environment. The modules of relax can also used as a vast library of dynamics related functions by your own software.

relax is free software (free as in freedom) which is licenced under the GNU General Public Licence (GPL). You are free to copy, modify, or redistribute relax under the terms of the GPL.

1.1 Program features

1.1.1 Literature

The primary references for the program relax are d'Auvergne and Gooley (2008a) and d'Auvergne and Gooley (2008b).

Other literature related to the improved model-free analysis used within relax, which can nevertheless be applied to other techniques such as SRLS, include model-free model selection (d'Auvergne and Gooley, 2003; Chen et al., 2004), model-free model elimination (d'Auvergne and Gooley, 2006), the theory (d'Auvergne and Gooley, 2007) behind the new model-free optimisation protocol (d'Auvergne and Gooley, 2008b), and the hybridisation of different models (Horne et al., 2007; d'Auvergne and Gooley, 2008b). Most of these details can be found in the PhD thesis of d'Auvergne (2006).

1.1.2 Supported NMR theories

The following relaxation data analysis techniques are currently supported by relax:

- Model-free analysis (Lipari and Szabo, 1982a,b; Clore et al., 1990).
- Reduced spectral density mapping (Farrow et al., 1995; Lefevre et al., 1996).
- Exponential curve fitting (to find the R_1 and R_2 relaxation rates).
- Steady-state NOE calculation.

The future

At some time in the future the following techniques are planned to be implemented within relax:

- Relaxation dispersion.
- SRLS – Slowly relaxing local structure (Tugarinov et al., 2001).

Because relax is free software, if you would like to contribute addition features, functions, or modules which you have written for your own publications for the benefit of the field, almost anything relating to molecular dynamics may be accepted. Please see the Open Source chapter for more details.

1.1.3 Data analysis tools

The following tools are implemented as modular components to be used by any data analysis technique:

- Numerous high-precision optimisation algorithms.
- Model selection (d'Auvergne and Gooley, 2003; Chen et al., 2004):
  - Akaike's Information Criteria (AIC).
  - Small sample size corrected AIC (AICc).
  - Bayesian or Schwarz Information Criteria (BIC).

  - Bootstrap model selection.
  - Single-item-out cross-validation (CV).
  - Hypothesis testing ANOVA model selection (only the model-free specific technique of Mandel et al. (1995) is supported).

- Monte Carlo simulations (error analysis for all data analysis techniques).
- Model elimination – the removal of failed models prior to model selection (d'Auvergne and Gooley, 2006).

1.1.4 Data visualisation

The results of an analysis, or any data input into relax, can be visualised using a number of programs:

MOLMOL 1D data can be mapped onto a structure either by the creation of MOLMOL macros or by direct control of the program.

PyMOL 3D objects such as the diffusion tensor representation can be displayed with the structure.

Grace any 2D data can be plotted.

OpenDX The chi-squared space of models with three parameters can be mapped and 3D images of the space produced.

1.1.5 Interfacing with other programs

relax can create the input files, execute in-line, and then read the output of the following programs. These programs can be used as optimisation engines replacing the minimisation algorithms built into relax:

- Dasha (model-free analysis).
- Modelfree (model-free analysis).

1.1.6 The user interfaces (UI)

relax can be used through the following UIs:

The prompt this is the primary interface of relax. Rather than reinventing a new command language, relax's interface is the powerful Python prompt. This gives the power user full access to a proven programming language.

Scripting this provides a more powerful and flexible framework for controlling the program. The script will be executed as Python code enabling advanced programming for automating data analysis. All the features available within the prompt environment are accessible to the script.

1.2 How to use relax

1.2.1 The prompt

The primary interface of relax is the prompt. After typing `'relax'` within a terminal you will be presented with

```
relax>
```

This is the Python prompt which has been tailored specifically for relax. You will hence have full access, if desired, to the power of the Python programing language to manipulate your data. You can for instance type

```
relax> print "Hello World"
```

the result being

```
relax> print "Hello World"
Hello World
relax>
```

Or using relax as a calculator

```
relax> (1.0 + (2 * 3))/10
0.69999999999999996
relax>
```

1.2.2 Python

relax has been designed such that knowledge about Python is not required to be able to fully use the program. A few basics though will aid in understanding relax.

A number of simple programming axioms includes that of strings, integers, floating point numbers, and lists. A string is text and within Python (as well as relax) this is delimited by either single or double quotes. An integer is a number with no decimal point whereas a float is a number with a decimal point. A list in Python (called an array in other languages) is a list of anything separated by commas and delimited by square brackets, an example is [0, 1, 2, 'a', 1.2143235].

Probably the most important detail is that functions in Python require brackets around their arguments. For example

```
relax> minimise()
```

will commence minimisation however

```
relax> minimise
```

will do nothing.

The arguments to a function are simply a comma separated list within the brackets of the function. For example to save the program's current state type

```
relax> state.save('save', force=True)
```

Two types of arguments exist in Python – standard arguments and keyword arguments. The majority of arguments you will encounter within relax are keyword arguments however you may, in rare cases, encounter a non-keyword argument. For these standard arguments just type the values in, although they must be in the correct order. Keyword arguments consist of two parts – the key and the value. For example the key may be `file` and the value you would like to supply is `'R1.out'`. Various methods exist for supplying this argument. Firstly you could simply type `'R1.out'` into the correct position in the argument list. Secondly you can type `file='R1.out'`. The power of this second option is that argument order is unimportant. Therefore if you would like to change the default value of the very last argument, you don't have to supply values for all other arguments. The only catch is that standard arguments must come before the keyword arguments.

1.2.3 User functions

For standard data analysis a large number of specially tailored functions called 'user functions' have been implemented. These are accessible from the relax prompt by simply typing the name of the function. An example is `'help()'`. An alphabetical listing of all accessible user functions together with full descriptions is presented later in this manual.

A few special objects which are available within the prompt are not actually functions. These objects do not require brackets at their end for them to function. For example to exit relax type

```
relax> exit
```

Another special object is that of the function class. This object is simply a container which holds a number of user functions. You can access the user function within the class by typing the name of the class, then a dot `'.'`, followed by the name of the user function. An example is the user function for reading relaxation data out of a file and loading the data into relax. The function is called `'read'` and the class is called `'relax_data'`. To execute the function, type something like

```
relax> relax_data.read('R1', '600', 600.0 * 1e6, 'r1.600.out')
```

On first usage the relax prompt can be quite daunting. Two features exist to increase the usability of the prompt – the help system and tab completion.

1.2.4 The help system

For assistance in using a function simply type

```
help(function)
```

In addition to functions if

```
help(object)
```

is typed the help for the python object is returned. This system is similar to the help function built into the python interpreter, which has been renamed to `help_python`, with the interactive component removed. For the standard interactive python help system type

```
help_python()
```

1.2.5 Tab completion

Tab completion is implemented to prevent insanity as the function names can be quite long – a deliberate feature to improve usability. The behaviour of the tab completion is very similar to that of the bash prompt.

Not only is tab completion useful for preventing RSI but it can also be used for listing all available functions. To begin with if you hit the [TAB] key without typing any text all available functions will be listed (along with function classes and other python objects). This extends to the exploration of user functions within a function class. For example to list the user functions within the function class `'model_free'` type

```
relax> model_free.
```

The dot character at the end is essential. After hitting the [TAB] key you should see something like

```
relax> model_free.
model_free.__class__
model_free.__doc__
model_free.__init__
model_free.__module__
model_free.__relax__
model_free.__relax_help__
model_free.create_model
model_free.delete
model_free.remove_tm
model_free.select_model
relax> model_free.
```

All the objects beginning with an underscore are "hidden", they contain information about the function class and should be ignored. From the listing the user functions `'copy'`, `'create_model'`, `'delete'`, `'remove_tm'`, and `'select_model'` contained within `'model_free'` are all visible.

1.2.6 The data pipe

Within relax all user functions operate on data stored within the current data pipe. This pipe stores data is input, processed, or output as user functions are called. There are different types of data pipe for different analyses, e.g. a reduced spectral density mapping pipe, a model-free pipe, an exponential curve-fitting pipe, etc. Multiple data pipes can be created within relax and various operations performed in sequence on these pipes. This is useful for operations such as model selection whereby the function `'model_selection'` can operate on a number of pipes corresponding to different models and then assign the results to a newly created pipe. When running relax you choose which pipe you are currently in by using the `'pipe.switch'` user function to jump between pipes.

The flow of data through relax can be thought of as travelling through these pipes. User functions exist to transfer data between these pipes and other functions combine data from multiple pipes into one or vice versa. The simplest invocation of relax would be the

creation of a single data pipe and with the data being processed as it is passing through this pipe.

The primary method for creating a data pipe is through the user function `'pipe.create'`. For example

```
relax> pipe.create('m1', 'mf')
```

will create a model-free data pipe labelled `'m1'`. The following is a table of all the types which can be assigned to a data pipe.

Data pipe type	Description
`'jw'`	Reduced spectral density mapping
`'mf'`	Model-free data analysis
`'N-state'`	N-state model of domain motions
`'noe'`	Steady state NOE calculation
`'relax_fit'`	Relaxation curve-fitting
`'srls'`	SRLS analysis

Currently the NOE calculation, relaxation curve-fitting, model-free analysis, and reduced spectral density mapping features of relax are implemented (if this documentation is out of date then you may be able to do a lot more).

1.2.7 Scripting

What ever is done within the prompt is also accessible through scripting. Just type your commands into a text file and then at the terminal type

```
$ relax your_script
```

An example of a simple script which will minimise the model-free model 'm4' after loading six relaxation data sets is

```
# Create the data pipe.
name = 'm4'
pipe.create(name, 'mf')

# Nuclei type
nuclei('N')

# Load the sequence.
sequence.read('noe.500.out')

# Load the relaxation data.
relax_data.read('R1', '600', 600.0 * 1e6, 'r1.600.out')
relax_data.read('R2', '600', 600.0 * 1e6, 'r2.600.out')
relax_data.read('NOE', '600', 600.0 * 1e6, 'noe.600.out')
relax_data.read('R1', '500', 500.0 * 1e6, 'r1.500.out')
relax_data.read('R2', '500', 500.0 * 1e6, 'r2.500.out')
```

```
relax_data.read('NOE', '500', 500.0 * 1e6, 'noe.500.out')

# Setup other values.
diffusion_tensor.init((2e-8, 1.3, 60, 290), param_types=1, axial_type='prolate',
fixed=True)
value.set(1.02 * 1e-10, 'bond_length')
value.set(-160 * 1e-6, 'csa')
value.set('15N', 'heteronucleus')
value.set('1H', 'proton')

# Select a preset model-free model.
model_free.select_model(model=name)

# Grid search.
grid_search(inc=11)

# Minimise.
minimise('newton')

# Finish.
results.write(file='results', force=True)
state.save('save', force=True)
```

Scripting is much more powerful than the prompt as advanced Python programming can be employed (see the file 'full_analysis.py' in the 'sample_scripts' directory for an example).

1.2.8 Sample scripts

A few sample scripts have been provided in the directory 'sample_scripts'. These can be copied and modified for different types of data analysis.

1.2.9 The test suite

To test that the program functions correctly, relax possesses an inbuilt test suite. The suite is a collection of simple tests which execute or probe different parts of the program checking that the software runs without problem. The test suite is executed by running relax using the command `'relax --test-suite'`.

1.2.10 The GUI

relax has been designed primarily for scripting and, as such, no graphical user interface (GUI) currently exists. The internal structure of the program has been specifically designed so any type of control mechanism can be easily added, including a GUI, therefore in the future one may be written. A GUI will, however, detract from the power and flexibility inherent in the control by scripting.

1.2.11 Access to the internals of relax

To enable advanced Python scripting and control many parts of relax have been designed in an object oriented fashion. If you would like to play with internals of the program the entirety of relax is accessible by importation. For example all data is contained within the object called the relax data store which, to be able to access it, needs be imported by typing:

```
relax> from data import Data as relax_data_store
```

This is a dictionary type object which contains the multiple data pipes. All of relax's packages, modules, functions, and classes are also accessible by import statements. For example to create a rotation matrix from three Euler angles in the z-y-z notation, type:

```
relax> alpha = 0.1342
relax> beta = 1.0134
relax> gamma = 2.4747
relax> from maths_fns.rotation_matrix import R_euler_zyz
relax> from numpy import float64, zeros
relax> R = zeros((3,3), float64)
relax> R_euler_zyz(R, alpha, beta, gamma)
relax> R
```

1.3 Usage of the name relax

The program relax is so relaxed that the first letter should always be in lower case!

Chapter 2

Installation instructions

2.1 Dependencies

The following packages need to be installed before using relax:

Python: Version 2.4 or higher (although any 2.x version may work).

Numeric: Version 21 or higher.

ScientificPython: Version 2.2 or higher.

Optik: Version 1.4 or higher. This is only needed if running python <= 2.2.

Older versions of these packages may work, use them at your own risk. If, for older dependency versions, errors do occur please submit a bug report to the bug tracker at https://gna.org/bugs/?group=relax. That way a solution may be created for the next relax release.

2.2 Installation

2.2.1 The precompiled verses source distribution

Two types of software packages are available for download – the precompiled and source distribution. Currently only relaxation curve-fitting requires compilation to function and all other features of relax will be fully functional without compilation. If relaxation curve-fitting is required but no precompiled version of relax exists for your operating system or architecture then, if a C compiler is present, the C code can be compiled into the shared objects files `*.so` which are loaded as modules into relax. To build these modules the Sconstruct system from http://scons.org/ is required. This software only depends on Python which is essential for running relax anyway. Once Sconstruct is installed type

```
$ scons
```

in the base directory where relax has been installed and the C modules should, hopefully, compile without any problems. Otherwise please submit a bug report to the bug tracker at https://gna.org/bugs/?group=relax.

2.2.2 Installation on GNU/Linux

To install the program relax on a GNU/Linux system download either the precompiled distribution labelled `relax-x.x.x.GNU-Linux.`*`arch`*`.tar.bz2` matching your machine architecture or the source distribution `relax-x.x.x.src.tar.bz2`. A number of installation methods are possible. The simplest way is to switch to the user 'root', unpack and decompress the archive within the `/usr/local` directory by typing, for instance

```
$ tar jxvf relax-x.x.x.GNU-Linux.i686.tar.bz2
```

then create a symbolic link in `/usr/local/bin` by moving to that directory and typing

```
$ ln -s ../relax/relax .
```

and finally running relax to create the byte-compiled Python `*.pyc` files to speed up the start time of relax by typing

```
$ relax --test
```

Alternatively if the Sconstruct system is installed typing

```
$ scons install
```

in the relax base directory will create a directory in `/usr/local/` called `relax`, copy all the uncompressed and untarred files into this directory, create a symbolic link in `/usr/local/bin` to the file `/usr/local/relax/relax`, and then finally run relax to create the byte-compiled Python `*.pyc`. To change the installation path to a non-standard location the Sconstruct script `sconstruct` in the base relax directory should be modified by changing the variable `INSTALL_PATH` to point to the desired location.

2.2.3 Installation on MS Windows

In addition to the above dependencies, running relax on MS Windows requires a number of additional programs. These include:

pyreadline: Version 1.3 or higher (download).

ctypes: The pyreadline package requires ctypes (download).

To install, simply download the pre-compiled binary distribution `relax-x.x.x.Win32.zip` or the source distribution `relax-x.x.x.src.zip` and extract the files to `C:\Program Files\relax-x.x.x`. Then add this directory to the system environment path (in Windows XP, right click on 'My Computer', go to 'Properties', click on the 'Advanced' tab, and click on the 'Envirnment Variables' button. Then double click on the 'Path' system variable and add the text ";C:\Program Files\relax-x.x.x" to the end of variable value field. The Python installation must also be located on the path (add the text ";C:\Program

Files\Python24", changing the text to point to the correct directory, to the field). To run the program from any directory inside the Windows command prompt (or dos prompt) type:

```
C:\> relax
```

2.2.4 Installation on Mac OS X

Please write me if you know how to do this!

2.2.5 Installation on your OS

Please write me if you know how to do this!

2.2.6 Running a non-compiled version

Compilation of the C code is not essential for running relax, however certain features of the program will be disabled. Currently only the exponential curve-fitting for determining the R_1 and R_2 relaxation rates requires compilation. To run relax without compilation install the dependencies detailed above, download the source distribution which should be named `relax-x.x.x.src.tar.bz2`, extract the files, and then run the file called `relax` in the base directory.

2.3 Optional programs

The following is a list of programs which can be used by relax although they are not essential for normal use.

2.3.1 Grace

Grace is a program for plotting two dimensional data sets in a professional looking manner. It is used to visualise parameter values. It can be downloaded from http://plasma-gate.weizmann.ac.il/Grace/.

2.3.2 OpenDX

Version 4.1.3 or compatible. OpenDX is used for viewing the output of the space mapping function and is executed by passing the command `dx` to the command line with various options. The program is designed for visualising multidimensional data and can be found at http://www.opendx.org/.

2.3.3 Molmol

Molmol is used for viewing the PDB structures loaded into the program and to display parameter values mapped onto the structure.

2.3.4 PyMOL

PDB structures can also be viewed using PyMOL. Although the mapping of parameter values onto the structure is not yet supported, this program can be used to display geometric objects generated by relax for representing physical concepts such as the diffusion tensor.

2.3.5 Dasha

Dasha is a program used for model-free analysis of NMR relaxation data. It can be used as an optimisation engine to replace the minimisation algorithms implemented within relax.

2.3.6 Modelfree4

Art Palmer's Modelfree4 program is also designed for model-free analysis and can be used as an optimisation engine to replace relax's high precision minimisation algorithms.

Chapter 3

Open source infrastructure

3.1 The relax web sites

The main web site for relax is http://nmr-relax.com. From these pages general information about the program, links to the latest documentation, links to the most current software releases, and information about the mailing lists are available. There are also Google search capabilities built into the pages for searching both the HTML version of the manual and the archives of the mailing lists.

The relax web site is hosted by the Gna! project (https://gna.org/) which is described as "a central point for development, distribution and maintenance of Libre Software (Free Software) projects". relax is a registered Gna! project and its primary Gna! web site is https://gna.org/projects/relax. This site contains many more technical details than the main web site.

3.2 The mailing lists

A number of mailing lists have been created covering different aspects of relax. These include the announcement list, the relax users list, the relax development list, and the relax committers list.

3.2.1 relax-announce

The relax announcement list "relax-announce at gna.org" is reserved for important announcements about the program including the release of new program versions. The amount of traffic on this list is relatively low. If you would like to receive information about relax you can subscribe to the list by vising the information page at https://mail.gna.org/listinfo/relax-announce/. Previous announcements can be viewed at https://mail.gna.org/public/relax-announce/.

3.2.2 relax-users

If you would like to ask questions about relax, discuss certain features, receive help, or to communicate on any other subject related to relax the mailing list "relax-users at gna.org" is the place to post your message. To subscribe to the list go to the relax-users information page at https://mail.gna.org/listinfo/relax-users/. You can also browse the mailing list archives at https://mail.gna.org/public/relax-users/.

3.2.3 relax-devel

A second mailing list exists for posts relating to the development of relax. The list is "relax-devel at gna.org" and to subscribe go to the relax-devel information page at https://mail.gna.org/listinfo/relax-devel/. Feature requests, program design, or any other posts relating to relax's structure or code should be sent to this list instead. The mailing list archives can be browsed at https://mail.gna.org/public/relax-devel/.

3.2.4 relax-commits

One last mailing list is the relax commits list. This list is reserved for automatically generated posts created by the version control software which looks after the relax source code and these web pages. If you would like to become a developer you can subscribe to the list at relax-commits information page https://mail.gna.org/listinfo/relax-commits/. The list can also be browsed at https://mail.gna.org/public/relax-commits/.

3.2.5 Replying to a message

When replying to a message on these lists remember to hit 'respond to all' so that the mailing list is included in the CC field. Otherwise your message will only be sent to the original poster and not return back to the list. Only messages to relax-users and relax-devel will be accepted. If you are using Gmail's web based interface, please do not click on 'Edit Subject' as this currently mangles the email headers, creates a new thread on the mailing list, and makes it difficult to follow the thread.

3.3 Reporting bugs

One of the philosophies in the construction of relax is that if there is something which is not immediately obvious then that is considered a design bug. If any flaws in relax are uncovered including general design flaws, bugs in the code, or documentation issues these can be reported within relax's bug tracker system. The link to submit a bug is https://gna.org/bugs/?group=relax&func=additem while the main page for browsing, submitting, viewing the statistics, or searching through the database is at https://gna.org/bugs/?group=relax. Please do not report bugs to personal email addresses or to the mailing lists.

When reporting a bug please include as much information as possible so that the problem can be reproduced. Include information such as the release version or the revision number if the repository sources are being used. Also include all the steps performed in order to trigger the bug. Attachment of files is allowed so scripts and subsets of the input data can be included. However please do not attach large files to the report. Prior to reporting the bug try to make sure that the problem is indeed a bug and if you have any doubts please feel free to ask on the relax-users mailing list. To avoid duplicates be sure that the bug has not already been submitted to the bug tracker. You can search the bugs from the page https://gna.org/project/search.php?group=relax.

Once the bug has been confirmed by one of the relax developers you may speed up the resolution of the problem by trying to fixing the bug yourself. If you do wish to play with the source code and try to fix the issue see the relax development chapter of this manual on how to check out the latest sources, how to generate a patch (which is just the output of diff in the 'unified' format), and the guidelines for the format of the code.

3.4 Latest sources – the relax repositories

relax's source code is kept within a version control system called Subversion (http://subversion.tigris.org/). Subversion or SVN allows fine control over the development of the program. The repository contains all information about every change ever made to the program. To learn more about the system the Subversion book located at http://svnbook.red-bean.com/ is a good place to start. The contents of the relax repository can be viewed on-line at http://svn.gna.org/viewcvs/relax/. The current sources, assuming that the most recent minor version number is 1.2, can be downloaded using the SVN protocol by typing

```
$ svn co svn://svn.gna.org/svn/relax/1.2 relax
```

however if this does not work, try the command

```
$ svn co http://svn.gna.org/svn/relax/1.2 relax
```

to download using the HTTP protocol. The entire relax repository is backed up daily to http://svn.gna.org/daily/relax.dump.gz.

3.5 News

Summaries of the latest news on relax can be found on the relax web site https://gna.org/projects/relax. However more information can be found at the dedicated news page https://gna.org/news/?group=relax.

3.6 The relax distribution archives

The relax distribution archives, the files to download to install relax, can be found at http://download.gna.org/relax/. If a compiled binary distribution for your architecture

does not exist you are welcome to create this distribution yourself and submit it for inclusion in the relax project. To do this a number of steps are required. Firstly, the code to each relax release or version resides in the 'tags' directory of the relax repository. To check out version 1.2.0 for example type

```
$ svn co svn://svn.gna.org/svn/relax/tags/1.2.0 relax
```

Again the sources are available through HTTP by typing

```
$ svn co http://svn.gna.org/svn/relax/tags/1.2.0 relax
```

The binary distribution can then be created for your architecture by shifting to the main directory of the checked out sources and typing

```
$ cd relax
$ scons binary_dist
```

At the end SCons will attempt to make a GPG signature for the newly created archive. However this will fail as the current relax private GPG key is not available for security reasons. If the SCons command fails, excluding the GPG signing, please submit a bug report with as much information possible including the details described next to https://gna.org/bugs/?group=relax&func=additem (the python and SCons version numbers may also be useful). Once the file has been created post a message to the relax development mailing list describing the compilation and the creation of the archive, the relax version number, the machine architecture, operating system, and the name of the new file. Do not attach the file though. You will then receive a response explaining where to send the file to. For security the archive will be thoroughly checked and if the source code is identical to that in the repository and the C modules are okay, the file will be GPG signed and uploaded to http://download.gna.org/relax/.

Chapter 4

Calculating the NOE

4.1 Introduction

The calculation of NOE values is a straight forward and quick procedure which involves two components – the calculation of the value itself and the calculation of the errors. To understand the steps involved the execution of a sample NOE calculation script will be followed in detail.

4.2 The sample script

```
# Script for calculating NOEs.

# Create the data pipe.
pipe.create('NOE', 'noe')

# Load the sequence from a PDB file.
structure.read_pdb(name, 'Ap4Aase_new_3.pdb')
structure.load_spins(spin_id='@N')
# Load the reference spectrum and saturated spectrum peak intensities.
noe.read(file='ref.list', spectrum_type='ref')
noe.read(file='sat.list', spectrum_type='sat')

# Set the errors.
noe.error(error=3600, spectrum_type='ref')
noe.error(error=3000, spectrum_type='sat')

# Individual residue errors.
noe.error(error=122000, spectrum_type='ref', res_num=114)
noe.error(error=8500, spectrum_type='sat', res_num=114)

# Deselect unresolved residues.
deselect.read(file='unresolved')
```

```
# Calculate the NOEs.
calc()

# Save the NOEs.
value.write(param='noe', file='noe.out', force=True)

# Create grace files.
grace.write(y_data_type='ref', file='ref.agr', force=True)
grace.write(y_data_type='sat', file='sat.agr', force=True)
grace.write(y_data_type='noe', file='noe.agr', force=True)

# View the grace files.
grace.view(file='ref.agr')
grace.view(file='sat.agr')
grace.view(file='noe.agr')

# Write the results.
results.write(file='results', dir=None, force=True)

# Save the program state.
state.save('save', force=True)
```

4.3 Initialisation of the data pipe

The data pipe is simply created by the command

```
pipe.create('NOE', 'noe')
```

This user function will then create a NOE calculation specific data pipe labelled `'NOE'`. The second argument sets the pipe type to that of the NOE calculation. Setting the pipe type is important so that the program knows which user functions are compatible with the data pipe, for example the function `minimise()` is meaningless in this sample script as the NOE values are directly calculated rather than optimised.

4.4 Loading the data

The first thing which need to be completed prior to any residue specific command is to generate the sequence from a PDB file. In this case the command

```
structure.read_pdb(name, 'Ap4Aase_new_3.pdb')
```

will load the PDB file 'Ap4Aase_new_3.pdb' into relax. Then

```
structure.load_spins(spin_id='@N')
```

will generate the molecule, residue, and spin sequence for the current data pipe. In this situation there will be a single spin system per residue generated corresponding to the backbone amide nitrogens. Although the PDB coordinates have been loaded into the program, the structural information serves no purpose when calculating NOE values.

The next two commands

```
noe.read(file='ref.list', spectrum_type='ref')
noe.read(file='sat.list', spectrum_type='sat')
```

load the peak heights of the reference and saturated NOE experiments (although the volume could be used instead). The keyword argument `format` has not been specified hence the default format of a Sparky peak list (saved after typing `'lt'`) is assumed. If the program XEasy was used to analyse the spectra the argument `format='xeasy'` is necessary. The first column of the file should be the Sparky assignment string and it is assumed that the 4^{th} column contains either the peak height or peak volume. For example:

```
Assignment          w1          w2   Data Height

   LEU3N-HN    122.454       8.397        129722
   GLY4N-HN    111.999       8.719        422375
   SER5N-HN    115.085       8.176        384180
   MET6N-HN    120.934       8.812        272100
   ASP7N-HN    122.394       8.750        174970
   SER8N-HN    113.916       7.836        218762
  GLU11N-HN    122.194       8.604         30412
  GLY12N-HN    110.525       9.028         90144
```

If you have any other format you would like read by relax please send an email to the relax development mailing list detailing the software used, the format of the file (specifically where the residue number and peak intensity are located), and possibly attaching an example of the file itself.

4.5 Setting the errors

In this example the errors where measured from the base plain noise. The Sparky RMSD function was used to estimate the maximal noise levels across the spectrum in regions containing no peaks. For the reference spectrum the RMSD was approximately 3600 whereas in the saturated spectrum the RMSD was 3000. These errors are set by the commands

```
noe.error(error=3600, spectrum_type='ref')
noe.error(error=3000, spectrum_type='sat')
```

For the residue G114, the noise levels are significantly increased compared to the rest of the protein as the peak is located close to the water signal. The higher errors for this residue are specified by the commands

```
noe.error(error=122000, spectrum_type='ref', res_num=114)
noe.error(error=8500, spectrum_type='sat', res_num=114)
```

4.6 Unresolved residues

As the peaks of certain residues overlap to such an extent that the heights cannot be resolved, a simple text file was created called `unresolved` in which each line consists of a

single residue number. By using the command

```
deselect.read(name, file='unresolved')
```

all residues in the file `unresolved` are excluded from the analysis.

4.7 The NOE

At this point the NOE can be calculated. The user function

```
calc()
```

will calculate both the NOE and the errors. The NOE value will be calculated using the formula

$$NOE = \frac{I_{sat}}{I_{ref}}, \tag{4.1}$$

where I_{sat} is the intensity of the peak in the saturated spectrum and I_{ref} is that of the reference spectrum. The error is calculated by

$$\sigma_{NOE} = \sqrt{\frac{(\sigma_{sat} \cdot I_{ref})^2 + (\sigma_{ref} \cdot I_{sat})^2}{I_{ref}}}, \tag{4.2}$$

where σ_{sat} and σ_{ref} are the peak intensity errors in the saturated and reference spectra respectively. To create a file of the NOEs the command

```
value.write(param='noe', file='noe.out', force=True)
```

will create a file called `noe.out` with the NOE values and errors. The force flag will cause any file with the same name to be overwritten. An example of the format of `noe.out` is

```
Num  Name  Value                 Error
1    GLY   None                  None
2    PRO   None                  None
3    LEU   None                  None
4    GLY   0.12479588727508535   0.020551827436105764
5    SER   0.42240815792914105   0.02016346825976852
6    MET   0.45281703194372114   0.026272719841642134
7    ASP   0.60727570079478255   0.032369427242382849
8    SER   0.63871921623680161   0.024695665815261791
9    PRO   None                  None
10   PRO   None                  None
11   GLU   None                  None
12   GLY   0.92927160307645906   0.059569089743604184
13   TYR   0.88832516377296256   0.044119641308479306
14   ARG   0.84945042565860407   0.060533543601110441
```

4.8 Viewing the results

Any two dimensional data set can be plotted in relax in conjunction with the program Grace. The program is also known as Xmgrace and was previously known as ACE/gr or

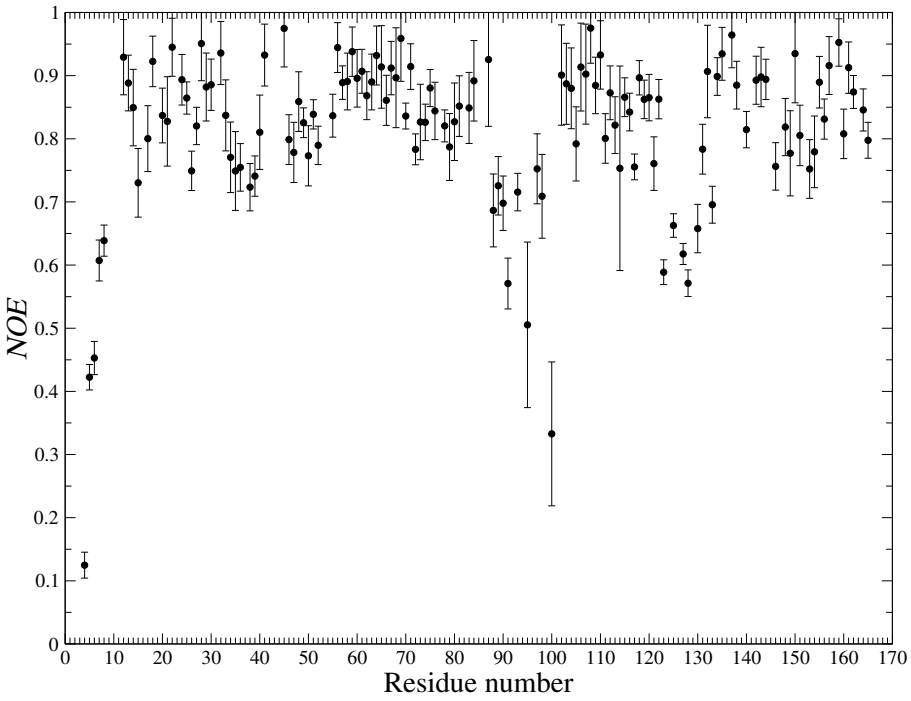

Figure 4.1: A Grace plot of the NOE value and error against the residue number. This is an example of the output of the user function `grace.write()`.

Xmgr. The highly flexible relax user function `grace.write` is capable of producing 2D plots of any x-y data sets. The three commands

```
grace.write(y_data_type='ref', file='ref.agr', force=True)
grace.write(y_data_type='sat', file='sat.agr', force=True)
grace.write(y_data_type='noe', file='noe.agr', force=True)
```

create three separate plots of the peak intensity of the reference and saturated spectra as well as the NOE. The x-axis in all three defaults to the residue number. As the x and y-axes can be any parameter the command

```
grace.write(x_data_type='ref', y_data_type='sat', file='ref_vs_sat.agr', force=True)
```

would create a plot of the reference verses the saturated intensity with one point per residue. Returning to the sample script three Grace data files are created `ref.agr`, `sat.agr`, and `noe.agr` and placed in the default directory `./grace`. These can be visualised by opening the file within Grace. However relax will do that for you with the commands

```
grace.view(file='ref.agr')
grace.view(file='sat.agr')
grace.view(file='noe.agr')
```

An example of the output after modifying the axes is shown in figure 4.1.

Chapter 5

The R_1 and R_2 relaxation rates – relaxation curve-fitting

5.1 Introduction

Relaxation curve-fitting involves a number of steps including the loading of data, the calculation of both the average peak intensity across replicated spectra and the standard deviations of those peak intensities, selection of the experiment type, optimisation of the parameters of the fit, Monte Carlo simulations to find the parameter errors, and saving and viewing the results. To simplify the process a sample script will be followed step by step as was done with the NOE calculation.

5.2 The sample script

```
# Script for relaxation curve-fitting.

# Create the 'rx' data pipe.
pipe.create('rx', 'relax_fit')

# Load the backbone amide 15N spins from a PDB file.
structure.read_pdb('Ap4Aase_new_3.pdb')
structure.load_spins(spin_id='@N')

# Load the peak intensities.
relax_fit.read(file='T2_ncyc1.list', relax_time=0.0176)
relax_fit.read(file='T2_ncyc1b.list', relax_time=0.0176)
relax_fit.read(file='T2_ncyc2.list', relax_time=0.0352)
relax_fit.read(file='T2_ncyc4.list', relax_time=0.0704)
relax_fit.read(file='T2_ncyc4b.list', relax_time=0.0704)
relax_fit.read(file='T2_ncyc6.list', relax_time=0.1056)
relax_fit.read(file='T2_ncyc9.list', relax_time=0.1584)
relax_fit.read(file='T2_ncyc9b.list', relax_time=0.1584)
```

```
relax_fit.read(file='T2_ncyc11.list', relax_time=0.1936)
relax_fit.read(file='T2_ncyc11b.list', relax_time=0.1936)

# Calculate the peak intensity averages and the standard deviation of all spectra.
relax_fit.mean_and_error()

# Deselect unresolved residues.
deselect.read(file='unresolved')

# Set the relaxation curve type.
relax_fit.select_model('exp')

# Grid search.
grid_search(inc=11)

# Minimise.
minimise('simplex', scaling=False, constraints=False)

# Monte Carlo simulations.
monte_carlo.setup(number=500)
monte_carlo.create_data()
monte_carlo.initial_values()
minimise('simplex', scaling=False, constraints=False)
monte_carlo.error_analysis()

# Save the relaxation rates.
value.write(param='rx', file='rx.out', force=True)

# Grace plots of the relaxation rate.
grace.write(y_data_type='rx', file='rx.agr', force=True)
grace.view(file='rx.agr')

# Save the program state.
state.save(file='rx.save', force=True)
```

5.3 Initialisation of the data pipe and loading of the data

The start of this sample script is very similar to that of the NOE calculation on page 20. The command

```
pipe.create('rx', 'relax_fit')
```

initialises the data pipe labelled `'rx'`. The data pipe type is set to relaxation curve-fitting by the argument `'relax_fit'`. The backbone amide nitrogen sequence is extracted from a PDB file using the same commands as the NOE calculation script

```
structure.read_pdb(name, 'Ap4Aase_new_3.pdb')

structure.load_spins(spin_id='@N')
```

To load the peak intensities into relax the user function `relax_fit.read` is executed. Two important keyword arguments to this command are the file name and the relaxation time period of the experiment in seconds. It is assumed that the file format is that of a Sparky peak list. Using the format argument, this can be changed to XEasy text window output format. To be able to import any other type of format please send an email to the relax development mailing list with the details of the format. Adding support for new formats is trivial. The following series of commands will load peak intensities from six different relaxation periods, four of which have been duplicated

```
relax_fit.read(file='T2_ncyc1.list', relax_time=0.0176)
relax_fit.read(file='T2_ncyc1b.list', relax_time=0.0176)
relax_fit.read(file='T2_ncyc2.list', relax_time=0.0352)
relax_fit.read(file='T2_ncyc4.list', relax_time=0.0704)
relax_fit.read(file='T2_ncyc4b.list', relax_time=0.0704)
relax_fit.read(file='T2_ncyc6.list', relax_time=0.1056)
relax_fit.read(file='T2_ncyc9.list', relax_time=0.1584)
relax_fit.read(file='T2_ncyc9b.list', relax_time=0.1584)
relax_fit.read(file='T2_ncyc11.list', relax_time=0.1936)
relax_fit.read(file='T2_ncyc11b.list', relax_time=0.1936)
```

The format for the Sparky peak list is assumed to have the intensity value in the 4^{th} column, e.g.:

```
Assignment          w1          w2   Data Height

   LEU3N-HN    122.454       8.397        129722
   GLY4N-HN    111.999       8.719        422375
   SER5N-HN    115.085       8.176        384180
   MET6N-HN    120.934       8.812        272100
   ASP7N-HN    122.394       8.750        174970
   SER8N-HN    113.916       7.836        218762
  GLU11N-HN    122.194       8.604         30412
  GLY12N-HN    110.525       9.028         90144
```

5.4 The rest of the setup

Once all the peak intensity data has been loaded a few calculations are required prior to optimisation. Firstly the peak intensities for individual residues needs to be averaged across replicated spectra. The peak intensity errors also have to be calculated using the standard deviation formula. These two operations are executed by the user function

```
relax_fit.mean_and_error()
```

Any residues which cannot be resolved due to peak overlap were included in a file called `'unresolved'`. This file consists solely of one residue number per line. These residues are excluded from the analysis by the user function

```
deselect.read(file='unresolved')
```

Finally the experiment type is specified by the command

```
relax_fit.select_model('exp')
```

The argument `'exp'` sets the relaxation curve to a two parameter $\{R_x, I_0\}$ exponential which decays to zero. The formula of this function is

$$I(t) = I_0 e^{-R_x \cdot t}, \tag{5.1}$$

where $I(t)$ is the peak intensity at any time point t, I_0 is the initial intensity, and R_x is the relaxation rate (either the R_1 or R_2). Changing the user function argument to `'inv'` will select the inversion recovery experiment. This curve consists of three paremeters $\{R_1, I_0, I_\infty\}$ and does not decay to zero. The formula is

$$I(t) = I_\infty - I_0 e^{-R_1 \cdot t}. \tag{5.2}$$

5.5 Optimisation

Now that everything has been setup minimision can be used to optimise the parameter values. Firstly a grid search is applied to find a rough starting position for the subsequent optimisation algorithm. Eleven increments per dimension of the model (in this case the two dimensions $\{R_x, I_0\}$) is sufficient. The user function for executing the grid search is

```
grid_search(inc=11)
```

The next step is to select one of the minimisation algorithms to optimise the model parameters. Currently for relaxation curve-fitting only simplex minimisation is supported. This is because the relaxation curve-fitting C module is incomplete only implementing the chi-squared function. The chi-squared gradient (the vector of first partial derivatives) and chi-squared Hessian (the matrix of second partial derivatives) are not yet implemented in the C modules and hence optimisation algorithms which only employ function calls are supported. Simplex minimisation is the only technique in relax which fits this criteron. In addition constraints cannot be used as the constraint algorithm is dependent on gradient calls. Therefore the minimisation command for relaxation curve-fitting is forced to be

```
minimise('simplex', constraints=False)
```

5.6 Error analysis

Only one technique adequately estimates parameter errors when the parameter values where found by optimisation – Monte Carlo simulations. In relax this can be implemented by using a series of functions from the `monte_carlo` user function class. Firstly the number of simulations needs to be set

```
monte_carlo.setup(number=500)
```

For each simulation, randomised relaxation curves will be fit using exactly the same methodology as the original exponential curves. These randomised curves are created by back calculation from the fitted model parameter values and then each point on the curve randomised using the error values set earlier in the script

```
monte_carlo.create_data()
```

As a grid search for each simulation would be too computationally expensive, the starting point for optimisation for each simulation can be set to the position of the optimised parameter values of the model

```
monte_carlo.initial_values()
```

Then exactly the same optimisation as was used for the model can be performed

```
minimise('simplex', constraints=False)
```

The parameter errors are then determined as the standard deviation of the optimised parameter values of the simulations

```
monte_carlo.error_analysis()
```

5.7 Finishing off

To finish off, the script first saves the relaxation rates together with their errors in a simple text file

```
value.write(param='rx', file='rx.out', force=True)
```

Grace plots are created and viewed

```
grace.write(y_data_type='rx', file='rx.agr', force=True)
```

```
grace.view(file='rx.agr')
```

and finally the program state is saved for future reference

```
state.save(file='rx.save', force=True)
```

Chapter 6

Model-free analysis

6.1 Theory

6.1.1 The chi-squared function – $\chi^2(\theta)$

For the minimisation of the model-free models a chain of calculations, each based on a different theory, is required. At the highest level the equation which is actually minimised is the chi-squared function

$$\chi^2(\theta) = \sum_{i=1}^{n} \frac{(\mathrm{R}_i - \mathrm{R}_i(\theta))^2}{\sigma_i^2}, \tag{6.1}$$

where the index i is the summation index ranging over all the experimentally collected relaxation data of all residues used in the analysis; R_i belongs to the relaxation data set R for an individual residue, a collection of residues, or the entire macromolecule and includes the R_1, R_2, and NOE data at all field strengths; $\mathrm{R}_i(\theta)$ is the back-calculated relaxation value belonging to the set $\mathrm{R}(\theta)$; θ is the model parameter vector which when minimised is denoted by $\hat{\theta}$; and σ_i is the experimental error.

The significance of the chi-squared equation (6.1) is that the function returns a single value which is then minimised by the optimisation algorithm to find the model-free parameter values of the given model.

6.1.2 The transformed relaxation equations – $\mathrm{R}_i(\theta)$

The chi-squared equation is itself dependent on the relaxation equations through the back-calculated relaxation data $\mathrm{R}(\theta)$. Letting the relaxation values of the set $\mathrm{R}(\theta)$ be the $\mathrm{R}_1(\theta)$, $\mathrm{R}_2(\theta)$, and $\mathrm{NOE}(\theta)$ an additional layer of abstraction can be used to simplify the calculation of the gradients and Hessians. This involves decomposing the NOE equation into the cross relaxation rate constant $\sigma_{\mathrm{NOE}}(\theta)$ and the auto relaxation rate $\mathrm{R}_1(\theta)$. Taking

equation (6.6) below the transformed relaxation equations are

$$\mathrm{R}_1(\theta) = \mathrm{R}'_1(\theta), \tag{6.2a}$$
$$\mathrm{R}_2(\theta) = \mathrm{R}'_2(\theta), \tag{6.2b}$$
$$\mathrm{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X}\frac{\sigma_{\mathrm{NOE}}(\theta)}{\mathrm{R}_1(\theta)}. \tag{6.2c}$$

whereas the relaxation equations are the $\mathrm{R}_1(\theta)$, $\mathrm{R}_2(\theta)$, $\sigma_{\mathrm{NOE}}(\theta)$.

6.1.3 The relaxation equations – $\mathrm{R}'_i(\theta)$

The relaxation values of the set $\mathrm{R}'(\theta)$ include the spin-lattice, spin-spin, and cross-relaxation rates at all field strengths. These rates are respectively (Abragam, 1961)

$$\mathrm{R}_1(\theta) = d\Big(J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H + \omega_X)\Big) + cJ(\omega_X), \tag{6.3a}$$
$$\begin{aligned}\mathrm{R}_2(\theta) = \frac{d}{2}\Big(&4J(0) + J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H) \\ &+ 6J(\omega_H + \omega_X)\Big) + \frac{c}{6}\Big(4J(0) + 3J(\omega_X)\Big) + R_{ex},\end{aligned} \tag{6.3b}$$
$$\sigma_{\mathrm{NOE}}(\theta) = d\Big(6J(\omega_H + \omega_X) - J(\omega_H - \omega_X)\Big), \tag{6.3c}$$

where $J(\omega)$ is the power spectral density function and R_{ex} is the relaxation due to chemical exchange. The dipolar and CSA constants are defined in SI units as

$$d = \frac{1}{4}\left(\frac{\mu_0}{4\pi}\right)^2 \frac{(\gamma_H\gamma_X\hbar)^2}{\langle r^6\rangle}, \tag{6.4}$$
$$c = \frac{(\omega_H\Delta\sigma)^2}{3}, \tag{6.5}$$

where μ_0 is the permeability of free space, γ_H and γ_X are the gyromagnetic ratios of the H and X spins respectively, $\hbar$ is Plank's constant divided by 2π, r is the bond length, and $\Delta\sigma$ is the chemical shift anisotropy measured in ppm. The cross-relaxation rate σ_{NOE} is related to the steady state NOE by the equation

$$\mathrm{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X}\frac{\sigma_{\mathrm{NOE}}(\theta)}{\mathrm{R}_1(\theta)}. \tag{6.6}$$

6.1.4 The spectral density functions – $J(\omega)$

The relaxation equations are themselves dependent on the calculation of the spectral density values $J(\omega)$. Within model-free analysis these are modelled by the original model-free formula (Lipari and Szabo, 1982a,b)

$$J(\omega) = \frac{2}{5}\sum_{i=-k}^{k} c_i \cdot \tau_i\left(\frac{S^2}{1 + (\omega\tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega\tau_e\tau_i)^2}\right), \tag{6.7}$$

where S^2 is the square of the Lipari and Szabo generalised order parameter and τ_e is the effective correlation time. The order parameter reflects the amplitude of the motion and the correlation time in an indication of the time scale of that motion. The theory was extended by Clore et al. (1990) by the modelling of two independent internal motions using the equation

$$J(\omega) = \frac{2}{5}\sum_{i=-k}^{k} c_i \cdot \tau_i \left(\frac{S^2}{1+(\omega\tau_i)^2} + \frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right). \quad (6.8)$$

where S_f^2 and τ_f are the amplitude and timescale of the faster of the two motions whereas S_s^2 and τ_s are those of the slower motion. S_f^2 and S_s^2 are related by the formula $S^2 = S_f^2 \cdot S_s^2$.

6.1.5 Brownian rotational diffusion

In equations (6.7) and (6.8) the generic Brownian diffusion NMR correlation function presented in d'Auvergne (2006) has been used. This function is

$$C(\tau) = \frac{1}{5}\sum_{i=-k}^{k} c_i \cdot e^{-\tau/\tau_i}, \quad (6.9)$$

where the summation index i ranges over the number of exponential terms within the correlation function. This equation is generic in that it can describe the diffusion of an ellipsoid, a spheroid, or a sphere.

Diffusion as an ellipsoid

For the ellipsoid defined by the parameter set $\{\mathfrak{D}_{iso},\ \mathfrak{D}_a,\ \mathfrak{D}_r,\ \alpha,\ \beta,\ \gamma\}$ the variable k is equal to two and therefore the index $i \in \{-2,-1,0,1,2\}$. The geometric parameters $\{\mathfrak{D}_{iso},\ \mathfrak{D}_a,\ \mathfrak{D}_r\}$ are defined as

$$\mathfrak{D}_{iso} = \tfrac{1}{3}(\mathfrak{D}_x + \mathfrak{D}_y + \mathfrak{D}_z), \quad (6.10a)$$
$$\mathfrak{D}_a = \mathfrak{D}_z - \tfrac{1}{2}(\mathfrak{D}_x + \mathfrak{D}_y), \quad (6.10b)$$
$$\mathfrak{D}_r = \frac{\mathfrak{D}_y - \mathfrak{D}_x}{2\mathfrak{D}_a}, \quad (6.10c)$$

and are constrained by

$$0 < \mathfrak{D}_{iso} < \infty, \quad (6.11a)$$
$$0 \le \mathfrak{D}_a < \frac{\mathfrak{D}_{iso}}{\frac{1}{3}+\mathfrak{D}_r} \le 3\mathfrak{D}_{iso}, \quad (6.11b)$$
$$0 \le \mathfrak{D}_r \le 1. \quad (6.11c)$$

The orientational parameters $\{\alpha,\ \beta,\ \gamma\}$ are the Euler angles using the z-y-z rotation notation.

The five weights c_i are defined as

$$c_{-2} = \tfrac{1}{4}(d - e), \tag{6.12a}$$
$$c_{-1} = 3\delta_y^2\delta_z^2, \tag{6.12b}$$
$$c_0 = 3\delta_x^2\delta_z^2, \tag{6.12c}$$
$$c_1 = 3\delta_x^2\delta_y^2, \tag{6.12d}$$
$$c_2 = \tfrac{1}{4}(d + e), \tag{6.12e}$$

where

$$d = 3\left(\delta_x^4 + \delta_y^4 + \delta_z^4\right) - 1, \tag{6.13}$$
$$e = \frac{1}{\mathfrak{R}}\left[(1 + 3\mathfrak{D}_r)\left(\delta_x^4 + 2\delta_y^2\delta_z^2\right) + (1 - 3\mathfrak{D}_r)\left(\delta_y^4 + 2\delta_x^2\delta_z^2\right) - 2\left(\delta_z^4 + 2\delta_x^2\delta_y^2\right)\right], \tag{6.14}$$

and where

$$\mathfrak{R} = \sqrt{1 + 3\mathfrak{D}_r^2}. \tag{6.15}$$

The five correlation times τ_i are

$$1/\tau_{-2} = 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R}, \tag{6.16a}$$
$$1/\tau_{-1} = 6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r), \tag{6.16b}$$
$$1/\tau_0 = 6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r), \tag{6.16c}$$
$$1/\tau_1 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a, \tag{6.16d}$$
$$1/\tau_2 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R}. \tag{6.16e}$$

Diffusion as a spheroid

The variable k is equal to one in the case of the spheroid defined by the parameter set $\{\mathfrak{D}_{iso},\ \mathfrak{D}_a,\ \theta,\ \phi\}$, hence $i \in \{-1, 0, 1\}$. The geometric parameters $\{\mathfrak{D}_{iso},\ \mathfrak{D}_a\}$ are defined as

$$\mathfrak{D}_{iso} = \tfrac{1}{3}(\mathfrak{D}_\parallel + 2\mathfrak{D}_\perp), \tag{6.17a}$$
$$\mathfrak{D}_a = \mathfrak{D}_\parallel - \mathfrak{D}_\perp. \tag{6.17b}$$

and are constrained by

$$0 < \mathfrak{D}_{iso} < \infty, \tag{6.18a}$$
$$-\tfrac{3}{2}\mathfrak{D}_{iso} < \mathfrak{D}_a < 3\mathfrak{D}_{iso}. \tag{6.18b}$$

The orientational parameters $\{\theta,\ \phi\}$ are the spherical angles defining the orientation of the major axis of the diffusion frame within the lab frame.

The three weights c_i are

$$c_{-1} = \tfrac{1}{4}(3\delta_z^2 - 1)^2, \tag{6.19a}$$
$$c_0 = 3\delta_z^2(1 - \delta_z^2), \tag{6.19b}$$
$$c_1 = \tfrac{3}{4}(\delta_z^2 - 1)^2. \tag{6.19c}$$

The five correlation times τ_i are

$$1/\tau_{-1} = 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a, \tag{6.20a}$$
$$1/\tau_0 = 6\mathfrak{D}_{iso} - \mathfrak{D}_a, \tag{6.20b}$$
$$1/\tau_1 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a. \tag{6.20c}$$

Diffusion as a sphere

In the situation of a molecule diffusing as a sphere either described by the single parameter τ_m or $\mathfrak{D}_{iso}$, the variable k is equal to zero. Therefore $i \in \{0\}$. The single weight c_0 is equal to one and the single correlation time τ_0 is equivalent to the global tumbling time τ_m given by

$$1/\tau_m = 6\mathfrak{D}_{iso}. \tag{6.21}$$

This is diffusion equation presented in Bloembergen et al. (1948).

6.1.6 The model-free models

Extending the list of models given in Mandel et al. (1995); Fushman et al. (1997); Orekhov et al. (1999); Korzhnev et al. (2001); Zhuravleva et al. (2004), the models built into relax include

$$m0 = \{\}, \tag{6.22.0}$$
$$m1 = \{S^2\}, \tag{6.22.1}$$
$$m2 = \{S^2, \tau_e\}, \tag{6.22.2}$$
$$m3 = \{S^2, R_{ex}\}, \tag{6.22.3}$$
$$m4 = \{S^2, \tau_e, R_{ex}\}, \tag{6.22.4}$$
$$m5 = \{S^2, S_f^2, \tau_s\}, \tag{6.22.5}$$
$$m6 = \{S^2, \tau_f, S_f^2, \tau_s\}, \tag{6.22.6}$$
$$m7 = \{S^2, S_f^2, \tau_s, R_{ex}\}, \tag{6.22.7}$$
$$m8 = \{S^2, \tau_f, S_f^2, \tau_s, R_{ex}\}, \tag{6.22.8}$$
$$m9 = \{R_{ex}\}. \tag{6.22.9}$$

The parameter R_{ex} is scaled quadratically with field strength in these models as it is assumed to be fast. In the set theory notation, the model-free model for the spin system i is represented by the symbol $\mathfrak{F}_i$. Through the addition of the local τ_m to each of these

models, only the component of Brownian rotational diffusion experienced by the spin system is probed. These models, represented in set notation by the symbol $\mathfrak{T}_i$, are

$$tm0 = \{\tau_m\}, \tag{6.23.0}$$
$$tm1 = \{\tau_m, S^2\}, \tag{6.23.1}$$
$$tm2 = \{\tau_m, S^2, \tau_e\}, \tag{6.23.2}$$
$$tm3 = \{\tau_m, S^2, R_{ex}\}, \tag{6.23.3}$$
$$tm4 = \{\tau_m, S^2, \tau_e, R_{ex}\}, \tag{6.23.4}$$
$$tm5 = \{\tau_m, S^2, S_f^2, \tau_s\}, \tag{6.23.5}$$
$$tm6 = \{\tau_m, S^2, \tau_f, S_f^2, \tau_s\}, \tag{6.23.6}$$
$$tm7 = \{\tau_m, S^2, S_f^2, \tau_s, R_{ex}\}, \tag{6.23.7}$$
$$tm8 = \{\tau_m, S^2, \tau_f, S_f^2, \tau_s, R_{ex}\}, \tag{6.23.8}$$
$$tm9 = \{\tau_m, R_{ex}\}. \tag{6.23.9}$$

6.1.7 Model-free optimisation theory

The model-free space

The optimisation of the parameters of an arbitrary model is dependent on a function f which takes the current parameter values $\theta \in \mathbb{R}^n$ and returns a single real value $f(\theta) \in \mathbb{R}$ corresponding to position θ in the n-dimensional space. For it is that single value which is minimised as

$$\hat{\theta} = \arg\min_{\theta} f(\theta), \tag{6.24}$$

where $\hat{\theta}$ is the parameter vector which is equal to the argument which minimises the function $f(\theta)$. In model-free analysis $f(\theta)$ is the chi-squared equation

$$\chi^2(\theta) = \sum_{i=1}^{n} \frac{(\mathrm{R}_i - \mathrm{R}_i(\theta))^2}{\sigma_i^2}, \tag{6.25}$$

where i is the summation index, R_i is the experimental relaxation data which belongs to the data set R and includes the R_1, R_2, and NOE values at all field strengths, $\mathrm{R}_i(\theta)$ is the back calculated relaxation data belonging to the set $\mathrm{R}(\theta)$, and σ_i is the experimental error. For the optimisation of the model-free parameters while the diffusion tensor is held fixed, the summation index ranges over the relaxation data of an individual residue. If the diffusion parameters are optimised simultaneously with the model-free parameters the summation index ranges over all relaxation data of all selected residues of the macromolecule.

Given the current parameter values the model-free function provided to the algorithm will calculate the value of the model-free spectral density function $J(\omega)$ at the five frequencies which induce NMR relaxation by using Equations (6.7) and (6.8). The theoretical R_1, R_2, and NOE values are then back-calculated using Equations (6.3a), (6.3b), (6.3c), and (6.6). Finally, the chi-squared value is calculated using Equation (6.25).

Topology of the space

The problem of finding the minimum is complicated by the fact that optimisation algorithms are blind to the curvature of the complete space. Instead they rely on topological information about the current and, sometimes, the previous parameter positions in the space. The techniques use this information to walk iteratively downhill to the minimum. Very few optimisation algorithms rely solely on the function value, conceptually the height of the space, at the current position. Most techniques also utilise the gradient at the current position. Although symbolically complex in the case of model-free analysis, the gradient can simply be calculated as the vector of first partial derivatives of the chi-squared equation with respect to each model-free parameter. The gradient is supplied as a second function to the algorithm which is then utilised in diverse ways by different optimisation techniques. The function value together with the gradient can be combined to construct a linear or planar description of the space at the current parameter position by first-order Taylor series approximation

$$f(\theta_k + x) \approx f_k + x^T \nabla f_k, \tag{6.26}$$

where f_k is the function value at the current parameter position θ_k, ∇f_k is the gradient at the same position, and x is an arbitrary vector. By accumulating information from previous parameter positions a more comprehensive geometric description of the curvature of the space can be exploited by the algorithm for more efficient optimisation.

The best and most comprehensive description of the space is given by the quadratic approximation of the topology which is generated from the combination of the function value, the gradient, and the Hessian. From the second-order Taylor series expansion the quadratic model of the space is

$$f(\theta_k + x) \approx f_k + x^T \nabla f_k + \tfrac{1}{2} x^T \nabla^2 f_k x, \tag{6.27}$$

where $\nabla^2 f_k$ is the Hessian, which is the symmetric matrix of second partial derivatives of the function, at the position θ_k. As the Hessian is computationally expensive a number of optimisation algorithms try to approximate it.

To produce the gradient and Hessian required for model-free optimisation a large chain of first and second partial derivatives needs to be calculated. Firstly the partial derivatives of the spectral density functions (6.7) and (6.8) are necessary. Then the partial derivatives of the relaxation equations (6.3a) to (6.3c) followed by the NOE equation (6.6) are needed. Finally the partial derivative of the chi-squared formula (6.25) is required. These first and second partial derivatives, as well as those of the components of the Brownian diffusion correlation function for non-isotropic tumbling, are presented in Chapter 8.

Optimisation algorithms

Prior to minimisation, all optimisation algorithms investigated require a starting position within the model-free space. This initial parameter vector is found by employing a coarse grid search – chi-squared values at regular positions spanning the space are calculated and the grid point with the lowest value becomes the starting position. The grid search

itself is an optimisation technique. As it is computationally expensive the number of grid points needs to be kept to a minimum. Hence the initial parameter values are a rough and imprecise approximation of the local minimum. Due to the complexity of the curvature of the model-free space, the grid point with the lowest chi-squared value may in fact be on the opposite side of the space to the local minimum.

Once the starting position has been determined by the grid search the optimisation algorithm can be executed. The number of algorithms developed within the mathematical field of optimisation is considerable. They can nevertheless be grouped into one of a small number of major categories based on the fundamental principles of the technique. These include the line search methods, the trust region methods, and the conjugate gradient methods. For more details on the algorithms described below see Nocedal and Wright (1999).

Line search methods

The defining characteristic of a line search algorithm is to choose a search direction p_k and then to find the minimum along that vector starting from θ_k (Nocedal and Wright, 1999). The distance travelled along p_k is the step length α_k and the parameter values for the next iteration are

$$\theta_{k+1} = \theta_k + \alpha_k p_k. \tag{6.28}$$

The line search algorithm determines the search direction p_k whereas the value of α_k is found using an auxiliary step-length selection algorithm.

One of the simplest line search methods is the steepest descent algorithm. The search direction is simply the negative gradient, $p_k = -\nabla f_k$, and hence the direction of maximal descent is always followed. This method is inefficient – the linear rate of convergence requires many iterations of the algorithm to reach the minimum and it is susceptible to being trapped on saddle points within the space.

The coordinate descent algorithms are a simplistic group of line search methods whereby the search directions alternate between vectors parallel to the parameter axes. For the back-and-forth coordinate descent the search directions cycle in one direction and then back again. For example for a three parameter model the search directions cycle $\theta_1, \theta_2, \theta_3, \theta_2, \theta_1, \theta_2, \ldots$, which means that each parameter of the model is optimised one by one. The method becomes less efficient when approaching the minimum as the step length α_k continually decreases (ibid.).

The quasi-Newton methods begin with an initial guess of the Hessian and update it at each iteration using the function value and gradient. Therefore the benefits of using the quadratic model of (6.27) are obtained without calculating the computationally expensive Hessian. The Hessian approximation B_k is updated using various formulae, the most common being the BFGS formula (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). The search direction is given by the equation $p_k = -B_k^{-1} \nabla f_k$. The quasi-Newton algorithms can attain a superlinear rate of convergence, being superior to the steepest descent or coordinate descent methods.

The most powerful line search method when close to the minimum is the Newton search direction

$$p_k = -\nabla^2 f_k^{-1} \nabla f_k. \tag{6.29}$$

This direction is obtained from the derivative of (6.27) which is assumed to be zero at the minimum of the quadratic model. The vector p_k points from the current position to the exact minimum of the quadratic model of the space. The rate of convergence is quadratic, being superior to both linear and superlinear convergence. The technique is computationally expensive due to the calculation of the Hessian. It is also susceptible to failure when optimisation commences from distant positions in the space as the Hessian may not be positive definite and hence not convex, a condition required for the search direction both to point downhill and to be reasonably oriented. In these cases the quadratic model is a poor description of the space.

A practical Newton algorithm which is robust for distant starting points is the Newton conjugate gradient method (Newton-CG). This line search method, which is also called the truncated Newton algorithm, finds an approximate solution to Equation (6.29) by using a conjugate gradient (CG) sub-algorithm. Retaining the performance of the pure Newton algorithm, the CG sub-algorithm guarantees that the search direction is always downhill as the method terminates when negative curvature is encountered. This algorithm is similar to the Newton-Raphson-CG algorithm implemented within Dasha. Newton optimisation is sometimes also known as the Newton-Raphson algorithm and, as documented in the source code, the Newton algorithm in Dasha is coupled to a conjugate gradient algorithm. The auxiliary step-length selection algorithm in Dasha is undocumented and may not be employed.

Once the search direction has been determined by the above algorithms the minimum along that direction needs to be determined. Not to be confused with the methodology for determining the search direction p_k, the line search itself is performed by an auxiliary step-length selection algorithm to find the value α_k. A number of step-length selection methods can be used to find a minimum along the line $\theta_k + \alpha_k p_k$, although only two will be investigated. The first is the backtracking line search of Nocedal and Wright (1999). This method is inexact – it takes a starting step length α_k and decreases the value until a sufficient decrease in the function is found. The second is the line search method of Moré and Thuente (1994). Designed to be robust, the MT algorithm finds the exact minimum along the search direction and guarantees sufficient decrease.

Trust region methods

In the trust region class of algorithms the curvature of the space is modelled quadratically by (6.27). This model is assumed to be reliable only within a region of trust defined by the inequality $\|p\| \leqslant \Delta_k$ where p is the step taken by the algorithm and Δ_k is the radius of the n-dimensional sphere of trust (Nocedal and Wright, 1999). The solution sought for each iteration of the algorithm is

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + p^T \nabla f_k + \tfrac{1}{2} p^T B_k p, \qquad \text{s.t. } \|p\| \leqslant \Delta_k, \tag{6.30}$$

where $m_k(p)$ is the quadratic model, B_k is a positive definite matrix which can be the true Hessian as in the Newton model or an approximation such as the BFGS matrix, and $\|p\|$

is the Euclidean norm of p. The trust region radius Δ_k is modified dynamically during optimisation – if the quadratic model is found to be a poor representation of the space the radius is decreased whereas if the quadratic model is found to be reasonable the radius is increased to allow larger, more efficient steps to be taken.

The Cauchy point algorithm is similar in concept to the steepest descent line search algorithm. The Cauchy point is the point lying on the gradient which minimises the quadratic model subject to the step being within the trust region. By iteratively finding the Cauchy point the local minimum can be found. The convergence of the technique is inefficient, being similar to that of the steepest descent algorithm.

In changing the trust region radius the exact solutions to (6.30) map out a curved trajectory which starts parallel to the gradient for small radii. The end of the trajectory, which occurs for radii greater than the step length, is the bottom of the quadratic model. The dogleg algorithm attempts to follow a similar path by first finding the minimum along the gradient and then finding the minimum along a trajectory from the current point to the bottom of the quadratic model. The minimum along the second path is either the trust region boundary or the quadratic solution. The matrix B_k of (6.30) can be the BFGS matrix, the unmodified Hessian, or a Hessian modified to be positive definite.

Another trust region algorithm is Steihaug's modified conjugate gradient approach (Steihaug, 1983). For each step k an iterative technique is used which is almost identical to the standard conjugate gradient procedure except for two additional termination conditions. The first is if the next step is outside the trust region, the second is if a direction of zero or negative curvature is encountered.

An almost exact solution to (6.30) can be found using an algorithm described in Nocedal and Wright (1999). This exact trust region algorithm aims to precisely find the minimum of the quadratic model m_k of the space within the trust region Δ_k. Any matrix B_k can be used to construct the quadratic model. However, the technique is computationally expensive.

Conjugate gradient methods

The conjugate gradient algorithm (CG) was originally designed as a mathematical technique for solving a large system of linear equations Hestenes and Stiefel (1952), but was later adapted to solving nonlinear optimisation problems (Fletcher and Reeves, 1964). The technique loops over a set of directions $p_0, p_1, \ldots, p_n$ which are all conjugate to the Hessian (Nocedal and Wright, 1999), a property defined as

$$p_i^T \nabla^2 f_k p_j = 0, \qquad \text{for all } i \neq j. \tag{6.31}$$

By performing line searches over all directions p_j the solution to the quadratic model (6.27) of the position θ_k will be found in n or less iterations of the CG algorithm where n is the total number of parameters in the model. The technique performs well on large problems with many parameters as no matrices are calculated or stored. The algorithms perform better than the steepest descent method and preconditioning of the system is used to improve optimisation. A number of preconditioned techniques will be investigated including the Fletcher-Reeves algorithm which was the original conjugate

gradient optimisation technique (Fletcher and Reeves, 1964), the Polak-Ribière method (Polak and Ribière, 1969), a modified Polak-Ribière method called the Polak-Ribière + method (Nocedal and Wright, 1999), and the Hestenes-Stiefel algorithm which originates from a formula in Hestenes and Stiefel (1952). As a line search is performed to find the minimum along each conjugate direction both the backtracking and Moré and Thuente auxiliary step-length selection algorithms will be tested with the CG algorithms.

Hessian modifications

The Newton search direction, used in both the line search and trust region methods, is dependent on the Hessian being positive definite for the quadratic model to be convex so that the search direction points sufficiently downhill. This is not always the case as saddle points and other non-quadratic features of the space can be problematic. Two classes of algorithms can be used to handle this situation. The first involves using the conjugate gradient method as a sub-algorithm for solving the Newton problem for the step k. The Newton-CG line search algorithm described above is one such example. The second class involves modifying the Hessian prior to, or at the same time as, finding the Newton step to guarantee that the replacement matrix B_k is positive definite. The convexity of B_k is ensured by its eigenvalues all being positive. The performance of two of these methods within the model-free space will be investigated.

The first modification uses the Cholesky factorisation of the matrix B_k, initialised to the true Hessian, to test for convexity (Algorithm 6.3 of Nocedal and Wright (1999)). If factorisation fails the matrix is not positive definite and a constant τ_k times the identity matrix I is then added to B_k. The constant originates from the Robbins norm of the Hessian $\|\nabla^2 f_k\|_F$ and is steadily increased until the factorisation is successful. The resultant Cholesky lower triangular matrix L can then be used to find the approximate Newton direction. If τ_k is too large the convergence of this technique can approach that of the steepest descent algorithm.

The second method is the Gill, Murray, and Wright (GMW) algorithm (Gill et al., 1981) which modifies the Hessian during the execution of the Cholesky factorisation $\nabla^2 f_k = LIL^T$, where L is a lower triangular matrix and D is a diagonal matrix. Only a single factorisation is required. As rows and columns are interchanged during the algorithm the technique may be slow for large problems such as the optimisation of the model-free parameters of all residues together with the diffusion tensor parameters. The rate of convergence of the technique is quadratic.

Other methods

Two other optimisation algorithms which cannot be classified within line search, trust region, or conjugate gradient categories will also be investigated. The first is the well known simplex optimisation algorithm. The technique is often used as the only the function value is employed and hence the derivation of the gradient and Hessian can be avoided. The simplex is created as an n-dimensional geometric object with $n+1$ vertices. The first vertex is the starting position. Each of the other n vertices are created by shifting the starting position by a small amount parallel to one of unit vectors defining the coordinate system of the space. Four simple rules are used to move the simplex through the space:

reflection, extension, contraction, and a shrinkage of the entire simplex. The result of these movements is that the simplex moves in an ameoboid-like fashion downhill, shrinking to pass through tight gaps and expanding to quickly move through non-convoluted space, eventually finding the minimum.

Key to these four movements is the pivot point, the centre of the face created by the n vertices with the lowest function values. The first movement is a reflection – the vertex with the greatest function value is reflected through the pivot point on the opposite face of the simplex. If the function value at this new position is less than all others the simplex is allowed to extend – the point is moved along the line to twice the distance between the current position and the pivot point. Otherwise if the function value is greater than the second highest value but less than the highest value, the reflected simplex is contracted. The reflected point is moved to be closer to the simplex, its position being half way between the reflected position and the pivot point. Otherwise if the function value at the reflected point is greater than all other vertices, then the original simplex is contracted – the highest vertex is moved to a position half way between the current position and the pivot point. Finally if none of these four movements yield an improvement, then the simplex is shrunk halfway towards the vertex with the lowest function value.

The other algorithm is the commonly used Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963) which is implemented in Modelfree4, Dasha, and Tensor2. This technique is designed for least-squares problems to which the chi-squared equation (6.25) belongs. The key to the algorithm is the replacement of the Hessian with the Levenberg-Marquardt matrix $J^TJ + \lambda I$, where J is the Jacobian of the system calculated as the matrix of partial derivatives of the residuals, $\lambda > 0$ is a factor related to the trust-region radius, and I is the identity matrix. The algorithm is conceptually allied to the trust region methods and its performance varies finely between that of the steepest descent and the pure Newton step. When far from the minimum λ is large and the algorithm takes steps close to the gradient; when in vicinity of the minimum λ heads towards zero and the steps taken approximate the Newton direction. Hence the algorithm avoids the problems of the Newton algorithm when non-convex curvature is encountered and approximates the Newton step in convex regions of the space.

Constraint algorithms

To guarantee that the minimum will still be reached the implementation of constraints limiting the parameter values together with optimisation algorithms is not a triviality. For this to occur the space and its boundaries must remain smooth thereby allowing the algorithm to move along the boundary to either find the minimum along the limit or to slide along the limit and then move back into the centre of the constrained space once the curvature allows it. One of the most powerful approaches is the Method of Multipliers (Nocedal and Wright, 1999), also known as the Augmented Lagrangian. Instead of a single optimisation the algorithm is iterative with each iteration consisting of an independent unconstrained minimisation on a sequentially modified space. When inside the limits the function value is unchanged but when outside a penalty, which is proportional to the distance outside the limit, is added to the function value. This penalty, which is based on the Lagrange multipliers, is smooth and hence the gradient and Hessian are continuous at and beyond the constraints. For each iteration of the Method of Multipliers the penalty is increased until it becomes impossible for the parameter vector to be in violation of the

limits. This approach allows the parameter vector θ outside the limits yet the successive iterations ensure that the final results will not be in violation of the constraint.

For inequality constraints, each iteration of the Method of Multipliers attempts to solve the quadratic sub-problem

$$\min_{\theta} \mathfrak{L}_A(\theta, \lambda^k; \mu_k) \stackrel{\text{def}}{=} f(\theta) + \sum_{i \in \mathfrak{I}} \Psi(c_i(\theta), \lambda_i^k; \mu_k), \tag{6.32}$$

where the function Ψ is defined as

$$\Psi(c_i(\theta), \lambda^k; \mu_k) = \begin{cases} -\lambda^k c_i(\theta) + \frac{1}{2\mu_k} c_i^2(\theta) & \text{if } c_i(\theta) - \mu_k \lambda^k \leqslant 0, \\ -\frac{\mu_k}{2}(\lambda^k)^2 & \text{otherwise.} \end{cases} \tag{6.33}$$

In (6.32), θ is the parameter vector; $\mathfrak{L}_A$ is the Augmented Lagrangian function; k is the current iteration of the Method of Multipliers; λ^k are the Lagrange multipliers which are positive factors such that, at the minimum $\hat{\theta}$, $\nabla f(\hat{\theta}) = \lambda_i \nabla c_i(\hat{\theta})$; $\mu_k > 0$ is the penalty parameter which decreases to zero as $k \to \infty$; $\mathfrak{I}$ is the set of inequality constraints; and $c_i(\theta)$ is an individual constraint value. The Lagrange multipliers are updated using the formula

$$\lambda_i^{k+1} = \max(\lambda_i^k - c_i(\theta)/\mu_k, 0), \qquad \text{for all } i \in \mathfrak{I}. \tag{6.34}$$

The gradient of the Augmented Lagrangian is

$$\nabla \mathfrak{L}_A(\theta, \lambda^k; \mu_k) = \nabla f(\theta) - \sum_{i \in \mathfrak{I} | c_i(\theta) \leqslant \mu_k \lambda_i^k} \left(\lambda_i^k - \frac{c_i(\theta)}{\mu_k} \right) \nabla c_i(\theta), \tag{6.35}$$

and the Hessian is

$$\nabla^2 \mathfrak{L}_A(\theta, \lambda^k; \mu_k) = \nabla^2 f(\theta) + \sum_{i \in \mathfrak{I} | c_i(\theta) \leqslant \mu_k \lambda_i^k} \left[\frac{1}{\mu_k} \nabla c_i^2(\theta) - \left(\lambda_i^k - \frac{c_i(\theta)}{\mu_k} \right) \nabla^2 c_i(\theta) \right]. \tag{6.36}$$

The Augmented Lagrangian algorithm can accept any set of three arbitrary constraint functions $c(\theta)$, $\nabla c(\theta)$, and $\nabla^2 c(\theta)$. When given the current parameter values $c(\theta)$ returns a vector of constraint values whereby each position corresponds to one of the model parameters. The constraint is defined as $c_i \geqslant 0$. The function $\nabla c(\theta)$ returns the matrix of constraint gradients and $\nabla^2 c(\theta)$ is the constraint Hessian function which should return the 3D matrix of constraint Hessians.

A more specific set of constraints accepted by the Method of Multipliers are bound constraints. These are defined by the function

$$l \leqslant \theta \leqslant u, \tag{6.37}$$

where l and u are the vectors of lower and upper bounds respectively and θ is the parameter vector. For example for model-free model $m4$ to place lower and upper bounds on the order

parameter and lower bounds on the correlation time and chemical exchange parameters, the vectors are

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \leqslant \begin{pmatrix} S^2 \\ \tau_e \\ R_{ex} \end{pmatrix} \leqslant \begin{pmatrix} 1 \\ \infty \\ \infty \end{pmatrix}. \tag{6.38}$$

The default setting in the program relax is to use linear constraints which are defined as

$$A \cdot \theta \geqslant b, \tag{6.39}$$

where A is an $m \times n$ matrix where the rows are the transposed vectors a_i of length n; the elements of a_i are the coefficients of the model parameters; θ is the vector of model parameters of dimension n; b is the vector of scalars of dimension m; m is the number of constraints; and n is the number of model parameters. For model-free analysis, linear constraints are the most useful type of constraint as the correlation time τ_f can be restricted to being less than τ_s by using the inequality $\tau_s - \tau_f \geqslant 0$.

In rearranging (6.39) the linear constraint function $c(\theta)$ returns the vector $A \cdot \theta - b$. Because of the linearity of the constraints the gradient and Hessian are greatly simplified. The gradient $\nabla c(\theta)$ is simply the matrix A and the Hessian $\nabla^2 c(\theta)$ is zero. For the parameters specific to individual residues the linear constraints in the notation of (6.39) are

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} S^2 \\ S_f^2 \\ S_s^2 \\ \tau_e \\ \tau_f \\ \tau_s \\ R_{ex} \\ r \\ CSA \end{pmatrix} \geqslant \begin{pmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.9e^{-10} \\ 2e^{-10} \\ 300e^{-6} \\ 0 \end{pmatrix}. \tag{6.40}$$

Through the isolation of each individual element, the constraints can be see to be equivalent

to

$$0 \leqslant S^2 \leqslant 1, \tag{6.41a}$$
$$0 \leqslant S_f^2 \leqslant 1, \tag{6.41b}$$
$$0 \leqslant S_s^2 \leqslant 1, \tag{6.41c}$$
$$S^2 \leqslant S_f^2, \tag{6.41d}$$
$$S^2 \leqslant S_s^2, \tag{6.41e}$$
$$\tau_e \geqslant 0, \tag{6.41f}$$
$$\tau_f \geqslant 0, \tag{6.41g}$$
$$\tau_s \geqslant 0, \tag{6.41h}$$
$$\tau_s \geqslant 0, \tag{6.41i}$$
$$\tau_f \leqslant \tau_s, \tag{6.41j}$$
$$R_{ex} \geqslant 0, \tag{6.41k}$$
$$0.9e^{-10} \leqslant r \leqslant 2e^{-10}, \tag{6.41l}$$
$$-300e^{-6} \leqslant CSA \leqslant 0. \tag{6.41m}$$

To prevent the computationally expensive optimisation of failed models in which the internal correlation times minimise to infinity (d'Auvergne and Gooley, 2006), the constraint $\tau_e, \tau_f, \tau_s \leqslant 2\tau_m$ was implemented. When the global correlation time is fixed the constraints in the matrix notation of (6.39) are

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_e \\ \tau_f \\ \tau_s \end{pmatrix} \geqslant \begin{pmatrix} -2\tau_m \\ -2\tau_m \\ -2\tau_m \end{pmatrix}. \tag{6.42}$$

However when the global correlation time τ_m is one of the parameters being optimised the constraints become

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ 2 & 0 & -1 & 0 \\ 2 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_m \\ \tau_e \\ \tau_f \\ \tau_s \end{pmatrix} \geqslant \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{6.43}$$

For the parameters of the diffusion tensor the constraints utilised are

$$0 \leqslant \tau_m \leqslant 200.0e^{-9}, \tag{6.44a}$$
$$\mathfrak{D}_a \geqslant 0, \tag{6.44b}$$
$$0 \leqslant \mathfrak{D}_r \leqslant 1, \tag{6.44c}$$

which in the matrix notation of (6.39) become

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_m \\ \mathfrak{D}_a \\ \mathfrak{D}_r \end{pmatrix} \geqslant \begin{pmatrix} 0 \\ -200.0e^{-9} \\ 0 \\ 0 \\ -1 \end{pmatrix}. \tag{6.45}$$

The upper limit of 200 ns on τ_m prevents the parameter from heading towards infinity when model failure occurs (see Chapter **??**). This can significantly decrease the computation time. To isolate the prolate spheroid the constraint

$$(1) \cdot (\mathfrak{D}_a) \geqslant (0), \tag{6.46}$$

is used whereas to isolate the oblate spheroid the constraint used is

$$(-1) \cdot (\mathfrak{D}_a) \geqslant (0). \tag{6.47}$$

Dependent on the model optimised, the matrix A and vector b are constructed from combinations of the above linear constraints.

Diagonal scaling

Model scaling can have a significant effect on the optimisation algorithm – a poorly scaled model can cause certain techniques to fail. When two parameters of the model lie on very different numeric scales the model is said to be poorly scaled. For example in model-free analysis the order of magnitude of the order parameters is one whereas for the internal correlation times the order of magnitude is between $1e^{-12}$ to $1e^{-8}$. Most effected are the trust region algorithms – the multidimensional sphere of trust will either be completely ineffective against the correlation time parameters or severely restrict optimisation in the order parameter dimensions. In model-free analysis the significant scaling disparity can even cause failure of optimisation due to amplified effects of machine precision. Therefore the model parameters need to be scaled.

This can be done by supplying the optimisation algorithm with the scaled rather than unscaled parameters. When the chi-squared function, gradient, and Hessian are called the vector is then premultiplied with a diagonal matrix in which the diagonal elements are the scaling factors. For the model-free analysis the scaling factor of one was used for the order parameter and a scaling factor of $1e^{-12}$ was used for the correlation times. The R_{ex} parameter was scaled to be the chemical exchange rate of the first field strength. The scaling matrix for the parameters $\{S^2, S_f^2, S_s^2, \tau_e, \tau_f, \tau_s, R_{ex}, r, CSA\}$ of individual residues is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e^{-12} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e^{-12} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e^{-12} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & (2\pi\omega_H)^{-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-4} \end{pmatrix}. \tag{6.48}$$

For the ellipsoidal diffusion parameters $\{\tau_m, \mathfrak{D}_a, \mathfrak{D}_r, \alpha, \beta, \gamma\}$ the scaling matrix is

$$\begin{pmatrix} 1e^{-12} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e^{7} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.49}$$

For the spheroidal diffusion parameters $\{\tau_m, \mathfrak{D}_a, \theta, \phi\}$ the scaling matrix is

$$\begin{pmatrix} 1e^{-12} & 0 & 0 & 0 \\ 0 & 1e^{7} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.50}$$

6.2 Optimisation of a single model-free model

6.2.1 The sample script

The sample script which demonstrates the optimisation of model-free model $m4$ which consists of the parameters $\{S^2, \tau_e, R_{ex}\}$ is `'model-free.py'`. The text of the script is:

```
# Script for model-free analysis.

# Create the data pipe.
name = 'm4'
pipe.create(name, 'mf')

# Load the sequence.
sequence.read('noe.500.out')

# Load the relaxation data.
relax_data.read('R1', '600', 600.0 * 1e6, 'r1.600.out')
relax_data.read('R2', '600', 600.0 * 1e6, 'r2.600.out')
relax_data.read('NOE', '600', 600.0 * 1e6, 'noe.600.out')
relax_data.read('R1', '500', 500.0 * 1e6, 'r1.500.out')
relax_data.read('R2', '500', 500.0 * 1e6, 'r2.500.out')
relax_data.read('NOE', '500', 500.0 * 1e6, 'noe.500.out')

# Setup other values.
diffusion_tensor.init(10e-9, fixed=True)
value.set(1.02 * 1e-10, 'bond_length')
value.set(-160 * 1e-6, 'csa')
value.set('15N', 'heteronucleus')
value.set('1H', 'proton')

# Select the model-free model.
```

```
model_free.select_model(model=name)

# Grid search.
grid_search(inc=11)

# Minimise.
minimise('newton')

# Monte Carlo simulations.
monte_carlo.setup(number=100)
monte_carlo.create_data()
monte_carlo.initial_values()
minimise('newton')
eliminate()
monte_carlo.error_analysis()

# Finish.
results.write(file='results', force=True)
state.save('save', force=True)
```

6.2.2 The rest

Please write me!

Until this section is completed please look at the sample script `'model-free.py'`.

6.3 Optimisation of all model-free models

6.3.1 The sample script

The sample script which demonstrates the optimisation of all model-free models from $m0$ to $m9$ of individual residues is `'mf_multimodel.py'`. The important parts of the script are:

```
# Set the data pipe names (also the names of preset model-free models).
pipes = ['m0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9']

# Loop over the pipes.
for name in pipes:
    # Create the data pipe.
    pipe.create(name, 'mf')

    # Load the sequence.
    sequence.read('noe.500.out')

    # Load the relaxation data.
    relax_data.read('R1', '600', 600.0 * 1e6, 'r1.600.out')
```

```
    relax_data.read('R2', '600', 600.0 * 1e6, 'r2.600.out')
    relax_data.read('NOE', '600', 600.0 * 1e6, 'noe.600.out')
    relax_data.read('R1', '500', 500.0 * 1e6, 'r1.500.out')
    relax_data.read('R2', '500', 500.0 * 1e6, 'r2.500.out')
    relax_data.read('NOE', '500', 500.0 * 1e6, 'noe.500.out')

    # Setup other values.
    diffusion_tensor.init(1e-8, fixed=True)
    value.set(1.02 * 1e-10, 'bond_length')
    value.set(-160 * 1e-6, 'csa')
    value.set('15N', 'heteronucleus')
    value.set('1H', 'proton')

    # Select the model-free model.
    model_free.select_model(model=name)

    # Minimise.
    grid_search(inc=11)
    minimise('newton')

    # Write the results.
    results.write(file='results', force=True)

# Save the program state.
state.save('save', force=True)
```

6.3.2 The rest

Please write me!

Until this section is completed please look at the sample script '`mf_multimodel.py`'.

6.4 Model-free model selection

6.4.1 The sample script

The sample script which demonstrates both model-free model elimination and model-free model selection between models from $m0$ to $m9$ is '`modsel.py`'. The text of the script is:

```
# Set the data pipe names.
pipes = ['m0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9']

# Loop over the data pipe names.
for name in pipes:
    print ''\n\n# '' + name + '' #''

    # Create the data pipe.
```

```
    pipe.create(name, 'mf')

    # Reload precalculated results from the file 'm1/results', etc.
    results.read(file='results', dir=name)

# Model elimination.
eliminate()

# Model selection.
model_selection(method='AIC', modsel_pipe='aic')

# Write the results.
state.save('save', force=True)
results.write(file='results', force=True)
```

6.4.2 The rest

Please write me!

Until this section is completed please look at the sample script `'modsel.py'`.

6.5 The methodology of Mandel et al., 1995

By presenting a systematic methodology for obtaining a consistent model-free description of the dynamics of the system, the manuscript of Mandel et al. (1995) revolutionised the application of model-free analysis. The full protocol is presented in Figure 6.1.

All of the data analysis techniques required for this protocol can be implemented within relax. The chi-squared distributions required for the chi-squared tests are constructed by Modelfree4 from the Monte Carlo simulations. If the optimisation algorithms and Monte Carlo simulations built into relax are utilised, then the relax script will need to construct the chi-squared distributions from the results as this is not yet coded into relax. The specific step-up hypothesis testing model selection of Mandel et al. (1995) is available through the `model_selection()` user function. Coding the rest of the protocol into a script should be straightforward.

6.6 The diffusion seeded paradigm

Ever since the original Lipari and Szabo papers (Lipari and Szabo, 1982a,b), the question of how to obtain the model-free description of the system has followed the route in which the diffusion tensor is initially estimated. Using this rough estimate, the model-free models are optimised for each spin system i, the best model selected, and then the global model $\mathfrak{S}$ of the diffusion model $\mathfrak{D}$ with each model-free model $\mathfrak{F}_i$ is optimised. This procedure is then repeated using the diffusion tensor parameters of $\mathfrak{S}$ as the initial input. Finally the global model is selected. The full protocol is illustrated in Figure 6.2.

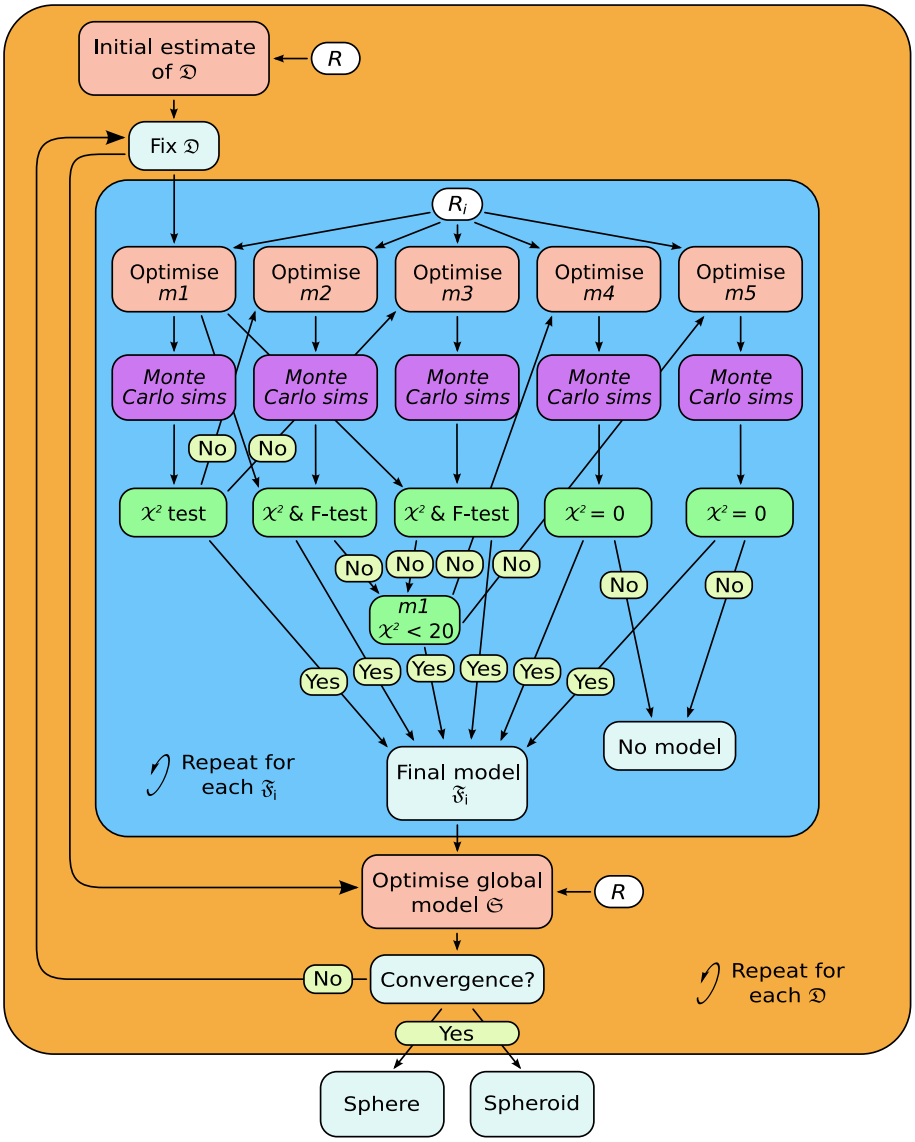

Figure 6.1: A schematic of the model-free optimisation protocol of Mandel et al. (1995). This specific protocol is for single field strength data. The initial diffusion tensor estimate is calculated using the R_2/R_1 ratio. The diffusion parameters of $\mathfrak{D}$ are held constant while model-free models $m1$ to $m5$ (6.22.1–6.22.5) of the set $\mathfrak{F}_i$ for each residue i are optimised and 500 Monte Carlo simulations executed. Using a web of ANOVA statistical tests, specifically χ^2 and F-tests, a step-up hypothesis testing model selection procedure is used to choose the best model-free model. These steps are repeated for all residues of the protein. The global model $\mathfrak{S}$, the union of $\mathfrak{D}$ and all $\mathfrak{F}_i$, is then optimised. These steps are repeated until convergence of the global model. The iterative process is repeated for both isotropic diffusion (sphere) and anisotropic diffusion (spheroid).

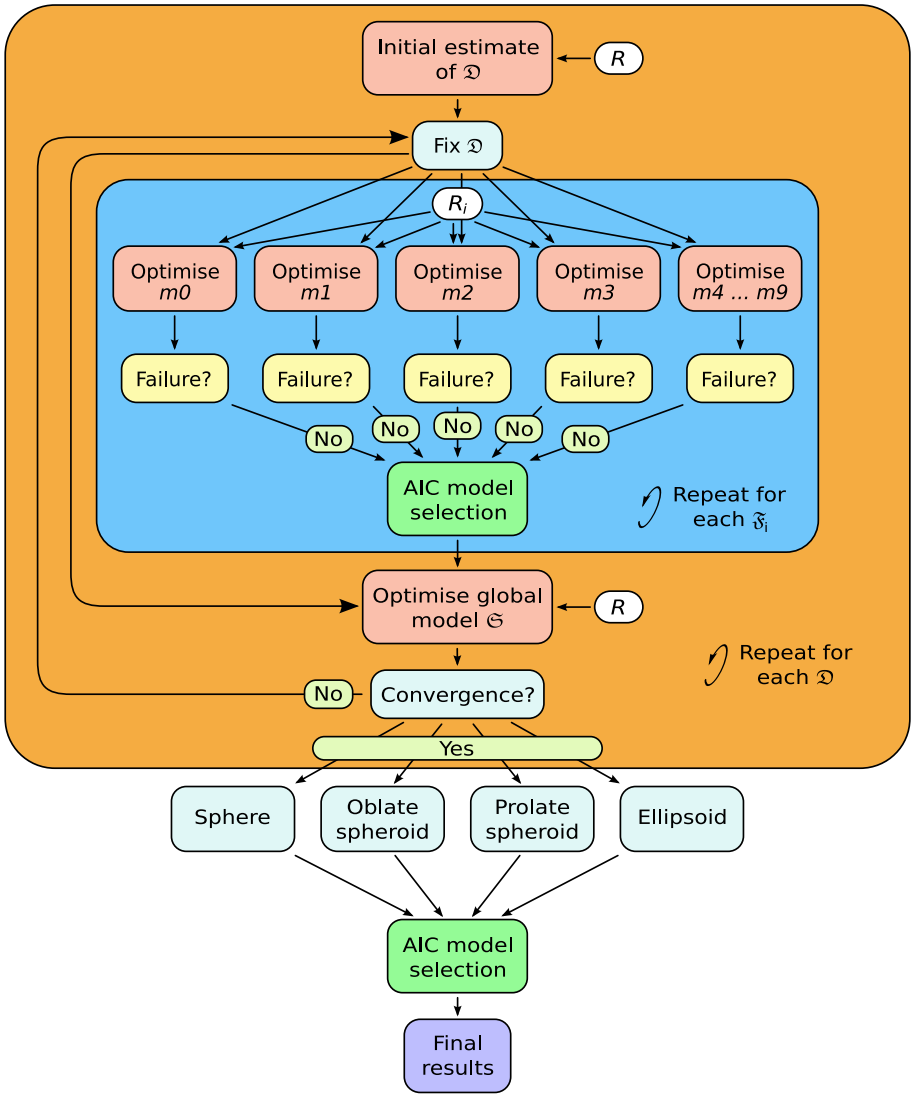

Figure 6.2: A schematic of model-free analysis using the diffusion seeded paradigm – the initial diffusion tensor estimate – together with AIC model selection and model elimination. The initial estimates of the parameters of $\mathfrak{D}$ are held constant while model-free models $m0$ to $m9$ (6.22.0–6.22.9) of the set $\mathfrak{F}_i$ for each spin system i are optimised, model elimination applied to remove failed models, and AIC model selection used to determine the best model. The global model $\mathfrak{S}$, the union of $\mathfrak{D}$ and all $\mathfrak{F}_i$, is then optimised. These steps are repeated until convergence of the global model. The entire iterative process is repeated for each of the Brownian diffusion models. Finally AIC model selection is used to determine the best description of the dynamics of the molecule by selecting between the global models $\mathfrak{S}$ including the sphere, oblate spheroid, prolate spheroid, and ellipsoid. Once the solution has been found, Monte Carlo simulations can be utilised for error analysis.

6.7 The new model-free optimisation protocol

Please write me!

Until this section is written please look at the sample script '`full_analysis.py`'. A description of the protocol is given at the top of the script. The protocol is summarised in Figure 6.3.

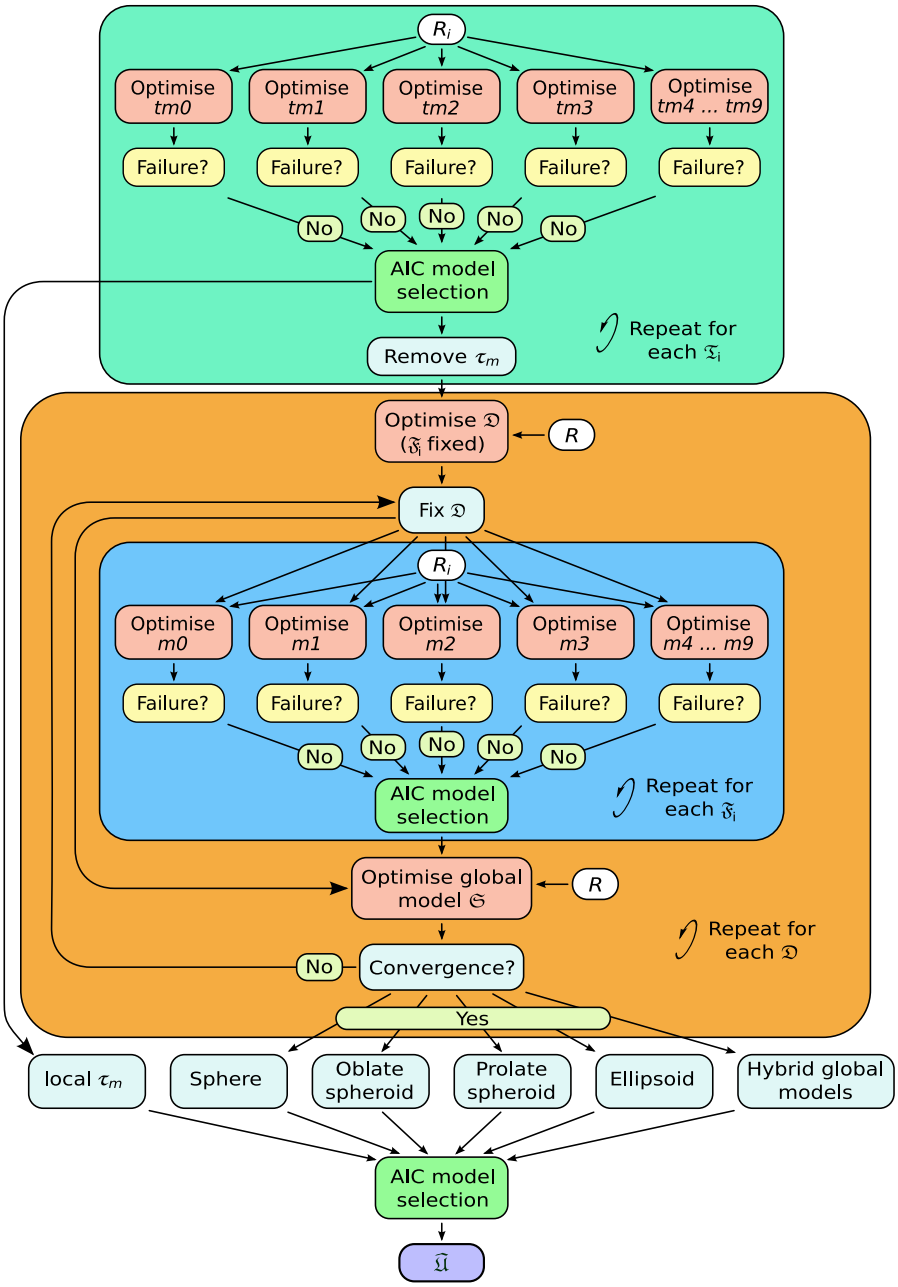

Figure 6.3: A schematic of the new model-free optimisation protocol. Initially models $tm0$ to $tm9$ (6.23.0–6.23.9) of the set $\mathfrak{T}_i$ for each spin system i are optimised, model elimination used to remove failed models, and AIC model selection used to pick the best model. Once all the $\mathfrak{T}_i$ have been determined for the system the the local τ_m parameter is removed, the model-free parameters are held fixed, and the global diffusion parameters of $\mathfrak{D}$ are optimised. These parameters are used as input for the central part of the schematic which follows the same procedure as that of Figure 6.2. Convergence is however precisely defined as identical models $\mathfrak{S}$, identical χ^2 values, and identical parameters θ between two iterations. The universal solution $\widehat{\mathfrak{U}}$, the best description of the dynamics of the molecule, is determined using AIC model selection to select between the local τ_m models for all spins, the sphere, oblate spheroid, prolate spheroid, ellipsoid, and possibly hybrid models whereby multiple diffusion tensors have been applied to different parts of the molecule.

Chapter 7

Reduced spectral density mapping

Please write me!

Until this chapter is written please look at the sample script '`jw_mapping.py`'.

Chapter 8

Values, gradients, and Hessians

8.1 Introduction

A word of warning before reading this chapter, the topics covered here are quite advanced and are not necessary for understanding how to either use relax or to implement any of the data analysis techniques present within relax. The material of this chapter is intended as an in-depth explanation of the mathematics involved in the optimisation of the parameters of the model-free models. As such it contains the chi-squared equation, relaxation equations, spectral density functions, and diffusion tensor equations as well as their gradients (the vector of first partial derivatives) and Hessians (the matrix of second partial derivatives). All these equations are used in the optimisation of models $m0$ to $m9$; models $tm0$ to $tm9$; the ellipsoidal, spheroidal, and spherical diffusion tensors; and the combination of the diffusion tensor and the model-free models.

8.2 Minimisation concepts

8.2.1 The function value

At the simplest level all minimisation techniques require at least a function which will supply a single value for different parameter values θ. For the modelling of NMR relaxation data this function is the chi-squared equation (6.1) on page 31. For certain algorithms, such a simplex minimisation, this single value suffices.

8.2.2 The gradient

The majority of minimisation algorithms also require the gradient at the point in the space represented by the parameter values θ. The gradient is a vector of partial derivatives and

is defined as

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial\theta_1} \\ \frac{\partial}{\partial\theta_2} \\ \vdots \\ \frac{\partial}{\partial\theta_n} \end{pmatrix} \tag{8.1}$$

where n is the total number of parameters in the model.

An example of a powerful algorithm which requires both the value and gradient at current parameter values is the BFGS quasi-Newton minimisation. The gradient is also essential for the use of the Method of Multipliers constraints algorithm (also known as the Augmented Lagrangian algorithm).

8.2.3 The Hessian

A few optimisation algorithms, which are among the most reliable for model-free analysis, additionally require the Hessian at current parameter values θ. The Hessian is the matrix of second partial derivatives and is defined as

$$\nabla^2 = \begin{pmatrix} \frac{\partial^2}{\partial{\theta_1}^2} & \frac{\partial^2}{\partial\theta_1\cdot\partial\theta_2} & \cdots & \frac{\partial^2}{\partial\theta_1\cdot\partial\theta_n} \\ \frac{\partial^2}{\partial\theta_2\cdot\partial\theta_1} & \frac{\partial^2}{\partial{\theta_2}^2} & \cdots & \frac{\partial^2}{\partial\theta_2\cdot\partial\theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial\theta_n\cdot\partial\theta_1} & \frac{\partial^2}{\partial\theta_n\cdot\partial\theta_2} & \cdots & \frac{\partial^2}{\partial{\theta_n}^2} \end{pmatrix}. \tag{8.2}$$

As the order in which the partial derivatives are calculated is inconsequential the Hessian is symmetric.

The most powerful minimisation algorithm for model-free analysis – Newton optimisation – requires the value, gradient, and Hessian at the current parameter values.

8.3 The four parameter combinations

In model-free analysis four different combinations of parameters can be optimised, each of which requires a different approach to the construction of the chi-squared value, gradient, and Hessian. These categories depend on whether the model-free parameter set $\mathfrak{F}$, the diffusion tensor parameter set $\mathfrak{D}$, or both sets are simultaneously optimised. The addition of the local τ_m parameter to the model-free set $\mathfrak{F}$ creates a fourth parameter combination.

8.3.1 Optimisation of the model-free models

This is the simplest category as it involves solely the optimisation of the model-free parameters of an individual residue while the diffusion tensor parameters are held constant. The model-free parameters belong to the set $\mathfrak{F}_i$ of the residue i. The models include $m0$ to $m9$ and the dimensionality is low with

$$\dim \mathfrak{F}_i = k \leqslant 5 \tag{8.3}$$

for the most complex model $m8 = \{S^2, \tau_f, S_f^2, \tau_s, R_{ex}\}$. The relaxation data of a single residue is used to build the chi-squared value, gradient, and Hessian.

8.3.2 Optimisation of the local τ_m models

The addition of the local τ_m parameter to the set $\mathfrak{F}_i$ creates a new set of models which will be labelled $\mathfrak{T}_i$. These include models $tm0$ to $tm9$. The local τ_m parameter is the single member of the set $\mathfrak{D}_i$ and in set notation

$$\mathfrak{T}_i = \mathfrak{D}_i \cup \mathfrak{F}_i. \tag{8.4}$$

Although the Brownian rotational diffusion parameter local τ_m is optimised, this category is residue specific. As such the complexity of the optimisation is lower than the next two categories. It is slightly greater than the optimisation of the set $\mathfrak{F}_i$ as

$$\dim \mathfrak{T}_i = 1 + k \leqslant 6, \tag{8.5}$$

where k is the number of model-free parameters.

8.3.3 Optimisation of the diffusion tensor parameters

The parameters of the Brownian rotational diffusion tensor belong to the set $\mathfrak{D}$. This set is the union of the geometric parameters $\mathfrak{G} = \{\mathfrak{D}_{iso}, \mathfrak{D}_a, \mathfrak{D}_r\}$ and the orientational parameters $\mathfrak{O}$,

$$\mathfrak{D} = \mathfrak{G} \cup \mathfrak{O}. \tag{8.6}$$

When diffusion is spherical solely the geometric parameter $\mathfrak{D}_{iso}$ is optimised. When the molecule diffuses as a spheroid the geometric parameters $\mathfrak{D}_{iso}$ and $\mathfrak{D}_a$ and the orientational parameters θ (the polar angle) and ϕ (the azimuthal angle) are optimised. If the molecule diffuses as an ellipsoid the geometric parameters $\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, and $\mathfrak{D}_r$ are optimised together with the Euler angles α, β, and γ.

This category is defined as the optimisation of solely the parameters of $\mathfrak{D}$. The model-free parameters of $\mathfrak{F}$ are held constant. As all selected residues of the macromolecule are involved in the optimisation, this category is global and can be more complex than the optimisation of $\mathfrak{F}_i$ or $\mathfrak{T}_i$. The dimensionality of the problem nevertheless low with

$$\dim \mathfrak{D} = 1, \qquad \dim \mathfrak{D} = 4, \qquad \dim \mathfrak{D} = 6, \tag{8.7}$$

for the diffusion as a sphere, spheroid, and ellipsoid respectively.

8.3.4 Optimisation of the global model $\mathfrak{S}$

The global model is defined as

$$\mathfrak{S} = \mathfrak{D} \cup \left(\bigcup_{i=1}^{l} \mathfrak{F}_i \right), \tag{8.8}$$

where i is the residue index and l is the total number of residues used in the analysis. This is the most complex of the four categories as both diffusion tensor parameters and model-free parameters of all selected residues are optimised simultaneously. The dimensionality of the model $\mathfrak{S}$ is much greater than the other categories and is equal to

$$\dim \mathfrak{S} = \dim \mathfrak{D} + \sum_{i=1}^{l} k_i \leqslant 6 + 5l, \tag{8.9}$$

where k_i is the number of model-free parameters for the residue i and is equal to $\dim \mathfrak{F}_i$, the number six corresponds to the maximum dimensionality of $\mathfrak{D}$, and the number five corresponds to the maximum dimensionality of $\mathfrak{F}_i$.

8.4 Construction of the values, gradients, and Hessians

8.4.1 The sum of chi-squared values

For the single residue models of $\mathfrak{F}_i$ and $\mathfrak{T}_i$ the chi-squared value χ_i^2 which is optimised is simply Equation (8.15) on page 62 in which the relaxation data is that of residue i. However for the global models $\mathfrak{D}$ and $\mathfrak{S}$ in which all selected residues are involved the optimised chi-squared value is the sum of those for each residue,

$$\chi^2 = \sum_{i=1}^{l} \chi_i^2, \tag{8.10}$$

where i is the residue index and l is the total number of residues used in the analysis. This is equivalent to Equation (8.15) when the index i ranges over the relaxation data of all selected residues.

8.4.2 Construction of the gradient

The construction of the gradient is significantly different for the models $\mathfrak{F}_i$, $\mathfrak{T}_i$, $\mathfrak{D}$, and $\mathfrak{S}$. In Figure 8.1 the construction of the chi-squared gradient $\nabla\chi^2$ for the global model $\mathfrak{S}$ is demonstrated. In this case

$$\nabla\chi^2 = \sum_{i=1}^{l} \nabla\chi_i^2, \tag{8.11}$$

where $\nabla\chi_i^2$ is the vector of partial derivatives of the chi-squared equation χ_i^2 for the residue i. The length of this vector is

$$\|\nabla\chi_i^2\| = \dim \mathfrak{S}, \tag{8.12}$$

with each position of the vector j equal to $\frac{\partial\chi_i^2}{\partial\theta_j}$ where each θ_j is a parameter of the model.

The construction of the gradient $\nabla\chi^2$ for the model $\mathfrak{D}$ is simply a subset of that of $\mathfrak{S}$. This is demonstrated in Figure 8.1 by simply taking the component of the gradient $\nabla\chi_i^2$

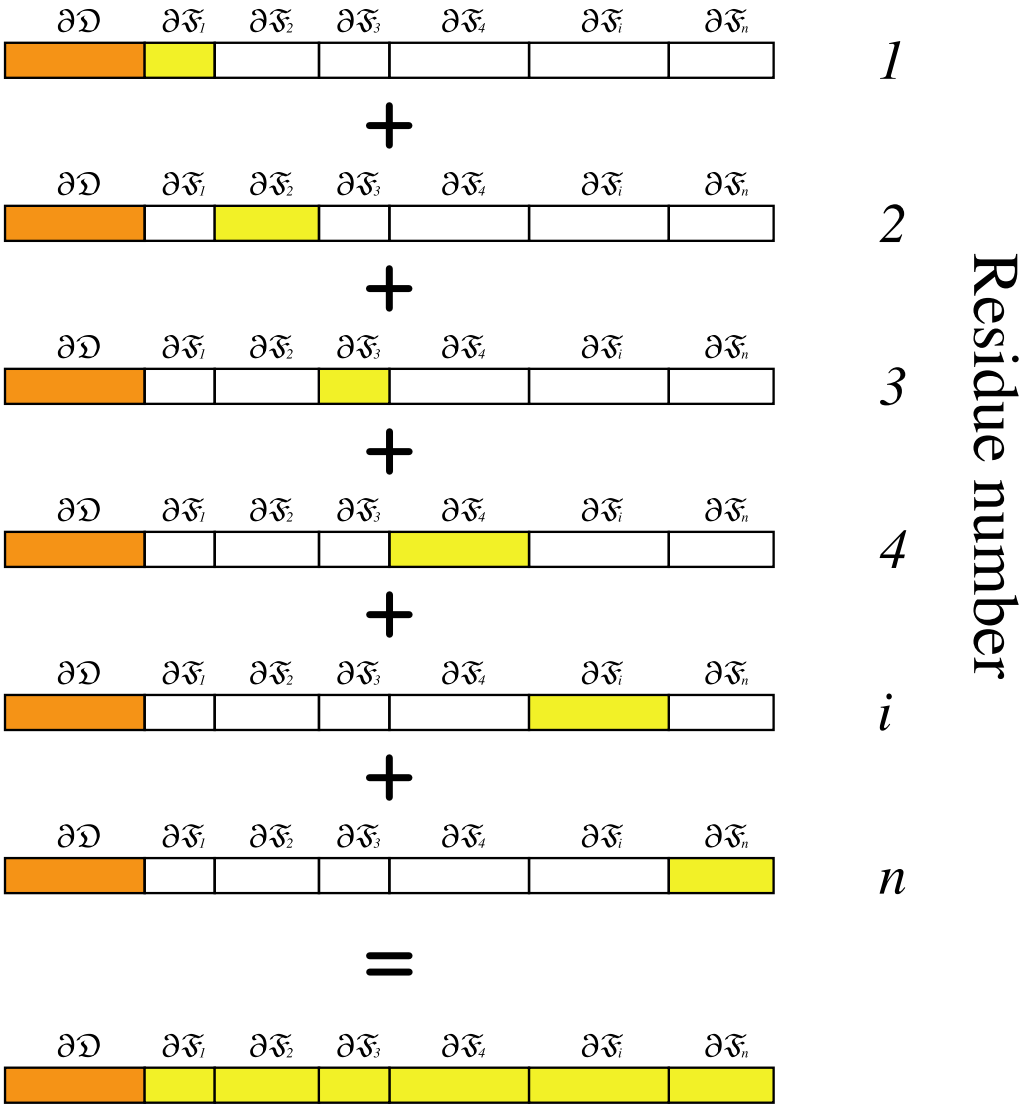

Figure 8.1: The construction of the model-free gradient $\nabla\chi^2$ for the global model $\mathfrak{S}$. For each residue i a different vector $\nabla\chi_i^2$ is constructed. The first element of the vector represented by the symbol $\partial\mathfrak{D}$ (the orange block) is the sub-vector of chi-squared partial derivatives with respect to each of the diffusion tensor parameters $\mathfrak{D}_j$. The rest of the elements, grouped into blocks for each residue denoted by the symbol $\partial\mathfrak{F}_i$, are the sub-vectors of chi-squared partial derivatives with respect to each of the model-free parameters $\mathfrak{F}_i^j$. For the residue dependent vector $\nabla\chi_i^2$ the partial derivatives with respect to the model-free parameters of $\mathfrak{F}_j$ where $i \neq j$ are zero. These blocks are left uncoloured. The complete gradient of $\mathfrak{S}$ is the sum of the vectors $\nabla\chi_i^2$.

denoted by the symbol $\partial\mathfrak{D}$ (the orange blocks) and summing these for all residues. This sum is given by (8.11) and

$$\|\nabla\chi_i^2\| = \dim\mathfrak{D}. \tag{8.13}$$

For the parameter set $\mathfrak{T}_i$, which consists of the local τ_m parameter and the model-free parameters of a single residue, the gradient $\nabla\chi_i^2$ for the residue i is simply the combination of the single orange block and single yellow block of the index i (Figure 8.1).

The model-free parameter set $\mathfrak{F}_i$ is even simpler. In Figure 8.1 the gradient $\nabla\chi_i^2$ is simply the vector denoted by the single yellow block for the residue i.

8.4.3 Construction of the Hessian

The construction of the Hessian for the models $\mathfrak{F}_i$, $\mathfrak{T}_i$, $\mathfrak{D}$, and $\mathfrak{S}$ is very similar to the procedure used for the gradient. The chi-squared Hessian for the global models $\mathfrak{D}$ and $\mathfrak{S}$ is

$$\nabla^2\chi^2 = \sum_{i=1}^{l} \nabla^2\chi_i^2. \tag{8.14}$$

Figure 8.2 demonstrates the construction of the full Hessian for the model $\mathfrak{S}$. The Hessian for the model $\mathfrak{D}$ is the sum of all the red blocks. The Hessian for the model $\mathfrak{T}_i$ is the combination of the single red block for residue i, the two orange blocks representing the sub-matrices of chi-squared second partial derivatives with respect to the diffusion parameter $\mathfrak{D}_j$ and the model-free parameter $\mathfrak{F}_i^k$, and the single yellow block for that residue. The Hessian for the model-free model $\mathfrak{F}_i$ is simply the sub-matrix for the residue i coloured yellow.

8.5 The value, gradient, and Hessian dependency chain

The dependency chain which was outlined in the model-free chapter – that the chi-squared function is dependent on the transformed relaxation equations which are dependent on the relaxation equations which themselves are dependent on the spectral density functions – combine with the values, gradients, and Hessians to create a complex web of dependencies. The relationship between all the values, gradients, and Hessians are outlined in Figure 8.3.

8.6 The χ^2 value, gradient, and Hessian

8.6.1 The χ^2 value

As was presented in Equation (6.1) on page 31 the χ^2 value is

$$\chi^2(\theta) = \sum_{i=1}^{n} \frac{(\mathrm{R}_i - \mathrm{R}_i(\theta))^2}{\sigma_i^2}, \tag{8.15}$$

where the summation index i ranges over all the relaxation data of all residues used in the analysis.

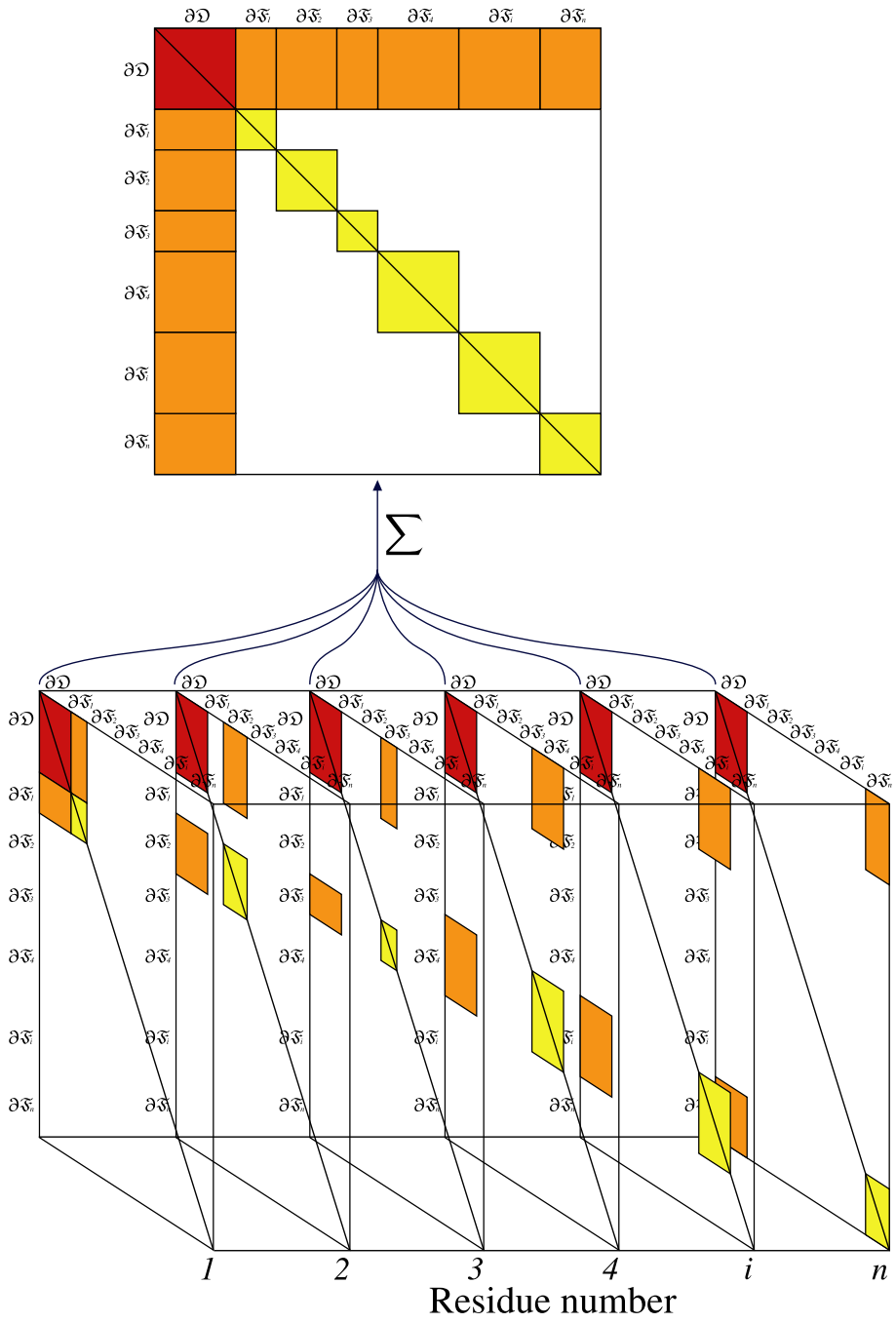

Figure 8.2: The model-free Hessian kite – a demonstration of the construction of the model-free Hessian $\nabla^2\chi^2$ for the global model $\mathfrak{S}$. For each residue i a different matrix $\nabla^2\chi_i^2$ is constructed. The first element of the matrix represented by the two symbols $\partial\mathfrak{D}$ (the red block) is the sub-matrix of chi-squared second partial derivatives with respect to the diffusion tensor parameters $\mathfrak{D}_j$ and $\mathfrak{D}_k$. The orange blocks are the sub-matrices of chi-squared second partial derivatives with respect to the diffusion parameter $\mathfrak{D}_j$ and the model-free parameter $\mathfrak{F}_i^k$. The yellow blocks are the sub-matrices of chi-squared second partial derivatives with respect to the model-free parameters $\mathfrak{F}_i^j$ and $\mathfrak{F}_i^k$. For the residue dependent matrix $\nabla^2\chi_i^2$ the second partial derivatives with respect to the model-free parameters $\mathfrak{F}_l^j$ and $\mathfrak{F}_l^k$ where $i \neq l$ are zero. In addition, the second partial derivatives with respect to the model-free parameters $\mathfrak{F}_i^j$ and $\mathfrak{F}_l^k$ where $i \neq l$ are also zero. These blocks of sub-matrices are left uncoloured. The complete Hessian of $\mathfrak{S}$ is the sum of the matrices $\nabla^2\chi_i^2$.

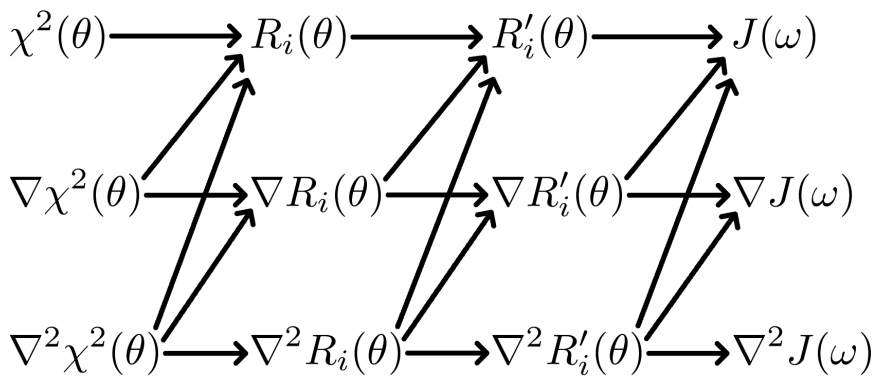

Figure 8.3: Dependencies between the χ^2, transformed relaxation, relaxation, and spectral density equations, gradients, and Hessians.

8.6.2 The χ^2 gradient

The χ^2 gradient in vector notation is

$$\nabla\chi^2(\theta) = 2\sum_{i=1}^{n}\frac{(\mathrm{R}_i - \mathrm{R}_i(\theta))^2}{\sigma_i^2}\nabla\mathrm{R}_i(\theta). \tag{8.16}$$

8.6.3 The χ^2 Hessian

The χ^2 Hessian in vector notation is

$$\nabla^2\chi^2(\theta) = 2\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\left(\nabla\mathrm{R}_i(\theta)\cdot\nabla\mathrm{R}_i(\theta)^T - (\mathrm{R}_i - \mathrm{R}_i(\theta))\nabla^2\mathrm{R}_i(\theta)\right). \tag{8.17}$$

8.7 The $\mathrm{R}_i(\theta)$ values, gradients, and Hessians

8.7.1 The $\mathrm{R}_i(\theta)$ values

The $\mathrm{R}_i(\theta)$ values are given by

$$\mathrm{R}_1(\theta) = \mathrm{R}_1'(\theta), \tag{8.18a}$$
$$\mathrm{R}_2(\theta) = \mathrm{R}_2'(\theta), \tag{8.18b}$$
$$\mathrm{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X}\frac{\sigma_{\mathrm{NOE}}(\theta)}{\mathrm{R}_1(\theta)}. \tag{8.18c}$$

8.7.2 The $\mathrm{R}_i(\theta)$ gradients

The $\mathrm{R}_i(\theta)$ gradients in vector notation are

$$\nabla\mathrm{R}_1(\theta) = \nabla\mathrm{R}_1'(\theta), \tag{8.19a}$$
$$\nabla\mathrm{R}_2(\theta) = \nabla\mathrm{R}_2'(\theta), \tag{8.19b}$$
$$\nabla\mathrm{NOE}(\theta) = \frac{\gamma_H}{\gamma_X}\frac{1}{\mathrm{R}_1(\theta)^2}\Big(\mathrm{R}_1(\theta)\nabla\sigma_{\mathrm{NOE}}(\theta) - \sigma_{\mathrm{NOE}}(\theta)\nabla\mathrm{R}_1(\theta)\Big). \tag{8.19c}$$

8.7.3 The $\mathrm{R}_i(\theta)$ Hessians

The $\mathrm{R}_i(\theta)$ Hessians in vector notation are

$$\nabla^2\mathrm{R}_1(\theta) = \nabla^2\mathrm{R}_1'(\theta), \tag{8.20a}$$
$$\nabla^2\mathrm{R}_2(\theta) = \nabla^2\mathrm{R}_2'(\theta), \tag{8.20b}$$
$$\begin{aligned}\nabla^2\mathrm{NOE}(\theta) = \frac{\gamma_H}{\gamma_X}\frac{1}{\mathrm{R}_1(\theta)^3}\Bigg[&\sigma_{\mathrm{NOE}}(\theta)\Big(2\nabla\mathrm{R}_1(\theta)\cdot\nabla\mathrm{R}_1(\theta)^T - \mathrm{R}_1(\theta)\nabla^2\mathrm{R}_1(\theta)\Big)\\ &- \mathrm{R}_1(\theta)\Big(\nabla\sigma_{\mathrm{NOE}}(\theta)\cdot\nabla\mathrm{R}_1(\theta)^T - \mathrm{R}_1(\theta)\nabla^2\sigma_{\mathrm{NOE}}(\theta)\Big)\Bigg].\end{aligned} \tag{8.20c}$$

8.8 $\mathrm{R}'_i(\theta)$ values, gradients, and Hessians

The partial and second partial derivatives of the relaxation equations of the set $\mathrm{R}'(\theta)$ are different for each parameter of the vector θ. The vector representation of the gradient $\nabla \mathrm{R}'_i(\theta)$ and the matrix representation of the Hessian $\nabla^2 \mathrm{R}'_i(\theta)$ can be reconstructed from the individual elements presented in the next section.

8.8.1 Components of the $\mathrm{R}'_i(\theta)$ equations

To simplify the calculations of the gradients and Hessians the $\mathrm{R}'_i(\theta)$ equations have been broken down into a number of components. These include the dipolar and CSA constants as well as the dipolar and CSA spectral density terms for each of the three transformed relaxation data types $\{\mathrm{R}_1, \mathrm{R}_2, \sigma_{\mathrm{NOE}}\}$. The segregation of these components simplifies the maths as many partial derivatives of the components are zero.

Dipolar constant

The dipolar constant is defined as

$$d = \frac{1}{4}\left(\frac{\mu_0}{4\pi}\right)^2 \frac{\left(\gamma_H \gamma_X \hbar\right)^2}{< r^6 >}. \tag{8.21}$$

This component of the relaxation equations is independent of the parameter of the spectral density function θ_j, the chemical exchange parameter ρ_{ex}, and the CSA parameter $\Delta\sigma$. Therefore the partial and second partial derivatives with respect to these parameters is zero. Only the derivative with respect to the bond length r is non-zero being

$$d' \equiv \frac{\mathrm{d}d}{\mathrm{d}r} = -\frac{3}{2}\left(\frac{\mu_0}{4\pi}\right)^2 \frac{\left(\gamma_H \gamma_X \hbar\right)^2}{< r^7 >}. \tag{8.22}$$

The second derivative with respect to the bond length is

$$d'' \equiv \frac{\mathrm{d}^2 d}{\mathrm{d}r^2} = \frac{21}{2}\left(\frac{\mu_0}{4\pi}\right)^2 \frac{\left(\gamma_H \gamma_X \hbar\right)^2}{< r^8 >}. \tag{8.23}$$

CSA constant

The CSA constant is defined as

$$c = \frac{\left(\omega_X \cdot \Delta\sigma\right)^2}{3}. \tag{8.24}$$

The partial derivative of this component with respect to all parameters but the CSA parameter $\Delta\sigma$ is zero. This derivative is

$$c' \equiv \frac{\mathrm{d}c}{\mathrm{d}\Delta\sigma} = \frac{2\omega_X^2 \cdot \Delta\sigma}{3}. \tag{8.25}$$

The CSA constant second derivative with respect to $\Delta\sigma$ is

$$c'' \equiv \frac{\mathrm{d}^2 c}{\mathrm{d}\Delta\sigma^2} = \frac{2\omega_X^2}{3}. \tag{8.26}$$

R_{ex} constant

The R_{ex} constant is defined as

$$R_{ex} = \rho_{ex}(2\pi\omega_H)^2. \tag{8.27}$$

The partial derivative of this component with respect to all parameters but the chemical exchange parameter ρ_{ex} is zero. This derivative is

$$R'_{ex} \equiv \frac{\mathrm{d}R_{ex}}{\mathrm{d}\rho_{ex}} = (2\pi\omega_H)^2. \tag{8.28}$$

The R_{ex} constant second derivative with respect to ρ_{ex} is

$$R''_{ex} \equiv \frac{\mathrm{d}^2R_{ex}}{\mathrm{d}\rho_{ex}^2} = 0. \tag{8.29}$$

Spectral density terms of the R_1 dipolar component

For the dipolar component of the R_1 equation (6.3a) on page 32 the spectral density terms are

$$J_d^{R_1} = J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H + \omega_X). \tag{8.30}$$

The partial derivative of these terms with respect to the spectral density function parameter θ_j is

$$J_d^{R_1\prime} \equiv \frac{\partial J_d^{R_1}}{\partial\theta_j} = \frac{\partial J(\omega_H - \omega_X)}{\partial\theta_j} + 3\frac{\partial J(\omega_X)}{\partial\theta_j} + 6\frac{\partial J(\omega_H + \omega_X)}{\partial\theta_j}. \tag{8.31}$$

The second partial derivative with respect to the spectral density function parameters θ_j and θ_k is

$$J_d^{R_1\prime\prime} \equiv \frac{\partial^2 J_d^{R_1}}{\partial\theta_j\cdot\partial\theta_k} = \frac{\partial^2 J(\omega_H - \omega_X)}{\partial\theta_j\cdot\partial\theta_k} + 3\frac{\partial^2 J(\omega_X)}{\partial\theta_j\cdot\partial\theta_k} + 6\frac{\partial^2 J(\omega_H + \omega_X)}{\partial\theta_j\cdot\partial\theta_k}. \tag{8.32}$$

Spectral density terms of the R_1 CSA component

For the CSA component of the R_1 equation (6.3a) on page 32 the spectral density terms are

$$J_c^{R_1} = J(\omega_X). \tag{8.33}$$

The partial derivative of these terms with respect to the spectral density function parameter θ_j is

$$J_c^{R_1\prime} \equiv \frac{\partial J_c^{R_1}}{\partial\theta_j} = \frac{\partial J(\omega_X)}{\partial\theta_j}. \tag{8.34}$$

The second partial derivative with respect to the spectral density function parameters θ_j and θ_k is

$$J_c^{R_1\prime\prime} \equiv \frac{\partial^2 J_c^{R_1}}{\partial\theta_j.\partial\theta_k} = \frac{\partial^2 J(\omega_X)}{\partial\theta_j\cdot\partial\theta_k}. \tag{8.35}$$

Spectral density terms of the R_2 dipolar component

For the dipolar component of the R_2 equation (6.3b) on page 32 the spectral density terms are

$$J_d^{R_2} = 4J(0) + J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H) + 6J(\omega_H + \omega_X). \tag{8.36}$$

The partial derivative of these terms with respect to the spectral density function parameter θ_j is

$$J_d^{R_2\prime} \equiv \frac{\partial J_d^{R_2}}{\partial \theta_j} = 4\frac{\partial J(0)}{\partial \theta_j} + \frac{\partial J(\omega_H - \omega_X)}{\partial \theta_j} + 3\frac{\partial J(\omega_X)}{\partial \theta_j} + 6\frac{\partial J(\omega_H)}{\partial \theta_j} + 6\frac{\partial J(\omega_H + \omega_X)}{\partial \theta_j}. \tag{8.37}$$

The second partial derivative with respect to the spectral density function parameters θ_j and θ_k is

$$J_d^{R_2\prime\prime} \equiv \frac{\partial^2 J_d^{R_2}}{\partial \theta_j \cdot \partial \theta_k} = 4\frac{\partial^2 J(0)}{\partial \theta_j \cdot \partial \theta_k} + \frac{\partial^2 J(\omega_H - \omega_X)}{\partial \theta_j \cdot \partial \theta_k} + 3\frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k} + 6\frac{\partial^2 J(\omega_H)}{\partial \theta_j \cdot \partial \theta_k} + 6\frac{\partial^2 J(\omega_H + \omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \tag{8.38}$$

Spectral density terms of the R_2 CSA component

For the CSA component of the R_2 equation (6.3b) on page 32 the spectral density terms are

$$J_c^{R_2} = 4J(0) + 3J(\omega_X). \tag{8.39}$$

The partial derivative of these terms with respect to the spectral density function parameter θ_j is

$$J_c^{R_2\prime} \equiv \frac{\partial J_c^{R_2}}{\partial \theta_j} = 4\frac{\partial J(0)}{\partial \theta_j} + 3\frac{\partial J(\omega_X)}{\partial \theta_j}. \tag{8.40}$$

The second partial derivative with respect to the spectral density function parameters θ_j and θ_k is

$$J_c^{R_2\prime\prime} \equiv \frac{\partial^2 J_c^{R_2}}{\partial \theta_j \cdot \partial \theta_k} = 4\frac{\partial^2 J(0)}{\partial \theta_j \cdot \partial \theta_k} + 3\frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \tag{8.41}$$

Spectral density terms of the σ_{NOE} dipolar component

For the dipolar component of the σ_{NOE} equation (6.3c) on page 32 the spectral density terms are

$$J_d^{\sigma_{NOE}} = 6J(\omega_H + \omega_X) - J(\omega_H - \omega_X). \tag{8.42}$$

The partial derivative of these terms with respect to the spectral density function parameter θ_j is

$$J_d^{\sigma_{NOE}\prime} \equiv \frac{\partial J_d^{\sigma_{NOE}}}{\partial \theta_j} = 6\frac{\partial J(\omega_H + \omega_X)}{\partial \theta_j} - \frac{\partial J(\omega_H - \omega_X)}{\partial \theta_j}. \tag{8.43}$$

The second partial derivative with respect to the spectral density function parameters θ_j and θ_k is

$$J_d^{\sigma_{\mathrm{NOE}}\prime\prime} \equiv \frac{\partial^2 J_d^{\sigma_{\mathrm{NOE}}}}{\partial\theta_j \cdot \partial\theta_k} = 6\frac{\partial^2 J(\omega_H + \omega_X)}{\partial\theta_j \cdot \partial\theta_k} - \frac{\partial^2 J(\omega_H - \omega_X)}{\partial\theta_j \cdot \partial\theta_k}. \tag{8.44}$$

8.8.2 $\mathrm{R}'_i(\theta)$ values

Using the components of the relaxation equations defined above the three relaxation equations can be re-expressed as

$$\mathrm{R}_1(\theta) = dJ_d^{\mathrm{R}_1} + cJ_c^{\mathrm{R}_1}, \tag{8.45a}$$

$$\mathrm{R}_2(\theta) = \frac{d}{2}J_d^{\mathrm{R}_2} + \frac{c}{6}J_c^{\mathrm{R}_2}, \tag{8.45b}$$

$$\sigma_{\mathrm{NOE}}(\theta) = dJ_d^{\sigma_{\mathrm{NOE}}}. \tag{8.45c}$$

8.8.3 $\mathrm{R}'_i(\theta)$ gradients

A different partial derivative exists for the spectral density function parameter θ_j, the chemical exchange parameter ρ_{ex}, CSA parameter $\Delta\sigma$, and bond length parameter r. In model-free analysis the spectral density parameters include both the parameters of the diffusion tensor and the parameters of the various model-free models.

θ_j partial derivative

The partial derivatives of the relaxation equations with respect to the spectral density function parameter θ_j are

$$\frac{\partial \mathrm{R}_1(\theta)}{\partial\theta_j} = dJ_d^{\mathrm{R}_1\prime} + cJ_c^{\mathrm{R}_1\prime}, \tag{8.46a}$$

$$\frac{\partial \mathrm{R}_2(\theta)}{\partial\theta_j} = \frac{d}{2}J_d^{\mathrm{R}_2\prime} + \frac{c}{6}J_c^{\mathrm{R}_2\prime}, \tag{8.46b}$$

$$\frac{\partial \sigma_{\mathrm{NOE}}(\theta)}{\partial\theta_j} = dJ_d^{\sigma_{\mathrm{NOE}}\prime}. \tag{8.46c}$$

ρ_{ex} partial derivative

The partial derivatives of the relaxation equations with respect to the chemical exchange parameter ρ_{ex} are

$$\frac{\partial \mathrm{R}_1(\theta)}{\partial\rho_{ex}} = 0, \tag{8.47a}$$

$$\frac{\partial \mathrm{R}_2(\theta)}{\partial\rho_{ex}} = (2\pi\omega_H)^2, \tag{8.47b}$$

$$\frac{\partial \sigma_{\mathrm{NOE}}(\theta)}{\partial\rho_{ex}} = 0. \tag{8.47c}$$

$\Delta\sigma$ partial derivative

The partial derivatives of the relaxation equations with respect to the CSA parameter $\Delta\sigma$ are

$$\frac{\partial \mathrm{R}_1(\theta)}{\partial \Delta\sigma} = c' J_c^{\mathrm{R}_1}, \tag{8.48a}$$

$$\frac{\partial \mathrm{R}_2(\theta)}{\partial \Delta\sigma} = \frac{c'}{6} J_c^{\mathrm{R}_2}, \tag{8.48b}$$

$$\frac{\partial \sigma_{\mathrm{NOE}}(\theta)}{\partial \Delta\sigma} = 0. \tag{8.48c}$$

r partial derivative

The partial derivatives of the relaxation equations with respect to the bond length parameter r are

$$\frac{\partial \mathrm{R}_1(\theta)}{\partial r} = d' J_d^{\mathrm{R}_1}, \tag{8.49a}$$

$$\frac{\partial \mathrm{R}_2(\theta)}{\partial r} = \frac{d'}{2} J_d^{\mathrm{R}_2}, \tag{8.49b}$$

$$\frac{\partial \sigma_{\mathrm{NOE}}(\theta)}{\partial r} = d' J_d^{\sigma_{\mathrm{NOE}}}. \tag{8.49c}$$

8.8.4 $\mathrm{R}_i'(\theta)$ Hessians

Again different second partial derivatives with respect to the spectral density function parameters θ_j and θ_k, the chemical exchange parameter ρ_{ex}, CSA parameter $\Delta\sigma$, and bond length parameter r. These second partial derivatives are the components of the $\mathrm{R}_i'(\theta)$ Hessian matrices.

$\theta_j - \theta_k$ partial derivative

The second partial derivatives of the relaxation equations with respect to the spectral density function parameters θ_j and θ_k are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial \theta_j \cdot \partial \theta_k} = d J_d^{\mathrm{R}_1\prime\prime} + c J_c^{\mathrm{R}_1\prime\prime}, \tag{8.50a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial \theta_j \cdot \partial \theta_k} = \frac{d}{2} J_d^{\mathrm{R}_2\prime\prime} + \frac{c}{6} J_c^{\mathrm{R}_2\prime\prime}, \tag{8.50b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial \theta_j \cdot \partial \theta_k} = d J_d^{\sigma_{\mathrm{NOE}}\prime\prime}. \tag{8.50c}$$

$\theta_j - \rho_{ex}$ partial derivative

The second partial derivatives of the relaxation equations with respect to the spectral density function parameter θ_j and the chemical exchange parameter ρ_{ex} are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\theta_j \cdot \partial\rho_{ex}} = 0, \tag{8.51a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\theta_j \cdot \partial\rho_{ex}} = 0, \tag{8.51b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\theta_j \cdot \partial\rho_{ex}} = 0. \tag{8.51c}$$

$\theta_j - \Delta\sigma$ partial derivative

The second partial derivatives of the relaxation equations with respect to the spectral density function parameter θ_j and the CSA parameter $\Delta\sigma$ are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\theta_j \cdot \partial\Delta\sigma} = c' J_c^{\mathrm{R}_1\prime}, \tag{8.52a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\theta_j \cdot \partial\Delta\sigma} = \frac{c'}{6} J_c^{\mathrm{R}_2\prime}, \tag{8.52b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\theta_j \cdot \partial\Delta\sigma} = 0. \tag{8.52c}$$

$\theta_j - r$ partial derivative

The second partial derivatives of the relaxation equations with respect to the spectral density function parameter θ_j and the bond length parameter r are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\theta_j \cdot \partial r} = d' J_d^{\mathrm{R}_1\prime}, \tag{8.53a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\theta_j \cdot \partial r} = \frac{d'}{2} J_d^{\mathrm{R}_2\prime}, \tag{8.53b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\theta_j \cdot \partial r} = d' J_d^{\sigma_{\mathrm{NOE}}\prime}. \tag{8.53c}$$

$\rho_{ex} - \rho_{ex}$ partial derivative

The second partial derivatives of the relaxation equations with respect to the chemical exchange parameter ρ_{ex} twice are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial{\rho_{ex}}^2} = 0, \tag{8.54a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial{\rho_{ex}}^2} = 0, \tag{8.54b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial{\rho_{ex}}^2} = 0. \tag{8.54c}$$

$\rho_{ex} - \Delta\sigma$ partial derivative

The second partial derivatives of the relaxation equations with respect to the chemical exchange parameter ρ_{ex} and the CSA parameter $\Delta\sigma$ are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\rho_{ex} \cdot \partial\Delta\sigma} = 0, \tag{8.55a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\rho_{ex} \cdot \partial\Delta\sigma} = 0, \tag{8.55b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\rho_{ex} \cdot \partial\Delta\sigma} = 0. \tag{8.55c}$$

$\rho_{ex} - r$ partial derivative

The second partial derivatives of the relaxation equations with respect to the chemical exchange parameter ρ_{ex} and the bond length parameter r are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\rho_{ex} \cdot \partial r} = 0, \tag{8.56a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\rho_{ex} \cdot \partial r} = 0, \tag{8.56b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\rho_{ex} \cdot \partial r} = 0. \tag{8.56c}$$

$\Delta\sigma - \Delta\sigma$ partial derivative

The second partial derivatives of the relaxation equations with respect to the CSA parameter $\Delta\sigma$ twice are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\Delta\sigma^2} = c'' J_c^{\mathrm{R}_1}, \tag{8.57a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\Delta\sigma^2} = \frac{c''}{6} J_c^{\mathrm{R}_2}, \tag{8.57b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\Delta\sigma^2} = 0. \tag{8.57c}$$

$\Delta\sigma - r$ partial derivative

The second partial derivatives of the relaxation equations with respect to the CSA parameter $\Delta\sigma$ and the bond length parameter r are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial\Delta\sigma \cdot \partial r} = 0, \tag{8.58a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial\Delta\sigma \cdot \partial r} = 0, \tag{8.58b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial\Delta\sigma \cdot \partial r} = 0. \tag{8.58c}$$

$r - r$ partial derivative

The second partial derivatives of the relaxation equations with respect to the bond length parameter r twice are

$$\frac{\partial^2 \mathrm{R}_1(\theta)}{\partial r^2} = d'' J_d^{\mathrm{R}_1}, \tag{8.59a}$$

$$\frac{\partial^2 \mathrm{R}_2(\theta)}{\partial r^2} = \frac{d''}{2} J_d^{\mathrm{R}_2}, \tag{8.59b}$$

$$\frac{\partial^2 \sigma_{\mathrm{NOE}}(\theta)}{\partial r^2} = d'' J_d^{\sigma_{\mathrm{NOE}}}. \tag{8.59c}$$

8.9 Model-free analysis

8.9.1 The model-free equations

In the original model-free analysis of Lipari and Szabo (1982a) the correlation function $C(\tau)$ of the XH bond vector is approximated by decoupling the internal fluctuations of the bond vector $C_{\mathrm{I}}(\tau)$ from the correlation function of the overall Brownian rotational diffusion $C_{\mathrm{O}}(\tau)$ by the equation

$$C(\tau) = C_{\mathrm{O}}(\tau) \cdot C_{\mathrm{I}}(\tau). \tag{8.60}$$

The overall correlation functions of the diffusion of a sphere, spheroid, and ellipsoid are presented respectively in section 8.10.1 on page 93, section 8.11.1 on page 106, and section 8.12.1 on page 110. These three different equations can be combined into one generic correlation function which is independent of the type of diffusion. This generic correlation function is

$$C_{\mathrm{O}}(\tau) = \frac{1}{5} \sum_{i=-k}^{k} c_i \cdot e^{-\tau/\tau_i}, \tag{8.61}$$

where c_i are the weights and τ_i are correlation times of the exponential terms. In the original model-free analysis of Lipari and Szabo (1982a,b) the internal motions are modelled by the correlation function

$$C_{\mathrm{I}}(\tau) = S^2 + (1 - S^2)e^{-\tau/\tau_e}, \tag{8.62}$$

where S^2 is the generalised Lipari and Szabo order parameter which is related to the amplitude of the motion and τ_e is the effective correlation time which is an indicator of the timescale of the motion, albeit being dependent on the value of the order parameter. The order parameter ranges from one for complete rigidity to zero for unrestricted motions. Model-free theory was extended by Clore et al. (1990) to include motions on two timescales by the correlation function

$$C_{\mathrm{I}}(\tau) = S^2 + (1 - S_f^2)e^{-\tau/\tau_f} + (S_f^2 - S^2)e^{-\tau/\tau_s}, \tag{8.63}$$

where the faster of the motions is defined by the order parameter S_f^2 and the correlation time τ_f, the slower by the parameters S_s^2 and τ_s, and the two order parameter are related by the equation $S^2 = S_f^2 \cdot S_s^2$.

The relaxation equations of Abragam (1961) are composed of a sum of power spectral density functions $J(\omega)$ at five frequencies. The spectral density function is related to the correlation function as the two are a Fourier pair. Applying the Fourier transform to the correlation function composed of the generic diffusion equation and the original model-free correlation function results in the equation

$$J(\omega) = \frac{2}{5} \sum_{i=-k}^{k} c_i \cdot \tau_i \left(\frac{S^2}{1 + (\omega\tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega\tau_e\tau_i)^2} \right). \tag{8.64}$$

The Fourier transform using the extended model-free correlation function is

$$J(\omega) = \frac{2}{5}\sum_{i=-k}^{k} c_i \cdot \tau_i \left(\frac{S^2}{1+(\omega\tau_i)^2} + \frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right). \quad (8.65)$$

8.9.2 The original model-free gradient

The model-free gradient of the original spectral density function (8.64) is the vector of partial derivatives of the function with respect to the geometric parameter $\mathfrak{G}_i$, the orientational parameter $\mathfrak{O}_i$, the order parameter S^2, and the internal correlation time τ_e. The positions in the vector correspond to the model parameters which are being optimised.

$\mathfrak{G}_j$ partial derivative

The partial derivative of (8.64) with respect to the geometric parameter $\mathfrak{G}_j$ is

$$\frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = \frac{2}{5}\sum_{i=-k}^{k} \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S^2 \frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} + (1-S^2)\tau_e^2 \frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^2} \right) + \frac{\partial c_i}{\partial \mathfrak{G}_j}\tau_i \left(\frac{S^2}{1+(\omega\tau_i)^2} + \frac{(1-S^2)(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2} \right) \right). \quad (8.66)$$

$\mathfrak{O}_j$ partial derivative

The partial derivative of (8.64) with respect to the orientational parameter $\mathfrak{O}_j$ is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5}\sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j}\tau_i \left(\frac{S^2}{1+(\omega\tau_i)^2} + \frac{(1-S^2)(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2} \right). \quad (8.67)$$

S^2 partial derivative

The partial derivative of (8.64) with respect to the order parameter S^2 is

$$\frac{\partial J(\omega)}{\partial S^2} = \frac{2}{5}\sum_{i=-k}^{k} c_i\tau_i \left(\frac{1}{1+(\omega\tau_i)^2} - \frac{(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2} \right). \quad (8.68)$$

τ_e partial derivative

The partial derivative of (8.64) with respect to the correlation time τ_e is

$$\frac{\partial J(\omega)}{\partial \tau_e} = \frac{2}{5}(1-S^2)\sum_{i=-k}^{k} c_i\tau_i^2 \frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^2}. \quad (8.69)$$

8.9.3 The original model-free Hessian

The model-free Hessian of the original spectral density function (8.64) is the matrix of second partial derivatives. The matrix coordinates correspond to the model parameters which are being optimised.

$\mathfrak{G}_j - \mathfrak{G}_k$ partial derivative

The second partial derivative of (8.64) with respect to the geometric parameters $\mathfrak{G}_j$ and $\mathfrak{G}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&- 2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j}\cdot\frac{\partial \tau_i}{\partial \mathfrak{G}_k}\Bigg(S^2\omega^2\tau_i\frac{3-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^3} \\ &+ (1-S^2)\tau_e^2\frac{(\tau_e+\tau_i)^3+3\omega^2\tau_e^3\tau_i(\tau_e+\tau_i)-(\omega\tau_e)^4\tau_i^3}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^3}\Bigg) \\ &+ \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\cdot\frac{\partial c_i}{\partial \mathfrak{G}_k}+\frac{\partial \tau_i}{\partial \mathfrak{G}_k}\cdot\frac{\partial c_i}{\partial \mathfrak{G}_j}+c_i\frac{\partial^2\tau_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{G}_k}\right)\Bigg(S^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} \\ &+ (1-S^2)\tau_e^2\frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^2}\Bigg) \\ &+ \left(\frac{\partial^2 c_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{G}_k}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S^2)(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2}\right)\right)\Bigg). \end{aligned} \quad (8.70)$$

$\mathfrak{G}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.64) with respect to the geometric parameter $\mathfrak{G}_j$ and the orientational parameter $\mathfrak{O}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\frac{\partial c_i}{\partial \mathfrak{O}_k}\left(S^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2}+(1-S^2)\tau_e^2\frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^2}\right) \\ &+ \frac{\partial^2 c_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{O}_k}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S^2)(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2}\right)\Bigg). \end{aligned} \quad (8.71)$$

$\mathfrak{G}_j - S^2$ partial derivative

The second partial derivative of (8.64) with respect to the geometric parameter $\mathfrak{G}_j$ and the order parameter S^2 is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S^2} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&c_i\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\left(\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2}-\tau_e^2\frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2)^2}\right) \\ &+ \frac{\partial c_i}{\partial \mathfrak{G}_j}\tau_i\left(\frac{1}{1+(\omega\tau_i)^2}-\frac{(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2}\right)\Bigg). \end{aligned} \quad (8.72)$$

$\mathfrak{G}_j - \tau_e$ partial derivative

The second partial derivative of (8.64) with respect to the geometric parameter $\mathfrak{G}_j$ and the correlation time τ_e is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_e} = \frac{2}{5}(1-S^2)\sum_{i=-k}^{k}\left(2c_i\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\tau_e\tau_i(\tau_e+\tau_i)\frac{(\tau_e+\tau_i)^2-3(\omega\tau_e\tau_i)^2}{\left((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2\right)^3}\right.$$
$$\left.+\frac{\partial c_i}{\partial \mathfrak{G}_j}\tau_i^2\frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{\left((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2\right)^2}\right). \quad (8.73)$$

$\mathfrak{O}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.64) with respect to the orientational parameters $\mathfrak{O}_j$ and $\mathfrak{O}_k$ is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5}\sum_{i=-k}^{k}\frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S^2)(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2}\right). \quad (8.74)$$

$\mathfrak{O}_j - S^2$ partial derivative

The second partial derivative of (8.64) with respect to the orientational parameter $\mathfrak{O}_j$ and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S^2} = \frac{2}{5}\sum_{i=-k}^{k}\frac{\partial c_i}{\partial \mathfrak{O}_j}\tau_i\left(\frac{1}{1+(\omega\tau_i)^2}-\frac{(\tau_e+\tau_i)\tau_e}{(\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2}\right). \quad (8.75)$$

$\mathfrak{O}_j - \tau_e$ partial derivative

The second partial derivative of (8.64) with respect to the orientational parameter $\mathfrak{O}_j$ and the correlation time τ_e is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_e} = \frac{2}{5}(1-S^2)\sum_{i=-k}^{k}\frac{\partial c_i}{\partial \mathfrak{O}_j}\tau_i^2\frac{(\tau_e+\tau_i)^2-(\omega\tau_e\tau_i)^2}{\left((\tau_e+\tau_i)^2+(\omega\tau_e\tau_i)^2\right)^2}. \quad (8.76)$$

$S^2 - S^2$ partial derivative

The second partial derivative of (8.64) with respect to the order parameter S^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S^2)^2} = 0. \quad (8.77)$$

$S^2 - \tau_e$ partial derivative

The second partial derivative of (8.64) with respect to the order parameter S^2 and correlation time τ_e is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_e} = -\frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2}. \tag{8.78}$$

$\tau_e - \tau_e$ partial derivative

The second partial derivative of (8.64) with respect to the correlation time τ_e twice is

$$\frac{\partial^2 J(\omega)}{\partial {\tau_e}^2} = -\frac{4}{5}(1 - S^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_e + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_e (\tau_e + \tau_i) - (\omega \tau_i)^4 \tau_e^3}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^3} \tag{8.79}$$

8.9.4 The extended model-free gradient

The model-free gradient of the extended spectral density function (8.65) is the vector of partial derivatives of the function with respect to the geometric parameter $\mathfrak{G}_i$, the orientational parameter $\mathfrak{O}_i$, the order parameters S^2 and S_f^2, and the internal correlation times τ_f and τ_s. The positions in the vector correspond to the model parameters which are being optimised.

$\mathfrak{G}_j$ partial derivative

The partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ is

$$\begin{aligned}\frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(c_i\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\Bigg(S^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} \\ + (1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2} \\ + (S_f^2-S^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2}\Bigg) \\ + \frac{\partial c_i}{\partial \mathfrak{G}_j}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right)\Bigg).\end{aligned} \tag{8.80}$$

$\mathfrak{O}_j$ partial derivative

The partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5}\sum_{i=-k}^{k}\frac{\partial c_i}{\partial \mathfrak{O}_j}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right). \tag{8.81}$$

S^2 partial derivative

The partial derivative of (8.65) with respect to the order parameter S^2 is

$$\frac{\partial J(\omega)}{\partial S^2} = \frac{2}{5}\sum_{i=-k}^{k}c_i\tau_i\left(\frac{1}{1+(\omega\tau_i)^2}-\frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right). \tag{8.82}$$

S_f^2 partial derivative

The partial derivative of (8.65) with respect to the order parameter S_f^2 is

$$\frac{\partial J(\omega)}{\partial S_f^2} = -\frac{2}{5}\sum_{i=-k}^{k}c_i\tau_i\left(\frac{(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}-\frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right). \tag{8.83}$$

τ_f partial derivative

The partial derivative of (8.65) with respect to the correlation time τ_f is

$$\frac{\partial J(\omega)}{\partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \tag{8.84}$$

τ_s partial derivative

The partial derivative of (8.65) with respect to the correlation time τ_s is

$$\frac{\partial J(\omega)}{\partial \tau_s} = \frac{2}{5}(S_f^2 - S^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.85}$$

8.9.5 The extended model-free Hessian

The model-free Hessian of the extended spectral density function (8.65) is the matrix of second partial derivatives. The matrix coordinates correspond to the model parameters which are being optimised.

$\mathfrak{G}_j - \mathfrak{G}_k$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameters $\mathfrak{G}_j$ and $\mathfrak{G}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial\mathfrak{G}_j\cdot\partial\mathfrak{G}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&-2c_i\frac{\partial\tau_i}{\partial\mathfrak{G}_j}\cdot\frac{\partial\tau_i}{\partial\mathfrak{G}_k}\Bigg(S^2\omega^2\tau_i\frac{3-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^3}\\
&+(1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^3+3\omega^2\tau_f^3\tau_i(\tau_f+\tau_i)-(\omega\tau_f)^4\tau_i^3}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^3}\\
&+(S_f^2-S^2)\tau_s^2\frac{(\tau_s+\tau_i)^3+3\omega^2\tau_s^3\tau_i(\tau_s+\tau_i)-(\omega\tau_s)^4\tau_i^3}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^3}\Bigg)\\
&+\left(\frac{\partial\tau_i}{\partial\mathfrak{G}_j}\cdot\frac{\partial c_i}{\partial\mathfrak{G}_k}+\frac{\partial\tau_i}{\partial\mathfrak{G}_k}\cdot\frac{\partial c_i}{\partial\mathfrak{G}_j}+c_i\frac{\partial^2\tau_i}{\partial\mathfrak{G}_j\cdot\partial\mathfrak{G}_k}\right)\Bigg(S^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2}\\
&+(1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2}\\
&+(S_f^2-S^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2}\Bigg)\\
&+\left(\frac{\partial^2 c_i}{\partial\mathfrak{G}_j\cdot\partial\mathfrak{G}_k}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right)\right)\Bigg).\end{aligned} \tag{8.86}$$

$\mathfrak{G}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the orientational parameter $\mathfrak{O}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial\mathfrak{G}_j\cdot\partial\mathfrak{O}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&\frac{\partial\tau_i}{\partial\mathfrak{G}_j}\frac{\partial c_i}{\partial\mathfrak{O}_k}\Bigg(S^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2}\\
&+(1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2}\\
&+(S_f^2-S^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2}\Bigg)\\
&+\frac{\partial^2 c_i}{\partial\mathfrak{G}_j\cdot\partial\mathfrak{O}_k}\tau_i\left(\frac{S^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right)\Bigg).\end{aligned} \tag{8.87}$$

$\mathfrak{G}_j - S^2$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S^2} = \frac{2}{5} \sum_{i=-k}^{k} \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\frac{1 - (\omega\tau_i)^2}{(1 + (\omega\tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega\tau_s\tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2)^2} \right) + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{1}{1 + (\omega\tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2} \right) \right). \quad (8.88)$$

$\mathfrak{G}_j - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_f^2} = -\frac{2}{5} \sum_{i=-k}^{k} \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega\tau_f\tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega\tau_f\tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega\tau_s\tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2)^2} \right) + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega\tau_f\tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2} \right) \right). \quad (8.89)$$

$\mathfrak{G}_j - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^{k} \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_f\tau_i(\tau_f + \tau_i) \frac{(\tau_f + \tau_i)^2 - 3(\omega\tau_f\tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega\tau_f\tau_i)^2)^3} + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega\tau_f\tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega\tau_f\tau_i)^2)^2} \right). \quad (8.90)$$

$\mathfrak{G}_j - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_s} = \frac{2}{5}(S_f^2 - S^2) \sum_{i=-k}^{k} \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_s\tau_i(\tau_s + \tau_i) \frac{(\tau_s + \tau_i)^2 - 3(\omega\tau_s\tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2)^3} + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega\tau_s\tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2)^2} \right). \quad (8.91)$$

$\mathfrak{O}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameters $\mathfrak{O}_j$ and $\mathfrak{O}_k$ is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5} \sum_{i=-k}^{k} \frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S^2}{1+(\omega\tau_i)^2} + \frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{(S_f^2-S^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right). \quad (8.92)$$

$\mathfrak{O}_j - S^2$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S^2} = \frac{2}{5} \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{1}{1+(\omega\tau_i)^2} - \frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right). \quad (8.93)$$

$\mathfrak{O}_j - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_f^2} = -\frac{2}{5} \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} - \frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right). \quad (8.94)$$

$\mathfrak{O}_j - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_f} = \frac{2}{5}(1-S_f^2) \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{\left((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2\right)^2}. \quad (8.95)$$

$\mathfrak{O}_j - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_s} = \frac{2}{5}(S_f^2-S^2) \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{\left((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2\right)^2}. \quad (8.96)$$

$S^2 - S^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S^2)^2} = 0. \tag{8.97}$$

$S^2 - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameters S^2 and S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial S_f^2} = 0. \tag{8.98}$$

$S^2 - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_f} = 0. \tag{8.99}$$

$S^2 - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_s} = -\frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{\left((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2\right)^2}. \tag{8.100}$$

$S_f^2 - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_f^2)^2} = 0. \tag{8.101}$$

$S_f^2 - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_f} = -\frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{\left((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2\right)^2}. \tag{8.102}$$

$S_f^2 - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_s} = \frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.103}$$

$\tau_f - \tau_f$ partial derivative

The second partial derivative of (8.64) with respect to the correlation time τ_f twice is

$$\frac{\partial^2 J(\omega)}{\partial {\tau_f}^2} = -\frac{4}{5}(1 - S_f^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_f (\tau_f + \tau_i) - (\omega \tau_i)^4 \tau_f^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \tag{8.104}$$

$\tau_f - \tau_s$ partial derivative

The second partial derivative of (8.64) with respect to the correlation times τ_f and τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f \cdot \partial \tau_s} = 0. \tag{8.105}$$

$\tau_s - \tau_s$ partial derivative

The second partial derivative of (8.64) with respect to the correlation time τ_s twice is

$$\frac{\partial^2 J(\omega)}{\partial {\tau_s}^2} = -\frac{4}{5}(S_f^2 - S^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_s (\tau_s + \tau_i) - (\omega \tau_i)^4 \tau_s^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \tag{8.106}$$

8.9.6 The alternative extended model-free gradient

Because of the equation $S^2 = S_f^2 \cdot S_s^2$ and the form of the extended spectral density function (8.65) a convolution of the model-free space occurs if the model-free parameters $\{S_f^2, S_s^2, \tau_f, \tau_s\}$ are optimised rather than the parameters $\{S^2, S_f^2, \tau_f, \tau_s\}$. This convolution increases the complexity of the gradient. For completeness the first partial derivatives are presented below.

$\mathfrak{G}_j$ partial derivative

The partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ is

$$\begin{aligned}\frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(c_i\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\Bigg(S_f^2\cdot S_s^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} \\ + (1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2} \\ + S_f^2(1-S_s^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2}\Bigg) \\ + \frac{\partial c_i}{\partial \mathfrak{G}_j}\tau_i\Bigg(\frac{S_f^2\cdot S_s^2}{1+(\omega\tau_i)^2} + \frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{S_f^2(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\Bigg)\Bigg).\end{aligned} \tag{8.107}$$

$\mathfrak{O}_j$ partial derivative

The partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5}\sum_{i=-k}^{k}\frac{\partial c_i}{\partial \mathfrak{O}_j}\tau_i\Bigg(\frac{S_f^2\cdot S_s^2}{1+(\omega\tau_i)^2} + \frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{S_f^2(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\Bigg). \tag{8.108}$$

S_f^2 partial derivative

The partial derivative of (8.65) with respect to the order parameter S_f^2 is

$$\frac{\partial J(\omega)}{\partial S_f^2} = \frac{2}{5}\sum_{i=-k}^{k}c_i\tau_i\Bigg(\frac{S_s^2}{1+(\omega\tau_i)^2} - \frac{(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} + \frac{(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\Bigg). \tag{8.109}$$

S_s^2 partial derivative

The partial derivative of (8.65) with respect to the order parameter S_s^2 is

$$\frac{\partial J(\omega)}{\partial S_s^2} = \frac{2}{5}S_f^2\sum_{i=-k}^{k}c_i\tau_i\Bigg(\frac{1}{1+(\omega\tau_i)^2} - \frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\Bigg). \tag{8.110}$$

τ_f partial derivative

The partial derivative of (8.65) with respect to the correlation time τ_f is

$$\frac{\partial J(\omega)}{\partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \tag{8.111}$$

τ_s partial derivative

The partial derivative of (8.65) with respect to the correlation time τ_s is

$$\frac{\partial J(\omega)}{\partial \tau_s} = \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.112}$$

8.9.7 The alternative extended model-free Hessian

The model-free Hessian of the extended spectral density function (8.65) is also complicated by the convolution resulting from the use of the parameters $\{S_f^2,\, S_s^2,\, \tau_f,\, \tau_s\}$. The second partial derivatives with respect to these parameters are presented below.

$\mathfrak{G}_j - \mathfrak{G}_k$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameters $\mathfrak{G}_j$ and $\mathfrak{G}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&- 2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j}\cdot\frac{\partial \tau_i}{\partial \mathfrak{G}_k}\Bigg(S_f^2\cdot S_s^2\omega^2\tau_i\frac{3-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^3} \\ &+ (1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^3+3\omega^2\tau_f^3\tau_i(\tau_f+\tau_i)-(\omega\tau_f)^4\tau_i^3}{\left((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2\right)^3} \\ &+ S_f^2(1-S_s^2)\tau_s^2\frac{(\tau_s+\tau_i)^3+3\omega^2\tau_s^3\tau_i(\tau_s+\tau_i)-(\omega\tau_s)^4\tau_i^3}{\left((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2\right)^3}\Bigg) \\ &+ \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\cdot\frac{\partial c_i}{\partial \mathfrak{G}_k}+\frac{\partial \tau_i}{\partial \mathfrak{G}_k}\cdot\frac{\partial c_i}{\partial \mathfrak{G}_j}+c_i\frac{\partial^2\tau_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{G}_k}\right)\Bigg(S_f^2\cdot S_s^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} \\ &+ (1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{\left((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2\right)^2} \\ &+ S_f^2(1-S_s^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{\left((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2\right)^2}\Bigg) \\ &+ \left(\frac{\partial^2 c_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{G}_k}\tau_i\left(\frac{S_f^2\cdot S_s^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{S_f^2(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right)\right)\Bigg).\end{aligned} \tag{8.113}$$

$\mathfrak{G}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the orientational parameter $\mathfrak{O}_k$ is

$$\begin{aligned}\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5}\sum_{i=-k}^{k}\Bigg(&\frac{\partial \tau_i}{\partial \mathfrak{G}_j}\frac{\partial c_i}{\partial \mathfrak{O}_k}\Bigg(S_f^2\cdot S_s^2\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} \\ &+ (1-S_f^2)\tau_f^2\frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{\left((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2\right)^2} \\ &+ S_f^2(1-S_s^2)\tau_s^2\frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{\left((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2\right)^2}\Bigg) \\ &+ \frac{\partial^2 c_i}{\partial \mathfrak{G}_j\cdot\partial \mathfrak{O}_k}\tau_i\left(\frac{S_f^2\cdot S_s^2}{1+(\omega\tau_i)^2}+\frac{(1-S_f^2)(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2}+\frac{S_f^2(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2}\right)\Bigg).\end{aligned} \tag{8.114}$$

$\mathfrak{G}_j - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_f^2} = \frac{2}{5} \sum_{i=-k}^{k} \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S_s^2 \frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} - \tau_f^2 \frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2} \right.\right.$$
$$\left. + (1-S_s^2)\tau_s^2 \frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2} \right)$$
$$+ \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{S_s^2}{1+(\omega\tau_i)^2} - \frac{(\tau_f+\tau_i)\tau_f}{(\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2} \right.$$
$$\left.\left. + \frac{(1-S_s^2)(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right) \right). \quad (8.115)$$

$\mathfrak{G}_j - S_s^2$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the order parameter S_s^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_s^2} = \frac{2}{5} S_f^2 \sum_{i=-k}^{k} \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\frac{1-(\omega\tau_i)^2}{(1+(\omega\tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s+\tau_i)^2-(\omega\tau_s\tau_i)^2}{((\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2)^2} \right) \right.$$
$$\left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{1}{1+(\omega\tau_i)^2} - \frac{(\tau_s+\tau_i)\tau_s}{(\tau_s+\tau_i)^2+(\omega\tau_s\tau_i)^2} \right) \right). \quad (8.116)$$

$\mathfrak{G}_j - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_f} = \frac{2}{5}(1-S_f^2) \sum_{i=-k}^{k} \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_f \tau_i (\tau_f+\tau_i) \frac{(\tau_f+\tau_i)^2-3(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^3} \right.$$
$$\left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_f+\tau_i)^2-(\omega\tau_f\tau_i)^2}{((\tau_f+\tau_i)^2+(\omega\tau_f\tau_i)^2)^2} \right). \quad (8.117)$$

$\mathfrak{G}_j - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the geometric parameter $\mathfrak{G}_j$ and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_s} = \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^{k} \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_s \tau_i (\tau_s + \tau_i) \frac{(\tau_s + \tau_i)^2 - 3(\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right). \quad (8.118)$$

$\mathfrak{O}_j - \mathfrak{O}_k$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameters $\mathfrak{O}_j$ and $\mathfrak{O}_k$ is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5} \sum_{i=-k}^{k} \frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{S_f^2 (1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (8.119)$$

$\mathfrak{O}_j - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_f^2} = \frac{2}{5} \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{S_s^2}{1 + (\omega \tau_i)^2} - \frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (8.120)$$

$\mathfrak{O}_j - S_s^2$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the order parameter S_s^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_s^2} = \frac{2}{5} S_f^2 \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (8.121)$$

$\mathfrak{O}_j - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_f} = \frac{2}{5} (1 - S_f^2) \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (8.122)$$

$\mathfrak{O}_j - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the orientational parameter $\mathfrak{O}_j$ and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_s} = \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^{k} \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.123}$$

$S_f^2 - S_f^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_f^2)^2} = 0. \tag{8.124}$$

$S_f^2 - S_s^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameters S_f^2 and S_s^2 is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial S_s^2} = \frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \tag{8.125}$$

$S_f^2 - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_f} = -\frac{2}{5} \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \tag{8.126}$$

$S_f^2 - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_f^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_s} = \frac{2}{5} (1 - S_s^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.127}$$

$S_s^2 - S_s^2$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_s^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_s^2)^2} = 0. \tag{8.128}$$

$S_s^2 - \tau_f$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_s^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_s^2 \cdot \partial \tau_f} = 0. \tag{8.129}$$

$S_s^2 - \tau_s$ partial derivative

The second partial derivative of (8.65) with respect to the order parameter S_s^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_s^2 \cdot \partial \tau_s} = -\frac{2}{5} S_f^2 \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \tag{8.130}$$

$\tau_f - \tau_f$ partial derivative

The second partial derivative of (8.64) with respect to the correlation time τ_f twice is

$$\frac{\partial^2 J(\omega)}{\partial {\tau_f}^2} = -\frac{4}{5}(1 - S_f^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_f (\tau_f + \tau_i) - (\omega \tau_i)^4 \tau_f^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \tag{8.131}$$

$\tau_f - \tau_s$ partial derivative

The second partial derivative of (8.64) with respect to the correlation times τ_f and τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f \cdot \partial \tau_s} = 0. \tag{8.132}$$

$\tau_s - \tau_s$ partial derivative

The second partial derivative of (8.64) with respect to the correlation time τ_s twice is

$$\frac{\partial^2 J(\omega)}{\partial {\tau_s}^2} = -\frac{4}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^{k} c_i \tau_i^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_s (\tau_s + \tau_i) - (\omega \tau_i)^4 \tau_s^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \tag{8.133}$$

8.10 Ellipsoidal diffusion tensor

8.10.1 The diffusion equation of the ellipsoid

The correlation function of the Brownian rotational diffusion of an ellipsoid is

$$C_{\mathrm{O}}(\tau) = \frac{1}{5}\sum_{i=-2}^{2} c_i e^{-\frac{\tau}{\tau_i}}. \tag{8.134}$$

where c_i are the weights of the five exponential terms which are dependent on the orientation of the XH bond vector and τ_i are the correlation times of the five exponential terms.

8.10.2 The weights of the ellipsoid

Definitions

The three direction cosines defining the XH bond vector within the diffusion frame are

$$\delta_x = \widehat{XH} \cdot \widehat{\mathfrak{D}_x}, \tag{8.135a}$$
$$\delta_y = \widehat{XH} \cdot \widehat{\mathfrak{D}_y}, \tag{8.135b}$$
$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}_z}. \tag{8.135c}$$

Let the set of geometric parameters be

$$\mathfrak{G} = \{\mathfrak{D}_{iso}, \mathfrak{D}_a, \mathfrak{D}_r\}, \tag{8.136}$$

and the set of orientational parameters be the Euler angles

$$\mathfrak{O} = \{\alpha, \beta, \gamma\}. \tag{8.137}$$

The weights

The five weights c_i in the correlation function of the Brownian rotational diffusion of an ellipsoid (8.134) are

$$c_{-2} = \tfrac{1}{4}(d - e), \tag{8.138a}$$
$$c_{-1} = 3\delta_y^2\delta_z^2, \tag{8.138b}$$
$$c_0 = 3\delta_x^2\delta_z^2, \tag{8.138c}$$
$$c_1 = 3\delta_x^2\delta_y^2, \tag{8.138d}$$
$$c_2 = \tfrac{1}{4}(d + e), \tag{8.138e}$$

where

$$d = 3\left(\delta_x^4 + \delta_y^4 + \delta_z^4\right) - 1, \tag{8.139}$$

$$e = \frac{1}{\mathfrak{R}}\Bigg[(1 + 3\mathfrak{D}_r)\left(\delta_x^4 + 2\delta_y^2\delta_z^2\right) + (1 - 3\mathfrak{D}_r)\left(\delta_y^4 + 2\delta_x^2\delta_z^2\right) - 2\left(\delta_z^4 + 2\delta_x^2\delta_y^2\right)\Bigg]. \tag{8.140}$$

The factor $\mathfrak{R}$ is defined as

$$\mathfrak{R} = \sqrt{1 + 3\mathfrak{D}_r^2}. \tag{8.141}$$

8.10.3 The weight gradients of the ellipsoid

$\mathfrak{O}_i$ partial derivative

The partial derivatives with respect to the orientational parameter $\mathfrak{O}_i$ are

$$\frac{\partial c_{-2}}{\partial \mathfrak{O}_i} = 3\left(\delta_x^3\frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y^3\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_z^3\frac{\partial \delta_z}{\partial \mathfrak{O}_i}\right) - \frac{\partial e}{\partial \mathfrak{O}_i}, \tag{8.142a}$$

$$\frac{\partial c_{-1}}{\partial \mathfrak{O}_i} = 6\delta_y\delta_z\left(\delta_y\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\right), \tag{8.142b}$$

$$\frac{\partial c_0}{\partial \mathfrak{O}_i} = 6\delta_x\delta_z\left(\delta_x\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right), \tag{8.142c}$$

$$\frac{\partial c_1}{\partial \mathfrak{O}_i} = 6\delta_x\delta_y\left(\delta_x\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_y\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right), \tag{8.142d}$$

$$\frac{\partial c_2}{\partial \mathfrak{O}_i} = 3\left(\delta_x^3\frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y^3\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_z^3\frac{\partial \delta_z}{\partial \mathfrak{O}_i}\right) + \frac{\partial e}{\partial \mathfrak{O}_i}, \tag{8.142e}$$

where

$$\begin{aligned}\frac{\partial e}{\partial \mathfrak{O}_i} = \frac{1}{\mathfrak{R}}\Bigg[&(1 + 3\mathfrak{D}_r)\left(\delta_x^3\frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y\delta_z\left(\delta_y\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\right)\right)\\ &+(1 - 3\mathfrak{D}_r)\left(\delta_y^3\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_x\delta_z\left(\delta_x\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right)\right)\\ &-2\left(\delta_z^3\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_x\delta_y\left(\delta_x\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_y\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right)\right)\Bigg].\end{aligned} \tag{8.143}$$

τ_m partial derivative

The partial derivatives with respect to the τ_m geometric parameter are

$$\frac{\partial c_{-2}}{\partial \tau_m} = 0, \tag{8.144a}$$
$$\frac{\partial c_{-1}}{\partial \tau_m} = 0, \tag{8.144b}$$
$$\frac{\partial c_0}{\partial \tau_m} = 0, \tag{8.144c}$$
$$\frac{\partial c_1}{\partial \tau_m} = 0, \tag{8.144d}$$
$$\frac{\partial c_2}{\partial \tau_m} = 0. \tag{8.144e}$$

$\mathfrak{D}_a$ partial derivative

The partial derivatives with respect to the $\mathfrak{D}_a$ geometric parameter are

$$\frac{\partial c_{-2}}{\partial \mathfrak{D}_a} = 0, \tag{8.145a}$$
$$\frac{\partial c_{-1}}{\partial \mathfrak{D}_a} = 0, \tag{8.145b}$$
$$\frac{\partial c_0}{\partial \mathfrak{D}_a} = 0, \tag{8.145c}$$
$$\frac{\partial c_1}{\partial \mathfrak{D}_a} = 0, \tag{8.145d}$$
$$\frac{\partial c_2}{\partial \mathfrak{D}_a} = 0. \tag{8.145e}$$

$\mathfrak{D}_r$ partial derivative

The partial derivatives with respect to the $\mathfrak{D}_r$ geometric parameter are

$$\frac{\partial c_{-2}}{\partial \mathfrak{D}_r} = -\frac{3}{4}\frac{\partial e}{\partial \mathfrak{D}_r}, \tag{8.146a}$$
$$\frac{\partial c_{-1}}{\partial \mathfrak{D}_r} = 0, \tag{8.146b}$$
$$\frac{\partial c_0}{\partial \mathfrak{D}_r} = 0, \tag{8.146c}$$
$$\frac{\partial c_1}{\partial \mathfrak{D}_r} = 0, \tag{8.146d}$$
$$\frac{\partial c_2}{\partial \mathfrak{D}_r} = \frac{3}{4}\frac{\partial e}{\partial \mathfrak{D}_r}, \tag{8.146e}$$

where

$$\frac{\partial e}{\partial \mathfrak{D}_r} = \frac{1}{\Re^3}\left[(1-\mathfrak{D}_r)\left(\delta_x^4 + 2\delta_y^2\delta_z^2\right) - (1+\mathfrak{D}_r)\left(\delta_y^4 + 2\delta_x^2\delta_z^2\right) + 2\mathfrak{D}_r\left(\delta_z^4 + 2\delta_x^2\delta_y^2\right)\right]. \tag{8.147}$$

8.10.4 The weight Hessians of the ellipsoid

$\mathfrak{O}_i - \mathfrak{O}_j$ partial derivative

The second partial derivatives with respect to the orientational parameters $\mathfrak{O}_i$ and $\mathfrak{O}_j$ are

$$\begin{aligned}\frac{\partial^2 c_{-2}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3\Bigg(&\delta_x^2\left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right)\\ &+\delta_y^2\left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j}\right)\\ &+\delta_z^2\left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j}\right)\Bigg) - \frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j},\end{aligned} \tag{8.148a}$$

$$\begin{aligned}\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = &6\delta_y^2\left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j}\right)\\ &+ 12\delta_y\delta_z\left(\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j}\right)\\ &+ 6\delta_z^2\left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j}\right),\end{aligned} \tag{8.148b}$$

$$\begin{aligned}\frac{\partial^2 c_{0}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = &6\delta_x^2\left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j}\right)\\ &+ 12\delta_x\delta_z\left(\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right)\\ &+ 6\delta_z^2\left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right),\end{aligned} \tag{8.148c}$$

$$\begin{aligned}\frac{\partial^2 c_{1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = &6\delta_x^2\left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j}\right)\\ &+ 12\delta_x\delta_y\left(\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right)\\ &+ 6\delta_y^2\left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right),\end{aligned} \tag{8.148d}$$

$$\begin{aligned}\frac{\partial^2 c_{2}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3\Bigg(&\delta_x^2\left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_x}{\partial \mathfrak{O}_j}\right)\\ &+\delta_y^2\left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_y}{\partial \mathfrak{O}_j}\right)\\ &+\delta_z^2\left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3\frac{\partial \delta_z}{\partial \mathfrak{O}_i}\cdot\frac{\partial \delta_z}{\partial \mathfrak{O}_j}\right)\Bigg) + \frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j},\end{aligned} \tag{8.148e}$$

where

$$\begin{aligned}\frac{\partial^2 e}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}=\frac{1}{\mathfrak{R}}\Bigg[&(1+3\mathfrak{D}_r)\Bigg(\delta_x^2\left(\delta_x\frac{\partial^2\delta_x}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+3\frac{\partial\delta_x}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_x}{\partial\mathfrak{O}_j}\right)\\&+\delta_y^2\left(\delta_z\frac{\partial^2\delta_z}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_z}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_z}{\partial\mathfrak{O}_j}\right)\\&+\delta_z^2\left(\delta_y\frac{\partial^2\delta_y}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_y}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_y}{\partial\mathfrak{O}_j}\right)\\&+2\delta_y\delta_z\left(\frac{\partial\delta_y}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_z}{\partial\mathfrak{O}_j}+\frac{\partial\delta_z}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_y}{\partial\mathfrak{O}_j}\right)\Bigg)\\&+(1-3\mathfrak{D}_r)\Bigg(\delta_y^2\left(\delta_y\frac{\partial^2\delta_y}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+3\frac{\partial\delta_y}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_y}{\partial\mathfrak{O}_j}\right)\\&+\delta_x^2\left(\delta_z\frac{\partial^2\delta_z}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_z}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_z}{\partial\mathfrak{O}_j}\right)\\&+\delta_z^2\left(\delta_x\frac{\partial^2\delta_x}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_x}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_x}{\partial\mathfrak{O}_j}\right)\\&+2\delta_x\delta_z\left(\frac{\partial\delta_x}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_z}{\partial\mathfrak{O}_j}+\frac{\partial\delta_z}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_x}{\partial\mathfrak{O}_j}\right)\Bigg)\\&-2\Bigg(\delta_z^2\left(\delta_z\frac{\partial^2\delta_z}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+3\frac{\partial\delta_z}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_z}{\partial\mathfrak{O}_j}\right)\\&+\delta_x^2\left(\delta_y\frac{\partial^2\delta_y}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_y}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_y}{\partial\mathfrak{O}_j}\right)\\&+\delta_y^2\left(\delta_x\frac{\partial^2\delta_x}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}+\frac{\partial\delta_x}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_x}{\partial\mathfrak{O}_j}\right)\\&+2\delta_x\delta_y\left(\frac{\partial\delta_x}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_y}{\partial\mathfrak{O}_j}+\frac{\partial\delta_y}{\partial\mathfrak{O}_i}\cdot\frac{\partial\delta_x}{\partial\mathfrak{O}_j}\right)\Bigg)\Bigg].\end{aligned}\tag{8.149}$$

$\mathfrak{O}_i - \tau_m$ partial derivative

The second partial derivatives with respect to the orientational parameter $\mathfrak{O}_i$ and the geometric parameter τ_m are

$$\frac{\partial^2 c_{-2}}{\partial\mathfrak{O}_i\cdot\partial\tau_m}=0,\tag{8.150a}$$

$$\frac{\partial^2 c_{-1}}{\partial\mathfrak{O}_i\cdot\partial\tau_m}=0,\tag{8.150b}$$

$$\frac{\partial^2 c_0}{\partial\mathfrak{O}_i\cdot\partial\tau_m}=0,\tag{8.150c}$$

$$\frac{\partial^2 c_1}{\partial\mathfrak{O}_i\cdot\partial\tau_m}=0,\tag{8.150d}$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{O}_i \cdot \partial \tau_m} = 0. \tag{8.150e}$$

$\mathfrak{O}_i - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the orientational parameter $\mathfrak{O}_i$ and the geometric parameter $\mathfrak{D}_a$ are

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_a} = 0, \tag{8.151a}$$
$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_a} = 0, \tag{8.151b}$$
$$\frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_a} = 0, \tag{8.151c}$$
$$\frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_a} = 0, \tag{8.151d}$$
$$\frac{\partial^2 c_2}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_a} = 0. \tag{8.151e}$$

$\mathfrak{O}_i - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the orientational parameter $\mathfrak{O}_i$ and the geometric parameter $\mathfrak{D}_r$ are

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = -3\frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r}, \tag{8.152a}$$
$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = 0, \tag{8.152b}$$
$$\frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = 0, \tag{8.152c}$$
$$\frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = 0, \tag{8.152d}$$
$$\frac{\partial^2 c_2}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = 3\frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r}, \tag{8.152e}$$

where

$$\begin{aligned}\frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{D}_r} = \frac{1}{\Re^3}\Bigg[&(1-\mathfrak{D}_r)\left(\delta_x^3\frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y\delta_z\left(\delta_y\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_y}{\partial \mathfrak{O}_i}\right)\right)\\ &-(1+\mathfrak{D}_r)\left(\delta_y^3\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_x\delta_z\left(\delta_x\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right)\right)\\ &+2\mathfrak{D}_r\left(\delta_z^3\frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_x\delta_y\left(\delta_x\frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_y\frac{\partial \delta_x}{\partial \mathfrak{O}_i}\right)\right)\Bigg].\end{aligned} \tag{8.153}$$

$\tau_m - \tau_m$ partial derivative

The second partial derivatives with respect to the geometric parameter τ_m twice are

$$\frac{\partial^2 c_{-2}}{\partial {\tau_m}^2} = 0, \tag{8.154a}$$

$$\frac{\partial^2 c_{-1}}{\partial {\tau_m}^2} = 0, \tag{8.154b}$$

$$\frac{\partial^2 c_0}{\partial {\tau_m}^2} = 0, \tag{8.154c}$$

$$\frac{\partial^2 c_1}{\partial {\tau_m}^2} = 0, \tag{8.154d}$$

$$\frac{\partial^2 c_2}{\partial {\tau_m}^2} = 0. \tag{8.154e}$$

$\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and $\mathfrak{D}_a$ are

$$\frac{\partial^2 c_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \tag{8.155a}$$

$$\frac{\partial^2 c_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \tag{8.155b}$$

$$\frac{\partial^2 c_0}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \tag{8.155c}$$

$$\frac{\partial^2 c_1}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \tag{8.155d}$$

$$\frac{\partial^2 c_2}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0. \tag{8.155e}$$

$\tau_m - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and $\mathfrak{D}_r$ are

$$\frac{\partial^2 c_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \tag{8.156a}$$

$$\frac{\partial^2 c_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \tag{8.156b}$$

$$\frac{\partial^2 c_0}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \tag{8.156c}$$

$$\frac{\partial^2 c_1}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \tag{8.156d}$$

$$\frac{\partial^2 c_2}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0. \tag{8.156e}$$

$\mathfrak{D}_a - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameter $\mathfrak{D}_a$ twice are

$$\frac{\partial^2 c_{-2}}{\partial {\mathfrak{D}_a}^2} = 0, \tag{8.157a}$$

$$\frac{\partial^2 c_{-1}}{\partial {\mathfrak{D}_a}^2} = 0, \tag{8.157b}$$

$$\frac{\partial^2 c_0}{\partial {\mathfrak{D}_a}^2} = 0, \tag{8.157c}$$

$$\frac{\partial^2 c_1}{\partial {\mathfrak{D}_a}^2} = 0, \tag{8.157d}$$

$$\frac{\partial^2 c_2}{\partial {\mathfrak{D}_a}^2} = 0. \tag{8.157e}$$

$\mathfrak{D}_a - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters $\mathfrak{D}_a$ and $\mathfrak{D}_r$ are

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \tag{8.158a}$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \tag{8.158b}$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \tag{8.158c}$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \tag{8.158d}$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0. \tag{8.158e}$$

$\mathfrak{D}_r - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameter $\mathfrak{D}_r$ twice are

$$\frac{\partial^2 c_{-2}}{\partial {\mathfrak{D}_r}^2} = -\frac{3}{4}\frac{\partial^2 e}{\partial \mathfrak{D}_r^2}, \tag{8.159a}$$

$$\frac{\partial^2 c_{-1}}{\partial {\mathfrak{D}_r}^2} = 0, \tag{8.159b}$$

$$\frac{\partial^2 c_0}{\partial {\mathfrak{D}_r}^2} = 0, \tag{8.159c}$$

$$\frac{\partial^2 c_1}{\partial {\mathfrak{D}_r}^2} = 0, \tag{8.159d}$$

$$\frac{\partial^2 c_2}{\partial {\mathfrak{D}_r}^2} = \frac{3}{4}\frac{\partial^2 e}{\partial \mathfrak{D}_r^2}, \tag{8.159e}$$

where

$$\frac{\partial^2 e}{\partial \mathfrak{D}_r^2} = \frac{1}{\mathfrak{R}^5}\Big[(6\mathfrak{D}_r^2 - 9\mathfrak{D}_r - 1)\left(\delta_x^4 + 2\delta_y^2\delta_z^2\right) + (6\mathfrak{D}_r^2 + 9\mathfrak{D}_r - 1)\left(\delta_y^4 + 2\delta_x^2\delta_z^2\right) - 2(6\mathfrak{D}_r^2 - 1)\left(\delta_z^4 + 2\delta_x^2\delta_y^2\right)\Big]. \tag{8.160}$$

8.10.5 The correlation times of the ellipsoid

The five correlation times τ_i in the correlation function of the Brownian rotational diffusion of an ellipsoid (8.134) on page 93 are

$$\tau_{-2} = (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-1}, \tag{8.161a}$$
$$\tau_{-1} = (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-1}, \tag{8.161b}$$
$$\tau_0 = (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-1}, \tag{8.161c}$$
$$\tau_1 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-1}, \tag{8.161d}$$
$$\tau_2 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-1}, \tag{8.161e}$$

where $\mathfrak{R}$ is defined in Equation (8.141) on page 94.

8.10.6 The correlation time gradients of the ellipsoid

τ_m partial derivative

The partial derivatives with respect to the geometric parameter τ_m are

$$\frac{\partial\tau_{-2}}{\partial\tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.162a}$$
$$\frac{\partial\tau_{-1}}{\partial\tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \tag{8.162b}$$
$$\frac{\partial\tau_0}{\partial\tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \tag{8.162c}$$
$$\frac{\partial\tau_1}{\partial\tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \tag{8.162d}$$
$$\frac{\partial\tau_2}{\partial\tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.162e}$$

$\mathfrak{D}_a$ partial derivative

The partial derivatives with respect to the geometric parameter $\mathfrak{D}_a$ are

$$\frac{\partial\tau_{-2}}{\partial\mathfrak{D}_a} = 2\mathfrak{R}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.163a}$$
$$\frac{\partial\tau_{-1}}{\partial\mathfrak{D}_a} = (1 + 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \tag{8.163b}$$
$$\frac{\partial\tau_0}{\partial\mathfrak{D}_a} = (1 - 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \tag{8.163c}$$
$$\frac{\partial\tau_1}{\partial\mathfrak{D}_a} = -2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \tag{8.163d}$$
$$\frac{\partial\tau_2}{\partial\mathfrak{D}_a} = -2\mathfrak{R}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.163e}$$

$\mathfrak{D}_r$ partial derivative

The partial derivatives with respect to the geometric parameter $\mathfrak{D}_r$ are

$$\frac{\partial \tau_{-2}}{\partial \mathfrak{D}_r} = 6\frac{\mathfrak{D}_a \mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.164a}$$

$$\frac{\partial \tau_{-1}}{\partial \mathfrak{D}_r} = 3\mathfrak{D}_a(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \tag{8.164b}$$

$$\frac{\partial \tau_0}{\partial \mathfrak{D}_r} = -3\mathfrak{D}_a(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \tag{8.164c}$$

$$\frac{\partial \tau_1}{\partial \mathfrak{D}_r} = 0, \tag{8.164d}$$

$$\frac{\partial \tau_2}{\partial \mathfrak{D}_r} = -6\frac{\mathfrak{D}_a \mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.164e}$$

8.10.7 The correlation time Hessians of the ellipsoid

$\tau_m - \tau_m$ partial derivative

The second partial derivatives with respect to the geometric parameter τ_m twice are

$$\frac{\partial^2 \tau_{-2}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.165a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \tag{8.165b}$$

$$\frac{\partial^2 \tau_{0}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \tag{8.165c}$$

$$\frac{\partial^2 \tau_{1}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \tag{8.165d}$$

$$\frac{\partial^2 \tau_{2}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.165e}$$

$\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and $\mathfrak{D}_a$ are

$$\frac{\partial^2 \tau_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 4\mathfrak{R}{\tau_m}^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \tag{8.166a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2(1 + 3\mathfrak{D}_r){\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \tag{8.166b}$$

$$\frac{\partial^2 \tau_{0}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2(1 - 3\mathfrak{D}_r){\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \tag{8.166c}$$

$$\frac{\partial^2 \tau_{1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4{\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}, \tag{8.166d}$$

$$\frac{\partial^2 \tau_{2}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4\mathfrak{R}{\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \tag{8.166e}$$

$\tau_m - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and $\mathfrak{D}_r$ are

$$\frac{\partial^2 \tau_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 12\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}{\tau_m}^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \tag{8.167a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 6\mathfrak{D}_a{\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \tag{8.167b}$$

$$\frac{\partial^2 \tau_{0}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = -6\mathfrak{D}_a{\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \tag{8.167c}$$

$$\frac{\partial^2 \tau_{1}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \tag{8.167d}$$

$$\frac{\partial^2 \tau_{2}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = -12\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}{\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \tag{8.167e}$$

$\mathfrak{D}_a - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameter $\mathfrak{D}_a$ twice are

$$\frac{\partial^2 \tau_{-2}}{\partial {\mathfrak{D}_a}^2} = 8\mathfrak{R}^2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \tag{8.168a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial {\mathfrak{D}_a}^2} = 2(1+3\mathfrak{D}_r)^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1+3\mathfrak{D}_r))^{-3}, \tag{8.168b}$$

$$\frac{\partial^2 \tau_{0}}{\partial {\mathfrak{D}_a}^2} = 2(1-3\mathfrak{D}_r)^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1-3\mathfrak{D}_r))^{-3}, \tag{8.168c}$$

$$\frac{\partial^2 \tau_{1}}{\partial {\mathfrak{D}_a}^2} = 8(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}, \tag{8.168d}$$

$$\frac{\partial^2 \tau_{2}}{\partial {\mathfrak{D}_a}^2} = 8\mathfrak{R}^2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \tag{8.168e}$$

$\mathfrak{D}_a - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters $\mathfrak{D}_a$ and $\mathfrak{D}_r$ are

$$\frac{\partial^2 \tau_{-2}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 24\mathfrak{D}_a\mathfrak{D}_r(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} + 6\frac{\mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.169a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 6\mathfrak{D}_a(1+3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1+3\mathfrak{D}_r))^{-3} + 3(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1+3\mathfrak{D}_r))^{-2}, \tag{8.169b}$$

$$\frac{\partial^2 \tau_{0}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = -6\mathfrak{D}_a(1-3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1-3\mathfrak{D}_r))^{-3} - 3(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1-3\mathfrak{D}_r))^{-2}, \tag{8.169c}$$

$$\frac{\partial^2 \tau_{1}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \tag{8.169d}$$

$$\frac{\partial^2 \tau_{2}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 24\mathfrak{D}_a\mathfrak{D}_r(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3} - 6\frac{\mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.169e}$$

$\mathfrak{D}_r - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameter $\mathfrak{D}_r$ twice are

$$\frac{\partial^2 \tau_{-2}}{\partial {\mathfrak{D}_r}^2} = 72\left(\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}\right)^2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} + 6\frac{\mathfrak{D}_a}{\mathfrak{R}^3}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \tag{8.170a}$$

$$\frac{\partial^2 \tau_{-1}}{\partial {\mathfrak{D}_r}^2} = 18\mathfrak{D}_a^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1+3\mathfrak{D}_r))^{-3}, \tag{8.170b}$$

$$\frac{\partial^2 \tau_{0}}{\partial {\mathfrak{D}_r}^2} = 18\mathfrak{D}_a^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1-3\mathfrak{D}_r))^{-3}, \tag{8.170c}$$

$$\frac{\partial^2 \tau_{1}}{\partial {\mathfrak{D}_r}^2} = 0, \tag{8.170d}$$

$$\frac{\partial^2 \tau_{2}}{\partial {\mathfrak{D}_r}^2} = 72\left(\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}\right)^2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} - 6\frac{\mathfrak{D}_a}{\mathfrak{R}^3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \tag{8.170e}$$

8.11 Spheroidal diffusion tensor

8.11.1 The diffusion equation of the spheroid

The correlation function of the Brownian rotational diffusion of a spheroid is

$$C_{\mathrm{O}}(\tau) = \frac{1}{5}\sum_{i=-1}^{1} c_i e^{-\frac{\tau}{\tau_i}}. \tag{8.171}$$

where c_i are the weights of the three exponential terms which are dependent on the orientation of the XH bond vector and τ_i are the correlation times of the three exponential terms.

8.11.2 The weights of the spheroid

Definitions

The direction cosine defining the XH bond vector within the spheroidal diffusion frame is

$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}_z}. \tag{8.172}$$

Let the set of geometric parameters be

$$\mathfrak{G} = \{\mathfrak{D}_{iso}, \mathfrak{D}_a\}, \tag{8.173}$$

and the set of orientational parameters be the spherical angles

$$\mathfrak{O} = \{\theta, \phi\}. \tag{8.174}$$

The weights

The three spheroid weights c_i in the correlation function of the Brownian rotational diffusion of a spheroid (8.171) are

$$c_{-1} = \tfrac{1}{4}(3\delta_z^2 - 1)^2, \tag{8.175a}$$
$$c_0 = 3\delta_z^2(1 - \delta_z^2), \tag{8.175b}$$
$$c_1 = \tfrac{3}{4}(\delta_z^2 - 1)^2. \tag{8.175c}$$

8.11.3 The weight gradients of the spheroid

$\mathfrak{O}_i$ partial derivative

The partial derivatives with respect to the orientational parameter $\mathfrak{O}_i$ are

$$\frac{\partial c_{-1}}{\partial \mathfrak{O}_i} = 3\delta_z(3\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}, \tag{8.176a}$$

$$\frac{\partial c_0}{\partial \mathfrak{O}_i} = 6\delta_z(1 - 2\delta_z^2)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}, \tag{8.176b}$$

$$\frac{\partial c_1}{\partial \mathfrak{O}_i} = 3\delta_z(\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}. \tag{8.176c}$$

8.11.4 The weight Hessians of the spheroid

$\mathfrak{O}_i - \mathfrak{O}_j$ partial derivative

The second partial derivatives with respect to the orientational parameters $\mathfrak{O}_i$ and $\mathfrak{O}_j$ are

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3\left((9\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(3\delta_z^2 - 1)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j}\right), \tag{8.177a}$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 6\left((1 - 6\delta_z^2)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(1 - 2\delta_z^2)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j}\right), \tag{8.177b}$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3\left((3\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(\delta_z^2 - 1)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j}\right). \tag{8.177c}$$

8.11.5 The correlation times of the spheroid

The three spheroid correlation times τ_i in the correlation function of the Brownian rotational diffusion of a spheroid (8.171) are

$$\tau_{-1} = (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-1}, \tag{8.178a}$$
$$\tau_0 = (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-1}, \tag{8.178b}$$
$$\tau_1 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-1}. \tag{8.178c}$$

8.11.6 The correlation time gradients of the spheroid

τ_m partial derivative

The partial derivatives with respect to the geometric parameter τ_m are

$$\frac{\partial \tau_{-1}}{\partial \tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \tag{8.179a}$$
$$\frac{\partial \tau_0}{\partial \tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \tag{8.179b}$$
$$\frac{\partial \tau_1}{\partial \tau_m} = {\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \tag{8.179c}$$

$\mathfrak{D}_a$ partial derivative

The partial derivatives with respect to the geometric parameter $\mathfrak{D}_a$ are

$$\frac{\partial \tau_{-1}}{\partial \mathfrak{D}_a} = 2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \tag{8.180a}$$
$$\frac{\partial \tau_0}{\partial \mathfrak{D}_a} = (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \tag{8.180b}$$
$$\frac{\partial \tau_1}{\partial \mathfrak{D}_a} = -2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \tag{8.180c}$$

8.11.7 The correlation time Hessians of the spheroid

$\tau_m - \tau_m$ partial derivative

The second partial derivatives with respect to the geometric parameter τ_m twice are

$$\frac{\partial^2 \tau_{-1}}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \tag{8.181a}$$
$$\frac{\partial^2 \tau_0}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \tag{8.181b}$$
$$\frac{\partial^2 \tau_1}{\partial {\tau_m}^2} = 2{\tau_m}^{-4}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3} - 2{\tau_m}^{-3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \tag{8.181c}$$

$\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and $\mathfrak{D}_a$ are

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 4{\tau_m}^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3}, \tag{8.182a}$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2{\tau_m}^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3}, \tag{8.182b}$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4{\tau_m}^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}. \tag{8.182c}$$

$\mathfrak{D}_a - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameter $\mathfrak{D}_a$ twice are

$$\frac{\partial^2 \tau_{-1}}{\partial {\mathfrak{D}_a}^2} = 8(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3}, \tag{8.183a}$$

$$\frac{\partial^2 \tau_0}{\partial {\mathfrak{D}_a}^2} = 2(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3}, \tag{8.183b}$$

$$\frac{\partial^2 \tau_1}{\partial {\mathfrak{D}_a}^2} = 8(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}. \tag{8.183c}$$

8.12 Spherical diffusion tensor

8.12.1 The diffusion equation of the sphere

The correlation function of the Brownian rotational diffusion of a sphere is

$$C_{\mathrm{O}}(\tau) = \frac{1}{5} e^{-\frac{\tau}{\tau_m}}, \tag{8.184}$$

$$= \frac{1}{5} \sum_{i=0}^{0} c_i e^{-\frac{\tau}{\tau_i}}. \tag{8.185}$$

where c_i is the weight of the single exponential term and τ_i is the correlation time of the single exponential term.

8.12.2 The weight of the sphere

Definitions

The entire diffusion parameter set consists of a single geometric parameter and is

$$\mathfrak{D} = \{\tau_m\}. \tag{8.186}$$

Summation terms

The summation indices of the correlation function of the Brownian rotational diffusion of a sphere (8.171) range from $k = 0$ to $k = 0$ therefore

$$i \in \{0\}. \tag{8.187}$$

The weights

The single weight c_i in the correlation function of the Brownian rotational diffusion of a sphere (8.171) is

$$c_0 = 1. \tag{8.188}$$

8.12.3 The weight gradient of the sphere

τ_m partial derivative

The partial derivative with respect to the geometric parameter τ_m is

$$\frac{\partial c_0}{\partial \tau_m} = 0. \tag{8.189}$$

8.12.4 The weight Hessian of the sphere

$\tau_m - \tau_m$ partial derivative

The second partial derivatives with respect to the geometric parameter τ_m twice is

$$\frac{\partial^2 c_0}{\partial {\tau_m}^2} = 0. \tag{8.190}$$

8.12.5 The correlation time of the sphere

The single correlation time τ_i of the correlation function of the Brownian rotational diffusion of a sphere (8.171) is

$$\tau_0 = \tau_m. \tag{8.191}$$

8.12.6 The correlation time gradient of the sphere

τ_m partial derivative

The partial derivative with respect to the geometric parameter τ_m is

$$\frac{\partial \tau_0}{\partial \tau_m} = 1. \tag{8.192}$$

8.12.7 The correlation time Hessian of the sphere

$\tau_m - \tau_m$ partial derivative

The second partial derivative with respect to the geometric parameter τ_m twice is

$$\frac{\partial^2 \tau_0}{\partial {\tau_m}^2} = 0. \tag{8.193}$$

8.13 Ellipsoidal dot product derivatives

8.13.1 The dot product of the ellipsoid

The dot product is defined as

$$\delta_i = \widehat{XH} \cdot \widehat{\mathfrak{D}_i}, \tag{8.194}$$

where i is one of $\{x, y, z\}$, $\widehat{XH}$ is a unit vector parallel to the XH bond vector, and $\widehat{\mathfrak{D}_i}$ is one of the unit vectors defining the diffusion frame. The three diffusion frame unit vectors can be expressed using the Euler angles α, β, and γ as

$$\widehat{\mathfrak{D}_x} = \begin{pmatrix} -\sin\alpha\sin\gamma + \cos\alpha\cos\beta\cos\gamma \\ -\sin\alpha\cos\gamma - \cos\alpha\cos\beta\sin\gamma \\ \cos\alpha\sin\beta \end{pmatrix}, \tag{8.195a}$$

$$\widehat{\mathfrak{D}_y} = \begin{pmatrix} \cos\alpha\sin\gamma + \sin\alpha\cos\beta\cos\gamma \\ \cos\alpha\cos\gamma - \sin\alpha\cos\beta\sin\gamma \\ \sin\alpha\sin\beta \end{pmatrix}, \tag{8.195b}$$

$$\widehat{\mathfrak{D}_z} = \begin{pmatrix} -\sin\beta\cos\gamma \\ \sin\beta\sin\gamma \\ \cos\beta \end{pmatrix}. \tag{8.195c}$$

8.13.2 The dot product gradient of the ellipsoid

The partial derivative of the dot product δ_i with respect to the orientational parameter $\mathfrak{O}_j$ is

$$\frac{\partial\delta_i}{\partial\mathfrak{O}_j} = \frac{\partial}{\partial\mathfrak{O}_j}\left(\widehat{XH}\cdot\widehat{\mathfrak{D}_i}\right) = \widehat{XH}\frac{\partial\widehat{\mathfrak{D}_i}}{\partial\mathfrak{O}_j} + \frac{\partial\widehat{XH}}{\partial\mathfrak{O}_j}\widehat{\mathfrak{D}_i}. \tag{8.196}$$

Because $\widehat{XH}$ is constant and not dependent on the Euler angles its derivative is zero. Therefore

$$\frac{\partial\delta_i}{\partial\mathfrak{O}_j} = \widehat{XH}\frac{\partial\widehat{\mathfrak{D}_i}}{\partial\mathfrak{O}_j}. \tag{8.197}$$

The $\widehat{\mathfrak{D}_x}$ gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}_x}$ with respect to the Euler angles are

$$\frac{\partial\widehat{\mathfrak{D}_x}}{\partial\alpha} = \begin{pmatrix} -\cos\alpha\sin\gamma - \sin\alpha\cos\beta\cos\gamma \\ -\cos\alpha\cos\gamma + \sin\alpha\cos\beta\sin\gamma \\ -\sin\alpha\sin\beta \end{pmatrix}, \tag{8.198a}$$

$$\frac{\partial\widehat{\mathfrak{D}_x}}{\partial\beta} = \begin{pmatrix} -\cos\alpha\sin\beta\cos\gamma \\ \cos\alpha\sin\beta\sin\gamma \\ \cos\alpha\cos\beta \end{pmatrix}, \tag{8.198b}$$

$$\frac{\partial\widehat{\mathfrak{D}_x}}{\partial\gamma} = \begin{pmatrix} -\sin\alpha\cos\gamma - \cos\alpha\cos\beta\sin\gamma \\ \sin\alpha\sin\gamma - \cos\alpha\cos\beta\cos\gamma \\ 0 \end{pmatrix}. \tag{8.198c}$$

The $\widehat{\mathfrak{D}_y}$ gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}_y}$ with respect to the Euler angles are

$$\frac{\partial\widehat{\mathfrak{D}_y}}{\partial\alpha} = \begin{pmatrix} -\sin\alpha\sin\gamma + \cos\alpha\cos\beta\cos\gamma \\ -\sin\alpha\cos\gamma - \cos\alpha\cos\beta\sin\gamma \\ \cos\alpha\sin\beta \end{pmatrix}, \tag{8.199a}$$

$$\frac{\partial\widehat{\mathfrak{D}_y}}{\partial\beta} = \begin{pmatrix} -\sin\alpha\sin\beta\cos\gamma \\ \sin\alpha\sin\beta\sin\gamma \\ \sin\alpha\cos\beta \end{pmatrix}, \tag{8.199b}$$

$$\frac{\partial\widehat{\mathfrak{D}_y}}{\partial\gamma} = \begin{pmatrix} \cos\alpha\cos\gamma - \sin\alpha\cos\beta\sin\gamma \\ -\cos\alpha\sin\gamma - \sin\alpha\cos\beta\cos\gamma \\ 0 \end{pmatrix}. \tag{8.199c}$$

The $\widehat{\mathfrak{D}_z}$ gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}_z}$ with respect to the Euler angles are

$$\frac{\partial\widehat{\mathfrak{D}_z}}{\partial\alpha} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{8.200a}$$

$$\frac{\partial\widehat{\mathfrak{D}_z}}{\partial\beta} = \begin{pmatrix} -\cos\beta\cos\gamma \\ \cos\beta\sin\gamma \\ -\sin\beta \end{pmatrix}, \tag{8.200b}$$

$$\frac{\partial\widehat{\mathfrak{D}_z}}{\partial\gamma} = \begin{pmatrix} \sin\beta\sin\gamma \\ \sin\beta\cos\gamma \\ 0 \end{pmatrix}. \tag{8.200c}$$

8.13.3 The dot product Hessian of the ellipsoid

The second partial derivative of the dot product δ_i with respect to the orientational parameters $\mathfrak{O}_j$ and $\mathfrak{O}_k$ is

$$\frac{\partial^2 \delta_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{\partial^2}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \left(\widehat{XH} \cdot \widehat{\mathfrak{D}_i} \right) = \widehat{XH} \frac{\partial^2 \widehat{\mathfrak{D}_i}}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k}. \tag{8.201}$$

The $\widehat{\mathfrak{D}_x}$ Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}_x}$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \alpha^2} = \begin{pmatrix} \sin\alpha \sin\gamma - \cos\alpha \cos\beta \cos\gamma \\ \sin\alpha \cos\gamma + \cos\alpha \cos\beta \sin\gamma \\ -\cos\alpha \sin\beta \end{pmatrix}, \tag{8.202a}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} \sin\alpha \sin\beta \cos\gamma \\ -\sin\alpha \sin\beta \sin\gamma \\ -\sin\alpha \cos\beta \end{pmatrix}, \tag{8.202b}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} -\cos\alpha \cos\gamma + \sin\alpha \cos\beta \sin\gamma \\ \cos\alpha \sin\gamma + \sin\alpha \cos\beta \cos\gamma \\ 0 \end{pmatrix}, \tag{8.202c}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \beta^2} = \begin{pmatrix} -\cos\alpha \cos\beta \cos\gamma \\ \cos\alpha \cos\beta \sin\gamma \\ -\cos\alpha \sin\beta \end{pmatrix}, \tag{8.202d}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \cos\alpha \sin\beta \sin\gamma \\ \cos\alpha \sin\beta \cos\gamma \\ 0 \end{pmatrix}, \tag{8.202e}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_x}}{\partial \gamma^2} = \begin{pmatrix} \sin\alpha \sin\gamma - \cos\alpha \cos\beta \cos\gamma \\ \sin\alpha \cos\gamma + \cos\alpha \cos\beta \sin\gamma \\ 0 \end{pmatrix}. \tag{8.202f}$$

The $\widehat{\mathfrak{D}_y}$ Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}_y}$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \alpha^2} = \begin{pmatrix} -\cos\alpha \sin\gamma - \sin\alpha \cos\beta \cos\gamma \\ -\cos\alpha \cos\gamma + \sin\alpha \cos\beta \sin\gamma \\ -\sin\alpha \sin\beta \end{pmatrix}, \tag{8.203a}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} -\cos\alpha \sin\beta \cos\gamma \\ \cos\alpha \sin\beta \sin\gamma \\ \cos\alpha \cos\beta \end{pmatrix}, \tag{8.203b}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} -\sin\alpha \cos\gamma - \cos\alpha \cos\beta \sin\gamma \\ \sin\alpha \sin\gamma - \cos\alpha \cos\beta \cos\gamma \\ 0 \end{pmatrix}, \tag{8.203c}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \beta^2} = \begin{pmatrix} -\sin\alpha\cos\beta\cos\gamma \\ \sin\alpha\cos\beta\sin\gamma \\ -\sin\alpha\sin\beta \end{pmatrix}, \tag{8.203d}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \sin\alpha\sin\beta\sin\gamma \\ \sin\alpha\sin\beta\cos\gamma \\ 0 \end{pmatrix}, \tag{8.203e}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_y}}{\partial \gamma^2} = \begin{pmatrix} -\cos\alpha\sin\gamma - \sin\alpha\cos\beta\cos\gamma \\ -\cos\alpha\cos\gamma + \sin\alpha\cos\beta\sin\gamma \\ 0 \end{pmatrix}. \tag{8.203f}$$

The $\widehat{\mathfrak{D}_z}$ Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}_z}$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \alpha^2} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{8.204a}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{8.204b}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{8.204c}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \beta^2} = \begin{pmatrix} \sin\beta\cos\gamma \\ -\sin\beta\sin\gamma \\ -\cos\beta \end{pmatrix}, \tag{8.204d}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \cos\beta\sin\gamma \\ \cos\beta\cos\gamma \\ 0 \end{pmatrix}, \tag{8.204e}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_z}}{\partial \gamma^2} = \begin{pmatrix} \sin\beta\cos\gamma \\ -\sin\beta\sin\gamma \\ 0 \end{pmatrix}. \tag{8.204f}$$

8.14 Spheroidal dot product derivatives

8.14.1 The dot product of the spheroid

The single dot product of the spheroid is defined as

$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}_\parallel}, \tag{8.205}$$

where $\widehat{XH}$ is a unit vector parallel to the XH vector. $\widehat{\mathfrak{D}_\parallel}$ is a unit vector parallel to the unique axis of the diffusion tensor and can be expressed using the spherical angles where θ is the polar angle and ϕ is the azimuthal angle as

$$\widehat{\mathfrak{D}_\parallel} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}. \tag{8.206}$$

8.14.2 The dot product gradient of the spheroid

The partial derivative of the dot product with respect to the orientational parameter $\mathfrak{O}_i$ is

$$\frac{\partial\delta_z}{\partial\mathfrak{O}_i} = \frac{\partial}{\partial\mathfrak{O}_i}\left(\widehat{XH}\cdot\widehat{\mathfrak{D}_\parallel}\right) = \widehat{XH}\frac{\partial\widehat{\mathfrak{D}_\parallel}}{\partial\mathfrak{O}_i} + \frac{\partial\widehat{XH}}{\partial\mathfrak{O}_i}\widehat{\mathfrak{D}_\parallel}. \tag{8.207}$$

Because the XH bond vector is constant and not dependent on the spherical angles its derivative is zero. Therefore

$$\frac{\partial\delta_z}{\partial\mathfrak{O}_i} = \widehat{XH}\frac{\partial\widehat{\mathfrak{D}_\parallel}}{\partial\mathfrak{O}_i}. \tag{8.208}$$

The $\widehat{\mathfrak{D}_\parallel}$ gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}_\parallel}$ with respect to the spherical angles are

$$\frac{\partial\widehat{\mathfrak{D}_\parallel}}{\partial\theta} = \begin{pmatrix} \cos\theta\cos\phi \\ \cos\theta\sin\phi \\ -\sin\theta \end{pmatrix}, \tag{8.209a}$$

$$\frac{\partial\widehat{\mathfrak{D}_\parallel}}{\partial\phi} = \begin{pmatrix} -\sin\theta\sin\phi \\ \sin\theta\cos\phi \\ 0 \end{pmatrix}. \tag{8.209b}$$

8.14.3 The dot product Hessian of the spheroid

The second partial derivative of the single spheroidal dot product δ_z with respect to the orientational parameters $\mathfrak{O}_i$ and $\mathfrak{O}_j$ is

$$\frac{\partial^2\delta_z}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j} = \frac{\partial^2}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}\left(\widehat{XH}\cdot\widehat{\mathfrak{D}_\parallel}\right) = \widehat{XH}\frac{\partial^2\widehat{\mathfrak{D}_\parallel}}{\partial\mathfrak{O}_i\cdot\partial\mathfrak{O}_j}. \tag{8.210}$$

The $\widehat{\mathfrak{D}_\parallel}$ Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}_\parallel}$ with respect to the spherical angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}_\parallel}}{\partial \theta^2} = \begin{pmatrix} -\sin\theta\cos\phi \\ -\sin\theta\sin\phi \\ -\cos\theta \end{pmatrix}, \tag{8.211a}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_\parallel}}{\partial \theta \cdot \partial \phi} = \begin{pmatrix} -\cos\theta\sin\phi \\ \cos\theta\cos\phi \\ 0 \end{pmatrix}, \tag{8.211b}$$

$$\frac{\partial^2 \widehat{\mathfrak{D}_\parallel}}{\partial \phi^2} = \begin{pmatrix} -\sin\theta\cos\phi \\ -\sin\theta\sin\phi \\ 0 \end{pmatrix}. \tag{8.211c}$$

Chapter 9

relax development

This chapter is for developers or those who would like to extend the functionality of relax. It is not required for using relax. If you would like to make modifications to the relax source code please subscribe to all the relax mailing lists (see the open source infrastructure chapter for more details). Announcements are sent to "relax-announce at gna.org" whereas "relax-users at gna.org" is the list where discussions about the usage of relax should be posted. "relax-devel at gna.org" is where all discussions about the development of relax including feature requests, program design, or any other discussions relating to relax's structure or code should be posted. Finally, "relax-commits at gna.org" is where all changes to relax's code and documentation, as well as changes to the web pages, are automatically sent to. Anyone interested in joining the project should subscribe to all four lists.

9.1 Version control using Subversion

The development of relax requires the use of the Subversion (SVN) version control software downloadable from http://subversion.tigris.org/. The source code to relax is stored in an SVN repository located at http://svn.gna.org/svn/relax/. Every single change which has ever made to the program is recorded within this repository. For more information see the open source infrastructure chapter.

Although the downloadable distribution archives can be modified it is best that the most current and up to date revision (the *head* revision) is modified instead. More information about the basics of version control and how this is implemented in Subversion can be found in the Subversion book located at http://svnbook.red-bean.com/.

If you are not currently a relax developer you can check out the head revision, assuming that 1.2 is the current major version number, by typing

```
$ svn co svn://svn.gna.org/svn/relax/1.2 relax
```

Otherwise if you are a developer type

```
$ svn co svn+ssh://xxxxx@svn.gna.org/svn/relax/1.2 relax
```

replacing `xxxxx` with your Gna! login name. If your version is out of date it can be updated to the latest revision by typing

```
$ svn up
```

Modifications can be made to these sources.

9.2 Coding conventions

The following conventions should be followed at all times for any code to be accepted into the relax repository. A Python script which tests if code meets relax's coding conventions can be downloaded from http://nmr-relax.com/scripts/code_validator. The main reason for these conventions is for readability. By using a consistent coding style and a high comment ratio, the code becomes much easier to read for non-coders and those new to Python. It significantly decreases the barrier of entry into the relax source code for NMR spectroscopists.

9.2.1 Indentation

Indentation should be set to four spaces rather than a tab character. This is the recommendation given in the Python style guide found at http://www.python.org/doc/essays/styleguide.html. Emacs should automatically set the tabstop correctly. For vi add the following lines to the `vimrc` file:

```
set tabstop=4
set shiftwidth=4
set expandtab
```

For UNIX systems, including Linux and Mac OS X, the `vimrc` file is '`~/.vimrc`' whereas in MS Windows the file is '`$VIM/_vimrc`' which is usually '`C:\Program Files\vim\_vimrc`'. Certain versions of vim, those within the 6.2 series, contain a bug where the tabstop value cannot be changed using the `vimrc` file (although typing '`:set tabstop=4`' in vim will fix it). One solution is to edit the file '`python.vim`' which on GNU/Linux systems is located in the path '`/usr/share/vim/ftplugin/`'. It contains the two lines

```
" Python always uses a 'tabstop' of 8.
setlocal ts=8
```

If these lines are deleted the bug will be removed. Another way to fix the problem is to install newer versions of the run-time files (which will do the same thing).

9.2.2 Doc strings

The following are relax's conventions for docstrings. Many of these are Python conventions.

- Each line of the should be set to no more than 100 characters long (this includes all leading white space). For one line docstrings, the trailing double quotes are ignored.

- The standard Python convention of a one line description separated from a detailed description by an empty line should be adhered to. This line must start with a capital letter and end in a period. This convention is required for certain docstring parsers (see the Python docs).
- All functions should have a docstring describing in detail the function, structure, and organisation of the code.
- A docstring should be followed by an empty line.
- Indentation of the docstring should be the same as that of the first line of code, excluding indented lists, etc.

An example of a single line docstring is:

```
    def delete(self):
        """Function for deleting all model-free data."""
```

An example of a multiline docstring is:

```
def aic(chi2, k, n):
    """Akaike's Information Criteria (AIC).

    The formula is::

        AIC = chi2 + 2k


    @param chi2:    The minimised chi-squared value.
    @type chi2:     float
    @param k:       The number of parameters in the model.
    @type k:        int
    @param n:       The dimension of the relaxation data set.
    @type n:        int
    @return:        The AIC value.
    @rtype:         float
    """

    return chi2 + 2.0*k
```

9.2.3 Variable, function, and class names

In relax a mixture of both camel case (eg. CamelCase) and lower case with underscores is used. Despite the variability there are fixed rules which should be adhered to. These naming conventions should be observed at all times.

Variables and functions

For both variables and functions lower case with underscores between words is always used. This is for readability as the convention is much more fluent than camel case. A few rare

exceptions exist, an example is the Brownian diffusion tensor parameter of anisotropy $\mathfrak{D}_a$ which is referenced as `cdp.diff_tensor.Da`. As a rule though all new variable or function names should be kept as lower case. An example of this convention for both the variable name and function name is:

```
    def assemble_param_vector(self, spin=None, spin_id=None, sim_index=None, model_type=None):
        """Assemble the model-free parameter vector (as numpy array).

        If the spin argument is supplied, then the spin_id argument will be ignored.

        @keyword spin:          The spin data container.
        @type spin:             SpinContainer instance
        @keyword spin_id:       The spin identification string.
        @type spin_id:          str
        @keyword sim_index:     The optional MC simulation index.
        @type sim_index:        int
        @keyword model_type:    The optional parameter set, one of 'all', 'diff', 'mf', or
                                'local_tm'.
        @type model_type:       str or None
        @return:                An array of the parameter values of the model-free model.
        @rtype:                 numpy array
        """

        # Initialise.
        param_vector = []

        # Determine the model type.
        if not model_type:
            model_type = self.determine_param_set_type()

        # Diffusion tensor parameters.
        if model_type == 'diff' or model_type == 'all':
            # Normal parameters.
            if sim_index == None:
                # Spherical diffusion.
                if cdp.diff_tensor.type == 'sphere':
                    param_vector.append(cdp.diff_tensor.tm)
```

Classes

For classes relax uses a mix of camel case (for example all the `RelaxError` objects) and underscores (for example `Model_free`). The first letter in all cases is always capitalised. Generally the camel case is reserved for very low level classes which are involved in the program's infrastructure. Examples include the RelaxError code, the threading code, and the `self.relax.data` code. All the data analysis specific code, generic code, interface code, etc. uses underscores between the words with only the first letter capitalised. One exception is the space mapping class `OpenDX`, the reason being that the program is called 'OpenDX'. An example is:

```
class Model_free_main:
    """Class containing functions specific to model-free analysis."""

    def are_mf_params_set(self, spin):
        """Test if the model-free parameter values are set.
```

```
        @param spin:    The spin container object.
        @type spin:     SpinContainer instance
        @return:        The name of the first parameter in the parameter list in which the
                        corresponding parameter value is None.  If all parameters are set, then None
                        is returned.
        @rtype:         str or None
        """

        # Deselected residue.
        if spin.select == 0:
            return
```

Long names

If you have a look at a few relax source files, you will notice that the variable, function, and class names can be quite long. For example the model-free function '`disassemble_param_vector()`' and the RelaxError class '`RelaxNoSequenceError`'. While this is not normal for coding, it is an important component of relax as it facilitates the reading of the source code by a non-coder or someone not familiar with the codebase. Iteration counters can be single letter variables such as '`i`', '`j`', '`k`', etc., however for all other variables, functions, and classes please attempt to use descriptive names which are instantly identifiable. Please minimise the amount of abbreviations used and only use those which are obvious. For example naming the parameter vector '`self.param_vector`', the relaxation data '`relax_data`', the model selection class '`class Model_selection`', the type of spheroidal diffusion '`spheroid_type`', etc.

9.2.4 Whitespace

The following conventions are for general code cleanliness and readability:

- Trailing whitespace should be avoided, although this is not very important.
- All functions should be preceded by two empty lines. The only exception is the first function of the class definition.
- Function arguments should be separated by a comma followed by a single space.
- The assignment operator should be surrounded by spaces, for example '`tm␣=␣1e-8`'. The exception is function arguments where for example '`self.classic_colour(res_num=None,␣width=0.3)`'.
- The comparison operators should also be surrounded by spaces, e.g. '`␣<␣`', '`␣>␣`', '`␣==␣`', '`␣<=␣`', '`␣=>␣`', '`␣<>␣`', '`␣!=␣`', '`␣is␣`', and '`␣in␣`'.

An example which shows most of these conventions is:

```
class␣Scientific_data(Base_struct_API):
␣␣␣␣"""The␣Scientific␣Python␣specific␣data␣object."""

␣␣␣␣#␣Identification␣string.
```

```
    id = 'scientific'


    def __find_bonded_atom(self, attached_atom, mol_type, res):
        """Find the atom named attached_atom directly bonded to the desired atom.

        @param attached_atom:    The name of the attached atom to return.
        @type attached_atom:     str
        @param mol_type:         The type of the molecule.  This can be one of 'protein', 'nucleic acid',
                                 or 'other'.
        @type mol_type:          str
        @param res:              The Scientific Python residue object.
        @type res:               Scientific Python residue instance
        @return:                 A tuple of information about the bonded atom.
        @rtype:                  tuple consisting of the atom number (int), atom name (str), element
                                 name (str), and atomic position (Numeric array of len 3)
        """

        # Init.
        bonded_found = False

        # The attached atom is in the residue.
        if attached_atom in res.atoms:
            # The bonded atom object.
            bonded = res[attached_atom]
```

9.2.5 Comments

Comments are a very important component within relax. In the current source code the percentage of comment lines relative to lines of code ranges from 15% to over 30% for different files. The average comment density would be close to 25%. The purpose of having so many comment lines, much more than you would expect from source code, is so that the relax's code is fully self documented. It allows someone who is not familiar with the codebase to read the code and quickly understand what is happening. It simplifies the process of learning and allows NMR spectroscopists who are not coders to dive into the code. If writing code for relax, please attempt to maintain the tradition by aiming towards a 25% comment ratio. The comment should be descriptive of what the code below it is supposed to do. Most importantly the comment explains why that code exists. The script http://nmr-relax.com/scripts/code_validator can be used to check the comment density.

9.3 Submitting changes to the relax project

9.3.1 Submitting changes as a patch

The preferred method for submitting fixes and improvements to the relax source code is by the creation of a patch. If your changes are a fix make sure you have submitted a bug report to the bug tracker located at https://gna.org/bugs/?group=relax first. See section 3.3 on page 16 for more details. Two methods can be used to generate the patch – the Unix command `diff` or the Subversion program. The resultant file `patch` of either the `diff` or `svn` command described below can be posted to the "relax-devel at gna.org" mailing list. Please label within your post which version of relax you modified or which

revision the patch is for. Also try to create a commit log message according to the format described in section 9.4.4 on page 126 for one of the relax committers to use as a template for committing the change.

9.3.2 Modification of official releases – creating patches with diff

If your modifications have been made to the source code of one of the official relax releases (for example 1.2.2) then the Unix command `diff` can be used to create a patch. A patch file is simply the output of the diff command run with the recursive flag and presented in the 'unified' format. Therefore two directories need to be compared. If the original sources are located in the directory `relax_orig` and the modified sources in `relax_mod` then the patch can be created by typing

```
$ diff -ur relax_orig relax_mod > patch
```

9.3.3 Modification of the latest sources – creating patches with Subversion

If possible changes to the latest sources is preferred. Using the most up to date sources from the relax SVN repository will significantly aid the relax developers to incorporate your changes back into the main development line. To check out the current development line see section 9.1 on page 119 for details. Prior to submitting a patch to the mailing list your sources should be updated to include the most recent changes. To do this type

```
$ svn up
```

and note the revision number to include in your post. The update may cause a conflict if changes added to the repository clash with your modifications. If this occurs see the Subversion book at http://svnbook.red-bean.com/ for details on how to resolve the conflict or submit a message to the relax-devel list.

Once the sources are up to date your changes can be can be converted into the patch text file. Using SVN creating a patch is easy. Just type

```
$ svn diff > patch
```

in the base relax directory.

9.4 Committers

9.4.1 Becoming a committer

Anyone can become a relax developer and obtain commit access to the relax repository. The main criteria for selection by the relax developers is to show good judgement, competence in producing good patches, compliance with the coding and commit log conventions, comportment on the mailing lists, not producing too many bugs, only taking on challenges which can be handled, and the skill in judging your own abilities. You will also need to

have an understanding of the concepts of version control specifically those relating to Subversion. The SVN book at http://svnbook.red-bean.com/ contains all the version control information you will need. After a number of patches have been submitted and accepted any of the relax developers can propose that you receive commit access. If a number of developers agree while no one says no then commit access will be offered.

One area where coding ability can be demonstrated is within the relax test suite. The addition of tests, especially those where the relax internal data structures of `self.relax.data` are scrutinised, can be a good starting point for learning the structure of relax. This is because the introduction of bugs has no effect on normal program execution. The relax test suite is an ideal proving ground.

If skills in only certain areas of relax development, for example in creation of the documentation, an understanding of C but not python, an understanding of solely the code of the user interface, or an understanding of the code specific to a certain type of data analysis methodology, then partial commit access may be granted. Although you will have the ability to make modifications to any part of the repository please make modifications only those areas for which you have permission.

9.4.2 Joining Gna!

The first step in becoming a committer is to create a Gna! account. Go to https://gna.org/account/register.php and type in a login name, password, real name, and the email address you would like to use. You will then get an automatic email from Gna! which will contain a link to activate your registration.

9.4.3 Joining the relax project

The second step in becoming a committer is to register to become a member of the relax project. Go to the Gna! website (https://gna.org/) and login. Click on 'My Groups' to go to https://gna.org/my/groups.php. In the section 'Request for inclusion' type 'relax' and hit enter. Select relax and write something in the comments. If you have been approved (see section 9.4.1) you will be added to the project committers list.

9.4.4 Format of the commit logs

If you are a relax developer and you have commit access to the repository the following conventions should be followed for all commit messages.

- The length of all lines in the commit log should never exceed 100 characters. This is so that the log message viewed in either emails or by the command prompt command `svn log` is legible.
- The first line of the commit log should be a short description or synopsis of the changes. If the change relates to a bug or a task, include the bug and task number using the notation `type #num` where type is either `bug`, `task` or `support` and `num` is the id number (for example `bug #6503`). This terminology is important because the

Gna! infrastructure knows how to translate this into a link to the issue. Also include a link to the issue.

- The second line should be blank.
- If the commit is a bug fix reported by a non-committer or if the commit originates from a patch posted by a non-committer the next lines should be reserved for crediting. The name of the person and their obfuscated email address (for example edward _at_ nmr-relax _dot_ com) should be included in the message.
- Next should be another blank line.
- If the commit relates to an entry in the bug tracker or to a discussion on the mailing lists then the web address of either the bug report or the mailing list archive message should be cited in the next section (separated from the synopsis or credit section by a blank line). All relevant links should be included to allow easy navigation between the repository, mailing lists, bug tracker, etc. An example is bug #5559 which is located at https://gna.org/bugs/?func=detailitem&item_id=5559 and the post to "relax-devel at gna.org" describing the fix to that bug which is located at https://mail.gna.org/public/relax-devel/2006-03/msg00013.html.
- A full description with all the details can follow. This description should follow a blank line, can itself be sectioned using blank lines, and finally the log is terminated by one blank line at the end of the message.

An example of a commit message for the closure of a bug is:

```
Fixing the rest of bug #7241 (https://gna.org/bugs/?7241).

Bug #7241 was thought to be fixed in in r2591 and r2593, the commit messages describing
the solution being located at https://mail.gna.org/public/relax-commits/2006-09/msg00064.html
(Message-id:  <E1GTgBi-0000R6-4h@subversion.gna.org>) for r2591 and
https://mail.gna.org/public/relax-commits/2006-10/msg00001.html (Message-id:
<E1GTt6C-0005rk-8q@subversion.gna.org>) for r2593.

However this was not the only place that the Scientific Python PDB data structure
peptide_chains was being accessed.  The chains were being accessed in the file
'generic_fns/sequence.py' when the sequence was being read out of the PDB file.  This
has now been modified with changes similar to r2591 and r2593.
```

An example of a commit message for changes relating to a task is:

```
This change implements half of task #3630 (https://gna.org/task/?3630).

The docstring in the generic optimisation function has been modified to clear up the
ambiguity cased by supplying the option 'None' together with Newton optimisation.
```

One last commit message example is:

```
Added the API documentation creation (using Epydoc) to the Scons scripts.

The Scons target to create the HTML API documentation is called 'api_manual_html'.  The
documentation can be created by typing:  $ scons api_manual_html

The function 'compile_api_manual_html()' was added to the 'scons/manuals.py' file.
This function runs the 'epydoc' command.  All the Epydoc options are specified in that
function.
```

9.4.5 Discussing major changes

If you are contemplating major changes, either for a bug fix, to add a completely new feature or user function for your own work, to improve the behaviour of part the program, or any other potentially disruptive modifications, please discuss these ideas on the relax-devel mailing list. If the planned changes have the potential to introduce problems the creation of a private branch may be suggested.

9.5 Branches

9.5.1 Branch creation

If a change is likely to be disruptive or cause breakages in the program, the use of your own temporary branch is recommended. This private branch is a complete copy of one of the main development lines wherein you can make changes without disrupting the other developers. Although called a private branch every change is visible to all other developers and each commit will result in an automatic email to the relax-commits mailing list. Other developers are even able to check out your branch and make modifications to it. Private branches can also be used for testing ideas. If the idea does not work the branch can be deleted from the repository (in reality the branch will always exist between the revision numbers of its creation and deletion and can always be resurrected). For example to create a branch from the main 1.3 development line called `molmol_macros` whereby new Molmol macros are to be written, type

```
$ svn cp svn+ssh://xxxxx@svn.gna.org/svn/relax/1.3 \
svn+ssh://xxxxx@svn.gna.org/svn/relax/branches/molmol_macros
```

replacing `xxxxx` with your login name. You can then check out your private branch by typing

```
$ svn co svn+ssh://xxxxx@svn.gna.org/svn/relax/branches/molmol_macros
```

which will create a directory called `molmol_macros` containing all the relax source files. To have the files placed into a different directory, type the name of that directory at the end of the last command. Modifications can be made to this copy while normal development continues on the main line.

9.5.2 Keeping the branch up to date using `svnmerge.py`

As you develop your branch, changes will be occurring simultaneously within the main line. These changes should be merged into your branch on a regular basis to avoid large incompatible changes from forming between the two branches. To simplify this process, the `svnmerge.py` script located at http://www.orcaware.com/svn/wiki/Svnmerge.py can be used. It is best to download the trunk version from that page, unless that version is non-functional. Once you have this script, the merging from the main line to your private branch must be initialised by typing, from within the checked out copy of your branch,

```
$ svnmerge.py init
```

This then needs to be committed using the automatically generated log

```
$ svn ci -F svnmerge-commit-message.txt
```

To keep up to date, simply type

```
$ svnmerge.py merge
```

If conflicts have occurred please refer to the Subversion book at http://svnbook.red-bean.com/ for information on how to resolve the problem. Otherwise, or once fixed, the main line revisions merged into your branch can be committed using the automatically generated log file:

```
$ svn ci -F svnmerge-commit-message.txt
```

9.5.3 Merging the branch back into the main line

Once you have completed the modifications desired for your branch, all changes which have occurred in the main line have been merged using `svnmerge.py`, and the changes have been approved for merging back into the main line – then your branch can be merged. First check out a copy of the main line,

```
$ svn co svn+ssh://xxxxx@svn.gna.org/svn/relax/1.3 relax-1.3
```

or update a previously checked out version,

```
$ svn up
```

Then `svnmerge.py` can be utilised again. First initialise the merging process by typing, from within the checked out copy of the main line,

```
$ svnmerge.py init svn+ssh://xxxxx@svn.gna.org/svn/relax/branches/molmol_macros
```

Then commit the change

```
$ svn ci -F svnmerge-commit-message.txt
```

To merge the branch and commit the changes, type

```
$ svnmerge.py merge --bidirectional
```

```
$ svn ci -F svnmerge-commit-message.txt
```

Finally the merge properties need to be removed

```
$ svnmerge.py uninit -S svn+ssh://xxxxx@svn.gna.org/svn/relax/branches/molmol_macros
```

the changes commited

```
$ svn ci -F svnmerge-commit-message.txt
```

and your private branch deleted

```
$ svn rm svn+ssh://xxxxx@svn.gna.org/svn/relax/branches/molmol_macros
```

9.6 The SCons build system

The SCons build system was chosen over other build systems including `'make'` as it is a cross-platform build system which can be used in Unix, GNU/Linux, Mac OS X, and even MS Windows (the correct compilers are nevertheless required). Various components of the program relax can be created using the SCons utility. This includes C module compilation, manual creation, distribution creation, and cleaning up and removing certain files. The file 'sconstruct' in the base relax directory, which consists of python code, directs the operation of SCons for the various functions.

9.6.1 SCons help

Multiple functions have been built into the `'sconstruct'` script and the modules of the `'scons'` directory. Each of these can be selected by specifying different 'targets' when running SCons. A description of each target is given by the SCons help system which is accessible by typing `'scons --help'` in the base relax directory.

9.6.2 C module compilation

As described in the installation chapter, typing `'scons'` in the base directory will create the shared objects or dll files which are imported into Python as modules.

9.6.3 Compilation of the user manual (PDF version)

To create the PDF version of the relax user manual type

```
$ scons user_manual_pdf
```

in the base directory. SCons will then run a series of shell commands to create the manual from the LaTeX sources located in the `'docs/latex'` directory. This is dependent on the programs `'latex'`, `'makeindex'`, `'dvips'`, and `'ps2pdf'` being located within the environment's path.

9.6.4 Compilation of the user manual (HTML version)

The HTML version of the relax user manual is made by typing

```
$ scons user_manual_html
```

in the base directory. One command calling the program `'latex2html'` will be executed which will create HTML pages from the LaTeX sources.

9.6.5 Compilation of the API documentation (HTML version)

The HTML version of the relax API documentation is made by typing

```
$ scons api_manual_html
```

in the base directory. The programs Epydoc and Graphviz are required for creating this documentation. The resultant HTML pages will be located in the director `'docs/api/index.html'`.

9.6.6 Making distribution archives

Two types of distribution archive can be created from the currently checked out sources – the source and binary distributions. To create the source distribution type

```
$ scons source_dist
```

whereas to create the binary distribution, whereby the C modules are compiled and the resultant shared objects are included in the bzipped tar file, type

```
$ scons binary_dist
```

If a binary distribution does not exist for your architecture feel free to create it yourself and contribute the archive to be included on the download pages. To do this you will need to check out the appropriate tagged branch from the relax subversion repository. If the current stable release is called 1.2.3 check out that branch by typing

```
$ svn co svn+ssh://bugman@svn.gna.org/svn/relax/tags/1.2.3 relax
```

replacing 'bugman' with your user name if you are a relax developer, otherwise typing

```
$ svn co svn://svn.gna.org/svn/relax/tags/1.2.3 relax
```

Then build the binary distribution and send a message to the relax development mailing list. If compilation does not work please submit a bug to the bug tracker system at https://gna.org/bugs/?group=relax detailing the relax version, operation system, architecture, and any other information you believe will help to solve the problem. More information about donating binary distributions to the relax project is given in the open source infrastructure chapter.

9.6.7 Cleaning up

If the command

```
$ scons clean
```

is run in the base directory all Python byte compiled files `'*.pyc'`, all C object files `'*.o'` and `'*.os'`, all C shared object files `'*.so'`, and any backup files with the extension `'*.bak'` are removed from all sub-directories. In addition any temporary LaTeX compilation files are removed from the `'docs/latex'` directory.

9.7 The core design of relax

To enable flexibility the internal structure of relax is modular. By construction the large collection of independent data analysis tools can be used individually and relatively easily

by any new code implementing other forms of relaxation data analysis or even by other programs. The core modular design of the program is shown in Figure 9.1.

9.7.1 The divisions of relax's source code

relax's source code can be divided into five major areas: the initialisation code, the user interface (UI) code, the functional code, the number crunching code, and the code storing the program state.

Initialisation: The code belonging to this section initialises the program, processes the command-line arguments, and determines what mode the program will be run in including the choice of the UI.

UI: The user interface. Currently the prompt and the script are the only user interfaces into relax. There are other program modes which are not part of a user interface. These include the test mode in which the program instantly exits and threading mode which is spawned by a parent process and waits for commands. In the future a graphical user interface (GUI), a web based interface, or any other type of interface may be added.

Functional code: This code is the main part of the program. It includes anything which does not fit into the other sections and comprises the generic code, the specific code, and the specific setup code used as an interface between the two.

Number crunching: The computationally expensive code belongs in this section.

Program state: The state of the program is contained within the data structure `self.relax.data` which is accessible from all parts of the program. It should only be read by the generic, specific, and number crunching code. Only the generic and specific code should change its contents.

9.7.2 The major components of relax

Each of the boxes in Figure 9.1 represents a different grouping of code.

relax: The top level module. This initialises the entire program, tests the dependencies, places the custom errors into the module `__builtin__`, and prints the program's introduction message.

Command line arguments: This code processes the arguments supplied to the program and decides whether to print the help message, initialise the prompt, execute a script, initialise a different UI, run the program in test mode, or run the program as a slave thread.

Prompt: The main user interface consisting of a Python prompt. The namespace of the interpreter contains the various user functions which are front ends to the generic code. The user functions are simply Python functions which test the supplied arguments to make sure they are of the correct type (string, integer, list, or any other

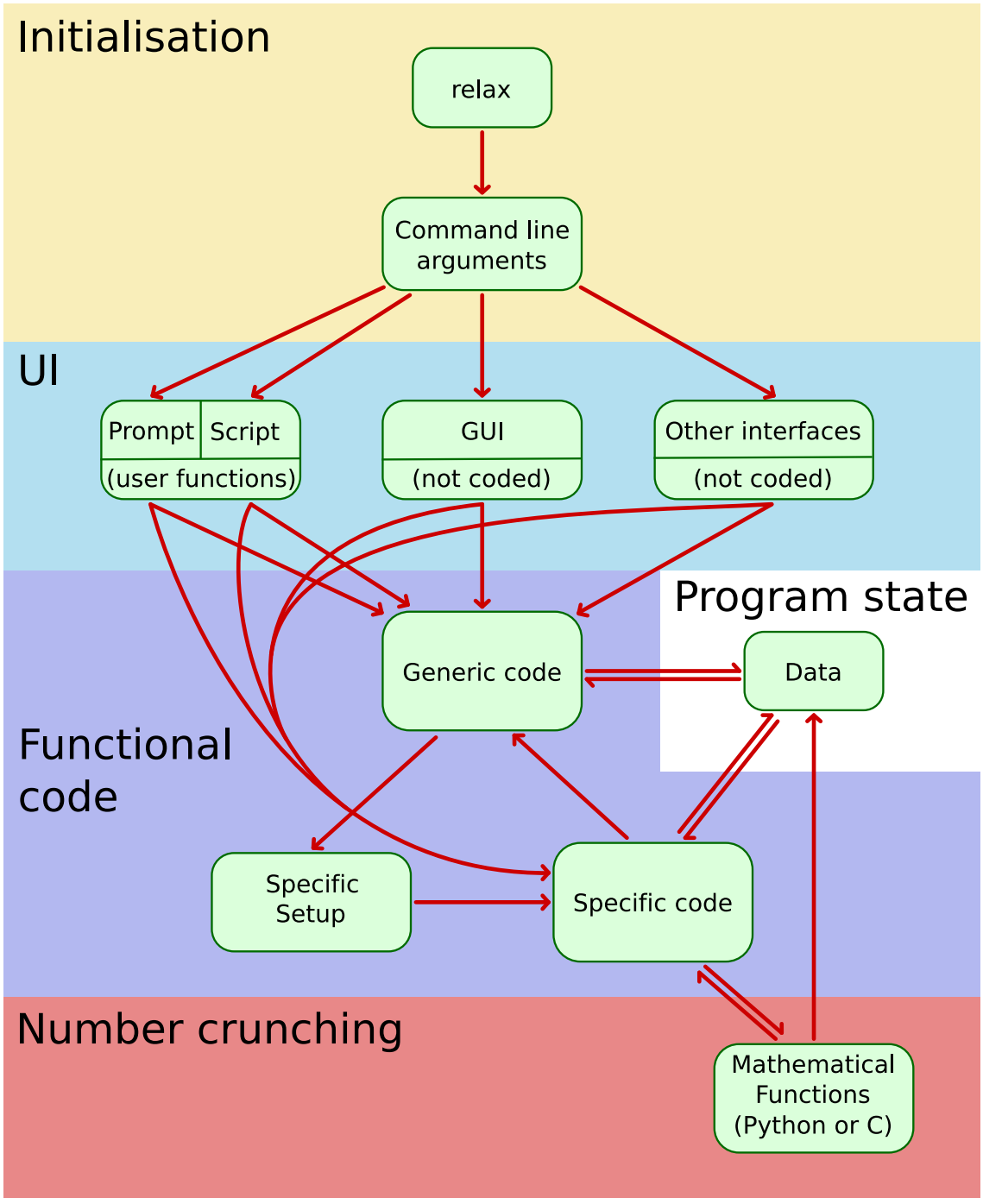

Figure 9.1: The core design of relax.

type) before sending the values to the generic code. The code for the prompt is located in the directory `prompt/`.

Script: If a script is supplied on the command line or executed within another user interface it will be run in the same namespace as that of the prompt. Hence the script has access to all the user functions available at the relax prompt. This allows commands which are typed at the prompt to be pasted directly and unmodified into a text file to be run as a script.

GUI: The graphical user interface. Although not coded the most mature and least destabilising widget set to use would be QT. The GUI should be relatively easy to tie into relax. The design is such that the GUI can be dropped straight into relax without effecting the normal prompt and script based operation of the program.

Other interfaces: Any number of interfaces for example other GUIs, an ncurses interface, a web based interface, or an MPI interface could be added to relax without modification of the current sources.

Generic code: This code includes classes and functions which are independent of the UI and not specific to a certain data pipe type, for example not being involved in model-free analysis, relaxation curve-fitting, the NOE calculation, and reduced spectral density mapping. All this code is located in the directory `generic_fns/`.

Specific setup: This code implements the internal interface between the generic and specific code. The generic code calls the specific setup asking for a specific function for the given data pipe type. For example by asking for the minimise function when the data pipe type is model-free analysis the function `self.relax.specific.model_free.minimise()` is returned. Although the generic code accesses the specific code solely through this interface the specific code can access the generic code directly. The code is located in the file `specific_fns/specific_setup.py`.

Specific code: This is the code which is specific to the data pipe type – model-free analysis, relaxation curve-fitting, reduced spectral density mapping, and the NOE calculation. Each type is located in a separate file in the directory `specific_fns/`.

Mathematical functions: This is reserved for CPU intensive code involved in calculations. The code may be written in Python however C code can be used to significantly increase the speed of the calculations. For optimisation the code can include function evaluations, calculation of gradients, and calculation of Hessians. These functions are located in the directory `maths_fns/`.

Data: The program state stored in the class `self.relax.data`. This class contains all the program data and is accessed by the generic and specific code. The mathematical functions may also access this data but this is not recommended. The structure is initialised by the file `data.py` and the data is modified solely by the generic and specific code.

9.8 The mailing lists

9.8.1 Private vs. public messages

If you would like to start a private discussion, please label your email as such. Private messages are however strongly discouraged, only start a private conversation if you really must.

If you receive a reply to a message you have sent, a bug report you have filed, etc. which has not been sent to the mailing list and has not been labelled as private, then the most likely explanation is that 'reply-to-all' has not been used and hence the mailing list has not been included on the CC list. If this occurs, please ask the person if the message was meant to be private and refrain from discussing any of the comments within the post. Save these comments until after the person responds by saying that the message was private or re-sends the message to the mailing list. Try to encourage public messages if you think that the post need not be private and if you think that it would be useful for the mailing list archives.

For thread consistency, if you send a message which accidentally misses the mailing list, please do not then forward the previously sent message to the list. For better readability of the mailing list archives, it is best that you create an entirely new message responding to the original post. Just cut and paste your miss-directed message into your new message. That way the thread will be continuous – there will not be any messages missing from the middle of the thread in between the original post and your forwarded message.

To simplify the process of checking if the message was supposed to be private, you could cut and paste the following message (modifying it as you see fit):

```
The following is the standard pre-composed response to a post not sent to the relax
mailing lists and not labelled as private.  If you would like to start a private
conversation about relax, please label your message as such.  If you really must start
a private exchange, please respond to this message saying so.  If your message was meant
to be sent to the relax mailing list, please resend the message hitting 'reply-to-all' or
making sure that the mailing list is in the CC field.  Please do not forward your message,
for thread consistency in the mailing list archives sending a new message in response to
the original post with the text of the old is best.
```

9.9 The bug, task, and support request trackers

relax's infrastructure includes three different issue trackers. These are the bug tracker, the task tracker, and the support request tracker.

9.9.1 Submitting a bug report

If someone reports a bug to one of the relax mailing lists, ask that person if they would like to create a bug report for that problem, pointing them to the submission web page. This is a good starting point to allow the person to become more involved in the relax project.

If they do not respond or say that they would prefer not to, then you can create bug report for the issue linking to the original message and crediting the person for reporting the issue.

9.9.2 Assigning an issue to yourself

If you are a relax committer and see an issue which you would like to solve, please assign that issue to yourself before you start work on it. The assignment will prevent duplicated efforts. If you can see an area where relax needs work, feel free to create a report within task tracker and then assign the task to yourself.

9.9.3 Closing an issue

When closing an issue (whether a bug report, a task, or a support request) a number of steps need to be taken. The tracker status should be changed to 'Done' and the issue 'Closed'. In addition, a message should be included which states the repository revision and the relax-commits mailing list archive link (with the message-id) in which the issue was solved. If multiple commits were required, then include all the revisions and as many links as possible (if a task required many commits, the relax-commits links could be skipped). An example is bug #7402 where the closing comment was:

```
This documentation bug was fixed in r2641.  The commit message is located at
https://mail.gna.org/public/relax-commits/2006-10/msg00073.html (Message-id:
<E1GYG4l-0002kK-Jx@subversion.gna.org>).
```

9.10 Links, links, and more links

Creating links throughout the relax infrastructure is important for two major reasons – navigation and search engine indexing. When including a link to a post within the mailing list archives, please include the message-id email header. This enables subscribers to the mailing lists to search for the specific message within their local copy of the email messages.

9.10.1 Navigation

To be able to easily navigate between the relax infrastructure components – the bug tracker, the task tracker, the support request tracker, the relax-devel mailing list, the commit logs, and the SVN and CVS repositories – try to include as many links as possible.

For example a bug may first be reported on the relax-users mailing list, then placed within the bug tracker, discussed on relax-devel, a fix committed to the repository, and finally the bug report closed. To be able to follow this chain, links are very important (email message-ids are also important). When the bug is first added to the bug tracker, a link to the relax-users mailing list archive message and the message-id should be included. If you start a discussion on relax-devel, try to include links to the bug tracker entry and the relax-users message. When committing a fix to the repository, include links to the bug

report, to the start of the thread in the mailing list archive, and the original message to relax-users. Then when the bug report is closed, include the revision number of the fix and a link to the relax-commits archive message (and message-id). By having all these links, it is then very easy for someone else to jump between the systems and follow the progression of the bug fix.

If you send a message referring to an old post which was sent the relax mailing lists, an old bug report, or any other archived information, please take the time to find that original information in the archives and include a link to it (including the message-id if relevant). It is much more efficient for a single person to hunt down that message than for the many recipients of your post to search for the message themselves. By including the link, you decrease the overhead of following the mailing list.

9.10.2 Search engine indexing

Having a large web of links across relax's infrastructure aids in the search engine indexing of the mailing list archives and the http://nmr-relax.com web site. The web of links is useful for catching those Google bots. That way the Google searching of the mailing list archives located on the communication web page will be more up to date. However to increase the search engine ranking of the web site, links to http://nmr-relax.com from external sites is required. This is one reason why relax is a registered freshmeat project.

Chapter 10

Alphabetical listing of user functions

The following is a listing with descriptions of all the user functions available within the relax prompt and scripting environments. These are simply an alphabetical list of the docstrings which can normally be viewed in prompt mode by typing '`help(function)`'.

10.1 A warning about the formatting

The following documentation of the user functions has been automatically generated by a script which extracts and formats the docstring associated with each function. There may therefore be instances where the formatting has failed or where there are inconsistencies.

10.2 The list of functions

Each user function is presented within it's own subsection with the documentation broken into multiple parts: the synopsis, the default arguments, and the sections from the function's docstring.

10.2.1 The synopsis

The synopsis presents a brief description of the function. It is taken as the first line of the docstring when browsing the help system.

10.2.2 Defaults

This section lists all the arguments taken by the function and their default values. To invoke the function type the function name then in brackets type a comma separated list of arguments.

The first argument printed is always 'self' but you can safely ignore it. 'self' is part of the object oriented programming within Python and is automatically prefixed to the list of arguments you supply. Therefore you can't provide 'self' as the first argument even if you do try.

10.2.3 Docstring sectioning

All other sections are created from the sectioning of the user function docstring.

10.2.4 align_tensor.copy()

Synopsis

Function for copying alignment tensor data.

Defaults

align_tensor.copy(self, tensor_from=None, pipe_from=None, tensor_to=None, pipe_to=None)

Keyword Arguments

tensor_from: The identification string of the alignment tensor to copy the data from.

pipe_from: The name of the data pipe to copy the alignment tensor data from.

tensor_to: The identification string of the alignment tensor to copy the data to.

pipe_to: The name of the data pipe to copy the alignment tensor data to.

Description

This function will copy the alignment tensor data to a new tensor or a new data pipe. The destination data pipe must not contain any alignment tensor data corresponding to the tensor_to label. If the pipe_from or pipe_to arguments are not supplied, then both will default to the current data pipe. Both the tensor_from and tensor_to arguments must be supplied.

Examples

To copy the alignment tensor data corresponding to '`Pf1`' from the data pipe '`old`' to the current data pipe, type one of:

```
relax> align_tensor.copy('Pf1', 'old')
```

```
relax> align_tensor.copy(tensor_from='Pf1',
pipe_from='old')
```

To copy the alignment tensor data corresponding to '`Otting`' from the current data pipe to the data pipe new, type one of:

```
relax> align_tensor.copy('Otting', pipe_to=
'new')
```

```
relax> align_tensor.copy(tensor_from=
'Otting', pipe_to='new')
```

To copy the alignment tensor data of '`Otting`' to that of '`Otting new`', type one of:

```
relax> align_tensor.copy('Otting',
tensor_to='Otting new')
```

```
relax> align_tensor.copy(tensor_from='Pf1',
tensor_to='Otting new')
```

10.2.5 align_tensor.delete()

Synopsis

Function for deleting alignment tensor data.

Defaults

align_tensor.delete(self, tensor=None)

Keyword Arguments

tensor: The alignment tensor identification string.

Description

This function will delete the specified alignment tensor data from the current data pipe.

10.2.6 align_tensor.display()

Synopsis

Function for displaying the alignment tensor information.

Defaults

align_tensor.display(self, tensor=None)

Keyword Arguments

tensor: The alignment tensor identification string.

10.2.7 align_tensor.fix()

Synopsis

Fix all alignment tensors so that they do not change during optimisation.

Defaults

align_tensor.fix(self, fixed=True)

Keyword Arguments

fixed: The flag specifying if the tensors should be fixed or variable.

10.2.8 align_tensor.init()

Synopsis

Function for initialising the alignment tensor.

Defaults

align_tensor.init(self, tensor=None, params=None, scale=1.0, angle_units='deg', param_types=0, errors=False)

Keyword Arguments

tensor: The alignment tensor identification string.

params: The alignment tensor data.

scale: The alignment tensor eigenvalue scaling value.

angle_units: The units for the angle parameters.

param_types: A flag to select different parameter combinations.

errors: A flag which determines if the alignment tensor data or its errors are being input.

Description

Using this function, the alignment tensor data can be set up. The params argument should be a tuple of floating point numbers (a list surrounded by round brakets). These correspond to the parameters of the tensor, which can be specified by the param_types argument, where the values correspond to

0 – {Sxx, Syy, Sxy, Sxz, Syz} (unitless),

1 – {Szz, Sxx-yy, Sxy, Sxz, Syz} (Pales default format),

2 – {Axx, Ayy, Axy, Axz, Ayz} (unitless),

3 – {Azz, Axx-yy, Axy, Axz, Ayz} (unitless),

4 – {Axx, Ayy, Axy, Axz, Ayz} (units of Hertz),

5 – {Azz, Axx-yy, Axy, Axz, Ayz} (units of Hertz),

6 – {Pxx, Pyy, Pxy, Pxz, Pyz} (unitless),

7 – {Pzz, Pxx-yy, Pxy, Pxz, Pyz} (unitless),

Other formats may be added later. The relationship between the Saupe order matrix S and the alignment tensor A is

$$S = 3/2\ A.$$

The probability matrix P is related to the alignment tensor A by

$$A = P - 1/3\ I,$$

where I is the identity matrix. For the alignment tensor to be supplied in Hertz, the bond vectors must all be of equal length.

Examples

To set a rhombic tensor to the run 'CaM', type one of:

```
relax> align_tensor.init('super media',
(-8.6322e-05, -5.5786e-04, -3.1732e-05,
2.2927e-05, 2.8599e-04), param_types=1)
```

```
relax> align_tensor.init(tensor='super
media', params=(-8.6322e-05, -5.5786e-04,
-3.1732e-05, 2.2927e-05, 2.8599e-04),
param_types=1)
```

10.2.9 align_tensor.matrix_angles()

Synopsis

Function for calculating the 5D angles between all alignment tensors.

Defaults

align_tensor.matrix_angles(self, basis_set=0, tensors=None)

Keyword Arguments

basis_set: The basis set to operate with.

tensors: A list of the tensors to apply the calculation to. If None, all tensors are used.

Description

This function will calculate the angles between all loaded alignment tensors for the current data pipe. The matrices are first converted to a 5D vector form and then then angles are calculated. The angles are dependent on the basis set. If the basis_set argument is set to the default of 0, the vectors {Sxx, Syy, Sxy, Sxz, Syz} are used. If the basis_set argument is set to 1, the vectors {Szz, Sxxyy, Sxy, Sxz, Syz} are used instead.

10.2.10 align_tensor.reduction()

Synopsis

Specify that one tensor is a reduction of another.

Defaults

align_tensor.reduction(self, full_tensor=None, red_tensor=None)

Keyword Arguments

full_tensor: The full alignment tensor.

red_tensor: The reduce alignment tensor.

Description

Prior to optimisation of the N-state model and Frame Order theories using alignment tensors, which tensor is a reduction of which other tensor must be specified through this user function.

Examples

To state that the alignment tensor loaded as '`chi3 C-dom`' is a reduction of '`chi3 N-dom`', type:

```
relax> align_tensor.reduction(full_tensor=
'chi3 N-dom', red_tensor='chi3 C-dom')
```

10.2.11 align_tensor.set_domain()

Synopsis

Set the domain label for the alignment tensor.

Defaults

align_tensor.set_domain(self, tensor=None, domain=None)

Keyword Arguments

tensor: The alignment tensor to assign the domain label to.

domain: The domain label.

Description

Prior to optimisation of the N-state model or Frame Order theories, the domain to which each alignment tensor belongs must be specified.

Examples

To link the alignment tensor loaded as '`chi3 C-dom`' to the C-terminal domain '`C`', type:

```
relax> align_tensor.set_domain(tensor='chi3
C-dom', domain='C')
```

10.2.12 align_tensor.svd()

Synopsis

Function for calculating the singular values for all tensors and the condition number.

Defaults

align_tensor.svd(self, basis_set=0, tensors=None)

Keyword Arguments

basis_set: The basis set to operate with.

tensors: A list of the tensors to apply the calculation to. If None, all tensors are used.

Description

This function will, using SVD, calculate the singular values of all tensors loaded for the current data pipe. If the basis_set argument is set to the default of 0, the matrix on which SVD will be performed is composed of the unitary basis set {Sxx, Syy, Sxy, Sxz, Syz} layed out as:

```
| Sxx1 Syy1 Sxy1 Sxz1 Syz1 |
| Sxx2 Syy2 Sxy2 Sxz2 Syz2 |
| Sxx3 Syy3 Sxy3 Sxz3 Syz3 |
|  .    .    .    .    .   |
|  .    .    .    .    .   |
|  .    .    .    .    .   |
| SxxN SyyN SxyN SxzN SyzN |
```

If basis_set is set to 1, the geometric basis set consisting of the stretching and skewing parameters Szz and Sxxyy respectively {Szz, Sxxyy, Sxy, Sxz, Syz} will be used instead. The matrix is:

```
| Szz1 Sxxyy1 Sxy1 Sxz1 Syz1 |
| Szz2 Sxxyy2 Sxy2 Sxz2 Syz2 |
| Szz3 Sxxyy3 Sxy3 Sxz3 Syz3 |
|  .     .     .    .    .   |
|  .     .     .    .    .   |
|  .     .     .    .    .   |
| SzzN SxxyyN SxyN SxzN SyzN |
```

The relationships between the geometric and unitary basis sets are:

```
Szz = - Sxx - Syy,
Sxxyy = Sxx - Syy,
```

The SVD values and condition number are dependendent upon the basis set chosen.

10.2.13 angle_diff_frame()

Synopsis

Calculate the angles defining the XH bond vector within the diffusion frame.

Defaults

angle_diff_frame(self)

Description

If the diffusion tensor is isotropic, then nothing will be done.

If the diffusion tensor is axially symmetric, then the angle α will be calculated for each XH bond vector.

If the diffusion tensor is asymmetric, then the three angles will be calculated.

10.2.14 calc()

Synopsis

Function for calculating the function value.

Defaults

calc(self, verbosity=1)

Keyword Arguments

verbosity: The amount of information to print to screen. Zero corresponds to minimal output while higher values increase the amount of output. The default value is 1.

10.2.15 consistency_tests.set_frq()

Synopsis

Function for selecting which relaxation data to use in the consistency tests.

Defaults

consistency_tests.set_frq(self, frq=None)

Keyword Arguments

frq: The spectrometer frequency in Hz.

Description

This function will select the relaxation data to use in the consistency tests corresponding to the given frequencies.

Examples

```
relax> consistency_tests.set_frq(600.0 *
1e6)
relax> consistency_tests.set_frq(frq=600.0 *
1e6)
```

10.2.16 dasha.create()

Synopsis

Function for creating the Dasha script.

Defaults

dasha.create(self, algor='LM', dir=None, force=False)

Keyword Arguments

algor: The minimisation algorithm.

dir: The directory to place the files.

force: A flag which if set to True will cause the results file to be overwritten if it already exists.

Description

The script file created is called '`dir/dasha_script`'.

Optimisation algorithms

The two minimisation algorithms within Dasha are accessible through the algor argument which can be set to:

'`LM`' - The Levenberg-Marquardt algorithm.

'`NR`' - Newton-Raphson algorithm.

For Levenberg-Marquardt minimisation, the function '`lmin`' will be called, while for Newton -Raphson, the function '`min`' will be executed.

10.2.17 dasha.execute()

Synopsis

Function for executing Dasha.

Defaults

dasha.execute(self, dir=None, force=False, binary='dasha')

Keyword Arguments

dir: The directory to place the files.

force: A flag which if set to True will cause the results file to be overwritten if it already exists.

binary: The name of the executable Dasha program file.

Execution

Dasha will be executed as

```
$ dasha < dasha_script | tee dasha_results
```

If you would like to use a different Dasha executable file, change the keyword argument '`binary`' to the appropriate file name. If the file is not located within the environment's path, include the full path in front of the binary file name.

10.2.18 dasha.extract()

Synopsis

Function for extracting data from the Dasha results file.

Defaults

dasha.extract(self, dir=None)

Keyword Arguments

dir: The directory where the file '`dasha_results`' is found.

10.2.19 deselect.all()

Synopsis

Function for deselecting all spins.

Defaults

deselect.all(self)

Examples

To deselect all spins, simply type:

```
relax> deselect.all()
```

10.2.20 deselect.read()

Synopsis

Function for deselecting the spins contained in a file.

Defaults

deselect.read(self, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, sep=None, spin_id=None, boolean='AND', change_all=False)

Keyword Arguments

file: The name of the file containing the list of spins to deselect.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

boolean: The boolean operator specifying how spins should be selected.

change_all: A flag specifying if all other spins should be changed.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Empty lines and lines beginning with a hash are ignored.

The '`change_all`' flag argument default is False meaning that all spins currently either selected or deselected will remain that way. Setting the argument to True will cause all spins not specified in the file to be selected.

Examples

To deselect all overlapped residues listed with residue numbers in the first column of the file '`unresolved`', type:

```
relax> deselect.read('unresolved',
res_num_col=1)
```

```
relax> deselect.read(file='unresolved',
res_num_col=1)
```

To deselect the spins in the second column of the relaxation data file '`r1.600`' while selecting all other spins, for example type:

```
relax> deselect.read('r1.600', spin_num_col=
2, change_all=True)
```

```
relax> deselect.read(file='r1.600',
spin_num_col=2, change_all=True)
```

10.2.21 deselect.reverse()

Synopsis

Function for the reversal of the spin selection.

Defaults

deselect.reverse(self, spin_id=None)

Keyword Arguments

spin_id: The spin identification string.

Description

By supplying the spin_id argument, a subset of spin can have their selection status reversed.

Examples

To deselect all currently selected spins and select those which are deselected type:

```
relax> deselect.reverse()
```

10.2.22 deselect.spin()

Synopsis

Function for deselecting specific spins.

Defaults

deselect.spin(self, spin_id=None, change_all=False)

Keyword Arguments

spin_id: The spin identification string.

change_all: A flag specifying if all other spins should be changed.

Description

The '`change_all`' flag argument default is False meaning that all spins currently either selected or deselected will remain that way. Setting the argument to True will cause all spins not specified by '`spin_id`' to be selected.

Examples

To deselect all glycines and alanines, type:

```
relax> deselect.spin(spin_id=':GLY|:ALA')
```

To deselect residue 12 MET type:

```
relax> deselect.spin(':12')
```

```
relax> deselect.spin(spin_id=':12')
```

```
relax> deselect.spin(spin_id=':12&:MET')
```

10.2.23 diffusion_tensor.copy()

Synopsis

Function for copying diffusion tensor data from one data pipe to another.

Defaults

diffusion_tensor.copy(self, pipe_from=None, pipe_to=None)

Keyword Arguments

pipe_from: The name of the data pipe to copy the diffusion tensor data from.

pipe_to: The name of the data pipe to copy the diffusion tensor data to.

Description

This function will copy the diffusion tensor data between data pipes. The destination data pipe must not contain any diffusion tensor data. If the pipe_from or pipe_to arguments are not supplied, then both will default to the current data pipe (hence giving one argument is essential).

Examples

To copy the diffusion tensor from the data pipe '`m1`' to the current data pipe, type:

```
relax> diffusion_tensor.copy('m1')
```

```
relax> diffusion_tensor.copy(pipe_from='m1')
```

To copy the diffusion tensor from the current data pipe to the data pipe '`m9`', type:

```
relax> diffusion_tensor.copy(pipe_to='m9')
```

To copy the diffusion tensor from the data pipe '`m1`' to '`m2`', type:

```
relax> diffusion_tensor.copy('m1', 'm2')
```

```
relax> diffusion_tensor.copy(pipe_from='m1',
pipe_to='m2')
```

10.2.24 diffusion_tensor.delete()

Synopsis

Function for deleting diffusion tensor data.

Defaults

diffusion_tensor.delete(self)

Description

This function will delete all diffusion tensor data from the current data pipe.

10.2.25 diffusion_tensor.display()

Synopsis

Function for displaying the diffusion tensor information.

Defaults

diffusion_tensor.display(self)

10.2.26 diffusion_tensor.init()

Synopsis

Function for initialising the diffusion tensor.

Defaults

diffusion_tensor.init(self, params=None, time_scale=1.0, d_scale=1.0, angle_units='deg', param_types=0, spheroid_type=None, fixed=True)

Keyword Arguments

params: The diffusion tensor data.

time_scale: The correlation time scaling value.

d_scale: The diffusion tensor eigenvalue scaling value.

angle_units: The units for the angle parameters.

param_types: A flag to select different parameter combinations.

spheroid_type: A string which, if supplied together with spheroid parameters, will restrict the tensor to either being '`oblate`' or '`prolate`'.

fixed: A flag specifying whether the diffusion tensor is fixed or can be optimised.

The sphere (isotropic diffusion)

When the molecule diffuses as a sphere, all three eigenvalues of the diffusion tensor are equal, $\mathfrak{D}_x = \mathfrak{D}_y = \mathfrak{D}_z$. In this case, the orientation of the XH bond vector within the diffusion frame is inconsequential to relaxation, hence, the spherical or Euler angles are undefined. Therefore solely a single geometric parameter, either τ_m or $\mathfrak{D}_{iso}$, can fully and sufficiently parameterise the diffusion tensor. The correlation function for the global rotational diffusion is

$$C(\tau) = \frac{1}{5} e^{-\tau / \tau_m},$$

To select isotropic diffusion, the parameters argument should be a single floating point number. The number is the value of the isotropic global correlation time, τ_m, in seconds. To specify the time in nanoseconds, set the '`time_scale`' argument to 1e-9. Alternative parameters can be used by changing the '`param_types`' flag to the following integers

0 – {tm} (Default),

1 – {Diso},

where

$$1 / \tau_m = 6\mathfrak{D}_{iso}.$$

The spheroid (axially symmetric diffusion)

When two of the three eigenvalues of the diffusion tensor are equal, the molecule diffuses as a spheroid. Four pieces of information are required to specify this tensor, the two geometric parameters, $\mathfrak{D}_{iso}$ and $\mathfrak{D}_a$, and the two orientational parameters, the polar angle θ and the azimuthal angle ϕ describing the orientation of the axis of symmetry. The correlation function of the global diffusion is

$$C(\tau) = \frac{1}{5} \sum_{i=-1}^{1} c_i \cdot e^{-\tau / \tau_i},$$

where

$$c\text{-}1 = 1/4\ (3\ \delta_z\hat{}2 - 1)\hat{}2,$$

c0 = 3 δ_z^2 (1 - δ_z^2),

c1 = 3/4 (δ_z^2 - 1)^2,

and

$1 / \tau$ -1 = $6\mathfrak{D}_{iso}$ - $2\mathfrak{D}_a$,

$1 / \tau$ 0 = $6\mathfrak{D}_{iso}$ - $\mathfrak{D}_a$,

$1 / \tau$ 1 = $6\mathfrak{D}_{iso}$ + $2\mathfrak{D}_a$.

The direction cosine δ_z is defined as the cosine of the angle α between the XH bond vector and the unique axis of the diffusion tensor.

To select axially symmetric anisotropic diffusion, the parameters argument should be a tuple of floating point numbers of length four. A tuple is a type of data structure enclosed in round brackets, the elements of which are separated by commas. Alternative sets of parameters, '`param_types`', are

0 – $\{\tau_m, \mathfrak{D}_a, \theta, \phi\}$ (Default),

1 – $\{\mathfrak{D}_{iso}, \mathfrak{D}_a, \theta, \phi\}$,

2 – $\{\tau_m, \mathfrak{D}_{ratio}, \theta, \phi\}$,

3 – $\{\mathfrak{D}_\parallel, \mathfrak{D}_\perp, \theta, \phi\}$,

4 – $\{\mathfrak{D}_{iso}, \mathfrak{D}_{ratio}, \theta, \phi\}$,

where

$\tau_m = 1 / 6\mathfrak{D}_{iso}$,

$\mathfrak{D}_{iso} = 1/3\ (\mathfrak{D}_\parallel + 2\mathfrak{D}_\perp)$,

$\mathfrak{D}_a = \mathfrak{D}_\parallel$ - $\mathfrak{D}_\perp$,

$\mathfrak{D}_{ratio} = \mathfrak{D}_\parallel / \mathfrak{D}_\perp$.

The spherical angles $\{\theta, \phi\}$ orienting the unique axis of the diffusion tensor within the PDB frame are defined between

$0 \leq \theta \leq \pi$,

$0 \leq \phi \leq 2\pi$,

while the angle α which is the angle between this axis and the given XH bond vector is defined between

$0 \leq \alpha \leq 2\pi$.

The '`spheroid_type`' argument should be '`oblate`', '`prolate`', or None. The argument will be ignored if the diffusion tensor is not axially symmetric. If '`oblate`' is given, then the constraint $\mathfrak{D}_a \leq 0$ is used while if '`prolate`' is given, then the constraint $\mathfrak{D}_a \geq 0$ is used. If nothing is supplied, then $\mathfrak{D}_a$ will be allowed to have any values. To prevent minimisation of diffusion tensor parameters in a space with two minima, it is recommended to specify which tensor is to be minimised, thereby partitioning the two minima into the two subspaces along the boundry $\mathfrak{D}_a = 0$.

The ellipsoid (rhombic diffusion)

When all three eigenvalues of the diffusion tensor are different, the molecule diffuses as an ellipsoid. This diffusion is also known as fully anisotropic, asymmetric, or rhombic. The full tensor is specified by six pieces of information, the three geometric parameters $\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, and $\mathfrak{D}_r$ representing the isotropic, anisotropic, and rhombic components of the tensor, and the three Euler angles α, β, and γ orienting the tensor within the PDB frame. The correlation function is

$$C(\tau) = \frac{1}{5}\sum_{i=-2}^{2} c_i \cdot e^{-\tau / \tau_i},$$

where the weights on the exponentials are

c-2 = 1/4 (d + e),

c-1 = 3 δ_y^2 δ_z^2,

c0 = 3 δ_x^2 δ_z^2,

c1 = 3 δ_x^2 δ_y^2,

c2 = 1/4 (d + e).

Let

$\mathfrak{R} = \mathrm{sqrt}(1 + 3\mathfrak{D}_r)$,

then

d = 3 (δ_x^4 + δ_y^4 + δ_z^4) - 1,

e = - 1 / $\Re$ ((1 + 3$\mathfrak{D}_r$)(δ_x^4 + 2δ_y^2 δ_z^2) + (1 - 3$\mathfrak{D}_r$)(δ_y^4 + 2δ_x^2 δ_z^2) - 2(δ_z^4 + 2δ_x^2 δ_y^2)).

The correlation times are

$1 / \tau$ -2 = $6\mathfrak{D}_{iso}$ - $2\mathfrak{D}_a$. $\Re$,

$1 / \tau$ -1 = $6\mathfrak{D}_{iso}$ - $\mathfrak{D}_a$ (1 + $3\mathfrak{D}_r$),

$1 / \tau$ 0 = $6\mathfrak{D}_{iso}$ - $\mathfrak{D}_a$ (1 - $3\mathfrak{D}_r$),

$1 / \tau$ 1 = $6\mathfrak{D}_{iso}$ + $2\mathfrak{D}_a$,

$1 / \tau$ 1 = $6\mathfrak{D}_{iso}$ + $2\mathfrak{D}_a$. $\Re$.

The three direction cosines δ_x, δ_y, and δ_z are the coordinates of a unit vector parallel to the XH bond vector. Hence the unit vector is [δ_x, δ_y, δ_z].

To select fully anisotropic diffusion, the parameters argument should be a tuple of length six. A tuple is a type of data structure enclosed in round brackets, the elements of which are separated by commas. Alternative sets of parameters, '`param_types`', are

0 – {τ_m, $\mathfrak{D}_a$, $\mathfrak{D}_r$, α, β, γ} (Default),

1 – {$\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, $\mathfrak{D}_r$, α, β, γ},

2 – {$\mathfrak{D}_x$, $\mathfrak{D}_y$, $\mathfrak{D}_z$, α, β, γ},

3 – {Dxx, Dyy, Dzz, Dxy, Dxz, Dyz},

where

$\tau_m = 1 / 6\mathfrak{D}_{iso}$,

$\mathfrak{D}_{iso} = 1/3\ (\mathfrak{D}_x + \mathfrak{D}_y + \mathfrak{D}_z)$,

$\mathfrak{D}_a = \mathfrak{D}_z - (\mathfrak{D}_x + \mathfrak{D}_y)/2$,

$\mathfrak{D}_r = (\mathfrak{D}_y - \mathfrak{D}_x)/2\mathfrak{D}_a$.

The angles α, β, and γ are the Euler angles describing the diffusion tensor within the PDB frame. These angles are defined using the z-y-z axis rotation notation where α is the initial rotation angle around the z-axis, β is the rotation angle around the y-axis, and γ is the final rotation around the z-axis again. The angles are defined between

$0 \leq \alpha \leq 2\pi$,

$0 \leq \beta \leq \pi$,

$0 \leq \gamma \leq 2\pi$.

Within the PDB frame, the XH bond vector is described using the spherical angles θ and ϕ where θ is the polar angle and ϕ is the azimuthal angle defined between

$0 \leq \theta \leq \pi$,

$0 \leq \phi \leq 2\pi$.

When param_types is set to 3, then the elements of the diffusion tensor matrix defined within the PDB frame can be supplied.

Units

The '`time_scale`' argument should be a floating point number. The only parameter affected by this value is τ_m.

The '`d_scale`' argument should also be a floating point number. Parameters affected by this value are $\mathfrak{D}_{iso}$, $\mathfrak{D}_\parallel$, $\mathfrak{D}_\perp$, $\mathfrak{D}_a$, $\mathfrak{D}_x$, $\mathfrak{D}_y$, and $\mathfrak{D}_z$. Significantly, $\mathfrak{D}_r$ is not affected.

The '`angle_units`' argument should either be the string '`deg`' or '`rad`'. Parameters affected are θ, ϕ, α, β, and γ.

Examples

To set an isotropic diffusion tensor with a correlation time of 10 ns, type:

```
relax> diffusion_tensor.init(10e-9)
```

```
relax> diffusion_tensor.init(params=10e-9)
```

```
relax> diffusion_tensor.init(10.0, 1e-9)
```

```
relax> diffusion_tensor.init(params=10.0,
time_scale=1e-9, fixed=True)
```

To select axially symmetric diffusion with a τ_m value of 8.5 ns, $\mathfrak{D}_{ratio}$ of 1.1, θ value of 20 degrees, and ϕ value of 20 degrees, type:

```
relax> diffusion_tensor.init((8.5e-9, 1.1,
20.0, 20.0), param_types=2)
```

To select a spheroid diffusion tensor with a $\mathfrak{D}_\parallel$ value of 1.698e7, $\mathfrak{D}_\perp$ value of 1.417e7, θ value of 67.174 degrees, and ϕ value of -83.718 degrees, type one of:

```
relax> diffusion_tensor.init((1.698e7,
1.417e7, 67.174, -83.718), param_types=3)
```

```
relax> diffusion_tensor.init(params=
(1.698e7, 1.417e7, 67.174, -83.718),
param_types=3)
```

```
relax> diffusion_tensor.init((1.698e-1,
1.417e-1, 67.174, -83.718), param_types=
3, d_scale=1e8)

relax> diffusion_tensor.init(params=(
1.698e-1, 1.417e-1, 67.174, -83.718),
param_types=3, d_scale=1e8)

relax> diffusion_tensor.init((1.698e-1,
1.417e-1, 1.1724, -1.4612), param_types=
3, d_scale=1e8, angle_units='rad')

relax> diffusion_tensor.init(params=(
1.698e-1, 1.417e-1, 1.1724, -1.4612),
param_types=3, d_scale=1e8, angle_units=
'rad', fixed=True)
```

To select ellipsoidal diffusion, type:

```
relax> diffusion_tensor.init((1.340e7,
1.516e7, 1.691e7, -82.027, -80.573, 65.568),
param_types=2)
```

10.2.27 dx.execute()

Synopsis

Function for running OpenDX.

Defaults

dx.execute(self, file='map', dir='dx', dx_exe='dx', vp_exec=True)

Keyword Arguments

file: The file name prefix. For example if file is set to '`temp`', then the OpenDX program temp.net will be loaded.

dir: The directory to change to for running OpenDX. If this is set to None, OpenDX will be run in the current directory.

δ_x_exe: The OpenDX executable file.

vp_exec: A flag specifying whether to execute the visual program automatically at start-up. The default of True causes the program to be executed.

10.2.28 dx.map()

Synopsis

Function for creating a map of the given space in OpenDX format.

Defaults

dx.map(self, params=None, map_type='Iso3D', spin_id=None, inc=20, lower=None, upper=None, axis_incs=5, file_prefix='map', dir='dx', point=None, point_file='point', remap=None)

Keyword Arguments

params: The parameters to be mapped. This argument should be an array of strings, the meanings of which are described below.

map_type: The type of map to create. For example the default, a 3D isosurface, the type is '`Iso3D`'. See below for more details.

spin_id: The spin identification numbe.

inc: The number of increments to map in each dimension. This value controls the resolution of the map.

lower: The lower bounds of the space. If you wish to change the lower bounds of the map then supply an array of length equal to the number of parameters in the model. A lower bound for each parameter must be supplied. If nothing is supplied then the defaults will be used.

upper: The upper bounds of the space. If you wish to change the upper bounds of the map then supply an array of length equal to the number of parameters in the model. A upper bound for each parameter must be supplied. If nothing is supplied then the defaults will be used.

axis_incs: The number of increments or ticks displaying parameter values along the axes of the OpenDX plot.

file_prefix: The file name. All the output files are prefixed with this name. The main file containing the data points will be called the value of '`file`'. The OpenDX program will be called '`file.net`' and the OpenDX import file will be called '`file.general`'.

dir: The directory to output files to. Set this to '`None`' if you do not want the files to be placed in subdirectory. If the directory does not exist, it will be created.

point: An array of parameter values where a point in the map, shown as a red sphere, will be placed. The length must be equal to the number of parameters.

point_file: The name of that the point output files will be prefixed with.

remap: A user supplied remapping function. This function will receive the parameter array and must return an array of equal length.

Map type

The map type can be changed by supplying the '`map_type`' keyword argument. Here is a list of currently supported map types:

(see table 10.1)

This argument is case insensitive.

Table 10.1: First table for the dx.map() user function.

Surface type	Pattern
3D isosurface	'Iso3D'

Examples

The following commands will generate a map of the extended model-free space for model 'm5' consisting of the parameters $\{S^2, S_f^2, \tau_s\}$. Files will be output into the directory 'dx' and will be prefixed by 'map'. In this case, the system is a protein and residue number 6 will be mapped.

```
relax> dx.map(['S2', 'S2f', 'ts'], spin_id=
':6')
```

```
relax> dx.map(['S2', 'S2f', 'ts'], spin_id=
':6', file_prefix='map', dir='dx')
```

```
relax> dx.map(params=['S2', 'S2f', 'ts'],
spin_id=':6', inc=20, file_prefix='map',
dir='dx')
```

```
relax> dx.map(params=['S2', 'S2f', 'ts'],
spin_id=':6', map_type='Iso3D', inc=20,
file_prefix='map', dir='dx')
```

To map the model-free space 'm4' for residue 2, spin N6 defined by the parameters $\{S^2, \tau_e, R_{ex}\}$, name the results 'test', and to place the files in the current directory, use one of the following commands:

```
relax> dx.map(['S2', 'te', 'Rex'], spin_id=
':2@N6', file_prefix='test', dir=None)
```

```
relax> dx.map(params=['S2', 'te', 'Rex'],
spin_id=':2@N6', inc=100, file_prefix=
'test', dir=None)
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Diffusion tensor parameter string matching patterns

(see table 10.2)

Model-free data type string matching patterns

(see table 10.3)

10.2.29 eliminate()

Synopsis

Function for model elimination.

Defaults

eliminate(self, function=None, args=None)

Keyword arguments

function: A user supplied function for model elimination.

args: A tuple of arguments for model elimination.

Description

This function is used for model validation to eliminate or reject models prior to model selection. Model validation is a part of mathematical modelling whereby models are either accepted or rejected.

Table 10.2: Second table for the dx.map() user function.

Data type	Object name	Patterns
Global correlation time - τ_m	'tm'	'^tm$'
Isotropic component of the diffusion tensor - $\mathfrak{D}_{iso}$	'Diso'	'[Dd]iso'
Anisotropic component of the diffusion tensor - $\mathfrak{D}_a$	'Da'	'[Dd]a'
Rhombic component of the diffusion tensor - $\mathfrak{D}_r$	'Dr'	'[Dd]r$'
Eigenvalue associated with the x-axis of the diffusion diffusion tensor - $\mathfrak{D}_x$	'Dx'	'[Dd]x'
Eigenvalue associated with the y-axis of the diffusion diffusion tensor - $\mathfrak{D}_y$	'Dy'	'[Dd]y'
Eigenvalue associated with the z-axis of the diffusion diffusion tensor - $\mathfrak{D}_z$	'Dz'	'[Dd]z'
Diffusion coefficient parallel to the major axis of the spheroid diffusion tensor - $\mathfrak{D}_{\parallel}$	'Dpar'	'[Dd]par'
Diffusion coefficient perpendicular to the major axis of the spheroid diffusion tensor - $\mathfrak{D}_{\perp}$	'Dper'	'[Dd]per'
Ratio of the parallel and perpendicular components of the spheroid diffusion tensor - $\mathfrak{D}_{ratio}$	'Dratio'	'[Dd]ratio'
The first Euler angle of the ellipsoid diffusion tensor - α	'alpha'	'^a$' or 'alpha'
The second Euler angle of the ellipsoid diffusion tensor - β	'beta'	'^b$' or 'beta'
The third Euler angle of the ellipsoid diffusion tensor - γ	'gamma'	'^g$' or 'gamma'
The polar angle defining the major axis of the spheroid diffusion tensor - θ	'theta'	'theta'
The azimuthal angle defining the major axis of the spheroid diffusion tensor - ϕ	'phi'	'phi'

Table 10.3: Third table for the dx.map() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Empirical rules are used for model rejection and are listed below. However these can be overridden by supplying a function. The function should accept five arguments, a string defining a certain parameter, the value of the parameter, the minimisation instance (ie the residue index if the model is residue specific), and the function arguments. If the model is rejected, the function should return True, otherwise it should return False. The function will be executed multiple times, once for each parameter of the model.

The 'args' keyword argument should be a tuple, a list enclosed in round brackets, and will be passed to the user supplied function or the inbuilt function. For a description of the arguments accepted by the inbuilt functions, see below.

Once a model is rejected, the select flag corresponding to that model will be set to False so that model selection, or any other function, will then skip the model.

Local τ_m model elimination rule

The local τ_m, in some cases, may exceed the value expected for a global correlation time. Generally the τ_m value will be stuck at the upper limit defined for the parameter. These models are eliminated using the rule:

$$\tau_m \geq \mathrm{c}$$

The default value of c is 50 ns, although this can be overridden by supplying the value (in seconds) as the first element of the args tuple.

Internal correlation times $\{\tau_e, \tau_f, \tau_s\}$ model elimination rules

These parameters may experience the same problem as the local τ_m in that the model fails and the parameter value is stuck at the upper limit. These parameters are constrained using the formula (τ_e, τ_f, $\tau_s \leq 2\tau_m$). These failed models are eliminated using the rule:

$$\tau_e,\ \tau_f,\ \tau_s \geq \mathrm{c}\ .\ \tau_m$$

The default value of c is 1.5. Because of round-off errors and the constraint algorithm, setting c to 2 will result in no models being eliminated as the minimised parameters will always be less than $2\tau_m$. The value can be changed by supplying the value as the second element of the tuple.

Arguments

The 'args' argument must be a tuple of length 2, the elements of which must be numbers. For example, to eliminate models which have a local τ_m value greater than 25 ns and models with internal correlation times greater than 1.5 times τ_m, set 'args' to (25 * 1e-9, 1.5).

10.2.30 fix()

Synopsis

Function for either fixing or allowing parameter values to change during optimisation.

Defaults

fix(self, element=None, fixed=True)

Keyword Arguments

element: Which element to fix.

fixed: A flag specifying if the parameters should be fixed or allowed to change.

Description

The keyword argument '`element`' can be any of the following:

'`diff`' - the diffusion tensor parameters. This will allow all diffusion tensor parameters to be toggled.

'`all_spins`' - using this keyword, all parameters from all spins will be toggled.

'`all`' - all parameter will be toggled. This is equivalent to combining both '`diff`' and '`all_spins`'.

The flag '`fixed`', if set to True, will fix parameters during optimisation whereas a value of False will allow parameters to vary.

10.2.31 frame_order.cone_pdb()

Synopsis

Create a PDB file representing the Frame Order cone models.

Defaults

frame_order.cone_pdb(self, size=30.0, inc=40, file='cone.pdb', dir=None, force=False)

Keyword Arguments

size: The size of the geometric object in Å.

inc: The number of increments used to create the geometric object.

file: The name of the PDB file to create.

dir: The directory where the file is to be located.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

This function creates a PDB file containing an artificial geometric structure representing the Frame Order cone models.

There are four different types of residue within the PDB. The pivot point is represented as as a single carbon atom of the residue '`PIV`'. The cone consists of numerous H atoms of the residue '`CON`'. The cone axis vector is presented as the residue '`AXE`' with one carbon atom positioned at the pivot and the other x Åaway on the cone axis (set by the size argument). Finally, if Monte Carlo have been performed, there will be multiple '`MCC`' residues representing the cone for each simulation, and multiple '`MCA`' residues representing the multiple cone axes.

To create the diffusion in a cone PDB representation, a uniform distribution of vectors on a sphere is generated using spherical coordinates with the polar angle defined by the cone axis. By incrementing the polar angle using an arccos distribution, a radial array of vectors representing latitude are created while incrementing the azimuthal angle evenly creates the longitudinal vectors. These are all placed into the PDB file as H atoms and are all connected using PDB CONECT records. Each H atom is connected to its two neighbours on the both the longitude and latitude. This creates a geometric PDB object with longitudinal and latitudinal lines representing the filled cone.

10.2.32 frame_order.domain_to_pdb()

Synopsis

Match the domains to PDB files.

Defaults

frame_order.domain_to_pdb(self, domain=None, pdb=None)

Keyword Arguments

domain: The domain to associate the PDB file to.

pdb: The PDB file to associate the domain to.

Description

To display the frame order cone models within Pymol, the two domains need to be associated with PDB files. Then the reference domain will be fixed in the PDB frame, and the moving domain will be rotated to its average position.

Examples

To set the '`N`' domain to the PDB file '`bax_N_1J7O_1st.pdb`', type one of:

```
relax> frame_order.domain_to_pdb('N',
'bax_N_1J7O_1st.pdb')
```

```
relax> frame_order.domain_to_pdb(domain='N',
pdb='bax_N_1J7O_1st.pdb')
```

10.2.33 frame_order.pivot()

Synopsis

Set the pivot point for the two body motion in the structural coordinate system.

Defaults

frame_order.pivot(self, pivot=None)

Keyword Arguments

pivot: The pivot point for the motion (e.g. the position between the 2 domains in PDB coordinates).

Examples

To set the pivot point, type one of:

```
relax> frame_order.pivot([12.067, 14.313,
-3.2675])
```

```
relax> frame_order.pivot(pivot=[12.067,
14.313, -3.2675])
```

10.2.34 frame_order.ref_domain()

Synopsis

Set the reference domain for the '`2-domain`' Frame Order theories.

Defaults

frame_order.ref_domain(self, ref=None)

Keyword Arguments

ref: The domain which will act as the frame of reference. This is only valid for the '`2-domain`' Frame Order theories.

Description

Prior to optimisation of the '`2-domain`' Frame Order theories, which of the two domains will act as the frame of reference must be specified. This is important for the attachment of cones to domains, etc.

Examples

To set up the isotropic cone frame order model with '`centre`' domain being the frame of reference, type:

```
relax> frame_order.ref_domain(ref='centre')
```

10.2.35 frame_order.select_model()

Synopsis

Select and set up the Frame Order model.

Defaults

frame_order.select_model(self, model=None)

Keyword Arguments

model: The name of the preset Frame Order model.

Description

Prior to optimisation, the Frame Order model should be selected. These models consist of three parameter categories:

- The average domain position. This includes the parameters ave_pos_alpha, ave_pos_beta, and ave_pos_gamma. These Euler angles rotate the tensors from the arbitrary PDB frame of the moving domain to the average domain position.

- The frame order eigenframe. This includes the parameters eigen_alpha, eigen_beta, and eigen_gamma. These Euler angles define the major modes of motion. The cone central axis is defined as the z-axis. The pseudo-elliptic cone x and y-axes are defined as the x and y-axes of the eigenframe.

- The cone parameters. These are defined as the tilt-torsion angles cone_theta_x, cone_theta_y, and cone_sigma_max. The cone_theta_x and cone_theta_y parameters define the two cone opening angles of the pseudo-ellipse. The amount of domain torsion is defined as the average domain position, plus and minus cone_sigma_max. The isotropic cones are defined by setting cone_theta_x = cone_theta_y and converting the single parameter into a 2^{nd} rank order parameter.

The list of available models are:

'`pseudo-ellipse`' - The pseudo-elliptic cone model. This is the full model consisting of the parameters ave_pos_alpha, ave_pos_beta, ave_pos_gamma, eigen_alpha, eigen_beta, eigen_gamma, cone_theta_x, cone_theta_y, and cone_sigma_max.

'`pseudo-ellipse, torsionless`' - The pseudo-elliptic cone with the torsion angle cone_sigma_max set to zero.

'`pseudo-ellipse, free rotor`' - The pseudo-elliptic cone with no torsion angle restriction.

'`iso cone`' - The isotropic cone model. The cone is defined by a single order parameter s1 which is related to the single cone opening angle cone_theta_x = cone_theta_y. Due to rotational symmetry about the cone axis, the average position α Euler angle ave_pos_alpha is dropped from the model. The symmetry also collapses the eigenframe to a single z-axis defined by the parameters axis_theta and axis_phi.

'`iso cone, torsionless`' - The isotropic cone model with the torsion angle cone_sigma_max set to zero.

'`iso cone, free rotor`' - The isotropic cone model with no torsion angle restriction.

'`line`' - The line cone model. This is the pseudo-elliptic cone with one of the cone angles, cone_theta_y, assumed to be statistically negligible. I.e. the cone angle is so small that it cannot be distinguished from noise.

'`line, torsionless`' - The line cone model with the torsion angle cone_sigma_max set to zero.

'`line, free rotor`' - The line cone model with no torsion angle restriction.

'`rotor`' - The only motion is a rotation about the cone axis restricted by the torsion angle cone_sigma_max.

'`rigid`' - No domain motions.

'`free rotor`' - The only motion is free rotation about the cone axis.

Examples

To select the isotropic cone model, type:

```
relax> frame_order.select_model(model='iso cone')
```

10.2.36 frq.set()

Synopsis

Set the spectrometer frequency of the experiment.

Defaults

frq.set(self, id=None, frq=None)

Keyword arguments

id: The experiment identification string.

frq: The spectrometer frequency in Hertz.

Description

This user function allows the spectrometer frequency of a given experiment to be set.

10.2.37 grace.view()

Synopsis

Function for running Grace.

Defaults

grace.view(self, file=None, dir='grace', grace_exe='xmgrace')

Keyword Arguments

file: The name of the file.

dir: The directory name.

grace_exe: The Grace executable file.

Description

This function can be used to execute Grace to view the specified file the Grace '`.agr`' file and the execute Grace. If the directory name is set to None, the file will be assumed to be in the current working directory.

Examples

To view the file '`s2.agr`' in the directory '`grace`', type:

```
relax> grace.view(file='s2.agr')
```

```
relax> grace.view(file='s2.agr', dir='grace')
```

10.2.38 grace.write()

Synopsis

Function for creating a grace '`.agr`' file.

Defaults

grace.write(self, x_data_type='spin', y_data_type=None, spin_id=None, plot_data='value', file=None, dir='grace', force=False, norm=False)

Keyword Arguments

x_data_type: The data type for the X-axis (no regular expression is allowed).

y_data_type: The data type for the Y-axis (no regular expression is allowed).

spin_id: The spin identification string.

plot_data: The data to use for the plot.

norm: Flag for the normalisation of series type data.

file: The name of the file.

dir: The directory name.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

This function is designed to be as flexible as possible so that any combination of data can be plotted. The output is in the format of a Grace plot (also known as ACE/gr, Xmgr, and xmgrace) which only supports two dimensional plots. Three types of keyword arguments can be used to create various types of plot. These include the X-axis and Y-axis data types, the spin identification string, and an argument for selecting what to actually plot.

The X-axis and Y-axis data type arguments should be plain strings, regular expression is not allowed. If the X-axis data type argument is not given, the plot will default to having the spin sequence along the x-axis. The two axes of the Grace plot can be absolutely any of the data types listed in the tables below. The only limitation, currently anyway, is that the data must belong to the same data pipe.

The spin identification string can be used to limit which spins are used in the plot. The default is that all spins will be used, however, these arguments can be used to select a subset of all spins, or a single spin for plots of Monte Carlo simulations, etc.

The property which is actually plotted can be controlled by the '`plot_data`' argument. It can be one of the following:

'`value`' – Plot values (with errors if they exist).

'`error`' – Plot errors.

'`sims`' – Plot the simulation values.

Normalisation is only allowed for series type data, for example the R_2 exponential curves, and will be ignored for all other data types. If the norm flag is set to True then the y-value of the first point of the series will be set to 1. This normalisation is useful for highlighting errors in the data sets.

Examples

To write the NOE values for all spins to the Grace file 'noe.agr', type one of:

```
relax> grace.write('spin', 'noe', file='noe.agr')
```

```
relax> grace.write(y_data_type='noe', file='noe.agr')
```

```
relax> grace.write(x_data_type='spin', y_data_type='noe', file='noe.agr')
```

```
relax> grace.write(y_data_type='noe', file='noe.agr', force=True)
```

To create a Grace file of 'S2' vs. 'te' for all spins, type one of:

```
relax> grace.write('S2', 'te', file='s2_te.agr')
```

```
relax> grace.write(x_data_type='S2', y_data_type='te', file='s2_te.agr')
```

```
relax> grace.write(x_data_type='S2', y_data_type='te', file='s2_te.agr', force=True)
```

To create a Grace file of the Monte Carlo simulation values of 'Rex' vs. 'te' for residue 123, type one of:

```
relax> grace.write('Rex', 'te', spin_id=':123', plot_data='sims', file='s2_te.agr')
```

```
relax> grace.write(x_data_type='Rex', y_data_type='te', spin_id=':123', plot_data='sims', file='s2_te.agr')
```

By plotting the peak intensities, the integrity of exponential relaxation curves can be checked and anomalies searched for prior to model-free analysis or reduced spectral density mapping. For example the normalised average peak intensities can be plotted verses the relaxation time periods for the relaxation curves of all residues of a protein. The normalisation, whereby the initial peak intensity of each residue I(0) is set to 1, emphasises any problems. To produce this Grace file, type:

```
relax> grace.write(x_data_type='relax_times', y_data_type='ave_int', file='intensities_norm.agr', force=True, norm=True)
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Minimisation statistic data type string matching patterns

(see table 10.4)

NOE calculation data type string matching patterns

(see table 10.5)

Relaxation curve fitting data type string matching patterns

(see table 10.6)

Reduced spectral density mapping data type string matching patterns

(see table 10.7)

Model-free data type string matching patterns

(see table 10.8)

10.2.39 grid_search()

Synopsis

The grid search function.

Table 10.4: First table for the grace.write() user function.

Data type	Object name	Patterns
Chi-squared statistic	'chi2'	'^[Cc]hi2$' or '^[Cc]hi[-_][Ss]quare'
Iteration count	'iter'	'^[Ii]ter'
Function call count	'f_count'	'^[Ff].*[-_][Cc]ount'
Gradient call count	'g_count'	'^[Gg].*[-_][Cc]ount'
Hessian call count	'h_count'	'^[Hh].*[-_][Cc]ount'

Table 10.5: Second table for the grace.write() user function.

Data type	Object name	Patterns
Reference intensity	'ref'	'^[Rr]ef$' or '[Rr]ef[-_][Ii]nt'
Saturated intensity	'sat'	'^[Ss]at$' or '[Ss]at[-_][Ii]nt'
NOE	'noe'	'^[Nn][Oo][Ee]$'

Table 10.6: Third table for the grace.write() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.7: Fourth table for the grace.write() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.8: Fifth table for the grace.write() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Defaults

grid_search(self, lower=None, upper=None, inc=21, constraints=True, verbosity=1)

Keyword Arguments

lower: An array of the lower bound parameter values for the grid search. The length of the array should be equal to the number of parameters in the model.

upper: An array of the upper bound parameter values for the grid search. The length of the array should be equal to the number of parameters in the model.

inc: The number of increments to search over. If a single integer is given then the number of increments will be equal in all dimensions. Different numbers of increments in each direction can be set if 'inc' is set to an array of integers of length equal to the number of parameters.

constraints: A boolean flag specifying whether the parameters should be constrained. The default is to turn constraints on (constraints=True).

verbosity: The amount of information to print to screen. Zero corresponds to minimal output while higher values increase the amount of output. The default value is 1.

10.2.40 intro_off()

Synopsis

Turn the function introductions off.

Defaults

intro_off(self, verbose=True)

10.2.41 intro_on()

Synopsis

Turn the function introductions on.

Defaults

intro_on(self, verbose=True)

10.2.42 jw_mapping.set_frq()

Synopsis

Function for selecting which relaxation data to use in the $J(\omega)$ mapping.

Defaults

jw_mapping.set_frq(self, frq=None)

Keyword Arguments

frq: The spectrometer frequency in Hz.

Description

This function will select the relaxation data to use in the reduced spectral density mapping corresponding to the given frequency.

Examples

```
relax> jw_mapping.set_frq(600.0 * 1e6)
relax> jw_mapping.set_frq(frq=600.0 * 1e6)
```

10.2.43 minimise()

Synopsis

Minimisation function.

Defaults

minimise(self, *args, **keywords)

Arguments

The arguments, which should all be strings, specify the minimiser as well as its options. A minimum of one argument is required. As this calls the minfx function '`generic_minimise`' the full list of allowed arguments is shown below in the reproduced '`generic_minimise`' docstring. Ignore all sections except those labelled as minimisation algorithms and minimisation options. Also do not select the Method of Multipliers constraint algorithm as this is used in combination with the given minimisation algorithm if the keyword argument '`constraints`' is set to 1. The grid search algorithm should also not be selected as this is accessed using the '`grid`' function instead. The first argument passed will be set to the minimisation algorithm while all other arguments will be set to the minimisation options.

Keyword arguments differ from normal arguments having the form '`keyword = value`'. All arguments must precede keyword arguments in python. For more information see the examples section below or the python tutorial.

Keyword Arguments

func_tol: The function tolerance. This is used to terminate minimisation once the function value between iterations is less than the tolerance. The default value is 1e-25.

grad_tol: The gradient tolerance. Minimisation is terminated if the current gradient value is less than the tolerance. The default value is None.

max_iterations: The maximum number of iterations. The default value is 1e7.

constraints: A boolean flag specifying whether the parameters should be constrained. The default is to turn constraints on (constraints=True).

scaling: The diagonal scaling boolean flag. The default that scaling is on (scaling=True).

verbosity: The amount of information to print to screen. Zero corresponds to minimal output while higher values increase the amount of output. The default value is 1.

Diagonal scaling

Diagonal scaling is the transformation of parameter values such that each value has a similar order of magnitude. Certain minimisation techniques, for example the trust region methods, perform extremely poorly with badly scaled problems. In addition, methods which are insensitive to scaling such as Newton minimisation may still benefit due to the minimisation of round off errors.

In Model-free analysis for example, if $S^2 = 0.5$, $\tau_e =$ 200 ps, and $R_{ex} = 15$ 1/s at 600 MHz, the unscaled parameter vector would be [0.5, 2.0e-10, 1.055e-18]. R_{ex} is divided by (2 * π * 600,000,000)**2 to make it field strength independent. The scaling vector for this model may be something like [1.0, 1e-9, 1/(2 * π * 6e8)**2]. By dividing the unscaled parameter vector by the scaling vector the scaled parameter vector is [0.5, 0.2, 15.0]. To revert to the original unscaled parameter vector, the scaled parameter vector and scaling vector are multiplied.

Examples

To apply Newton minimisation together with the GMW81 Hessian modification algorithm, the More and Thuente line search algorithm, a function tolerance of 1e-25, no gradient tolerance, a maximum of 10,000,000 iterations, constraints turned on to limit parameter values, and have normal printout, type any combination of:

```
relax> minimise('newton')
```

```
relax> minimise('Newton')
relax> minimise('newton', 'gmw')
relax> minimise('newton', 'mt')
relax> minimise('newton', 'gmw', 'mt')
relax> minimise('newton', 'mt', 'gmw')
relax> minimise('newton', func_tol=1e-25)
relax> minimise('newton', func_tol=1e-25,
grad_tol=None)
relax> minimise('newton', max_iter=1e7)
relax> minimise('newton', constraints=True,
max_iter=1e7)
relax> minimise('newton', verbosity=1)
```

To use constrained Simplex minimisation with a maximum of 5000 iterations, type:

```
relax> minimise('simplex', constraints=True,
max_iter=5000)
```

Note

All the text which follows is a reproduction of the docstring of the generic_minimise function from the minfx python package. Only take note of the minimisation algorithms and minimisation options sections, the other sections are not relevant for this function. The Grid search and Method of Multipliers algorithms CANNOT be selected as minimisation algorithms for this function.

The section entitled Keyword Arguments is also completely inaccessible therefore please ignore that text.

Generic minimisation function.

This is a generic function which can be used to access all minimisers using the same set of function arguments. These are the function tolerance value for convergence tests, the maximum number of iterations, a flag specifying which data structures should be returned, and a flag specifying the amount of detail to print to screen.

Keyword Arguments

func: The function which returns the value.

dfunc: The function which returns the gradient.

d2func: The function which returns the Hessian.

args: The tuple of arguments to supply to the functions func, dfunc, and d2func.

x0: The vector of initial parameter value estimates (as an array).

min_algor: A string specifying which minimisation technique to use.

min_options: A tuple to pass to the minimisation function as the min_options keyword.

func_tol: The function tolerance value. Once the function value between iterations decreases below this value, minimisation is terminated.

grad_tol: The gradient tolerance value.

maxiter: The maximum number of iterations.

A: Linear constraint matrix m*n (A.x $\geq b$).

b: Linear constraint scalar vector (A.x $\geq b$).

l: Lower bound constraint vector ($l \leq x \leq u$).

u: Upper bound constraint vector ($l \leq x \leq u$).

c: User supplied constraint function.

dc: User supplied constraint gradient function.

d2c: User supplied constraint Hessian function.

full_output: A flag specifying which data structures should be returned.

print_flag: A flag specifying how much information should be printed to standard output during minimisation. 0 means no output, 1 means minimal output, and values above 1 increase the amount of output printed.

Minimisation output

The following values of the '`full_output`' flag will return, in tuple form, the following data

0 – '`xk`',

1 – '`(xk, fk,` k`, f_count, g_count, h_count, warning)`',

where the data names correspond to

'`xk`' – The array of minimised parameter values,

'`fk`' – The minimised function value,

'`k`' – The number of iterations,

'`f_count`' – The number of function calls,

'`g_count`' – The number of gradient calls,

'`h_count`' – The number of Hessian calls,

'`warning`' – The warning string.

Minimisation algorithms

A minimisation function is selected if the minimisation algorithm argument, which should be a string, matches a certain pattern. Because the python regular expression '`match`' statement is used, various strings can be supplied to select the same minimisation algorithm. Below is a list of the minimisation algorithms available together with the corresponding patterns.

This is a short description of python regular expression, for more information, see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'`[]`' – A sequence or set of characters to match to a single character. For example, '`[Nn]ewton`' will match both '`Newton`' and '`newton`'.

'`^`' – Match the start of the string.

'`$`' – Match the end of the string. For example, '`^[Ll][Mm]$`' will match '`lm`' and '`LM`' but will not match if characters are placed either before or after these strings.

To select a minimisation algorithm, set the argument to a string which matches the given pattern.

Unconstrained line search methods:

(see table 10.9)

Unconstrained trust-region methods:

(see table 10.10)

Unconstrained conjugate gradient methods:

(see table 10.11)

Miscellaneous unconstrained methods:

(see table 10.12)

Constrained methods:

(see table 10.13)

Global minimisation methods:

(see table 10.14)

Minimisation options

The minimisation options can be given in any order.

Line search algorithms. These are used in the line search methods and the conjugate gradient methods. The default is the Backtracking line search.

(see table 10.15)

Hessian modifications. These are used in the Newton, Dogleg, and Exact trust region algorithms.

(see table 10.16)

Hessian type, these are used in a few of the trust region methods including the Dogleg and Exact trust region algorithms. In these cases, when the Hessian type is set to Newton, a Hessian modification can also be supplied as above. The default Hessian type is Newton, and the default Hessian modification when Newton is selected is the GMW algorithm.

(see table 10.17)

For Newton minimisation, the default line search algorithm is the More and Thuente line search, while the default Hessian modification is the GMW algorithm.

10.2.44 model_free.create_model()

Synopsis

Function to create a model-free model.

Defaults

model_free.create_model(self, model=None, equation=None, params=None, spin_id=None)

Keyword Arguments

model: The name of the model-free model.

equation: The model-free equation.

params: The array of parameter names of the model.

spin_id: The spin identification string.

Model-free equation

'`mf_orig`' selects the original model-free equations with parameters $\{S^2, \tau_e\}$. '`mf_ext`' selects the extended model-free equations with parameters $\{S_f^2, \tau_f, S^2, \tau_s\}$. '`mf_ext2`' selects the extended model-free equations with parameters $\{S_f^2, \tau_f, S_s^2, \tau_s\}$.

Model-free parameters

The following parameters are accepted for the original model-free equation:

'`S2`' – The square of the generalised order parameter.

'`te`' – The effective correlation time.

Table 10.9: First table for the minimise() user function.

Minimisation algorithm	Patterns
Back-and-forth coordinate descent	'`^[Cc][Dd]$`' or '`^[Cc]oordinate[ _-][Dd]escent$`'
Steepest descent	'`^[Ss][Dd]$`' or '`^[Ss]teepest[ _-][Dd]escent$`'
Quasi-Newton BFGS	'`^[Bb][Ff][Gg][Ss]$`'
Newton	'`^[Nn]ewton$`'
Newton-CG	'`^[Nn]ewton[ _-][Cc][Gg]$`' or '`^[Nn][Cc][Gg]$`'

Table 10.10: Second table for the minimise() user function.

Minimisation algorithm	Patterns
Cauchy point	'`^[Cc]auchy`'
Dogleg	'`^[Dd]ogleg`'
CG-Steihaug	'`^[Cc][Gg][-_ ][Ss]teihaug`' or '`^[Ss]teihaug`'
Exact trust region	'`^[Ee]xact`'

Table 10.11: Third table for the minimise() user function.

Minimisation algorithm	Patterns
Fletcher-Reeves	'`^[Ff][Rr]$`' or '`^[Ff]letcher[-_ ][Rr]eeves$`'
Polak-Ribière	'`^[Pp][Rr]$`' or '`^[Pp]olak[-_ ][Rr]ibiere$`'
Polak-Ribière +	'`^[Pp][Rr]\+$`' or '`^[Pp]olak[-_ ][Rr]ibiere\+$`'
Hestenes-Stiefel	'`^[Hh][Ss]$`' or '`^[Hh]estenes[-_ ][Ss]tiefel$`'

Table 10.12: Fourth table for the minimise() user function.

Minimisation algorithm	Patterns
Simplex	'`^[Ss]implex$`'
Levenberg-Marquardt	'`^[Ll][Mm]$`' or '`^[Ll]evenburg-[Mm]arquardt$`'

Table 10.13: Fifth table for the minimise() user function.

Minimisation algorithm	Patterns
Method of Multipliers	'`^[Mm][Oo][Mm]$`' or '`[Mm]ethod of [Mm]ultipliers$`'

Table 10.14: Sixth table for the minimise() user function.

Minimisation algorithm	Patterns
Simulated Annealing	'^[Ss][Aa]$' or '^[Ss]imulated [Aa]nnealing$'

Table 10.15: Seventh table for the minimise() user function.

Line search algorithm	Patterns
Backtracking line search	'^[Bb]ack'
Nocedal and Wright interpolation based line search	'^[Nn][Ww][Ii]' or '^[Nn]ocedal[_][Ww]right[_][Ii]nt'
Nocedal and Wright line search for the Wolfe conditions	'^[Nn][Ww][Ww]' or '^[Nn]ocedal[_][Ww]right[_][Ww]olfe'
More and Thuente line search	'^[Mm][Tt]' or '^[Mm]ore[_][Tt]huente$'
No line search	'^[Nn]o [Ll]ine [Ss]earch$'

Table 10.16: Eighth table for the minimise() user function.

Hessian modification	Patterns
Unmodified Hessian	'^[Nn]o [Hh]essian [Mm]od'
Eigenvalue modification	'^[Ee]igen'
Cholesky with added multiple of the identity	'^[Cc]hol'
The Gill, Murray, and Wright modified Cholesky algorithm	'^[Gg][Mm][Ww]$'
The Schnabel and Eskow 1999 algorithm	'^[Ss][Ee]99'

Table 10.17: Ninth table for the minimise() user function.

Hessian type	Patterns
Quasi-Newton BFGS	'^[Bb][Ff][Gg][Ss]$'
Newton	'^[Nn]ewton$'

The following parameters are accepted for the extended model-free equation:

'S2f' – The square of the generalised order parameter of the faster motion.

'tf' – The effective correlation time of the faster motion.

'S2' – The square of the generalised order parameter $S^2 = S_f^2 * S_s^2$.

'ts' – The effective correlation time of the slower motion.

The following parameters are accepted for the extended 2 model-free equation:

'S2f' – The square of the generalised order parameter of the faster motion.

'tf' – The effective correlation time of the faster motion.

'S2s' – The square of the generalised order parameter of the slower motion.

'ts' – The effective correlation time of the slower motion.

The following parameters are accepted for all equations:

'Rex' – The chemical exchange relaxation.

'r' – The average bond length $<r>$.

'CSA' – The chemical shift anisotropy.

Spin identification string

If 'spin_id' is supplied then the model will only be created for the corresponding spins. Otherwise the model will be created for all spins.

Examples

The following commands will create the model-free model 'm1' which is based on the original model-free equation and contains the single parameter 'S2'.

```
relax> model_free.create_model('m1', 'mf_orig', ['S2'])
```

```
relax> model_free.create_model(model='m1', params=['S2'], equation='mf_orig')
```

The following commands will create the model-free model 'large_model' which is based on the extended model-free equation and contains the seven parameters 'S2f', 'tf', 'S2', 'ts', 'Rex', 'CSA', 'r'.

```
relax> model_free.create_model('large_model', 'mf_ext', ['S2f', 'tf', 'S2', 'ts', 'Rex', 'CSA', 'r'])
```

```
relax> model_free.create_model(model='large_model', params=['S2f', 'tf', 'S2', 'ts', 'Rex', 'CSA', 'r'], equation='mf_ext')
```

10.2.45 model_free.delete()

Synopsis

Function for deleting all model-free data from the current data pipe.

Defaults

model_free.delete(self)

Examples

To delete all model-free data, type:

```
relax> model_free.delete()
```

10.2.46 model_free.remove_tm()

Synopsis

Function for removing the local τ_m parameter from a model.

Defaults

model_free.remove_tm(self, spin_id=None)

Keyword Arguments

spin_id: The spin identification string.

Description

This function will remove the local τ_m parameter from the model-free parameter set. If there is no local τ_m parameter within the set nothing will happen.

If no spin identification string is given, then the function will apply to all spins.

Examples

The following command will remove the parameter 'tm':

```
relax> model_free.remove_tm()
```

10.2.47 model_free.select_model()

Synopsis

Function for the selection of a preset model-free model.

Defaults

model_free.select_model(self, model=None, spin_id=None)

Keyword Arguments

model: The name of the preset model.

The preset models

The standard preset model-free models are

'm0' = {},

'm1' = {S2},

'm2' = {S^2, τ_e},

'm3' = {S^2, R_{ex}},

'm4' = {S^2, τ_e, R_{ex}},

'm5' = {S_f^2, S^2, τ_s},

'm6' = {S_f^2, τ_f, S^2, τ_s},

'm7' = {S_f^2, S^2, τ_s, R_{ex}},

'm8' = {S_f^2, τ_f, S^2, τ_s, R_{ex}},

'm9' = {Rex}.

The preset model-free models with optimisation of the CSA value are

'm10' = {CSA},

'm11' = {CSA, S^2},

'm12' = {CSA, S^2, τ_e},

'm13' = {CSA, S^2, R_{ex}},

'm14' = {CSA, S^2, τ_e, R_{ex}},

'm15' = {CSA, S_f^2, S^2, τ_s},

'm16' = {CSA, S_f^2, τ_f, S^2, τ_s},

'm17' = {CSA, S_f^2, S^2, τ_s, R_{ex}},

'm18' = {CSA, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'm19' = {CSA, R_{ex}}.

The preset model-free models with optimisation of the bond length are

'm20' = {r},

'm21' = {r, S^2},

'm22' = {r, S^2, τ_e},

'm23' = {r, S^2, R_{ex}},

'm24' = {r, S^2, τ_e, R_{ex}},

'm25' = {r, S_f^2, S^2, τ_s},

'm26' = {r, S_f^2, τ_f, S^2, τ_s},

'm27' = {r, S_f^2, S^2, τ_s, R_{ex}},

'm28' = {r, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'm29' = {r, CSA, R_{ex}}.

The preset model-free models with both optimisation of the bond length and CSA are

'm30' = {r, CSA},

'm31' = {r, CSA, S^2},

'm32' = {r, CSA, S^2, τ_e},

'm33' = {r, CSA, S^2, R_{ex}},

'm34' = {r, CSA, S^2, τ_e, R_{ex}},

'm35' = {r, CSA, S_f^2, S^2, τ_s},

'm36' = {r, CSA, S_f^2, τ_f, S^2, τ_s},

'm37' = {r, CSA, S_f^2, S^2, τ_s, R_{ex}},

'm38' = {r, CSA, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'm39' = {r, CSA, R_{ex}}.

Warning: The models in the thirties range fail when using standard R_1, R_2, and NOE relaxation data. This is due to the extreme flexibly of these models where a change in the parameter '`r`' is compensated by a corresponding change in the parameter '`CSA`' and vice versa.

Additional preset model-free models, which are simply extensions of the above models with the addition of a local τ_m parameter are:

'`tm0`' = {tm},

'`tm1`' = {τ_m, S^2},

'`tm2`' = {τ_m, S^2, τ_e},

'`tm3`' = {τ_m, S^2, R_{ex}},

'`tm4`' = {τ_m, S^2, τ_e, R_{ex}},

'`tm5`' = {τ_m, S_f^2, S^2, τ_s},

'`tm6`' = {τ_m, S_f^2, τ_f, S^2, τ_s},

'`tm7`' = {τ_m, S_f^2, S^2, τ_s, R_{ex}},

'`tm8`' = {τ_m, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'`tm9`' = {τ_m, R_{ex}}.

The preset model-free models with optimisation of the CSA value are

'`tm10`' = {τ_m, CSA},

'`tm11`' = {τ_m, CSA, S^2},

'`tm12`' = {τ_m, CSA, S^2, τ_e},

'`tm13`' = {τ_m, CSA, S^2, R_{ex}},

'`tm14`' = {τ_m, CSA, S^2, τ_e, R_{ex}},

'`tm15`' = {τ_m, CSA, S_f^2, S^2, τ_s},

'`tm16`' = {τ_m, CSA, S_f^2, τ_f, S^2, τ_s},

'`tm17`' = {τ_m, CSA, S_f^2, S^2, τ_s, R_{ex}},

'`tm18`' = {τ_m, CSA, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'`tm19`' = {τ_m, CSA, R_{ex}}.

The preset model-free models with optimisation of the bond length are

'`tm20`' = {τ_m, r},

'`tm21`' = {τ_m, r, S^2},

'`tm22`' = {τ_m, r, S^2, τ_e},

'`tm23`' = {τ_m, r, S^2, R_{ex}},

'`tm24`' = {τ_m, r, S^2, τ_e, R_{ex}},

'`tm25`' = {τ_m, r, S_f^2, S^2, τ_s},

'`tm26`' = {τ_m, r, S_f^2, τ_f, S^2, τ_s},

'`tm27`' = {τ_m, r, S_f^2, S^2, τ_s, R_{ex}},

'`tm28`' = {τ_m, r, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'`tm29`' = {τ_m, r, CSA, R_{ex}}.

The preset model-free models with both optimisation of the bond length and CSA are

'`tm30`' = {τ_m, r, CSA},

'`tm31`' = {τ_m, r, CSA, S^2},

'`tm32`' = {τ_m, r, CSA, S^2, τ_e},

'`tm33`' = {τ_m, r, CSA, S^2, R_{ex}},

'`tm34`' = {τ_m, r, CSA, S^2, τ_e, R_{ex}},

'`tm35`' = {τ_m, r, CSA, S_f^2, S^2, τ_s},

'`tm36`' = {τ_m, r, CSA, S_f^2, τ_f, S^2, τ_s},

'`tm37`' = {τ_m, r, CSA, S_f^2, S^2, τ_s, R_{ex}},

'`tm38`' = {τ_m, r, CSA, S_f^2, τ_f, S^2, τ_s, R_{ex}},

'`tm39`' = {τ_m, r, CSA, R_{ex}}.

Spin identification string

If '`spin_id`' is supplied then the model will only be selected for the corresponding spins. Otherwise the model will be selected for all spins.

Examples

To pick model '`m1`' for all selected spins, type:

```
relax> model_free.select_model('m1')
relax> model_free.select_model(model='m1')
```

10.2.48 model_selection()

Synopsis

Function for model selection.

Defaults

model_selection(self, method=None, modsel_pipe=None, pipes=None)

Keyword arguments

method: The model selection technique (see below).

modsel_pipe: The name of the new data pipe which will be created by this user function by the copying of the selected data pipe.

pipes: An array containing the names of all data pipes to include in model selection.

Description

The following model selection methods are supported:

AIC – Akaike's Information Criteria.

AICc – Small sample size corrected AIC.

BIC – Bayesian or Schwarz Information Criteria.

Bootstrap – Bootstrap model selection.

CV – Single-item-out cross-validation.

Expect – The expected overall discrepancy (the true values of the parameters are required).

Farrow – Old model-free method by Farrow et al., 1994.

Palmer – Old model-free method by Mandel et al., 1995.

Overall – The realised overall discrepancy (the true values of the parameters are required).

For the methods '`Bootstrap`', '`Expect`', and '`Overall`', the function '`monte_carlo`' should have previously been executed with the type argument set to the appropriate value to modify its behaviour.

If the pipes argument is not supplied then all data pipes will be used for model selection.

Example

For model-free analysis, if the preset models 1 to 5 are minimised and loaded into the program, the following commands will carry out AIC model selection and to place the selected results into the '`mixed`' data pipe, type one of:

```
relax> model_selection('AIC', 'mixed')

relax> model_selection(method='AIC',
modsel_pipe='mixed')

relax> model_selection('AIC', 'mixed',
['m1', 'm2', 'm3', 'm4', 'm5'])

relax> model_selection(method='AIC',
modsel_pipe='mixed', pipes=['m1', 'm2',
'm3', 'm4', 'm5'])
```

10.2.49 molecule.copy()

Synopsis

Function for copying all data associated with a molecule.

Defaults

molecule.copy(self, pipe_from=None, mol_from=None, pipe_to=None, mol_to=None)

Keyword Arguments

pipe_from: The data pipe containing the molecule from which the data will be copied. This defaults to the current data pipe.

mol_from: The molecule identifier string of the molecule to copy the data from.

pipe_to: The data pipe to copy the data to. This defaults to the current data pipe.

mol_to: The molecule identifier string of the molecule to copy the data to.

Description

This function will copy all the data associated with a molecule to a second molecule. This includes residue and spin system information. The new molecule must not yet exist.

Examples

To copy the molecule data from the molecule '`GST`' to the new molecule '`wt-GST`', type:

```
relax> molecule.copy('#GST', '#wt-GST')

relax> molecule.copy(mol_from='#GST',
mol_to='#wt-GST')
```

To copy the molecule data of the molecule '`Ap4Aase`' from the data pipe '`m1`' to '`m2`', assuming the current data pipe is '`m1`', type:

```
relax> molecule.copy(mol_from='#ApAase',
pipe_to='m2')

relax> molecule.copy(pipe_from='m1',
mol_from='#ApAase', pipe_to='m2', mol_to=
'#ApAase')
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.50 molecule.create()

Synopsis

Function for creating a new molecule.

Defaults

molecule.create(self, mol_name=None, type=None)

Keyword Arguments

mol_name: The name of the molecule.

type: The type of molecule.

Description

This function will add a new molecule data container to the relax data storage object. The same molecule name cannot be used more than once. The molecule type need not be specified. However if it given, it should be one of 'protein', 'RNA', 'DNA', 'organic molecule', 'inorganic molecule'.

Examples

To create the molecules 'Ap4Aase', 'ATP', and 'MgF4', type:

```
relax> molecule.create('Ap4Aase')

relax> molecule.create('ATP')

relax> molecule.create('MgF4')
```

10.2.51 molecule.delete()

Synopsis

Function for deleting molecules.

Defaults

molecule.delete(self, mol_id=None)

Keyword Arguments

mol_id: The molecule identifier string.

Description

This function can be used to delete a single or sets of molecules.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.52 molecule.display()

Synopsis

Function for displaying the molecule information.

Defaults

molecule.display(self, mol_id=None)

Keyword Arguments

mol_id: The molecule identifier string.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.53 molecule.name()

Synopsis

Function for naming a molecule.

Defaults

molecule.name(self, mol_id=None, name=None, force=False)

Keyword Arguments

mol_id: The molecule identification string corresponding to one or more molecules.

name: The new molecule name.

force: A flag which if True will cause the molecule to be renamed.

Description

This function simply allows molecules to be named (or renamed).

Examples

To rename the molecule 'Ap4Aase' to 'Inhib Ap4Aase', type one of:

```
relax> molecule.name('#Ap4Aase', 'Inhib
Ap4Aase', True)
```

```
relax> molecule.name(mol_id='#Ap4Aase',
name='Inhib Ap4Aase', force=True)
```

This assumes the molecule 'Ap4Aase' already exists.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.54 molmol.clear_history()

Synopsis

Function for clearing the Molmol command history.

Defaults

molmol.clear_history(self)

10.2.55 molmol.command()

Synopsis

Function for executing a user supplied Molmol command.

Defaults

molmol.command(self, command=None)

Keyword Arguments

command: The Molmol command to execute.

Description

This user function allows you to pass Molmol commands to the program. This can be useful for automation or scripting.

Example

To reinitialise the Molmol instance:

```
relax> molmol.command("InitAll yes")
```

10.2.56 molmol.macro_exec()

Synopsis

Function for executing Molmol macros.

Defaults

molmol.macro_exec(self, data_type=None, style='classic', colour_start=None, colour_end=None, colour_list=None)

Keyword Arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start: The starting colour, either an array or string, of the linear colour gradient.

colour_end: The ending colour, either an array or string, of the linear colour gradient.

colour_list: The list of colours to match the start and end strings.

Description

This function allows residues specific values to be mapped to a structure through Molmol macros. Currently only the '`classic`' style, which is described below, is available.

Colour

The values are coloured based on a linear colour gradient which is specified through the '`colour_start`' and '`colour_end`' arguments. These arguments can either be a string to identify one of the RGB (red, green, blue) colour arrays listed in the tables below, or you can give the RGB vector itself. For example, colour_start='`white`' and colour_start=[1.0, 1.0, 1.0] both select the same colour. Leaving both arguments at None will select the default colour gradient which for each type of analysis is described below.

When supplying the colours as strings, two lists of colours can be selected from which to match the strings. These are the default Molmol colour list and the X11 colour list, both of which are described in the tables below. The default behaviour is to first search the Molmol list and then the X11 colour list, raising an error if neither contain the string. To explicitly select these lists, set the '`colour_list`' argument to either '`molmol`' or '`x11`'.

Examples

To map the order parameter values, S^2, onto the structure using the classic style, type:

```
relax> molmol.macro_exec('S2')

relax> molmol.macro_exec(data_type='S2')

relax> molmol.macro_exec(data_type='S2',
style="classic")
```

10.2.57 molmol.ribbon()

Synopsis

Apply the Molmol ribbon style.

Defaults

molmol.ribbon(self)

Description

This function applies the Molmol ribbon style which is equivalent to clicking on '`ribbon`' in the Molmol side menu. To do this, the following commands are executed:

CalcAtom '`H`'

CalcAtom '`HN`'

CalcSecondary

XMacStand ribbon.mac

Example

To apply the ribbon style to the PDB file loaded, type:

```
relax> molmol.ribbon()
```

10.2.58 molmol.tensor_pdb()

Synopsis

Function displaying the diffusion tensor PDB geometric object over the loaded PDB.

Defaults

molmol.tensor_pdb(self, file=None)

Keyword Arguments

file: The name of the PDB file containing the tensor geometric object.

Description

In executing this user function, a PDB file must have previously been loaded , a geometric object or polygon representing the Brownian rotational diffusion tensor will be overlain with the loaded PDB file and displayed within Molmol. The PDB file containing the geometric object must be created using the complementary '`pdb.create_diff_tensor_pdb()`' user function.

To display the diffusion tensor, the multiple commands will be executed. To overlay the structure with the diffusion tensor, everything will be selected and reoriented and moved to their original PDB frame positions:

SelectAtom ''

SelectBond ''

SelectAngle ''

SelectDist ''

SelectPrim ''

RotateInit

MoveInit

Next the tensor PDB file is read in, selected, and the covalent bonds of the PDB CONECT records calculated:

ReadPdb file

SelectMol '`@file`'

CalcBond 1 1 1

Then only the atoms and bonds of the geometric object are selected and the '`ball/stick`' style applied:

SelectAtom '`0`'

SelectBond '`0`'

SelectAtom '`:TNS`'

SelectBond '`:TNS`'

XMacStand ball_stick.mac

The appearance is finally touched up:

RadiusAtom 1

SelectAtom '`:TNS@C*`'

RadiusAtom 1.5

10.2.59 molmol.view()

Synopsis

Function for viewing the collection of molecules extracted from the PDB file.

Defaults

molmol.view(self)

Example

```
relax> molmol.view()
```

```
relax> molmol.view()
```

10.2.60 molmol.write()

Synopsis

Function for creating Molmol macros.

Defaults

molmol.write(self, data_type=None, style='classic', colour_start=None, colour_end=None, colour_list=None, file=None, dir='molmol', force=False)

Keyword Arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start: The starting colour, either an array or string, of the linear colour gradient.

colour_end: The ending colour, either an array or string, of the linear colour gradient.

colour_list: The list of colours to match the start and end strings.

file: The name of the file.

dir: The directory name.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

This function allows residues specific values to be mapped to a structure through the creation of a Molmol '`*.mac`' macro which can be executed in Molmol by clicking on '`File, Macro, Execute User...`'. Currently only the '`classic`' style, which is described below, is available.

Colour

The values are coloured based on a linear colour gradient which is specified through the '`colour_start`' and '`colour_end`' arguments. These arguments can either be a string to identify one of the RGB (red, green, blue) colour arrays listed in the tables below, or you can give the RGB vector itself. For example, colour_start='`white`' and colour_start=[1.0, 1.0, 1.0] both select the same colour. Leaving both arguments at None will select the default colour gradient which for each type of analysis is described below.

When supplying the colours as strings, two lists of colours can be selected from which to match the strings. These are the default Molmol colour list and the X11 colour list, both of which are described in the tables below. The default behaviour is to first search the Molmol list and then the X11 colour list, raising an error if neither contain the string. To explicitly select these lists, set the '`colour_list`' argument to either '`molmol`' or '`x11`'.

Examples

To create a Molmol macro mapping the order parameter values, S^2, onto the structure using the classic style, type:

```
relax> molmol.write('S2')
```

```
relax> molmol.write(data_type='S2')
```

```
relax> molmol.write(data_type='S2', style=
"classic", file='s2.mac', dir='molmol')
```

Model-free classic style

Creator: Edward d'Auvergne

Argument string: "classic"

Description: The classic style draws the backbone of a protein in the Molmol '`neon`' style. Rather than colouring the amino acids to which the NH bond belongs, the three covalent bonds of the peptide bond from Ca to Ca in which the NH bond is located are coloured. Deselected residues are shown as black lines.

Supported data types:

(see table 10.18)

Molmol RGB colour arrays

The following table is a list of colours used in Molmol and their corresponding RGB colour values ranging from 0 to 1.

(see table 10.19)

Table 10.18: First table for the molmol.write() user function.

Data type	String	Description
S^2.	'`S2`'	The standard model-free order parameter, equal to S_f^2.S2s for the two timescale models. The default colour gradient starts at '`yellow`' and ends at '`red`'.
S_f^2.	'`S2f`'	The order parameter of the faster of two internal motions. Residues which are described by model-free models m1 to m4, the single timescale models, are illustrated as white neon bonds. The default colour gradient is the same as that for the S^2 data type.
S_s^2.	'`S2s`'	The order parameter of the slower of two internal motions. This functions exactly as S_f^2 except that S_s^2 is plotted instead.
Amplitude of fast motions.	'`amp_fast`'	Model independent display of the amplite of fast motions. For residues described by model-free models m5 to m8, the value plotted is that of S_f^2. However, for residues described by models m1 to m4, what is shown is dependent on the timescale of the motions. This is because these single timescale models can, at times, be perfect approximations to the more complex two timescale models. Hence if τ_e is less than 200 ps, S^2 is plotted. Otherwise the peptide bond is coloured white. The default colour gradient is the same as that for S^2.
Amplitude of slow motions.	'`amp_slow`'	Model independent display of the amplite of slow motions, arbitrarily defined as motions slower than 200 ps. For residues described by model-free models m5 to m8, the order parameter S^2 is plotted if $\tau_s > 200$ ps. For models m1 to m4, S^2 is plotted if $\tau_e > 200$ ps. The default colour gradient is the same as that for S^2.
τ_e.	'`te`'	The correlation time, τ_e. The default colour gradient starts at '`turquoise`' and ends at '`blue`'.
τ_f.	'`tf`'	The correlation time, τ_f. The default colour gradient is the same as that of τ_e.
τ_s.	'`ts`'	The correlation time, τ_s. The default colour gradient starts at '`blue`' and ends at '`black`'.
Timescale of fast motions	'`time_fast`'	Model independent display of the timescale of fast motions. For models m5 to m8, only the parameter τ_f is plotted. For models m2 and m4, the parameter τ_e is plotted only if it is less than 200 ps. All other residues are assumed to have a correlation time of zero. The default colour gradient is the same as that of τ_e.
Timescale of slow motions	'`time_slow`'	Model independent display of the timescale of slow motions. For models m5 to m8, only the parameter τ_s is plotted. For models m2 and m4, the parameter τ_e is plotted only if it is greater than 200 ps. All other residues are coloured white. The default colour gradient is the same as that of τ_s.
Chemical exchange	'`Rex`'	The chemical exchange, R_{ex}. Residues which experience no chemical exchange are coloured white. The default colour gradient starts at '`yellow`' and finishes at '`red`'.

Table 10.19: Second table for the molmol.write() user function.

Name	Red	Green	Blue
'black'	0.000	0.000	0.000
'navy'	0.000	0.000	0.502
'blue'	0.000	0.000	1.000
'dark green'	0.000	0.392	0.000
'green'	0.000	1.000	0.000
'cyan'	0.000	1.000	1.000
'turquoise'	0.251	0.878	0.816
'royal blue'	0.255	0.412	0.882
'aquamarine'	0.498	1.000	0.831
'sky green'	0.529	0.808	0.922
'dark violet'	0.580	0.000	0.827
'pale green'	0.596	0.984	0.596
'purple'	0.627	0.125	0.941
'brown'	0.647	0.165	0.165
'light blue'	0.678	0.847	0.902
'grey'	0.745	0.745	0.745
'light grey'	0.827	0.827	0.827
'violet'	0.933	0.510	0.933
'light coral'	0.941	0.502	0.502
'khaki'	0.941	0.902	0.549
'beige'	0.961	0.961	0.863
'red'	1.000	0.000	0.000
'magenta'	1.000	0.000	1.000
'deep pink'	1.000	0.078	0.576
'orange red'	1.000	0.271	0.000
'hot pink'	1.000	0.412	0.706
'coral'	1.000	0.498	0.314
'dark orange'	1.000	0.549	0.000
'orange'	1.000	0.647	0.000
'pink'	1.000	0.753	0.796
'gold'	1.000	0.843	0.000
'yellow'	1.000	1.000	0.000
'light yellow'	1.000	1.000	0.878
'ivory'	1.000	1.000	0.941
'white'	1.000	1.000	1.000

X11 RGB colour arrays

The following table is the list of X11 colour names and their corresponding RGB colour values ranging from 0 to 255.

(see table 10.20)

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
snow	255	250	250
ghost white	248	248	255
white smoke	245	245	245
gainsboro	220	220	220
floral white	255	250	240
old lace	253	245	230
linen	250	240	230
antique white	250	235	215
papaya whip	255	239	213
blanched almond	255	235	205
bisque	255	228	196
peach puff	255	218	185
navajo white	255	222	173
moccasin	255	228	181
cornsilk	255	248	220
ivory	255	255	240
lemon chiffon	255	250	205
seashell	255	245	238
honeydew	240	255	240
mint cream	245	255	250
azure	240	255	255
alice blue	240	248	255
lavender	230	230	250
lavender blush	255	240	245
misty rose	255	228	225
white	255	255	255
black	0	0	0
dark slate grey	47	79	79
dim grey	105	105	105
slate grey	112	128	144
light slate grey	119	136	153
grey	190	190	190
light grey	211	211	211
midnight blue	25	25	112
navy	0	0	128
cornflower blue	100	149	237
dark slate blue	72	61	139
slate blue	106	90	205
medium slate blue	123	104	238
light slate blue	132	112	255
medium blue	0	0	205
royal blue	65	105	225
blue	0	0	255
dodger blue	30	144	255
deep sky blue	0	191	255
sky blue	135	206	235
light sky blue	135	206	250
steel blue	70	130	180
light steel blue	176	196	222
light blue	173	216	230
powder blue	176	224	230
pale turquoise	175	238	238
dark turquoise	0	206	209
medium turquoise	72	209	204
turquoise	64	224	208
cyan	0	255	255
light cyan	224	255	255
cadet blue	95	158	160
medium aquamarine	102	205	170
aquamarine	127	255	212
dark green	0	100	0
dark olive green	85	107	47
dark sea green	143	188	143
sea green	46	139	87
medium sea green	60	179	113
light sea green	32	178	170

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
pale green	152	251	152
spring green	0	255	127
lawn green	124	252	0
green	0	255	0
chartreuse	127	255	0
medium spring green	0	250	154
green yellow	173	255	47
lime green	50	205	50
yellow green	154	205	50
forest green	34	139	34
olive drab	107	142	35
dark khaki	189	183	107
khaki	240	230	140
pale goldenrod	238	232	170
light goldenrod yellow	250	250	210
light yellow	255	255	224
yellow	255	255	0
gold	255	215	0
light goldenrod	238	221	130
goldenrod	218	165	32
dark goldenrod	184	134	11
rosy brown	188	143	143
indian red	205	92	92
saddle brown	139	69	19
sienna	160	82	45
peru	205	133	63
burlywood	222	184	135
beige	245	245	220
wheat	245	222	179
sandy brown	244	164	96
tan	210	180	140
chocolate	210	105	30
firebrick	178	34	34
brown	165	42	42
dark salmon	233	150	122
salmon	250	128	114
light salmon	255	160	122
orange	255	165	0
dark orange	255	140	0
coral	255	127	80
light coral	240	128	128
tomato	255	99	71
orange red	255	69	0
red	255	0	0
hot pink	255	105	180
deep pink	255	20	147
pink	255	192	203
light pink	255	182	193
pale violet red	219	112	147
maroon	176	48	96
medium violet red	199	21	133
violet red	208	32	144
magenta	255	0	255
violet	238	130	238
plum	221	160	221
orchid	218	112	214
medium orchid	186	85	211
dark orchid	153	50	204
dark violet	148	0	211
blue violet	138	43	226
purple	160	32	240
medium purple	147	112	219
thistle	216	191	216
snow 1	255	250	250
snow 2	238	233	233
snow 3	205	201	201

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
snow 4	139	137	137
seashell 1	255	245	238
seashell 2	238	229	222
seashell 3	205	197	191
seashell 4	139	134	130
antique white 1	255	239	219
antique white 2	238	223	204
antique white 3	205	192	176
antique white 4	139	131	120
bisque 1	255	228	196
bisque 2	238	213	183
bisque 3	205	183	158
bisque 4	139	125	107
peach puff 1	255	218	185
peach puff 2	238	203	173
peach puff 3	205	175	149
peach puff 4	139	119	101
navajo white 1	255	222	173
navajo white 2	238	207	161
navajo white 3	205	179	139
navajo white 4	139	121	94
lemon chiffon 1	255	250	205
lemon chiffon 2	238	233	191
lemon chiffon 3	205	201	165
lemon chiffon 4	139	137	112
cornsilk 1	255	248	220
cornsilk 2	238	232	205
cornsilk 3	205	200	177
cornsilk 4	139	136	120
ivory 1	255	255	240
ivory 2	238	238	224
ivory 3	205	205	193
ivory 4	139	139	131
honeydew 1	240	255	240
honeydew 2	224	238	224
honeydew 3	193	205	193
honeydew 4	131	139	131
lavender blush 1	255	240	245
lavender blush 2	238	224	229
lavender blush 3	205	193	197
lavender blush 4	139	131	134
misty rose 1	255	228	225
misty rose 2	238	213	210
misty rose 3	205	183	181
misty rose 4	139	125	123
azure 1	240	255	255
azure 2	224	238	238
azure 3	193	205	205
azure 4	131	139	139
slate blue 1	131	111	255
slate blue 2	122	103	238
slate blue 3	105	89	205
slate blue 4	71	60	139
royal blue 1	72	118	255
royal blue 2	67	110	238
royal blue 3	58	95	205
royal blue 4	39	64	139
blue 1	0	0	255
blue 2	0	0	238
blue 3	0	0	205
blue 4	0	0	139
dodger blue 1	30	144	255
dodger blue 2	28	134	238
dodger blue 3	24	116	205
dodger blue 4	16	78	139
steel blue 1	99	184	255

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
steel blue 2	92	172	238
steel blue 3	79	148	205
steel blue 4	54	100	139
deep sky blue 1	0	191	255
deep sky blue 2	0	178	238
deep sky blue 3	0	154	205
deep sky blue 4	0	104	139
sky blue 1	135	206	255
sky blue 2	126	192	238
sky blue 3	108	166	205
sky blue 4	74	112	139
light sky blue 1	176	226	255
light sky blue 2	164	211	238
light sky blue 3	141	182	205
light sky blue 4	96	123	139
slate grey 1	198	226	255
slate grey 2	185	211	238
slate grey 3	159	182	205
slate grey 4	108	123	139
light steel blue 1	202	225	255
light steel blue 2	188	210	238
light steel blue 3	162	181	205
light steel blue 4	110	123	139
light blue 1	191	239	255
light blue 2	178	223	238
light blue 3	154	192	205
light blue 4	104	131	139
light cyan 1	224	255	255
light cyan 2	209	238	238
light cyan 3	180	205	205
light cyan 4	122	139	139
pale turquoise 1	187	255	255
pale turquoise 2	174	238	238
pale turquoise 3	150	205	205
pale turquoise 4	102	139	139
cadet blue 1	152	245	255
cadet blue 2	142	229	238
cadet blue 3	122	197	205
cadet blue 4	83	134	139
turquoise 1	0	245	255
turquoise 2	0	229	238
turquoise 3	0	197	205
turquoise 4	0	134	139
cyan 1	0	255	255
cyan 2	0	238	238
cyan 3	0	205	205
cyan 4	0	139	139
dark slate grey 1	151	255	255
dark slate grey 2	141	238	238
dark slate grey 3	121	205	205
dark slate grey 4	82	139	139
aquamarine 1	127	255	212
aquamarine 2	118	238	198
aquamarine 3	102	205	170
aquamarine 4	69	139	116
dark sea green 1	193	255	193
dark sea green 2	180	238	180
dark sea green 3	155	205	155
dark sea green 4	105	139	105
sea green 1	84	255	159
sea green 2	78	238	148
sea green 3	67	205	128
sea green 4	46	139	87
pale green 1	154	255	154
pale green 2	144	238	144
pale green 3	124	205	124

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
pale green 4	84	139	84
spring green 1	0	255	127
spring green 2	0	238	118
spring green 3	0	205	102
spring green 4	0	139	69
green 1	0	255	0
green 2	0	238	0
green 3	0	205	0
green 4	0	139	0
chartreuse 1	127	255	0
chartreuse 2	118	238	0
chartreuse 3	102	205	0
chartreuse 4	69	139	0
olive drab 1	192	255	62
olive drab 2	179	238	58
olive drab 3	154	205	50
olive drab 4	105	139	34
dark olive green 1	202	255	112
dark olive green 2	188	238	104
dark olive green 3	162	205	90
dark olive green 4	110	139	61
khaki 1	255	246	143
khaki 2	238	230	133
khaki 3	205	198	115
khaki 4	139	134	78
light goldenrod 1	255	236	139
light goldenrod 2	238	220	130
light goldenrod 3	205	190	112
light goldenrod 4	139	129	76
light yellow 1	255	255	224
light yellow 2	238	238	209
light yellow 3	205	205	180
light yellow 4	139	139	122
yellow 1	255	255	0
yellow 2	238	238	0
yellow 3	205	205	0
yellow 4	139	139	0
gold 1	255	215	0
gold 2	238	201	0
gold 3	205	173	0
gold 4	139	117	0
goldenrod 1	255	193	37
goldenrod 2	238	180	34
goldenrod 3	205	155	29
goldenrod 4	139	105	20
dark goldenrod 1	255	185	15
dark goldenrod 2	238	173	14
dark goldenrod 3	205	149	12
dark goldenrod 4	139	101	8
rosy brown 1	255	193	193
rosy brown 2	238	180	180
rosy brown 3	205	155	155
rosy brown 4	139	105	105
indian red 1	255	106	106
indian red 2	238	99	99
indian red 3	205	85	85
indian red 4	139	58	58
sienna 1	255	130	71
sienna 2	238	121	66
sienna 3	205	104	57
sienna 4	139	71	38
burlywood 1	255	211	155
burlywood 2	238	197	145
burlywood 3	205	170	125
burlywood 4	139	115	85
wheat 1	255	231	186

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
wheat 2	238	216	174
wheat 3	205	186	150
wheat 4	139	126	102
tan 1	255	165	79
tan 2	238	154	73
tan 3	205	133	63
tan 4	139	90	43
chocolate 1	255	127	36
chocolate 2	238	118	33
chocolate 3	205	102	29
chocolate 4	139	69	19
firebrick 1	255	48	48
firebrick 2	238	44	44
firebrick 3	205	38	38
firebrick 4	139	26	26
brown 1	255	64	64
brown 2	238	59	59
brown 3	205	51	51
brown 4	139	35	35
salmon 1	255	140	105
salmon 2	238	130	98
salmon 3	205	112	84
salmon 4	139	76	57
light salmon 1	255	160	122
light salmon 2	238	149	114
light salmon 3	205	129	98
light salmon 4	139	87	66
orange 1	255	165	0
orange 2	238	154	0
orange 3	205	133	0
orange 4	139	90	0
dark orange 1	255	127	0
dark orange 2	238	118	0
dark orange 3	205	102	0
dark orange 4	139	69	0
coral 1	255	114	86
coral 2	238	106	80
coral 3	205	91	69
coral 4	139	62	47
tomato 1	255	99	71
tomato 2	238	92	66
tomato 3	205	79	57
tomato 4	139	54	38
orange red 1	255	69	0
orange red 2	238	64	0
orange red 3	205	55	0
orange red 4	139	37	0
red 1	255	0	0
red 2	238	0	0
red 3	205	0	0
red 4	139	0	0
deep pink 1	255	20	147
deep pink 2	238	18	137
deep pink 3	205	16	118
deep pink 4	139	10	80
hot pink 1	255	110	180
hot pink 2	238	106	167
hot pink 3	205	96	144
hot pink 4	139	58	98
pink 1	255	181	197
pink 2	238	169	184
pink 3	205	145	158
pink 4	139	99	108
light pink 1	255	174	185
light pink 2	238	162	173
light pink 3	205	140	149

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
light pink 4	139	95	101
pale violet red 1	255	130	171
pale violet red 2	238	121	159
pale violet red 3	205	104	137
pale violet red 4	139	71	93
maroon 1	255	52	179
maroon 2	238	48	167
maroon 3	205	41	144
maroon 4	139	28	98
violet red 1	255	62	150
violet red 2	238	58	140
violet red 3	205	50	120
violet red 4	139	34	82
magenta 1	255	0	255
magenta 2	238	0	238
magenta 3	205	0	205
magenta 4	139	0	139
orchid 1	255	131	250
orchid 2	238	122	233
orchid 3	205	105	201
orchid 4	139	71	137
plum 1	255	187	255
plum 2	238	174	238
plum 3	205	150	205
plum 4	139	102	139
medium orchid 1	224	102	255
medium orchid 2	209	95	238
medium orchid 3	180	82	205
medium orchid 4	122	55	139
dark orchid 1	191	62	255
dark orchid 2	178	58	238
dark orchid 3	154	50	205
dark orchid 4	104	34	139
purple 1	155	48	255
purple 2	145	44	238
purple 3	125	38	205
purple 4	85	26	139
medium purple 1	171	130	255
medium purple 2	159	121	238
medium purple 3	137	104	205
medium purple 4	93	71	139
thistle 1	255	225	255
thistle 2	238	210	238
thistle 3	205	181	205
thistle 4	139	123	139
grey 0	0	0	0
grey 1	3	3	3
grey 2	5	5	5
grey 3	8	8	8
grey 4	10	10	10
grey 5	13	13	13
grey 6	15	15	15
grey 7	18	18	18
grey 8	20	20	20
grey 9	23	23	23
grey 10	26	26	26
grey 11	28	28	28
grey 12	31	31	31
grey 13	33	33	33
grey 14	36	36	36
grey 15	38	38	38
grey 16	41	41	41
grey 17	43	43	43
grey 18	46	46	46
grey 19	48	48	48
grey 20	51	51	51

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
grey 21	54	54	54
grey 22	56	56	56
grey 23	59	59	59
grey 24	61	61	61
grey 25	64	64	64
grey 26	66	66	66
grey 27	69	69	69
grey 28	71	71	71
grey 29	74	74	74
grey 30	77	77	77
grey 31	79	79	79
grey 32	82	82	82
grey 33	84	84	84
grey 34	87	87	87
grey 35	89	89	89
grey 36	92	92	92
grey 37	94	94	94
grey 38	97	97	97
grey 39	99	99	99
grey 40	102	102	102
grey 41	105	105	105
grey 42	107	107	107
grey 43	110	110	110
grey 44	112	112	112
grey 45	115	115	115
grey 46	117	117	117
grey 47	120	120	120
grey 48	122	122	122
grey 49	125	125	125
grey 50	127	127	127
grey 51	130	130	130
grey 52	133	133	133
grey 53	135	135	135
grey 54	138	138	138
grey 55	140	140	140
grey 56	143	143	143
grey 57	145	145	145
grey 58	148	148	148
grey 59	150	150	150
grey 60	153	153	153
grey 61	156	156	156
grey 62	158	158	158
grey 63	161	161	161
grey 64	163	163	163
grey 65	166	166	166
grey 66	168	168	168
grey 67	171	171	171
grey 68	173	173	173
grey 69	176	176	176
grey 70	179	179	179
grey 71	181	181	181
grey 72	184	184	184
grey 73	186	186	186
grey 74	189	189	189
grey 75	191	191	191
grey 76	194	194	194
grey 77	196	196	196
grey 78	199	199	199
grey 79	201	201	201
grey 80	204	204	204
grey 81	207	207	207
grey 82	209	209	209
grey 83	212	212	212
grey 84	214	214	214
grey 85	217	217	217
grey 86	219	219	219

Table 10.20: Third table for the molmol.write() user function.

Name	Red	Green	Blue
grey 87	222	222	222
grey 88	224	224	224
grey 89	227	227	227
grey 90	229	229	229
grey 91	232	232	232
grey 92	235	235	235
grey 93	237	237	237
grey 94	240	240	240
grey 95	242	242	242
grey 96	245	245	245
grey 97	247	247	247
grey 98	250	250	250
grey 99	252	252	252
grey 100	255	255	255
dark grey	169	169	169
dark blue	0	0	139
dark cyan	0	139	139
dark magenta	139	0	139
dark red	139	0	0
light green	144	238	144

10.2.61 monte_carlo.create_data()

Synopsis

Function for creating simulation data.

Defaults

monte_carlo.create_data(self, method='back_calc')

Keyword Arguments

method: The simulation method.

Description

The method argument can either be set to '`back_calc`' or '`direct`', the choice of which determines the simulation type. If the values or parameters are calculated rather than minimised, this option will have no effect, hence, '`back_calc`' and '`direct`' are identical.

For error analysis, the method argument should be set to '`back_calc`' which will result in proper Monte Carlo simulations. The data used for each simulation is back calculated from the minimised model parameters and is randomised using Gaussian noise where the standard deviation is from the original error set. When the method is set to '`back_calc`', this function should only be called after the model is fully minimised.

The simulation type can be changed by setting the method argument to '`direct`'. This will result in simulations which cannot be used in error analysis and which are no longer Monte Carlo simulations. However, these simulations are required for certain model selection techniques (see the documentation for the model selection function for details), and can be used for other purposes. Rather than the data being back calculated from the fitted model parameters, the data is generated by taking the original data and randomising using Gaussian noise with the standard deviations set to the original error set.

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.

relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.
```

```
relax> monte_carlo.error_analysis() # Step
8.
```

10.2.62 monte_carlo.error_analysis()

Synopsis

Function for calculating parameter errors from the Monte Carlo simulations.

Defaults

monte_carlo.error_analysis(self, prune=0.0)

Keyword Arguments

prune: Legacy argument corresponding to '`trim`' in Art Palmer's Modelfree program.

Description

Parameter errors are calculated as the standard deviation of the distribution of parameter values. This function should never be used if parameter values are obtained by minimisation and the simulation data are generated using the method '`direct`'. The reason is because only true Monte Carlo simulations can give the true parameter errors.

The prune argument is legacy code which corresponds to the '`trim`' option in Art Palmer's Modelfree program. To remove failed simulations, the eliminate function should be used prior to this function. Eliminating the simulations specifically identifies and removes the failed simulations whereas the prune argument will only, in a few cases, positively identify failed simulations but only if severe parameter limits have been imposed. Most failed models will pass through the pruning process and hence cause a catastrophic increase in the parameter errors. If the argument must be used, the following must be taken into account. If the values or parameters are calculated rather than minimised, the prune argument must be set to zero. The value of this argument is proportional to the number of simulations removed prior to error calculation. If prune is set to 0.0, all simulations are used for calculating errors, whereas a value of 1.0 excludes all data. In almost all cases prune must be set to zero, any value greater than zero will result in an underestimation of the error values. If a value is supplied, the lower and upper tails of the distribution of chi-squared values will be excluded from the error calculation.

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.
```

```
relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.

relax> monte_carlo.error_analysis() # Step
8.
```

10.2.63 monte_carlo.initial_values()

Synopsis

Function for setting the initial simulation parameter values.

Defaults

monte_carlo.initial_values(self)

Description

This function only effects where minimisation occurs and can therefore be skipped if the values or parameters are calculated rather than minimised. However, if accidentally run in this case, the results will be unaffected. It should only be called after the model or run is fully minimised. Once called, the functions '`grid_search`' and '`minimise`' will only effect the simulations and not the model parameters.

The initial values of the parameters for each simulation is set to the minimised parameters of the model. A grid search can be undertaken for each simulation instead, although this is computationally expensive and unnecessary. The minimisation function should be executed for a second time after running this function.

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.

relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.
```

```
relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.

relax> monte_carlo.error_analysis() # Step
8.
```

10.2.64 monte_carlo.off()

Synopsis

Function for turning simulations off.

Defaults

monte_carlo.off(self)

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.

relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.

relax> monte_carlo.error_analysis() # Step
8.
```

10.2.65 monte_carlo.on()

Synopsis

Function for turning simulations on.

Defaults

monte_carlo.on(self)

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.

relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.

relax> monte_carlo.error_analysis() # Step
8.
```

10.2.66 monte_carlo.setup()

Synopsis

Function for setting up Monte Carlo simulations.

Defaults

monte_carlo.setup(self, number=500)

Keyword Arguments

number: The number of Monte Carlo simulations.

Description

This function must be called prior to any of the other Monte Carlo functions. The effect is that the number of simulations will be set and that simulations will be turned on.

Monte Carlo Simulation Overview

For proper error analysis using Monte Carlo simulations, a sequence of function calls is required for running the various simulation components. The steps necessary for implementing Monte Carlo simulations are:

1. The measured data set together with the corresponding error set should be loaded into relax.

2. Either minimisation is used to optimise the parameters of the chosen model, or a calculation is run.

3. To initialise and turn on Monte Carlo simulations, the number of simulations, n, needs to be set.

4. The simulation data needs to be created either by back calculation from the fully minimised model parameters from step 2 or by direct calculation when values are calculated rather than minimised. The error set is used to randomise each simulation data set by assuming Gaussian errors. This creates a synthetic data set for each Monte Carlo simulation.

5. Prior to minimisation of the parameters of each simulation, initial parameter estimates are required. These are taken as the optimised model parameters. An alternative is to use a grid search for each simulation to generate initial estimates, however this is extremely computationally expensive. For the case where values are calculated rather than minimised, this step should be skipped (although the results will be unaffected if this is accidentally run).

6. Each simulation requires minimisation or calculation. The same techniques as used in step 2, excluding the grid search when minimising, should be used for the simulations.

7. Failed simulations are removed using the techniques of model elimination.

8. The model parameter errors are calculated from the distribution of simulation parameters.

Monte Carlo simulations can be turned on or off using functions within this class. Once the function for setting up simulations has been called, simulations will be turned on. The effect of having simulations turned on is that the functions used for minimisation (grid search, minimise, etc) or calculation will only affect the simulation parameters and not the model parameters. By subsequently turning simulations off using the appropriate function, the functions used in minimisation will affect the model parameters and not the simulation parameters.

An example, for model-free analysis, which includes only the functions required for implementing the above steps is:

```
relax> grid_search(inc=11) # Step 2.

relax> minimise('newton') # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> monte_carlo.initial_values() # Step
5.

relax> minimise('newton') # Step 6.

relax> eliminate() # Step 7.

relax> monte_carlo.error_analysis() # Step
8.
```

An example for reduced spectral density mapping is:

```
relax> calc() # Step 2.

relax> monte_carlo.setup(number=500) # Step
3.

relax> monte_carlo.create_data(method=
'back_calc') # Step 4.

relax> calc() # Step 6.

relax> monte_carlo.error_analysis() # Step
8.
```

10.2.67 n_state_model.CoM()

Synopsis

Centre of mass (CoM) analysis.

Defaults

n_state_model.CoM(self, pivot_point=[0.0, 0.0, 0.0], centre=None)

Keyword Arguments

pivot_point: The pivot point of the motions between the two domains.

centre: The optional argument for manually specifying the CoM of the initial position prior to the N rotations to the positions of the N states.

Description

This function is used for analysing the domain motion information content of the N states from the N-state model. The states do not correspond to physical states, hence nothing can be extracted from the individual states. This analysis involves the calculation of the pivot to centre of mass (pivot-CoM) order parameter and subsequent cone of motions.

For the analysis, both the pivot point and centre of mass must be specified. The supplied pivot point must be a vector of floating point numbers of length 3. If the centre keyword argument is supplied, it must also be a vector of floating point numbers (of length 3). If the centre argument is not supplied, then the CoM will be calulcated from the selected parts of a previously loaded structure.

Examples

To perform an analysis where the pivot is at the origin and the CoM is set to the N-terminal domain of a previously loaded PDB file (the C-terminal domain has been deselected), type:

```
relax> n_state_model.CoM()
```

To perform an analysis where the pivot is at the origin (because the real pivot has been shifted to this position) and the CoM is at the position [0, 0, 1], type one of:

```
relax> n_state_model.CoM(centre=[0, 0, 1])
```

```
relax> n_state_model.CoM(centre=[0.0, 0.0,
1.0])

relax> n_state_model.CoM(pivot_point=[0.0,
0.0, 0.0], centre=[0.0, 0.0, 1.0])
```

10.2.68 n_state_model.cone_pdb()

Synopsis

Create a PDB file representing the cone models from the centre of mass (CoM) analysis.

Defaults

n_state_model.cone_pdb(self, cone_type=None, scale=1.0, file='cone.pdb', dir=None, force=False)

Keyword Arguments

cone_type: The type of cone model to represent.

scale: Value for scaling the pivot-CoM distance which the size of the cone defaults to.

file: The name of the PDB file.

dir: The directory where the file is located.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

This function creates a PDB file containing an artificial geometric structure to represent the various cone models. These models include:

'diff in cone'

'diff on cone'

The model can be selected by setting the cone_type argument to one of these strings. The cone is represented as an isotropic cone with its axis parallel to the average pivot-CoM vector, the vertex placed at the pivot point of the domain motions, and the length of the edge of the cone equal to the pivot-CoM distance multipled by the scaling argument. The resultant PDB file can subsequently read into any molecular viewer.

There are four different types of residue within the PDB. The pivot point is represented as as a single carbon atom of the residue 'PIV'. The cone consists of numerous H atoms of the residue 'CON'. The average pivot-CoM vector is presented as the residue 'AVE' with one carbon atom positioned at the pivot and the other at the head of the vector (after scaling by the scale argument). Finally, if Monte Carlo have been performed, there will be multiple 'MCC' residues representing the cone for each simulation, and multiple 'MCA' residues representing the varying average pivot-CoM vector for each simulation.

To create the diffusion in a cone PDB representation, a uniform distribution of vectors on a sphere is generated using spherical coordinates with the polar angle defined from the average pivot-CoM vector. By incrementing the polar angle using an arccos distribution, a radial array of vectors representing latitude are created while incrementing the azimuthal angle evenly creates the longitudinal vectors. These are all placed into the PDB file as H atoms and are all connected using PDB CONECT records. Each H atom is connected to its two neighbours on the both the longitude and latitude. This creates a geometric PDB object with longitudinal and latitudinal lines representing the filled cone.

10.2.69 n_state_model.elim_no_prob()

Synopsis

Eliminate the structures or states with no probability.

Defaults

n_state_model.elim_no_prob(self)

Examples

Simply type:

```
relax> n_state_model.elim_no_prob(N=8)
```

10.2.70 n_state_model.number_of_states()

Synopsis

Set the number of states in the N-state model.

Defaults

n_state_model.number_of_states(self, N=None)

Keyword Arguments

N: The number of states.

Description

Prior to optimisation, the number of states in the N-state model can be specified. If the number of states is not set, then this parameter will be equal to the number of loaded structures.

Examples

To set up an 8-state model, type:

```
relax> n_state_model.number_of_states(N=8)
```

10.2.71 n_state_model.ref_domain()

Synopsis

Set the reference domain for the '2-domain' N-state model.

Defaults

n_state_model.ref_domain(self, ref=None)

Keyword Arguments

ref: The domain which will act as the frame of reference. This is only valid for the '2-domain' N-state model.

Description

Prior to optimisation of the '2-domain' N-state model, which of the two domains will act as the frame of reference must be specified. The N-states will be rotations of the other domain, so to switch the frame of reference to the other domain simply transpose the rotation matrices.

Examples

To set up a 5-state model with 'C' domain being the frame of reference, type:

```
relax> n_state_model.ref_domain(ref='C')
```

10.2.72 n_state_model.select_model()

Synopsis

Select the N-state model type and set up the model.

Defaults

n_state_model.select_model(self, model=None)

Keyword Arguments

model: The name of the preset N-state model.

Description

Prior to optimisation, the N-state model type should be selected. The preset models are:

'2-domain' - The N-state model for a system of two domains, where one domain experiences a a reduced tensor.

'population' - The N-state model whereby only populations are optimised. The structures loaded into relax are assumed to be fixed. I.e. if two domains are present, the Euler angles for each state are fixed. The parameters of the model include the weight or probability for each state and the alignment tensor - {p0, p1, ..., pN, Axx, Ayy, Axy, Axz, Ayz}.

'fixed' - The N-state model whereby all motions are fixed and all populations are fixed to the set probabilities. The parameters of the model are simply the alignment tensor, {Axx, Ayy, Axy, Axz, Ayz}.

Examples

To analyse populations of states, type:

```
relax> n_state_model.select_model(model=
'populations')
```

10.2.73 noe.read_restraints()

Synopsis

Read NOESY or ROESY restraints from a file.

Defaults

noe.read_restraints(self, file=None, dir=None, proton1_col=None, proton2_col=None, lower_col=None, upper_col=None, sep=None)

Keyword Arguments

file: The name of the file containing the relaxation data.

dir: The directory where the file is located.

proton1_col: The column containing the first proton of the NOE or ROE cross peak.

proton2_col: The column containing the second proton of the NOE or ROE cross peak.

lower_col: The column containing the lower NOE bound.

upper_col: The column containing the upper NOE bound.

sep: The column separator (the default is white space).

Description

This function can automatically determine the format of the file, for example Xplor formatted restraint files. A generically formatted file is also supported if it contains minimally four columns with the two proton names and the upper and lower bounds, as specified by the *_col arguments. The proton names need to be in the spin identification string format.

Examples

To read the Xplor formatted restraint file '`NOE.xpl`', type one of:

```
relax> noe.read_restraints('NOE.xpl')
```

```
relax> noe.read_restraints(file='NOE.xpl')
```

To read the generic formatted file '`noes`', type one of:

```
relax> noe.read_restraints(file='NOE.xpl',
proton1_col=0, proton2_col=1, lower_col=2,
upper_col=3)
```

10.2.74 noe.spectrum_type()

Synopsis

Set the steady-state NOE spectrum type for pre-loaded peak intensities.

Defaults

noe.spectrum_type(self, spectrum_type=None, spectrum_id=None)

Keyword Arguments

spectrum_type: The type of steady-state NOE spectrum, one of '`ref`' or '`sat`'.

spectrum_id: The spectrum identification string.

Description

The spectrum_type argument can have the following values:

'`ref`' – The steady-state NOE reference spectrum.

'`sat`' – The steady-state NOE spectrum with proton saturation turned on.

Peak intensities should be loaded before calling this user function via the '`spectrum.read_intensities()`' user function. The intensity values will then be associated with a spectrum identifier which can be used here.

10.2.75 palmer.create()

Synopsis

Function for creating the Modelfree4 input files.

Defaults

palmer.create(self, dir=None, force=False, binary='modelfree4', diff_search='none', sims=0, sim_type='pred', trim=0, steps=20, constraints=True, heteronuc_type='15N', atom1='N', atom2='H', spin_id=None)

Keyword Arguments

dir: The directory to place the files.

force: A flag which if set to True will cause the results file to be overwritten if it already exists.

binary: The name of the executable Modelfree program file.

diff_search: See the Modelfree4 manual for '`diffusion_search`'.

sims: The number of Monte Carlo simulations.

sim_type: See the Modelfree4 manual.

trim: See the Modelfree4 manual.

steps: See the Modelfree4 manual.

constraints: A flag specifying whether the parameters should be constrained. The default is to turn constraints on (constraints=True).

heteronuc_type: A three letter string describing the heteronucleus type, ie 15N, 13C, etc.

atom1: The symbol of the X heteronucleus in the pdb file.

atom2: The symbol of the H nucleus in the pdb file.

spin_id: The spin identification string.

Description

The following files are created

'`dir/mfin`',

'`dir/mfdata`',

'`dir/mfpar`',

'`dir/mfmodel`',

'`dir/run.sh`'.

The file '`dir/run.sh`' contains the single command,

'`modelfree4 -i mfin -d mfdata -p mfpar -m mfmodel -o mfout -e out`',

which can be used to execute modelfree4.

If you would like to use a different Modelfree executable file, change the keyword argument '`binary`' to the appropriate file name. If the file is not located within the environment's path, include the full path infront of the binary file name.

10.2.76 palmer.execute()

Synopsis

Function for executing Modelfree4.

Defaults

palmer.execute(self, dir=None, force=False, binary='modelfree4')

Keyword Arguments

dir: The directory to place the files.

force: A flag which if set to True will cause the results file to be overwritten if it already exists.

binary: The name of the executable Modelfree program file.

Description

Modelfree 4 will be executed as

```
$ modelfree4 -i mfin -d mfdata -p mfpar -m
mfmodel -o mfout -e out
```

If a PDB file is loaded and non-isotropic diffusion is selected, then the file name will be placed on the command line as '`-s pdb_file_name`'.

If you would like to use a different Modelfree executable file, change the keyword argument '`binary`' to the appropriate file name. If the file is not located within the environment's path, include the full path in front of the binary file name.

10.2.77 palmer.extract()

Synopsis

Function for extracting data from the Modelfree4 '`mfout`' star formatted file.

Defaults

palmer.extract(self, dir=None)

Keyword Arguments

dir: The directory where the file '`mfout`' is found.

10.2.78 paramag.centre()

Synopsis

Specify which atom is the paramagnetic centre.

Defaults

paramag.centre(self, pos=None, atom_id=None, pipe=None, verbosity=1, fix=True, ave_pos=True, force=False)

Keyword Arguments

pos: The atomic position.

atom_id: The atom ID string.

pipe: The data pipe containing the structures to extract the centre from.

verbosity: The amount of information to print out.

fix: A flag specifying if the paramagnetic centre should be fixed during optimisation.

ave_pos: A flag specifying if the position of the atom is to be averaged across all models.

force: A flag which if True will cause the current paramagnetic centre to be overwritten.

Description

This function is required for specifying where the paramagnetic centre is located in the loaded structure file. If no structure number is given, then the average atom position will be calculated if multiple structures are loaded.

A different set of structures than those loaded into the current data pipe can also be used to determine the position, or its average. This can be achieved by loading the alternative structures into another data pipe, and then specifying that pipe through the pipe argument.

If the ave_pos flag is set to True, the average position from all models will be used as the position of the paramagnetic centre. If False, then the positions from all structures will be used. If multiple positions are used, then a fast paramagnetic centre motion will be assumed so that PCSs for a single tensor will be calculated for each position, and the PCS values linearly averaged.

Examples

If the paramagnetic centre is the lanthanide Dysprosium which is labelled as $\mathfrak{D}_y$ in a loaded PDB file, then type one of:

```
relax> paramag.centre('Dy')
```

```
relax> paramag.centre(atom_id='Dy')
```

If the carbon atom '`C1`' of residue '`4`' in the PDB file is to be used as the paramagnetic centre, then type:

```
relax> paramag.centre(':4@C1')
```

To state that the Dy3+ atomic position is [0.136, 12.543, 4.356], type one of:

```
relax> paramag.centre([0.136, 12.543,
4.356])
```

```
relax> paramag.centre(pos=[0.136, 12.543,
4.356])
```

To find an unknown paramagnetic centre, type:

```
relax> paramag.centre(fix=False)
```

10.2.79 pcs.back_calc()

Synopsis

Back calculate the pseudo-contact shifts.

Defaults

pcs.back_calc(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

10.2.80 pcs.calc_q_factors()

Synopsis

Calculate the PCS Q-factor for the selected spins.

Defaults

pcs.calc_q_factors(self, spin_id=None)

Keyword Arguments

spin_id: The spin ID string for restricting to subset of all selected spins.

Description

For this function to work, the back-calculated PCS data must first be generated by the analysis specific code. Otherwise a warning will be given.

Examples

To calculate the PCS Q-factor for only the spins '`@H26`', '`@H27`', and '`@H28`', type one of:

```
relax> pcs.calc_q_factors('@H26 & @H27 &
@H28')
```

```
relax> pcs.calc_q_factors(spin_id='@H26 &
@H27 & @H28')
```

10.2.81 pcs.copy()

Synopsis

Copy PCS data from pipe_from to pipe_to.

Defaults

pcs.copy(self, pipe_from=None, pipe_to=None, align_id=None)

Keyword Arguments

pipe_from: The name of the pipe to copy the PCS data from.

pipe_to: The name of the pipe to copy the PCS data to.

align_id: The alignment ID string.

Description

This function will copy PCS data from '`pipe_from`' to '`pipe_to`'. If align_id is not given then all PCS data will be copied, otherwise only a specific data set will be.

Examples

To copy all PCS data from pipe '`m1`' to pipe '`m9`', type one of:

```
relax> pcs.copy('m1', 'm9')
```

```
relax> pcs.copy(pipe_from='m1', pipe_to='m9')
```

```
relax> pcs.copy('m1', 'm9', None)
```

```
relax> pcs.copy(pipe_from='m1', pipe_to='m9', align_id=None)
```

To copy only the '`Th`' PCS data from '`m3`' to '`m6`', type one of:

```
relax> pcs.copy('m3', 'm6', 'Th')
```

```
relax> pcs.copy(pipe_from='m3', pipe_to='m6', align_id='Th')
```

10.2.82 pcs.corr_plot()

Synopsis

Generate a correlation plot of the measured vs. the back-calculated PCSs.

Defaults

pcs.corr_plot(self, format='grace', file='pcs_corr_plot.agr', dir=None, force=False)

Keyword Arguments

format: The format of the plot data.

file: The name of the file.

dir: The directory name.

force: A flag which if True will cause the file to be over-written.

Description

Two formats are currently supported. If format is set to '`grace`', then a Grace plot file will be created. If the format arg is set to None, then a plain text list of the measured and back-calculated data will be created.

Examples

To create a Grace plot of the data, type:

```
relax> pcs.corr_plot()
```

To create a plain text list of the measured and back-calculated data, type one of:

```
relax> pcs.corr_plot(None)
```

```
relax> pcs.corr_plot(format=None)
```

10.2.83 pcs.delete()

Synopsis

Delete the PCS data corresponding to the alignment ID.

Defaults

pcs.delete(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

Examples

To delete the PCS data corresponding to align_id='PH_gel', type:

```
relax> pcs.delete('PH_gel')
```

10.2.84 pcs.display()

Synopsis

Display the PCS data corresponding to the alignment ID.

Defaults

pcs.display(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

Examples

To display the 'phage' PCS data, type:

```
relax> pcs.display('phage')
```

10.2.85 pcs.read()

Synopsis

Read the PCS data from file.

Defaults

pcs.read(self, align_id=None, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, data_col=None, error_col=None, sep=None, spin_id=None)

Keyword Arguments

align_id: The alignment ID string.

file: The name of the file containing the PCS data.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The PCS data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Examples

The following commands will read the PCS data out of the file 'Tb.txt' where the columns are separated by the symbol ',', and store the PCSs under the ID 'Tb'.

```
relax> pcs.read('Tb', 'Tb.txt', sep=',')
```

To read the 15N and 1H PCSs from the file 'Eu.txt', where the 15N values are in the 4^{th} column and the 1H in the 9^{th}, type both the following:

```
relax> pcs.read('Tb', 'Tb.txt', spin_id=
'@N', res_num_col=1, data_col=4)
```

```
relax> pcs.read('Tb', 'Tb.txt', spin_id=
'@H', res_num_col=1, data_col=9)
```

10.2.86 pcs.weight()

Synopsis

Set optimisation weights on the PCS data.

Defaults

pcs.weight(self, align_id=None, spin_id=None, weight=1.0)

Keyword Arguments

align_id: The alignment ID string.

spin_id: The spin ID string.

weight: The weighting value.

Description

This function can be used to force the PCS to contribute more or less to the chi-squared optimisation statistic. The higher the value, the more importance the PCS will have.

10.2.87 pcs.write()

Synopsis

Write the PCS data to file.

Defaults

pcs.write(self, align_id=None, file=None, dir=None, force=False)

Keyword Arguments

align_id: The alignment ID string.

file: The name of the file.

dir: The directory name.

force: A flag which if True will cause the file to be over-written.

Description

If no directory name is given, the file will be placed in the current working directory. The '`align_id`' argument are required for selecting which PCS data set will be written to file.

10.2.88 pipe.copy()

Synopsis

Function for copying a data pipe.

Defaults

pipe.copy(self, pipe_from=None, pipe_to=None)

Keyword Arguments

pipe_from: The name of the source data pipe to copy the data from.

pipe_to: The name of the target data pipe to copy the data to.

Description

This user function allows the contents of a data pipe to be copied. If the '`pipe_from`' keyword argument is set to None the current data pipe is assumed. The data pipe corresponding to '`pipe_to`' must not yet exist.

Examples

To copy the contents of the '`m1`' data pipe to the '`m2`' data pipe, type:

```
relax> pipe.copy('m1', 'm2')
```

```
relax> pipe.copy(pipe_from='m1', pipe_to='m2')
```

If the current data pipe is '`m1`', then the following command can be used:

```
relax> pipe.copy(pipe_to='m2')
```

10.2.89 pipe.create()

Synopsis

Function for initialising a data pipe.

Defaults

pipe.create(self, pipe_name=None, pipe_type=None)

Keyword Arguments

pipe_name: The name of the data pipe.

pipe_type: The type of data pipe.

Description

The data pipe name can be any string however the data pipe type can only be one of the following:

'frame order' – The Frame Order theories,

'jw' – Reduced spectral density mapping,

'mf' – Model-free analysis,

'N-state' – N-state model of domain motions,

'noe' – Steady state NOE calculation,

'relax_fit' – Relaxation curve fitting,

'srls' – SRLS analysis.

Examples

To set up a model-free analysis data pipe with the name 'm5', type:

```
relax> pipe.create('m5', 'mf')
```

10.2.90 pipe.current()

Synopsis

Print the name of the current pipe.

Defaults

pipe.current(self)

Examples

To run the user function, type:

```
relax> pipe.current()
```

10.2.91 pipe.delete()

Synopsis

Function for deleting a data pipe.

Defaults

pipe.delete(self, pipe_name=None)

Keyword Arguments

pipe_name: The name of the data pipe.

Description

This function will permanently remove the data pipe and all of its contents. If the pipe name is not given, then all data pipes will be deleted.

10.2.92 pipe.display()

Synopsis

Print a list of all the data pipes.

Defaults

pipe.display(self)

Examples

To run the user function, type:

```
relax> pipe.display()
```

10.2.93 pipe.hybridise()

Synopsis

Create a hybrid data pipe by fusing a number of other data pipes.

Defaults

pipe.hybridise(self, hybrid=None, pipes=None)

Keyword Arguments

hybrid: The name of the hybrid data pipe to create.

pipes: An array containing the names of all data pipes to hybridise.

Description

This user function can be used to construct hybrid models. An example of the use of a hybrid model could be if the protein consists of two independent domains. These two domains could be analysed separately, each having their own optimised diffusion tensors. The N-terminal domain data pipe could be called '`N_sphere`' while the C-terminal domain could be called '`C_ellipsoid`'. These two data pipes could then be hybridised into a single data pipe called '`mixed model`' by typing:

```
relax> pipe.hybridise('mixed model',
['N_sphere', 'C_ellipsoid'])
```

```
relax> pipe.hybridise(hybrid='mixed model',
pipes=['N_sphere', 'C_ellipsoid'])
```

This hybrid data pipe can then be compared via model selection to a data pipe whereby the entire protein is assumed to have a single diffusion tensor.

The requirements for data pipes to be hybridised is that the molecules, sequences, and spin systems for all the data pipes is the same, and that no spin system is allowed to be selected in two or more data pipes. The selections must not overlap to allow for rigorous statistical comparisons.

10.2.94 pipe.switch()

Synopsis

Function for switching between data pipes.

Defaults

pipe.switch(self, pipe_name=None)

Keyword Arguments

pipe_name: The name of the data pipe.

Description

This function will switch from the current data pipe to the given data pipe.

Examples

To switch to the '`ellipsoid`' data pipe, type:

```
relax> pipe.switch('ellipsoid')
```

```
relax> pipe.switch(pipe_name='ellipsoid')
```

10.2.95 pymol.cartoon()

Synopsis

Apply the PyMOL cartoon style and colour by secondary structure.

Defaults

pymol.cartoon(self)

Description

This function applies the PyMOL cartoon style which is equivalent to hiding everything and clicking on show cartoon. It also colours the cartoon with red helices, yellow strands, and green loops. The following commands are executed:

cmd.hide('`everything`', file)

cmd.show('`cartoon`', file)

util.cbss(file, '`red`', '`yellow`', '`green`')

where file is the file name without the '`.pdb`' extension.

Example

To apply this user function, type:

```
relax> pymol.cartoon()
```

10.2.96 pymol.clear_history()

Synopsis

Function for clearing the PyMOL command history.

Defaults

pymol.clear_history(self)

10.2.97 pymol.command()

Synopsis

Function for executing a user supplied PyMOL command.

Defaults

pymol.command(self, command=None)

Keyword Arguments

command: The PyMOL command to execute.

Description

This user function allows you to pass PyMOL commands to the program. This can be useful for automation or scripting.

Example

To reinitialise the PyMOL instance, type:

```
relax> pymol.command("reinitialise")
```

10.2.98 pymol.cone_pdb()

Synopsis

Display, as designed, the cone PDB geometric object from the N-state model.

Defaults

pymol.cone_pdb(self, file=None)

Keyword Arguments

file: The name of the PDB file containing the cone geometric object.

Description

The PDB file containing the geometric object must be created using the complementary '`n_state_model.cone_pdb()`' user function.

The cone PDB file is read in using the command:

load file

The average CoM-pivot point vector, the residue '`AVE`' is displayed using the commands:

select resn AVE

show sticks, '`sele`'

color blue, '`sele`'

The cone object, the residue '`CON`', is displayed using the commands:

select resn CON

hide ('`sele`')

show sticks, '`sele`'

color white, '`sele`'

10.2.99 pymol.macro_exec()

Synopsis

Function for executing PyMOL macros.

Defaults

pymol.macro_exec(self, data_type=None, style='classic', colour_start=None, colour_end=None, colour_list=None)

Keyword Arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start: The starting colour, either an array or string, of the linear colour gradient.

colour_end: The ending colour, either an array or string, of the linear colour gradient.

colour_list: The list of colours to match the start and end strings.

Description

This function allows residues specific values to be mapped to a structure through PyMOL macros. Currently only the '`classic`' style, which is described below, is available.

Colour

The values are coloured based on a linear colour gradient which is specified through the '`colour_start`' and '`colour_end`' arguments. These arguments can either be a string to identify one of the RGB (red, green, blue) colour arrays listed in the tables below, or you can give the RGB vector itself. For example, colour_start='`white`' and colour_start=[1.0, 1.0, 1.0] both select the same colour. Leaving both arguments at None will select the default colour gradient which for each type of analysis is described below.

When supplying the colours as strings, two lists of colours can be selected from which to match the strings. These are the default PyMOL colour list and the X11 colour list, both of which are described in the tables below. The default behaviour is to first search the Molmol list and then the X11 colour list, raising an error if neither contain the string. To explicitly select these lists, set the '`colour_list`' argument to either '`molmol`' or '`x11`'.

Examples

To map the order parameter values, S^2, onto the structure using the classic style, type:

```
relax> pymol.macro_exec('S2')

relax> pymol.macro_exec(data_type='S2')

relax> pymol.macro_exec(data_type='S2',
style="classic")
```

10.2.100 pymol.tensor_pdb()

Synopsis

Function displaying the diffusion tensor PDB geometric object over the loaded PDB.

Defaults

pymol.tensor_pdb(self, file=None)

Keyword Arguments

file: The name of the PDB file containing the tensor geometric object.

Description

In executing this user function, a PDB file must have previously been loaded into this data pipe a geometric object or polygon representing the Brownian rotational diffusion tensor will be overlain with the loaded PDB file and displayed within PyMOL. The PDB file containing the geometric object must be created using the complementary '`pdb.create_diff_tensor_pdb()`' user function.

The tensor PDB file is read in using the command:

load file

The centre of mass residue '`COM`' is displayed using the commands:

select resn COM

show dots, '`sele`'

color blue, '`sele`'

The axes of the diffusion tensor, the residue '`AXS`', is displayed using the commands:

select resn AXS

hide ('`sele`')

show sticks, '`sele`'

color cyan, '`sele`'

label '`sele`', name

The simulation axes, the residues '`SIM`', are displayed using the commands:

select resn SIM

colour cyan, '`sele`'

10.2.101 pymol.vector_dist()

Synopsis

Function displaying the PDB file representation of the XH vector distribution.

Defaults

pymol.vector_dist(self, file='XH_dist.pdb')

Keyword Arguments

file: The name of the PDB file containing the vector distribution.

Description

A PDB file of the macromolecule must have previously been loaded as the vector distribution will be overlain with the macromolecule within PyMOL. The PDB file containing the vector distribution must be created using the complementary '`pdb.create_vector_dist()`' user function.

The vector distribution PDB file is read in using the command:

```
load file
```

10.2.102 pymol.view()

Synopsis

Function for viewing the collection of molecules extracted from the PDB file.

Defaults

pymol.view(self)

Example

```
relax> pymol.view()
```

10.2.103 pymol.write()

Synopsis

Function for creating PyMOL macros.

Defaults

pymol.write(self, data_type=None, style='classic', colour_start=None, colour_end=None, colour_list=None, file=None, dir='pymol', force=False)

Keyword Arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start: The starting colour, either an array or string, of the linear colour gradient.

colour_end: The ending colour, either an array or string, of the linear colour gradient.

colour_list: The list of colours to match the start and end strings.

file: The name of the file.

dir: The directory name.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

This function allows residues specific values to be mapped to a structure through the creation of a PyMOL macro which can be executed in PyMOL by clicking on '`File, Macro, Execute User...`'. Currently only the '`classic`' style, which is described below, is available.

Colour

The values are coloured based on a linear colour gradient which is specified through the '`colour_start`' and '`colour_end`' arguments. These arguments can either be a string to identify one of the RGB (red, green, blue) colour arrays listed in the tables below, or you can give the RGB vector itself. For example, colour_start='`white`' and colour_start=[1.0, 1.0, 1.0] both select the same colour. Leaving both arguments at None will select the default colour gradient which for each type of analysis is described below.

When supplying the colours as strings, two lists of colours can be selected from which to match the strings. These are the default PyMOL colour list and the X11 colour list, both of which are described in the tables below. The default behaviour is to first search the PyMOL list and then the X11 colour list, raising an error if neither contain the string. To explicitly select these lists, set the '`colour_list`' argument to either '`molmol`' or '`x11`'.

Examples

To create a PyMOL macro mapping the order parameter values, S^2, onto the structure using the classic style, type:

```
relax> pymol.write('S2')

relax> pymol.write(data_type='S2')

relax> pymol.write(data_type='S2', style=
"classic", file='s2.mac', dir='pymol')
```

Molmol RGB colour arrays

The following table is a list of colours used in Molmol and their corresponding RGB colour values ranging from 0 to 1.

(see table 10.21)

X11 RGB colour arrays

The following table is the list of X11 colour names and their corresponding RGB colour values ranging from 0 to 255.

(see table 10.22)

Table 10.21: First table for the pymol.write() user function.

Name	Red	Green	Blue
'black'	0.000	0.000	0.000
'navy'	0.000	0.000	0.502
'blue'	0.000	0.000	1.000
'dark green'	0.000	0.392	0.000
'green'	0.000	1.000	0.000
'cyan'	0.000	1.000	1.000
'turquoise'	0.251	0.878	0.816
'royal blue'	0.255	0.412	0.882
'aquamarine'	0.498	1.000	0.831
'sky green'	0.529	0.808	0.922
'dark violet'	0.580	0.000	0.827
'pale green'	0.596	0.984	0.596
'purple'	0.627	0.125	0.941
'brown'	0.647	0.165	0.165
'light blue'	0.678	0.847	0.902
'grey'	0.745	0.745	0.745
'light grey'	0.827	0.827	0.827
'violet'	0.933	0.510	0.933
'light coral'	0.941	0.502	0.502
'khaki'	0.941	0.902	0.549
'beige'	0.961	0.961	0.863
'red'	1.000	0.000	0.000
'magenta'	1.000	0.000	1.000
'deep pink'	1.000	0.078	0.576
'orange red'	1.000	0.271	0.000
'hot pink'	1.000	0.412	0.706
'coral'	1.000	0.498	0.314
'dark orange'	1.000	0.549	0.000
'orange'	1.000	0.647	0.000
'pink'	1.000	0.753	0.796
'gold'	1.000	0.843	0.000
'yellow'	1.000	1.000	0.000
'light yellow'	1.000	1.000	0.878
'ivory'	1.000	1.000	0.941
'white'	1.000	1.000	1.000

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
snow	255	250	250
ghost white	248	248	255
white smoke	245	245	245
gainsboro	220	220	220
floral white	255	250	240
old lace	253	245	230
linen	250	240	230
antique white	250	235	215
papaya whip	255	239	213
blanched almond	255	235	205
bisque	255	228	196
peach puff	255	218	185
navajo white	255	222	173
moccasin	255	228	181
cornsilk	255	248	220
ivory	255	255	240
lemon chiffon	255	250	205
seashell	255	245	238
honeydew	240	255	240
mint cream	245	255	250
azure	240	255	255
alice blue	240	248	255
lavender	230	230	250
lavender blush	255	240	245
misty rose	255	228	225
white	255	255	255
black	0	0	0
dark slate grey	47	79	79
dim grey	105	105	105
slate grey	112	128	144
light slate grey	119	136	153
grey	190	190	190
light grey	211	211	211
midnight blue	25	25	112
navy	0	0	128
cornflower blue	100	149	237
dark slate blue	72	61	139
slate blue	106	90	205
medium slate blue	123	104	238
light slate blue	132	112	255
medium blue	0	0	205
royal blue	65	105	225
blue	0	0	255
dodger blue	30	144	255
deep sky blue	0	191	255
sky blue	135	206	235
light sky blue	135	206	250
steel blue	70	130	180
light steel blue	176	196	222
light blue	173	216	230
powder blue	176	224	230
pale turquoise	175	238	238
dark turquoise	0	206	209
medium turquoise	72	209	204
turquoise	64	224	208
cyan	0	255	255
light cyan	224	255	255
cadet blue	95	158	160
medium aquamarine	102	205	170
aquamarine	127	255	212
dark green	0	100	0
dark olive green	85	107	47
dark sea green	143	188	143
sea green	46	139	87
medium sea green	60	179	113
light sea green	32	178	170

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
pale green	152	251	152
spring green	0	255	127
lawn green	124	252	0
green	0	255	0
chartreuse	127	255	0
medium spring green	0	250	154
green yellow	173	255	47
lime green	50	205	50
yellow green	154	205	50
forest green	34	139	34
olive drab	107	142	35
dark khaki	189	183	107
khaki	240	230	140
pale goldenrod	238	232	170
light goldenrod yellow	250	250	210
light yellow	255	255	224
yellow	255	255	0
gold	255	215	0
light goldenrod	238	221	130
goldenrod	218	165	32
dark goldenrod	184	134	11
rosy brown	188	143	143
indian red	205	92	92
saddle brown	139	69	19
sienna	160	82	45
peru	205	133	63
burlywood	222	184	135
beige	245	245	220
wheat	245	222	179
sandy brown	244	164	96
tan	210	180	140
chocolate	210	105	30
firebrick	178	34	34
brown	165	42	42
dark salmon	233	150	122
salmon	250	128	114
light salmon	255	160	122
orange	255	165	0
dark orange	255	140	0
coral	255	127	80
light coral	240	128	128
tomato	255	99	71
orange red	255	69	0
red	255	0	0
hot pink	255	105	180
deep pink	255	20	147
pink	255	192	203
light pink	255	182	193
pale violet red	219	112	147
maroon	176	48	96
medium violet red	199	21	133
violet red	208	32	144
magenta	255	0	255
violet	238	130	238
plum	221	160	221
orchid	218	112	214
medium orchid	186	85	211
dark orchid	153	50	204
dark violet	148	0	211
blue violet	138	43	226
purple	160	32	240
medium purple	147	112	219
thistle	216	191	216
snow 1	255	250	250
snow 2	238	233	233
snow 3	205	201	201

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
snow 4	139	137	137
seashell 1	255	245	238
seashell 2	238	229	222
seashell 3	205	197	191
seashell 4	139	134	130
antique white 1	255	239	219
antique white 2	238	223	204
antique white 3	205	192	176
antique white 4	139	131	120
bisque 1	255	228	196
bisque 2	238	213	183
bisque 3	205	183	158
bisque 4	139	125	107
peach puff 1	255	218	185
peach puff 2	238	203	173
peach puff 3	205	175	149
peach puff 4	139	119	101
navajo white 1	255	222	173
navajo white 2	238	207	161
navajo white 3	205	179	139
navajo white 4	139	121	94
lemon chiffon 1	255	250	205
lemon chiffon 2	238	233	191
lemon chiffon 3	205	201	165
lemon chiffon 4	139	137	112
cornsilk 1	255	248	220
cornsilk 2	238	232	205
cornsilk 3	205	200	177
cornsilk 4	139	136	120
ivory 1	255	255	240
ivory 2	238	238	224
ivory 3	205	205	193
ivory 4	139	139	131
honeydew 1	240	255	240
honeydew 2	224	238	224
honeydew 3	193	205	193
honeydew 4	131	139	131
lavender blush 1	255	240	245
lavender blush 2	238	224	229
lavender blush 3	205	193	197
lavender blush 4	139	131	134
misty rose 1	255	228	225
misty rose 2	238	213	210
misty rose 3	205	183	181
misty rose 4	139	125	123
azure 1	240	255	255
azure 2	224	238	238
azure 3	193	205	205
azure 4	131	139	139
slate blue 1	131	111	255
slate blue 2	122	103	238
slate blue 3	105	89	205
slate blue 4	71	60	139
royal blue 1	72	118	255
royal blue 2	67	110	238
royal blue 3	58	95	205
royal blue 4	39	64	139
blue 1	0	0	255
blue 2	0	0	238
blue 3	0	0	205
blue 4	0	0	139
dodger blue 1	30	144	255
dodger blue 2	28	134	238
dodger blue 3	24	116	205
dodger blue 4	16	78	139
steel blue 1	99	184	255

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
steel blue 2	92	172	238
steel blue 3	79	148	205
steel blue 4	54	100	139
deep sky blue 1	0	191	255
deep sky blue 2	0	178	238
deep sky blue 3	0	154	205
deep sky blue 4	0	104	139
sky blue 1	135	206	255
sky blue 2	126	192	238
sky blue 3	108	166	205
sky blue 4	74	112	139
light sky blue 1	176	226	255
light sky blue 2	164	211	238
light sky blue 3	141	182	205
light sky blue 4	96	123	139
slate grey 1	198	226	255
slate grey 2	185	211	238
slate grey 3	159	182	205
slate grey 4	108	123	139
light steel blue 1	202	225	255
light steel blue 2	188	210	238
light steel blue 3	162	181	205
light steel blue 4	110	123	139
light blue 1	191	239	255
light blue 2	178	223	238
light blue 3	154	192	205
light blue 4	104	131	139
light cyan 1	224	255	255
light cyan 2	209	238	238
light cyan 3	180	205	205
light cyan 4	122	139	139
pale turquoise 1	187	255	255
pale turquoise 2	174	238	238
pale turquoise 3	150	205	205
pale turquoise 4	102	139	139
cadet blue 1	152	245	255
cadet blue 2	142	229	238
cadet blue 3	122	197	205
cadet blue 4	83	134	139
turquoise 1	0	245	255
turquoise 2	0	229	238
turquoise 3	0	197	205
turquoise 4	0	134	139
cyan 1	0	255	255
cyan 2	0	238	238
cyan 3	0	205	205
cyan 4	0	139	139
dark slate grey 1	151	255	255
dark slate grey 2	141	238	238
dark slate grey 3	121	205	205
dark slate grey 4	82	139	139
aquamarine 1	127	255	212
aquamarine 2	118	238	198
aquamarine 3	102	205	170
aquamarine 4	69	139	116
dark sea green 1	193	255	193
dark sea green 2	180	238	180
dark sea green 3	155	205	155
dark sea green 4	105	139	105
sea green 1	84	255	159
sea green 2	78	238	148
sea green 3	67	205	128
sea green 4	46	139	87
pale green 1	154	255	154
pale green 2	144	238	144
pale green 3	124	205	124

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
pale green 4	84	139	84
spring green 1	0	255	127
spring green 2	0	238	118
spring green 3	0	205	102
spring green 4	0	139	69
green 1	0	255	0
green 2	0	238	0
green 3	0	205	0
green 4	0	139	0
chartreuse 1	127	255	0
chartreuse 2	118	238	0
chartreuse 3	102	205	0
chartreuse 4	69	139	0
olive drab 1	192	255	62
olive drab 2	179	238	58
olive drab 3	154	205	50
olive drab 4	105	139	34
dark olive green 1	202	255	112
dark olive green 2	188	238	104
dark olive green 3	162	205	90
dark olive green 4	110	139	61
khaki 1	255	246	143
khaki 2	238	230	133
khaki 3	205	198	115
khaki 4	139	134	78
light goldenrod 1	255	236	139
light goldenrod 2	238	220	130
light goldenrod 3	205	190	112
light goldenrod 4	139	129	76
light yellow 1	255	255	224
light yellow 2	238	238	209
light yellow 3	205	205	180
light yellow 4	139	139	122
yellow 1	255	255	0
yellow 2	238	238	0
yellow 3	205	205	0
yellow 4	139	139	0
gold 1	255	215	0
gold 2	238	201	0
gold 3	205	173	0
gold 4	139	117	0
goldenrod 1	255	193	37
goldenrod 2	238	180	34
goldenrod 3	205	155	29
goldenrod 4	139	105	20
dark goldenrod 1	255	185	15
dark goldenrod 2	238	173	14
dark goldenrod 3	205	149	12
dark goldenrod 4	139	101	8
rosy brown 1	255	193	193
rosy brown 2	238	180	180
rosy brown 3	205	155	155
rosy brown 4	139	105	105
indian red 1	255	106	106
indian red 2	238	99	99
indian red 3	205	85	85
indian red 4	139	58	58
sienna 1	255	130	71
sienna 2	238	121	66
sienna 3	205	104	57
sienna 4	139	71	38
burlywood 1	255	211	155
burlywood 2	238	197	145
burlywood 3	205	170	125
burlywood 4	139	115	85
wheat 1	255	231	186

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
wheat 2	238	216	174
wheat 3	205	186	150
wheat 4	139	126	102
tan 1	255	165	79
tan 2	238	154	73
tan 3	205	133	63
tan 4	139	90	43
chocolate 1	255	127	36
chocolate 2	238	118	33
chocolate 3	205	102	29
chocolate 4	139	69	19
firebrick 1	255	48	48
firebrick 2	238	44	44
firebrick 3	205	38	38
firebrick 4	139	26	26
brown 1	255	64	64
brown 2	238	59	59
brown 3	205	51	51
brown 4	139	35	35
salmon 1	255	140	105
salmon 2	238	130	98
salmon 3	205	112	84
salmon 4	139	76	57
light salmon 1	255	160	122
light salmon 2	238	149	114
light salmon 3	205	129	98
light salmon 4	139	87	66
orange 1	255	165	0
orange 2	238	154	0
orange 3	205	133	0
orange 4	139	90	0
dark orange 1	255	127	0
dark orange 2	238	118	0
dark orange 3	205	102	0
dark orange 4	139	69	0
coral 1	255	114	86
coral 2	238	106	80
coral 3	205	91	69
coral 4	139	62	47
tomato 1	255	99	71
tomato 2	238	92	66
tomato 3	205	79	57
tomato 4	139	54	38
orange red 1	255	69	0
orange red 2	238	64	0
orange red 3	205	55	0
orange red 4	139	37	0
red 1	255	0	0
red 2	238	0	0
red 3	205	0	0
red 4	139	0	0
deep pink 1	255	20	147
deep pink 2	238	18	137
deep pink 3	205	16	118
deep pink 4	139	10	80
hot pink 1	255	110	180
hot pink 2	238	106	167
hot pink 3	205	96	144
hot pink 4	139	58	98
pink 1	255	181	197
pink 2	238	169	184
pink 3	205	145	158
pink 4	139	99	108
light pink 1	255	174	185
light pink 2	238	162	173
light pink 3	205	140	149

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
light pink 4	139	95	101
pale violet red 1	255	130	171
pale violet red 2	238	121	159
pale violet red 3	205	104	137
pale violet red 4	139	71	93
maroon 1	255	52	179
maroon 2	238	48	167
maroon 3	205	41	144
maroon 4	139	28	98
violet red 1	255	62	150
violet red 2	238	58	140
violet red 3	205	50	120
violet red 4	139	34	82
magenta 1	255	0	255
magenta 2	238	0	238
magenta 3	205	0	205
magenta 4	139	0	139
orchid 1	255	131	250
orchid 2	238	122	233
orchid 3	205	105	201
orchid 4	139	71	137
plum 1	255	187	255
plum 2	238	174	238
plum 3	205	150	205
plum 4	139	102	139
medium orchid 1	224	102	255
medium orchid 2	209	95	238
medium orchid 3	180	82	205
medium orchid 4	122	55	139
dark orchid 1	191	62	255
dark orchid 2	178	58	238
dark orchid 3	154	50	205
dark orchid 4	104	34	139
purple 1	155	48	255
purple 2	145	44	238
purple 3	125	38	205
purple 4	85	26	139
medium purple 1	171	130	255
medium purple 2	159	121	238
medium purple 3	137	104	205
medium purple 4	93	71	139
thistle 1	255	225	255
thistle 2	238	210	238
thistle 3	205	181	205
thistle 4	139	123	139
grey 0	0	0	0
grey 1	3	3	3
grey 2	5	5	5
grey 3	8	8	8
grey 4	10	10	10
grey 5	13	13	13
grey 6	15	15	15
grey 7	18	18	18
grey 8	20	20	20
grey 9	23	23	23
grey 10	26	26	26
grey 11	28	28	28
grey 12	31	31	31
grey 13	33	33	33
grey 14	36	36	36
grey 15	38	38	38
grey 16	41	41	41
grey 17	43	43	43
grey 18	46	46	46
grey 19	48	48	48
grey 20	51	51	51

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
grey 21	54	54	54
grey 22	56	56	56
grey 23	59	59	59
grey 24	61	61	61
grey 25	64	64	64
grey 26	66	66	66
grey 27	69	69	69
grey 28	71	71	71
grey 29	74	74	74
grey 30	77	77	77
grey 31	79	79	79
grey 32	82	82	82
grey 33	84	84	84
grey 34	87	87	87
grey 35	89	89	89
grey 36	92	92	92
grey 37	94	94	94
grey 38	97	97	97
grey 39	99	99	99
grey 40	102	102	102
grey 41	105	105	105
grey 42	107	107	107
grey 43	110	110	110
grey 44	112	112	112
grey 45	115	115	115
grey 46	117	117	117
grey 47	120	120	120
grey 48	122	122	122
grey 49	125	125	125
grey 50	127	127	127
grey 51	130	130	130
grey 52	133	133	133
grey 53	135	135	135
grey 54	138	138	138
grey 55	140	140	140
grey 56	143	143	143
grey 57	145	145	145
grey 58	148	148	148
grey 59	150	150	150
grey 60	153	153	153
grey 61	156	156	156
grey 62	158	158	158
grey 63	161	161	161
grey 64	163	163	163
grey 65	166	166	166
grey 66	168	168	168
grey 67	171	171	171
grey 68	173	173	173
grey 69	176	176	176
grey 70	179	179	179
grey 71	181	181	181
grey 72	184	184	184
grey 73	186	186	186
grey 74	189	189	189
grey 75	191	191	191
grey 76	194	194	194
grey 77	196	196	196
grey 78	199	199	199
grey 79	201	201	201
grey 80	204	204	204
grey 81	207	207	207
grey 82	209	209	209
grey 83	212	212	212
grey 84	214	214	214
grey 85	217	217	217
grey 86	219	219	219

Table 10.22: Second table for the pymol.write() user function.

Name	Red	Green	Blue
grey 87	222	222	222
grey 88	224	224	224
grey 89	227	227	227
grey 90	229	229	229
grey 91	232	232	232
grey 92	235	235	235
grey 93	237	237	237
grey 94	240	240	240
grey 95	242	242	242
grey 96	245	245	245
grey 97	247	247	247
grey 98	250	250	250
grey 99	252	252	252
grey 100	255	255	255
dark grey	169	169	169
dark blue	0	0	139
dark cyan	0	139	139
dark magenta	139	0	139
dark red	139	0	0
light green	144	238	144

10.2.104 rdc.back_calc()

Synopsis

Back calculate RDCs.

Defaults

rdc.back_calc(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

10.2.105 rdc.calc_q_factors()

Synopsis

Calculate the RDC Q-factor for the selected spins.

Defaults

rdc.calc_q_factors(self, spin_id=None)

Keyword Arguments

spin_id: The spin ID string for restricting to subset of all selected spins.

Description

For this function to work, the back-calculated RDC data must first be generated by the analysis specific code. Otherwise a warning will be given.

Examples

To calculate the RDC Q-factor for only the spins '@H26', '@H27', and '@H28', type one of:

```
relax> rdc.calc_q_factors('@H26 & @H27 &
@H28')

relax> rdc.calc_q_factors(spin_id='@H26 &
@H27 & @H28')
```

10.2.106 rdc.copy()

Synopsis

Copy RDC data from pipe_from to pipe_to.

Defaults

rdc.copy(self, pipe_from=None, pipe_to=None, align_id=None)

Keyword Arguments

pipe_from: The name of the pipe to copy the RDC data from.

pipe_to: The name of the pipe to copy the RDC data to.

align_id: The alignment ID string.

Description

This function will copy RDC data from 'pipe_from' to 'pipe_to'. If align_id is not given then all RDC data will be copied, otherwise only a specific data set will be.

Examples

To copy all RDC data from pipe 'm1' to pipe 'm9', type one of:

```
relax> rdc.copy('m1', 'm9')

relax> rdc.copy(pipe_from='m1', pipe_to=
'm9')

relax> rdc.copy('m1', 'm9', None)

relax> rdc.copy(pipe_from='m1', pipe_to=
'm9', align_id=None)
```

To copy only the 'Th' RDC data from 'm3' to 'm6', type one of:

```
relax> rdc.copy('m3', 'm6', 'Th')

relax> rdc.copy(pipe_from='m3', pipe_to=
'm6', align_id='Th')
```

10.2.107 rdc.corr_plot()

Synopsis

Generate a correlation plot of the measured vs. the back-calculated RDCs.

Defaults

rdc.corr_plot(self, format='grace', file='rdc_corr_plot.agr', dir=None, force=False)

Keyword Arguments

format: The format of the plot data.

file: The name of the file.

dir: The directory name.

force: A flag which if True will cause the file to be over-written.

Description

Two formats are currently supported. If format is set to '`grace`', then a Grace plot file will be created. If the format arg is set to None, then a plain text list of the measured and back-calculated data will be created.

Examples

To create a Grace plot of the data, type:

```
relax> rdc.corr_plot()
```

To create a plain text list of the measured and back-calculated data, type one of:

```
relax> rdc.corr_plot(None)
```

```
relax> rdc.corr_plot(format=None)
```

10.2.108 rdc.delete()

Synopsis

Delete the RDC data corresponding to the alignment ID.

Defaults

rdc.delete(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

Examples

To delete the RDC data corresponding to align_id='`PH_gel`', type:

```
relax> rdc.delete('PH_gel')
```

10.2.109 rdc.display()

Synopsis

Display the RDC data corresponding to the alignment ID.

Defaults

rdc.display(self, align_id=None)

Keyword Arguments

align_id: The alignment ID string.

Examples

To display the '`phage`' RDC data, type:

```
relax> rdc.display('phage')
```

10.2.110 rdc.read()

Synopsis

Read the RDC data from file.

Defaults

rdc.read(self, align_id=None, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, data_col=None, error_col=None, sep=None, spin_id=None, neg_g_corr=False)

Keyword Arguments

align_id: The alignment ID string.

file: The name of the file containing the RDC data.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

neg_g_corr: A flag which is used to correct for the negative gyromagnetic ratio of 15N.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

If neg_g_corr is set to True, a sign inversion will be applied to all RDC values to be loaded.

Examples

The following commands will read the RDC data out of the file '`Tb.txt`' where the columns are separated by the symbol '`,`', and store the RDCs under the ID '`Tb`'.

```
relax> rdc.read('Tb', 'Tb.txt', sep=',')
```

If the individual spin RDC errors are located in the file '`rdc_err.txt`' in column number 5, then to read these values into relax, type one of:

```
relax> rdc.read('phage', 'rdc_err.txt',
error_col=5)

relax> rdc.read(align_id='phage', file=
'rdc_err.txt', error_col=5)
```

If the RDCs correspond to the '`N`' spin and other spin types such as 1H, 13C, etc. are loaded into relax, then type:

```
relax> rdc.read('Tb', 'Tb.txt', spin_id=
'@N')
```

10.2.111 rdc.weight()

Synopsis

Set optimisation weights on the RDC data.

Defaults

rdc.weight(self, align_id=None, spin_id=None, weight=1.0)

Keyword Arguments

align_id: The alignment ID string.

spin_id: The spin ID string.

weight: The weighting value.

Description

This function can be used to force the RDC to contribute more or less to the chi-squared optimisation statistic. The higher the value, the more importance the RDC will have.

10.2.112 rdc.write()

Synopsis

Write the RDC data to file.

Defaults

rdc.write(self, align_id=None, file=None, dir=None, force=False)

Keyword Arguments

align_id: The alignment ID string.

file: The name of the file.

dir: The directory name.

force: A flag which if True will cause the file to be overwritten.

Description

If no directory name is given, the file will be placed in the current working directory. The '`align_id`' argument are required for selecting which RDC data set will be written to file.

10.2.113 relax_data.back_calc()

Synopsis

Function for back calculating relaxation data.

Defaults

relax_data.back_calc(self, ri_label=None, frq_label=None, frq=None)

Keyword Arguments

ri_label: The relaxation data type, ie '`R1`', '`R2`', or '`NOE`'.

frq_label: The field strength label.

frq: The spectrometer frequency in Hz.

10.2.114 relax_data.copy()

Synopsis

Function for copying relaxation data from pipe_from to pipe_to.

Defaults

relax_data.copy(self, pipe_from=None, pipe_to=None, ri_label=None, frq_label=None)

Keyword Arguments

pipe_from: The name of the pipe to copy the relaxation data from.

pipe_to: The name of the pipe to copy the relaxation data to.

ri_label: The relaxation data type, ie '`R1`', '`R2`', or '`NOE`'.

frq_label: The field strength label.

Description

This function will copy relaxation data from '`pipe_from`' to '`pipe_to`'. If ri_label and frq_label are not given then all relaxation data will be copied, otherwise only a specific data set will be copied.

Examples

To copy all relaxation data from pipe '`m1`' to pipe '`m9`', type one of:

```
relax> relax_data.copy('m1', 'm9')
```

```
relax> relax_data.copy(pipe_from='m1',
pipe_to='m9')
```

```
relax> relax_data.copy('m1', 'm9', None,
None)
```

```
relax> relax_data.copy(pipe_from='m1',
pipe_to='m9', ri_label=None, frq_label=None)
```

To copy only the NOE relaxation data with the frq_label of '`800`' from '`m3`' to '`m6`', type one of:

```
relax> relax_data.copy('m3', 'm6', 'NOE',
'800')
```

```
relax> relax_data.copy(pipe_from='m3',
pipe_to='m6', ri_label='NOE', frq_label=
'800')
```

10.2.115 relax_data.delete()

Synopsis

Function for deleting the relaxation data corresponding to ri_label and frq_label.

Defaults

relax_data.delete(self, ri_label=None, frq_label=None)

Keyword Arguments

ri_label: The relaxation data type, ie '`R1`', '`R2`', or '`NOE`'.

frq_label: The field strength label.

Examples

To delete the relaxation data corresponding to ri_label='NOE', frq_label='600', type:

```
relax> relax_data.delete('NOE', '600')
```

10.2.116 relax_data.display()

Synopsis

Function for displaying the relaxation data corresponding to ri_label and frq_label.

Defaults

relax_data.display(self, ri_label=None, frq_label=None)

Keyword Arguments

ri_label: The relaxation data type, ie 'R1', 'R2', or 'NOE'.

frq_label: The field strength label.

Examples

To display the NOE relaxation data at 600 MHz, type:

```
relax> relax_data.display('NOE', '600')
```

10.2.117 relax_data.read()

Synopsis

Function for reading R_1, R_2, or NOE relaxation data from a file.

Defaults

relax_data.read(self, ri_label=None, frq_label=None, frq=None, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, data_col=None, error_col=None, sep=None, spin_id=None)

Keyword Arguments

ri_label: The relaxation data type, ie 'R1', 'R2', or 'NOE'.

frq_label: The field strength label.

frq: The spectrometer frequency in Hz.

file: The name of the file containing the relaxation data.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The relaxation data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

The frequency label argument can be anything as long as data collected at the same field strength have the same label.

Examples

The following commands will read the protein NOE relaxation data collected at 600 MHz out of a file called 'noe.600.out' where the residue numbers, residue names, data, errors are in the first, second, third, and forth columns respectively.

```
relax> relax_data.read('NOE', '600',
599.7 * 1e6, 'noe.600.out', res_num_col=1,
res_name_col=2, data_col=3, error_col=4)
```

```
relax> relax_data.read(ri_label='NOE',
frq_label='600', frq=600.0 * 1e6, file=
'noe.600.out', res_num_col=1, res_name_col=2,
data_col=3, error_col=4)
```

The following commands will read the R_2 data out of the file '`r2.out`' where the residue numbers, residue names, data, errors are in the second, third, fifth, and sixth columns respectively. The columns are separated by commas.

```
relax> relax_data.read('R2', '800 MHz', 8.0
* 1e8, 'r2.out', res_num_col=2, res_name_col=
3, data_col=5, error_col=6, sep=',')
```

```
relax> relax_data.read(ri_label='R2',
frq_label='800 MHz', frq=8.0*1e8, file=
'r2.out', res_num_col=2, res_name_col=3,
data_col=5, error_col=6, sep=',')
```

The following commands will read the R_1 data out of the file '`r1.out`' where the columns are separated by the symbol '`%`'

```
relax> relax_data.read('R1', '300', 300.1 *
1e6, 'r1.out', sep='%')
```

10.2.118 relax_data.write()

Synopsis

Function for writing R_1, R_2, or NOE relaxation data to a file.

Defaults

relax_data.write(self, ri_label=None, frq_label=None, file=None, dir=None, force=False)

Keyword Arguments

ri_label: The relaxation data type, ie '`R1`', '`R2`', or '`NOE`'.

frq_label: The field strength label.

file: The name of the file.

dir: The directory name.

force: A flag which if True will cause the file to be over-written.

Description

If no directory name is given, the file will be placed in the current working directory. The '`ri_label`' and '`frq_label`' arguments are required for selecting which relaxation data to write to file.

10.2.119 relax_fit.relax_time()

Synopsis

Function for setting the relaxation time period associated with each spectrum.

Defaults

relax_fit.relax_time(self, time=0.0, spectrum_id=None)

Keyword Arguments

time: The time, in seconds, of the relaxation period.

spectrum_id: The spectrum identification string.

Description

Peak intensities should be loaded before calling this user function via the '`spectrum.read_intensities()`' user function. The intensity values will then be associated with a spectrum identifier. To associate each spectrum identifier with a time point in the relaxation curve prior to optimisation, this user function should be called.

10.2.120 relax_fit.select_model()

Synopsis

Function for the selection of the relaxation curve type.

Defaults

relax_fit.select_model(self, model='exp')

Keyword Arguments

model: The type of relaxation curve to fit.

The preset models

The supported relaxation experiments include the default two parameter exponential fit, selected by setting the '`fit_type`' argument to '`exp`', and the three parameter inversion recovery experiment in which the peak intensity limit is a non-zero value, selected by setting the argument to '`inv`'.

The parameters of these two models are

'`exp`' – [Rx, I0],

'`inv`' – [Rx, I0, Iinf].

10.2.121 reset()

Synopsis

Reset relax.

Defaults

reset(self)

All of the data of the relax data storage object will be erased and hence relax will return to its initial state.

10.2.122 residue.copy()

Synopsis

Function for copying all data associated with a residue.

Defaults

residue.copy(self, pipe_from=None, res_from=None, pipe_to=None, res_to=None)

Keyword Arguments

pipe_from: The data pipe containing the residue from which the data will be copied. This defaults to the current data pipe.

res_from: The residue identifier string of the residue to copy the data from.

pipe_to: The data pipe to copy the data to. This defaults to the current data pipe.

res_to: The residue identifier string of the residue to copy the data to.

Description

This function will copy all the data associated with the identified residue to the new, non-existent residue. The new residue must not already exist.

Examples

To copy the residue data from residue 1 to the new residue 2, type:

```
relax> residue.copy(res_from='':1'', res_to=
'':2'')
```

To copy residue 1 of the molecule '`Old mol`' to residue 5 of the molecule '`New mol`', type:

```
relax> residue.copy(res_from='#Old mol:1',
res_to='#New mol:5')
```

To copy the residue data of residue 1 from the data pipe '`m1`' to '`m2`', assuming the current data pipe is '`m1`', type:

```
relax> residue.copy(res_from='':1'', pipe_to=
'm2'')
```

```
relax> residue.copy(pipe_from='m1',
res_from='':1'', pipe_to='m2', res_to='':1'')
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '`#`' character, the residue id token beginning with the '`:`' character, and the atom or spin system id token beginning with the '`@`' character. Each token can be composed of multiple elements separated by the '`,`' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '`-`' character. Negative numbers are supported. The full id string specification is '`#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]`', where the token elements are '`<mol_name>`', the name of the molecule, '`<res_id>`', the residue identifier which can be a number, name, or range of numbers, '`<atom_id>`', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '`#`' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '`@H*`' will select the protons '`H`', '`H2`', '`H98`'.

10.2.123 residue.create()

Synopsis

Function for creating a new residue.

Defaults

residue.create(self, res_num=None, res_name=None, mol_name=None)

Keyword Arguments

res_num: The residue number.

res_name: The name of the residue.

mol_name: The name of the molecule to add the residue to.

Description

Using this function a new sequence can be generated without using the sequence user functions. However if the sequence already exists, the new residue will be added to the end of the residue list (the residue numbers of this list need not be sequential). The same residue number cannot be used more than once. A corresponding single spin system will be created for this residue. The spin system number and name or additional spin systems can be added later if desired.

Examples

The following sequence of commands will generate the sequence 1 ALA, 2 GLY, 3 LYS:

```
relax> residue.create(1, 'ALA')
relax> residue.create(2, 'GLY')
relax> residue.create(3, 'LYS')
```

10.2.124 residue.delete()

Synopsis

Function for deleting residues.

Defaults

residue.delete(self, res_id=None)

Keyword Arguments

res_id: The residue identifier string.

Description

This function can be used to delete a single or sets of residues. See the identification string documentation below for more information. If spin system/atom ids are included a RelaxError will be raised.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '`#`' character, the residue id token beginning with the '`:`' character, and the atom or spin system id token beginning with the '`@`' character. Each token can be composed of multiple elements separated by the '`,`' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '`-`' character. Negative numbers are supported. The full id string specification is '`#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]`', where the token elements are '`<mol_name>`', the name of the molecule, '`<res_id>`', the residue identifier which can be a number, name, or range of numbers, '`<atom_id>`', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '`#`' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '`@H*`' will select the protons '`H`', '`H2`', '`H98`'.

10.2.125 residue.display()

Synopsis

Function for displaying information about the residue(s).

Defaults

residue.display(self, res_id=None)

Keyword Arguments

res_id: The residue identification string.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '`#`' character, the residue id token beginning with the '`:`'

character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.126 residue.name()

Synopsis

Function for naming residues.

Defaults

residue.name(self, res_id=None, name=None, force=False)

Keyword Arguments

res_id: The residue identification string corresponding to one or more residues.

name: The new name.

force: A flag which if True will cause the residue to be renamed.

Description

This function simply allows residues to be named (or renamed).

Examples

The following sequence of commands will rename the sequence {1 ALA, 2 GLY, 3 LYS} to {1 XXX, 2 XXX, 3 XXX}:

```
relax> residue.name(':1', 'XXX', force=True)
relax> residue.name(':2', 'XXX', force=True)
relax> residue.name(':3', 'XXX', force=True)
```

Alternatively:

```
relax> residue.name(':1,2,3', 'XXX', force=True)
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.127 residue.number()

Synopsis

Function for numbering residues.

Defaults

residue.number(self, res_id=None, number=None, force=False)

Keyword Arguments

res_id: The residue identification string corresponding to a single residue.

number: The new residue number.

force: A flag which if True will cause the residue to be renumbered.

Description

This function simply allows residues to be numbered. The new number cannot correspond to an existing residue.

Examples

The following sequence of commands will renumber the sequence {1 ALA, 2 GLY, 3 LYS} to {101 ALA, 102 GLY, 103 LYS}:

```
relax> residue.number(':1', 101, force=True)
relax> residue.number(':2', 102, force=True)
relax> residue.number(':3', 103, force=True)
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.128 results.display()

Synopsis

Function for displaying the results.

Defaults

results.display(self)

Description

This will print to screen (STDOUT) the results contained within the current data pipe.

10.2.129 results.read()

Synopsis

Function for reading results from a file.

Defaults

results.read(self, file='results', dir=None)

Keyword Arguments

file: The name of the file to read results from.

dir: The directory where the file is located.

Description

To search for the results file in the current working directory, set dir to None.

This function is able to handle uncompressed, bzip2 compressed files, or gzip compressed files automatically. The full file name including extension can be supplied, however, if the file cannot be found, this function will search for the file name with '.bz2' appended followed by the file name with '.gz' appended.

10.2.130 results.write()

Synopsis

Function for writing results to a file.

Defaults

results.write(self, file='results', dir='pipe_name', compress_type=1, force=False)

Keyword Arguments

file: The name of the file to output results to. The default is '`results`'. Optionally this can be a file object, or any object with a write() method.

dir: The directory name.

compress_type: The type of compression to use when creating the file.

force: A flag which if True will cause the results file to be overwritten.

Description

To place the results file in the current working directory, set dir to None. If dir is set to the special name '`pipe_name`', then the results file will be placed into a directory with the same name as the current data pipe.

The default behaviour of this function is to compress the file using bzip2 compression. If the extension '`.bz2`' is not included in the file name, it will be added. The compression can, however, be changed to either no compression or gzip compression. This is controlled by the compress_type argument which can be set to

0 – No compression (no file extension),

1 – bzip2 compression ('`.bz2`' file extension),

2 – gzip compression ('`.gz`' file extension).

The complementary read function will automatically handle the compressed files.

10.2.131 select.all()

Synopsis

Function for selecting all spins.

Defaults

select.all(self)

Examples

To select all spins, simply type:

```
relax> select.all()
```

10.2.132 select.read()

Synopsis

Function for selecting the spins contained in a file.

Defaults

select.read(self, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, sep=None, spin_id=None, boolean='OR', change_all=False)

Keyword Arguments

file: The name of the file containing the list of spins to select.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

boolean: The boolean operator specifying how spins should be selected.

change_all: A flag specifying if all other spins should be changed.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts

at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Empty lines and lines beginning with a hash are ignored.

The '`change_all`' flag argument default is False meaning that all spins currently either selected or deselected will remain that way. Setting the argument to True will cause all spins not specified in the file to be deselected.

Examples

To select all residues listed with residue numbers in the first column of the file '`isolated_peaks`', type one of:

```
relax> select.read('isolated_peaks', res_num_col=1)
```

```
relax> select.read(file='isolated_peaks', res_num_col=1)
```

To select the spins in the second column of the relaxation data file '`r1.600`' while deselecting all other spins, for example type:

```
relax> select.read('r1.600', spin_num_col=2, change_all=True)
```

```
relax> select.read(file='r1.600', spin_num_col=2, change_all=True)
```

Boolean operators

The '`boolean`' keyword argument can be used to change how spin systems are selected. The allowed values are: '`OR`', '`NOR`', '`AND`', '`NAND`', '`XOR`', '`XNOR`'. The following table details how the selections will occur for the different boolean operators.

(see table 10.23)

10.2.133 select.reverse()

Synopsis

Function for the reversal of the spin selection for the given spins.

Defaults

select.reverse(self, spin_id=None)

Keyword Arguments

spin_id: The spin identification string.

Description

By supplying the spin_id argument, a subset of spin can have their selection status reversed.

Examples

To select all currently deselected spins and deselect those which are selected type:

```
relax> select.reverse()
```

10.2.134 select.spin()

Synopsis

Function for selecting specific spins.

Defaults

select.spin(self, spin_id=None, boolean='OR', change_all=False)

Keyword Arguments

spin_id: The spin identification string.

boolean: The boolean operator specifying how spins should be selected.

change_all: A flag specifying if all other spins should be changed.

Description

The '`change_all`' flag argument default is False meaning that all spins currently either selected or deselected will remain that way. Setting the argument to True will cause all spins not specified by '`spin_id`' to be selected.

Examples

To select only glycines and alanines, assuming they have been loaded with the names GLY and ALA, type one of:

```
relax> select.spin(spin_id=':GLY|:ALA')
```

To select residue 5 CYS in addition to the currently selected residues, type one of:

```
relax> select.spin(':5')
```

```
relax> select.spin(':5&:CYS')
```

```
relax> select.spin(spin_id=':5&:CYS')
```

Table 10.23: First table for the select.read() user function.

Spin system	1	2	3	4	5	6	7	8	9
Original selection	0	1	1	1	1	0	1	0	1
New selection	0	1	1	1	1	1	0	0	0
OR	0	1	1	1	1	1	1	0	1
NOR	1	0	0	0	0	0	0	1	0
AND	0	1	1	1	1	0	0	0	0
NAND	1	0	0	0	0	1	1	1	1
XOR	0	0	0	0	0	1	1	0	1
XNOR	1	1	1	1	1	0	0	1	0

10.2.135 sequence.copy()

Synopsis

Copy the molecule, residue, and spin sequence data from one data pipe to another.

Defaults

sequence.copy(self, pipe_from=None, pipe_to=None)

Keyword Arguments

pipe_from: The name of the data pipe to copy the sequence data from.

pipe_to: The name of the data pipe to copy the sequence data to.

Description

This function will copy the sequence data between data pipes. The destination data pipe must not contain any sequence data. If the pipe_from or pipe_to arguments are not supplied, then both will default to the current data pipe (hence giving one argument is essential).

Examples

To copy the sequence from the data pipe 'm1' to the current data pipe, type:

```
relax> sequence.copy('m1')
```

```
relax> sequence.copy(pipe_from='m1')
```

To copy the sequence from the current data pipe to the data pipe 'm9', type:

```
relax> sequence.copy(pipe_to='m9')
```

To copy the sequence from the data pipe 'm1' to 'm2', type:

```
relax> sequence.copy('m1', 'm2')
```

```
relax> sequence.copy(pipe_from='m1',
pipe_to='m2')
```

10.2.136 sequence.display()

Synopsis

Function for displaying sequences of molecules, residues, and/or spins.

Defaults

sequence.display(self, sep=None, mol_name_flag=True, res_num_flag=True, res_name_flag=True, spin_num_flag=True, spin_name_flag=True)

Keyword Arguments

sep: The column separator (the default of None corresponds to white space).

mol_name_flag: A flag whic if True will cause the molecule name column to be shown.

res_num_flag: A flag whic if True will cause the residue number column to be shown.

res_name_flag: A flag whic if True will cause the residue name column to be shown.

spin_num_flag: A flag whic if True will cause the spin number column to be shown.

spin_name_flag: A flag whic if True will cause the spin name column to be shown.

10.2.137 sequence.read()

Synopsis

Read the molecule, residue, and spin sequence from a file.

Defaults

sequence.read(self, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, sep=None, spin_id=None)

Keyword Arguments

file: The name of the file containing the sequence data.

dir: The directory where the file is located.

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Examples

The following commands will read protein backbone 15N sequence data out of a file called '`seq`' where the residue numbers and names are in the first and second columns respectively:

```
relax> sequence.read('seq')
```

```
relax> sequence.read('seq', res_num_col=1,
res_name_col=2)
```

```
relax> sequence.read(file='seq',
res_num_col=1, res_name_col=2, sep=None)
```

The following commands will read the residue sequence out of the file '`noe.out`' which also contains the NOE values:

```
relax> sequence.read('noe.out')
```

```
relax> sequence.read('noe.out', res_num_col=
1, res_name_col=2)
```

```
relax> sequence.read(file='noe.out',
res_num_col=1, res_name_col=2)
```

The following commands will read the sequence out of the file '`noe.600.out`' where the residue numbers are in the second column, the names are in the sixth column and the columns are separated by commas:

```
relax> sequence.read('noe.600.out',
res_num_col=2, res_name_col=6, sep=',')
```

```
relax> sequence.read(file='noe.600.out',
res_num_col=2, res_name_col=6, sep=',')
```

The following commands will read the RNA residues and atoms (including C2, C5, C6, C8, N1, and N3) from the file '`500.NOE`', where the residue number, residue name, spin number, and spin name are in the first to fourth columns respectively:

```
relax> sequence.read('500.NOE', res_num_col=
1, res_name_col=2, spin_num_col=3,
spin_name_col=4)
```

```
relax> sequence.read(file='500.NOE',
res_num_col=1, res_name_col=2, spin_num_col=
3, spin_name_col=4)
```

10.2.138 sequence.write()

Synopsis

Write the molecule, residue, and spin sequence to a file.

Defaults

sequence.write(self, file, dir=None, sep=None, mol_name_flag=False, res_num_flag=False, res_name_flag=False, spin_num_flag=False, spin_name_flag=False, force=False)

Keyword Arguments

file: The name of the file.

dir: The directory name.

sep: The column separator (the default of None corresponds to white space).

mol_name_flag: A flag whic if True will cause the molecule name column to be shown.

res_num_flag: A flag whic if True will cause the residue number column to be shown.

res_name_flag: A flag whic if True will cause the residue name column to be shown.

spin_num_flag: A flag whic if True will cause the spin number column to be shown.

spin_name_flag: A flag whic if True will cause the spin name column to be shown.

force: A flag which if True will cause the file to be overwritten.

Description

If no directory name is given, the file will be placed in the current working directory.

10.2.139 spectrum.baseplane_rmsd()

Synopsis

Set the baseplane RMSD of a given spin in a spectrum for error analysis.

Defaults

spectrum.baseplane_rmsd(self, error=0.0, spectrum_id=None, spin_id=None)

Keyword Arguments

error: The baseplane RMSD error value.

spectrum_id: The spectrum identification string.

spin_id: The spin identification string.

Description

The spectrum_id argument identifies the spectrum associated with the error and must correspond to a previously loaded set of intensities. If the '`spin_id`' argument is left on the default of None, then the error value for all spins will be set to the supplied value.

10.2.140 spectrum.error_analysis()

Synopsis

Function for performing an error analysis for peak intensities.

Defaults

spectrum.error_analysis(self)

Description

This user function must only be called after all peak intensities have been loaded and all other necessary spectral information set. This includes the baseplane RMSD and the number of points used in volume integration, both of which are only used if spectra have not been replicated.

Six different types of error analysis are supported depending on whether peak heights or volumes are supplied, whether noise is determined from replicated spectra or the RMSD of the baseplane noise, and whether all spectra or only a subset have been duplicated. These are:

(see table 10.24)

Peak heights with baseplane noise RMSD

When none of the spectra have been replicated, then the peak height errors are calculated using the RMSD of the baseplane noise, the value of which is set by the spectrum.baseplane_rmsd() user function. This results in a different error per peak per spectrum. The standard deviation error measure for the peak height, sigma_I, is set to the RMSD value.

Peak heights with partially replicated spectra

When spectra are replicated, the variance for a single spin at a single replicated spectra set is calculated by the formula

Table 10.24: First table for the spectrum.error_analysis() user function.

Int type	Noise source	Error scope
Heights	RMSD baseplane	One sigma per peak per spectrum
Heights	Partial duplicate + variance averaging	One sigma for all peaks, all spectra
Heights	All replicated + variance averaging	One sigma per replicated spectra set
Volumes	RMSD baseplane	One sigma per peak per spectrum
Volumes	Partial duplicate + variance averaging	One sigma for all peaks, all spectra
Volumes	All replicated + variance averaging	One sigma per replicated spectra set

```
sigma^2 =  sum({Ii - Iav}^2) / (n - 1) ,
```

where sigma^2 is the variance, sigma is the standard deviation, n is the size of the replicated spectra set with i being the corresponding index, Ii is the peak intensity for spectrum i, and Iav is the mean over all spectra i.e. the sum of all peak intensities divided by n.

As the value of n in the above equation is always very low since normally only a couple of spectra are collected per replicated spectra set, the variance of all spins is averaged for a single replicated spectra set. Although this results in all spins having the same error, the accuracy of the error estimate is significantly improved.

If there are in addition to the replicated spectra loaded peak intensities which only consist of a single spectrum, i.e. not all spectra are replicated, then the variances of replicated replicated spectra sets will be averaged. This will be used for the entire experiment so that there will be only a single error value for all spins and for all spectra.

Peak heights with all spectra replicated

If all spectra are collected in duplicate (triplicate or higher number of spectra are supported), the each replicated spectra set will have its own error estimate. The error for a single peak is calculated as when partially replicated spectra are collected, and these are again averaged to give a single error per replicated spectra set. However as all replicated spectra sets will have their own error estimate, variance averaging across all spectra sets will not be performed.

Peak volumes with baseplane noise RMSD

The method of error analysis when no spectra have been replicated and peak volumes are used is highly dependent on the integration method. Many methods simply sum the number of points within a fixed region, either a box or oval object. The number of points used, N, must be specified by another user function in this class. Then the error is simply given by the sum of variances:

```
sigma_vol^2 = sigma_i^2 * N,
```

where sigma_vol is the standard deviation of the volume, sigma_i is the standard deviation of a single point assumed to be equal to the RMSD of the baseplane noise, and N is the total number of points used in the summation integration method. For a box integration method, this converts to the Nicholson, Kay, Baldisseri, Arango, Young, Bax, and Torchia (1992) Biochemistry, 31: 5253-5263 equation:

```
sigma_vol = sigma_i * sqrt(n*m),
```

where n and m are the dimensions of the box. Note that a number of programs, for example peakint (http://hugin.ethz.ch/wuthrich/software/xeasy/xeasy_m15.html) does not use all points within the box. And if the number N can not be determined, this category of error analysis is not possible.

Also note that non-point summation methods, for example when line shape fitting is used to determine peak volumes, the equations above cannot be used. Hence again this category of error analysis cannot be used. This is the case for one of the three integration methods used by Sparky (http://www.cgl.ucsf.edu/home/sparky/manual/peaks.html#Integration). And if fancy techniques are used, for example as Cara does to deconvolute overlapping peaks (http://www.cara.ethz.ch/Wiki/Integration), this again makes this error analysis impossible.

Peak volumes with partially replicated spectra

When peak volumes are measured by any integration method and a few of the spectra are replicated, then the intensity errors are calculated identically as described in the '`Peak heights with partially replicated spectra`' section above.

Peak volumes with all spectra replicated

With all spectra replicated and again using any integration methodology, the intensity errors can be calculated as described in the '`Peak heights with all spectra replicated`' section above.

10.2.141 spectrum.integration_points()

Synopsis

Set the number of summed points used in volume integration of a given spin in a spectrum.

Defaults

spectrum.integration_points(self, N=None, spectrum_id=None, spin_id=None)

Keyword Arguments

N: The number of points used by the summation volume integration method.

spectrum_id: The spectrum identification string.

spin_id: The spin identification string.

Description

For a complete description of which integration methods and how many points N are used for different integration techniques, please read the spectrum.error_analysis() documentation.

The spectrum_id argument identifies the spectrum associated with the value of N and must correspond to a previously loaded set of intensities. If the '`spin_id`' argument is left on the default of None, then the number of summed points for all spins will be set to the supplied value.

10.2.142 spectrum.read_intensities()

Synopsis

Function for reading peak intensities from a file for NOE calculations.

Defaults

spectrum.read_intensities(self, file=None, dir=None, spectrum_id=None, heteronuc='N', proton='HN', int_col=None, int_method='height', spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, sep=None, spin_id=None, ncproc=None)

Keyword Arguments

file: The name of the file containing the intensity data.

dir: The directory where the file is located.

spectrum_id: The spectrum identification string.

heteronuc: The name of the heteronucleus as specified in the peak intensity file.

proton: The name of the proton as specified in the peak intensity file.

int_col: The column containing the peak intensity data (used by the generic intensity file format).

int_method: The integration method.

spin_id_col: The spin ID string column used by the generic intensity file format (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column used by the generic intensity file format (alternative to the spin_id_col).

res_num_col: The residue number column used by the generic intensity file format (alternative to the spin_id_col).

res_name_col: The residue name column used by the generic intensity file format (alternative to the spin_id_col).

spin_num_col: The spin number column used by the generic intensity file format (alternative to the spin_id_col).

spin_name_col: The spin name column used by the generic intensity file format (alternative to the spin_id_col).

sep: The column separator used by the generic intensity format (the default is white space).

spin_id: The spin ID string used by the generic intensity file format to restrict the loading of data to certain spin subsets.

ncproc: The Bruker specific FID intensity scaling factor.

Description

The peak intensity can either be from peak heights or peak volumes.

The '`spectrum_id`' argument is a label which is subsequently utilised by other user functions. If this identifier matches that of a previously loaded set of intensities, then this indicates a replicated spectrum.

The '`heteronuc`' and '`proton`' arguments should be set respectively to the name of the heteronucleus and proton in the file. Only those lines which match these labels will be used.

The '`int_method`' argument is required for the subsequent error analysis. When peak heights are measured, this argument should be set to '`height`'. Volume integration methods are a bit varied and hence two values are accepted. If the volume integration involves pure point summation, with no deconvolution algorithms or other methods affecting peak heights, then the argument should be set to '`point sum`'. All other volume integration methods, e.g. line shape fitting, the argument should be set to '`other`'.

If a series of intensities extracted from Bruker FID files processed in Topspin or XWinNMR are to be compared, the ncproc parameter may need to be supplied. This is because this FID is stored using integer representation and is scaled using ncproc to avoid numerical truncation artifacts. If two spectra have significantly different maximal intensities, then ncproc will be different for both. The intensity scaling is binary, *i*.e. 2**ncproc. Therefore if spectrum A has an ncproc of 6 and and spectrum B a value of 7, then a reference intensity in B will be double that of A. Internally, relax stores the intensities scaled by 2**ncproc.

File formats

The peak list or intensity file will be automatically determined.

Sparky peak list: The file should be a Sparky peak list saved after typing the command '`lt`'. The default is to assume that columns 0, 1, 2, and 3 (1^{st}, 2^{nd}, 3^{rd}, and 4^{th}) contain the Sparky assignment, w1, w2, and peak intensity data respectively. The frequency data w1 and w2 are ignored while the peak intensity data can either be the peak height or volume displayed by changing the window options. If the peak intensity data is not within column 3, set the argument '`int_col`' to the appropriate value (column numbering starts from 0 rather than 1).

XEasy peak list: The file should be the saved XEasy text window output of the list peak entries command, '`tw`' followed by '`le`'. As the columns are fixed, the peak intensity column is hardwired to number 10 (the 11^{th} column) which contains either the peak height or peak volume data. Because the columns are fixed, the '`int_col`' argument will be ignored.

NMRView: The file should be a NMRView peak list. The default is to use column 16 (which contains peak heights) for peak intensities. To use use peak volumes (or evolumes), int_col must be set to 15.

Generic intensity file: This is a generic format which can be created by scripting to support non-supported peak lists. It should contain in the first few columns enough information to identify the spin. This can include columns for the molecule name, residue number, residue name, spin number, and spin name, with each optional type positioned with the *name_col and *num_col arguments. Alternatively a spin ID string column can be used. The peak intensities can be placed in another column specified by the int_col argument. Intensities from multiple spectra can be placed into different columns, and these can then be specified simultaneously by setting the int_col argument to a list of columns. This list must be matched by setting the spectrum_id argument to list of the same length. If columns are delimited by a character other than whitespace, this can be specified with the sep argument. The spin_id argument can be used to restrict the loading to specific spin subsets.

Examples

To read the reference and saturated spectra peak heights from the Sparky formatted files '`ref.list`' and '`sat.list`', type:

```
relax> spectrum.read_intensities(file='ref.
list', spectrum_id='ref')

relax> spectrum.read_intensities(file='sat.
list', spectrum_id='sat')
```

To read the reference and saturated spectra peak heights from the XEasy formatted files '`ref.text`' and '`sat.text`', type:

```
relax> spectrum.read_intensities(file='ref.
text', spectrum_id='ref')

relax> spectrum.read_intensities(file='sat.
text', spectrum_id='sat')
```

10.2.143 spectrum.replicated()

Synopsis

Function for specifying which spectra are replicates.

Defaults

spectrum.replicated(self, spectrum_ids=None)

Keyword Arguments

spectrum_ids: The list of replicated spectra identification strings.

Description

This user function is used to identify which loaded spectra are replicates of each other. This is very important for error analysis.

Examples

To specify that the NOE spectra labelled '`ref1`', '`ref2`', and '`ref3`' are the same spectrum replicated, type one of:

```
relax> spectrum.replicated(['ref1', 'ref2',
'ref3'])
```

```
relax> spectrum.replicated(spectrum_ids=
['ref1', 'ref2', 'ref3'])
```

To specify that the two R_2 spectra '`ncyc2`' and '`ncyc2b`' are the same time point, type:

```
relax> spectrum.replicated(['ncyc2',
'ncyc2b'])
```

10.2.144 spin.copy()

Synopsis

Function for copying all data associated with a spin.

Defaults

spin.copy(self, pipe_from=None, spin_from=None, pipe_to=None, spin_to=None)

Keyword Arguments

pipe_from: The data pipe containing the spin from which the data will be copied. This defaults to the current data pipe.

spin_from: The spin identifier string of the spin to copy the data from.

pipe_to: The data pipe to copy the data to. This defaults to the current data pipe.

spin_to: The spin identifier string of the spin to copy the data to.

Description

This function will copy all the data associated with the identified spin to the new, non-existent spin. The new spin must not already exist.

Examples

To copy the spin data from spin 1 to the new spin 2, type:

```
relax> spin.copy(spin_from='@1', spin_to=
'@2')
```

To copy spin 1 of the molecule '`Old mol`' to spin 5 of the molecule '`New mol`', type:

```
relax> spin.copy(spin_from='#Old mol@1',
spin_to='#New mol@5')
```

To copy the spin data of spin 1 from the data pipe '`m1`' to '`m2`', assuming the current data pipe is '`m1`', type:

```
relax> spin.copy(spin_from='@1', pipe_to=
'm2')
```

```
relax> spin.copy(pipe_from='m1', spin_from=
'@1', pipe_to='m2', spin_to='@1')
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '`#`' character, the residue id token beginning with the '`:`' character, and the atom or spin system id token beginning with the '`@`' character. Each token can be composed of multiple elements separated by the '`,`' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '`-`' character. Negative numbers are supported. The full id string specification is '`#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]`', where the token elements are '`<mol_name>`', the name of the molecule, '`<res_id>`', the residue identifier which can be a number, name, or range of numbers, '`<atom_id>`', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '`#`' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '`@H*`' will select the protons '`H`', '`H2`', '`H98`'.

10.2.145 spin.create()

Synopsis

Function for creating a new spin.

Defaults

spin.create(self, spin_num=None, spin_name=None, res_num=None, res_name=None, mol_name=None)

Keyword Arguments

spin_num: The spin number.

spin_name: The name of the spin.

res_num: The number of the residue to add the spin to.

res_name: The name of the residue to add the spin to.

mol_name: The name of the molecule to add the spin to.

Description

This function will add a new spin data container to the relax data storage object. The same spin number cannot be used more than once.

Examples

The following sequence of commands will generate the sequence 1 C4, 2 C9, 3 C15:

```
relax> spin.create(1, 'C4')
relax> spin.create(2, 'C9')
relax> spin.create(3, 'C15')
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.146 spin.create_pseudo()

Synopsis

Function for creating a spin system representing a pseudo-atom.

Defaults

spin.create_pseudo(self, spin_name=None, spin_num=None, res_id=None, members=None, averaging='linear')

Keyword Arguments

spin_name: The name of the pseudo-atom spin.

spin_num: The spin number.

res_id: The molecule and residue ID string identifying the position to add the pseudo-spin to.

mol_id: The molecule ID string identifying the molecule to add the pseudo-spin to.

members: A list of the atoms the pseudo-atom is composed of.

averaging: The positional averaging technique.

Description

This function will create a spin data container representing a number of pre-existing spin containers as a pseudo-atom. The optional spin number must not already exist.

Examples

The following will create the pseudo-atom named 'Q9' consisting of the protons '@H16', '@H17', '@H18':

```
relax> spin.create_pseudo('Q9', members=
['@H16', '@H17', '@H18'])
```

10.2.147 spin.delete()

Synopsis

Function for deleting spins.

Defaults

spin.delete(self, spin_id=None)

Keyword Arguments

spin_id: The spin identifier string.

Description

This function can be used to delete a single or sets of spins. See the identification string documentation below for more information.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.148 spin.display()

Synopsis

Function for displaying information about the spin(s).

Defaults

spin.display(self, spin_id=None)

Keyword Arguments

spin_id: The spin identification string.

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.149 spin.name()

Synopsis

Function for naming spins.

Defaults

spin.name(self, spin_id=None, name=None, force=False)

Keyword Arguments

spin_id: The spin identification string corresponding to one or more spins.

name: The new name.

force: A flag which if True will cause the spin to be renamed.

Description

This function simply allows spins to be named (or renamed).

Examples

The following sequence of commands will rename the sequence {1 C1, 2 C2, 3 C3} to {1 C11, 2 C12, 3 C13}:

```
relax> spin.name('@1', 'C11', force=True)
relax> spin.name('@2', 'C12', force=True)
relax> spin.name('@3', 'C13', force=True)
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.150 spin.number()

Synopsis

Function for numbering spins.

Defaults

spin.number(self, spin_id=None, number=None, force=False)

Keyword Arguments

spin_id: The spin identification string corresponding to a single spin.

number: The new spin number.

force: A flag which if True will cause the spin to be renumbered.

Description

This function simply allows spins to be numbered. The new number cannot correspond to an existing spin number.

Examples

The following sequence of commands will renumber the sequence {1 C1, 2 C2, 3 C3} to {-1 C1, -2 C2, -3 C3}:

```
relax> spin.number('@1', -1, force=True)
relax> spin.number('@2', -2, force=True)
relax> spin.number('@3', -3, force=True)
```

Identification string documentation

The identification string is composed of three components: the molecule id token beginning with the '#' character, the residue id token beginning with the ':' character, and the atom or spin system id token beginning with the '@' character. Each token can be composed of multiple elements separated by the ',' character and each individual element can either be a number (which must be an integer, in string format), a name, or a range of numbers separated by the '-' character. Negative numbers are supported. The full id string specification is '#<mol_name> :<res_id>[, <res_id>[, <res_id>, ...]] @<atom_id>[, <atom_id>[, <atom_id>, ...]]', where the token elements are '<mol_name>', the name of the molecule, '<res_id>', the residue identifier which can be a number, name, or range of numbers, '<atom_id>', the atom or spin system identifier which can be a number, name, or range of numbers.

If one of the tokens is left out then all elements will be assumed to match. For example if the string does not contain the '#' character then all molecules will match the string.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

10.2.151 state.load()

Synopsis

Function for loading a saved program state.

Defaults

state.load(self, state=None, dir=None, force=False)

Keyword Arguments

state: The file name, which can be a string or a file descriptor object, of a saved program state.

dir: The name of the directory in which the file is found.

force: A boolean flag which if True will cause the current program state to be overwritten.

Description

This function is able to handle uncompressed, bzip2 compressed files, or gzip compressed files automatically. The full file name including extension can be supplied, however, if the file cannot be found, this function will search for the file name with '`.bz2`' appended followed by the file name with '`.gz`' appended.

Both the XML and pickled saved state formats are supported and automatically determined. For more advanced users, file descriptor objects are also supported. If the force flag is set to True, then the relax data store will be reset prior to the loading of the saved state.

Examples

The following commands will load the state saved in the file '`save`'.

```
relax> state.load('save')
```

```
relax> state.load(state='save')
```

Use one of the following commands to load the state saved in the bzip2 compressed file '`save.bz2`':

```
relax> state.load('save')
```

```
relax> state.load(state='save')
```

```
relax> state.load('save.bz2')
```

```
relax> state.load(state='save.bz2', force=
True)
```

10.2.152 state.save()

Synopsis

Function for saving the program state.

Defaults

state.save(self, state=None, dir=None, compress_type=1, force=False, pickle=False)

Keyword Arguments

state: The file name, which can be a string or a file descriptor object, to save the current program state in.

dir: The name of the directory in which to place the file.

compress_type: The type of compression to use when creating the file.

force: A boolean flag which if set to True will cause the file to be overwritten.

pickle: A flag which if true will cause the state file to be a pickled object rather than the default XML format.

Description

This user function will place the program state - the relax data store - into a file for later reloading or reference. The default format is an XML formatted file, but this can be changed to a Python pickled object through the pickle flag. Note, the pickle format is not human readable and often is not compatible with newer relax versions.

The default behaviour of this function is to compress the file using bzip2 compression. If the extension '`.bz2`' is not included in the file name, it will be added. The compression can, however, be changed to either no compression or gzip compression. This is controlled by the compress_type argument which can be set to

0 − No compression (no file extension).

1 − bzip2 compression ('`.bz2`' file extension).

2 − gzip compression ('`.gz`' file extension).

Examples

The following commands will save the current program state, uncompressed, into the file '`save`':

```
relax> state.save('save', compress_type=0)
```

```
relax> state.save(state='save',
compress_type=0)
```

The following commands will save the current program state into the bzip2 compressed file '`save.bz2`':

```
relax> state.save('save')
```

```
relax> state.save(state='save')
```

```
relax> state.save('save.bz2')
```

```
relax> state.save(state='save.bz2')
```

If the file '`save`' already exists, the following commands will save the current program state by overwriting the file.

```
relax> state.save('save', force=True)
```

```
relax> state.save(state='save', force=True)
```

10.2.153 structure.create_diff-_tensor_pdb()

Synopsis

Create a PDB file to represent the diffusion tensor.

Defaults

structure.create_diff_tensor_pdb(self, scale= 1.7999999999999999e-06, file='tensor.pdb', dir=None, force=False)

Keyword Arguments

scale: Value for scaling the diffusion rates.

file: The name of the PDB file.

dir: The directory where the file is located.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

This function creates a PDB file containing an artificial geometric structure to represent the diffusion tensor. A structure must have previously been read into relax. The diffusion tensor is represented by an ellipsoidal, spheroidal, or spherical geometric object with its origin located at the centre of mass (of the selected residues). This diffusion tensor PDB file can subsequently read into any molecular viewer.

There are four different types of residue within the PDB. The centre of mass of the selected residues is represented as a single carbon atom of the residue '`COM`'. The ellipsoidal geometric shape consists of numerous H atoms of the residue '`TNS`'. The axes of the tensor, when defined, are presented as the residue '`AXS`' and consist of carbon atoms: one at the centre of mass and one at the end of each eigenvector. Finally, if Monte Carlo simulations were run and the diffusion tensor parameters were allowed to vary then there will be multiple '`SIM`' residues, one for each simulation. These are essentially the same as the '`AXS`' residue, representing the axes of the simulated tensors, and they will appear as a distribution.

As the Brownian rotational diffusion tensor is a measure of the rate of rotation about different axes - the larger the geometric object, the faster the diffusion of a molecule. For example the diffusion tensor of a water molecule is much larger than that of a macromolecule.

The effective global correlation time experienced by an XH bond vector, not to be confused with the Lipari and Szabo parameter τ_e, will be approximately proportional to the component of the diffusion tensor parallel to it. The approximation is not exact due to the multiexponential form of the correlation function of Brownian rotational diffusion. If an XH bond vector is parallel to the longest axis of the tensor, it will be unaffected by rotations about that axis, which are the fastest rotations of the molecule, and therefore its effective global correlation time will be maximal.

To set the size of the diffusion tensor within the PDB frame the unit vectors used to generate the geometric object are first multiplied by the diffusion tensor (which has the units of inverse seconds) then by the scaling factor (which has the units of second Åand has the default value of 1.8e-6 s.Angstrom). Therefore the rotational diffusion rate per Åis equal the inverse of the scale value (which defaults to 5.56e5 sˆ-1.Angstromˆ-1). Using the default scaling value for spherical diffusion, the correspondence between global correlation time, $\mathfrak{D}_{iso}$ diffusion rate, and the radius of the sphere for a number of discrete cases will be:

(see table 10.25)

The scaling value has been fixed to facilitate comparisons within or between publications, but can be changed to vary the size of the tensor geometric object if necessary. Reporting the rotational diffusion rate per Åwithin figure legends would be useful.

To create the tensor PDB representation, a number of algorithms are utilised. Firstly the centre of mass is calculated for the selected residues and is represented in the PDB by a C atom. Then the axes of the diffusion are calculated, as unit vectors scaled to the appropriate length (multiplied by the eigenvalue $\mathfrak{D}_x$, $\mathfrak{D}_y$, $\mathfrak{D}_z$, $\mathfrak{D}_\parallel$, $\mathfrak{D}_\perp$, or $\mathfrak{D}_{iso}$ as well as the scale value), and a C atom placed at the position of this vector plus the centre of mass. Finally a uniform distribution of vectors on a sphere is generated using spherical coordinates. By incrementing the polar angle using an arccos distribution, a radial array of vectors representing latitude are created while incrementing the azimuthal angle evenly creates the longitudinal vectors. These unit vectors, which are distributed within the PDB frame and are of 1 Åin length, are first rotated into the diffusion frame using a rotation matrix (the spherical diffusion tensor is not rotated). Then they are multiplied by the diffusion tensor matrix to extend the vector out to the correct length, and finally multiplied by the scale value so that the vectors reasonably superimpose onto the macromolecular structure. The last set of algorithms place all this information into a PDB file. The distribution of vectors are represented by H atoms and are all connected using PDB CONECT records. Each H atom is connected to its two neighbours on the both the longitude and latitude. This creates a geometric PDB object with longitudinal and latitudinal lines.

10.2.154 structure.create_vector-_dist()

Synopsis

Create a PDB file representation of the distribution of XH bond vectors.

Table 10.25: First table for the structure.create_diff_tensor_pdb() user function.

τ_m (ns)	$\mathfrak{D}_{iso}$ (s^-1)	Radius (Å)
1	1.67e8	300
3	5.56e7	100
10	1.67e7	30
30	5.56e6	10

Defaults

structure.create_vector_dist(self, length= 2.0000000000000001e-09, file='XH_dist.pdb', dir=None, symmetry=True, force=False)

Keyword Arguments

length: The length of the vectors in the PDB representation (meters).

file: The name of the PDB file.

dir: The directory to place the file into.

symmetry: A flag which if True will create a second chain with reversed XH bond orientations.

force: A flag which if True will overwrite the file if it already exists.

Description

This function creates a PDB file containing an artificial vectors, the length of which default to the length argument of 20 Å. A structure must have previously been read into relax. The origin of the vector distribution is located at the centre of mass (of the selected residues). This vector distribution PDB file can subsequently be read into any molecular viewer.

Because of the symmetry of the diffusion tensor reversing the orientation of the XH bond vector has no effect. Therefore by setting the symmetry flag two chains '`A`' and '`B`' will be added to the PDB file whereby chain '`B`' is chain '`A`' with the XH bonds reversed.

10.2.155 structure.delete()

Synopsis

Delete all structural information from the current data pipe.

Defaults

structure.delete(self)

Description

This function will delete all the structural information.

Example

Simply type:

```
relax> structure.delete()
```

10.2.156 structure.get_pos()

Synopsis

Extract the atomic positions from the loaded structures for the given spins.

Defaults

structure.get_pos(self, spin_id=None, ave_pos=True)

Keyword Arguments

spin_id: The spin identification string.

ave_pos: A flag specifying if the position of the atom is to be averaged across models.

Description

This function allows the atomic positions of the spins to be extracted from the loaded structures. This is automatically performed by the structure.load_spins() user function, but if the sequence information is generated in

other ways, this user function allows the structural information to be obtained.

If the ave_pos flag is True, the average position of all models will be loaded into the spin container. If False, then the positions from all models will be loaded.

Example

For a model-free backbone amide nitrogen analysis whereby the N spins have already been created, to obtain the backbone N positions from the file '1F3Y.pdb' (which is a single protein), type the following two user functions:

```
relax> structure.read_pdb('1F3Y.pdb')
relax> structure.get_pos(spin_id='@N')
```

10.2.157 structure.load_spins()

Synopsis

Load spins from the structure into the relax data store.

Defaults

structure.load_spins(self, spin_id=None, combine_models=True, ave_pos=True)

Keyword Arguments

spin_id: The spin identification string.

combine_models: A flag which specifies if spins from separate models should be combined.

ave_pos: A flag specifying if the position of the atom is to be averaged across models.

Description

This function allows a sequence to be generated within the relax data store using the atomic information from the structure already associated with this data pipe. The spin_id string is used to select which molecules, which residues, and which atoms will be recognised as spin systems within relax. If spin_id is left as None, then all molecules, residues, and atoms will be placed within the data store.

If the combine_models flag is True, then the spins from only a single structure from the ensemble of models will be taken. If False, then spins will be loaded for each model.

If the ave_pos flag is True, the average position of all models will be loaded into the spin container. If False, then the positions from all models will be loaded.

Example

For a model-free backbone amide nitrogen analysis, to load just the backbone N sequence from the file '1F3Y.pdb' (which is a single protein), type the following two user functions:

```
relax> structure.read_pdb('1F3Y.pdb')
relax> structure.load_spins(spin_id='@N')
```

For an RNA analysis of adenine C8 and C2, guanine C8 and N1, cytidine C5 and C6, and uracil N3, C5, and C6, type the following series of commands (assuming that the PDB file with this atom naming has already been read):

```
relax> structure.load_spins(spin_id=":A@C8")
relax> structure.load_spins(spin_id=":A@C2")
relax> structure.load_spins(spin_id=":G@C8")
relax> structure.load_spins(spin_id=":G@N1")
relax> structure.load_spins(spin_id=":C@C5")
relax> structure.load_spins(spin_id=":C@C6")
relax> structure.load_spins(spin_id=":U@N3")
relax> structure.load_spins(spin_id=":U@C5")
relax> structure.load_spins(spin_id=":U@C6")
```

Alternatively using some Python programming:

```
relax> for id in [":A@C8", ":A@C2", ":G@C8",
":G@N1", ":C@C5", ":C@C6", ":U@N3", ":U@C5",
":U@C6"]:
relax> structure.load_spins(spin_id=id)
```

10.2.158 structure.read_pdb()

Synopsis

The PDB loading function.

Defaults

structure.read_pdb(self, file=None, dir=None, read_mol=None, set_mol_name=None, read_model=None, set_model_num=None, parser='internal')

Keyword Arguments

file: The name of the PDB file.

dir: The directory where the file is located.

read_mol: If set, only the given molecule(s) will be read.

set_mol_name: Set the names of the read molecules.

read_model: If set, only the given model number(s) from the PDB file will be read.

set_model_num: Set the model numbers of the read molecules.

parser: The PDB parser used to read the file.

Description

The reading of PDB files into relax is quite a flexible procedure allowing for both models, defined as an ensemble of the same molecule but with different atomic positions, and different molecules within the same model. One of more molecules can exist in one or more models. The flexibility allows PDB models to be converted into different molecules and different PDB files loaded as the same molecule but as different models. This flexibility is controlled by the four keyword arguments '`read_mol`', '`set_mol_name`', '`read_model`', and '`set_model_num`'.

A few different PDB parsers can be used to read the structural data. The choice of which to use depends on whether your PDB file is supported by that reader. These are selected by setting the '`parser`' argument to one of:

'`scientific`' - the Scientific Python PDB parser.

'`internal`' - a lower quality and less reliable, although faster, PDB parser built into

relax.

In a PDB file, the models are specified by the MODEL PDB record. All the supported PDB readers in relax recognise this. The molecule level is quite different between the Scientific Python and internal readers. For how Scientific Python defines molecules, please see its documentation. The internal reader is far simpler as it defines molecules using the TER PDB record. In both cases, the molecules will be numbered consecutively from 1.

The '`set_mol_name`' argument is used to name the molecules within the PDB (within one model). If not set, then the molecules will be named after the file name, with the molecule number appended if more than one exists.

Note that relax will complain if it cannot work out what to do.

Examples

To load all structures from the PDB file '`test.pdb`' in the directory '`~/pdb`', including all models and all molecules, type one of:

```
relax> structure.read_pdb('test.pdb',
'~/pdb')
```

```
relax> structure.read_pdb(file='test.pdb',
dir='pdb')
```

To load the 10^{th} model from the file '`test.pdb`' using the Scientific Python PDB parser and naming it '`CaM`', use one of:

```
relax> structure.read_pdb('test.pdb',
read_model=10, set_mol_name='CaM', parser=
'scientific')
```

```
relax> structure.read_pdb(file='test.pdb',
read_model=10, set_mol_name='CaM', parser=
'scientific')
```

To load models 1 and 5 from the file '`test.pdb`' as two different structures of the same model, type one of:

```
relax> structure.read_pdb('test.pdb',
read_model=[1, 5], set_model_num=[1, 1])
```

```
relax> structure.read_pdb('test.pdb',
set_mol_name=['CaM_1', 'CaM_2'], read_model=
[1, 5], set_model_num=[1, 1])
```

To load the files '`lactose_MCMM4_S1_1.pdb`', '`lactose_MCMM4_S1_2.pdb`', '`lactose_MCMM4_S1_3.pdb`' and '`lactose_MCMM4_S1_4.pdb`' as models, type the following sequence of commands:

```
relax> structure.read_pdb(
'lactose_MCMM4_S1_1.pdb', set_mol_name=
'lactose_MCMM4_S1', set_model_num=1)
```

```
relax> structure.read_pdb(
'lactose_MCMM4_S1_2.pdb', set_mol_name=
'lactose_MCMM4_S1', set_model_num=2)
```

```
relax> structure.read_pdb(
'lactose_MCMM4_S1_3.pdb', set_mol_name=
'lactose_MCMM4_S1', set_model_num=3)
```

```
relax> structure.read_pdb(
'lactose_MCMM4_S1_4.pdb', set_mol_name=
'lactose_MCMM4_S1', set_model_num=4)
```

10.2.159 structure.vectors()

Synopsis

Extract and store the bond vectors from the loaded structures in the spin container.

Defaults

structure.vectors(self, attached='H', spin_id=None, model=None, verbosity=1, ave=True, unit=True)

Keyword arguments

attached: The name of the second atom which attached to the spin of interest. Regular expression is allowed, for example '`H*`'.

spin_id: The spin identification string.

model: The model to extract bond vectors from (which if set to None will cause the vectors of all models to be extracted).

verbosity: The amount of information to print to screen. Zero corresponds to minimal output while higher values increase the amount of output. The default value is 1.

ave: A flag which if True will cause the bond vectors from all models to be averaged. If vectors from only one model is extracted, this argument will have no effect.

unit: A flag which if True will cause the unit vector to calculated rather than the full length bond vector.

Description

For a number of types of analysis, bond vectors or unit bond vectors are required for the calculations. This user function allows these vectors to be extracted from the loaded structures. The bond vector will be that from the atom associated with the spin system loaded in relax to the bonded atom specified by the '`attached`' argument. For example if '`attached`' is set to '`H`' and the protein backbone amide spins '`N`' are loaded, the all '`N-H`' vectors will be extracted. But if set to '`CA`', all atoms named '`CA`' in the structures will be searched for and all '`N-Ca`' bond vectors will be extracted.

The extraction of vectors can occur in a number of ways. For example if an NMR structure with N models is loaded or if multiple molecules, from any source, of the same compound are loaded as different models, there are three options for extracting the bond vector. Firstly the bond vector of a single model can be extracted by setting the '`model`' argument. Secondly the bond vectors from all models can be extracted if '`model`' is None and '`ave`' is set to False. Thirdly, if '`model`' is None and '`ave`' is set to True, then a single vector which is the average for all models will be calculated.

Example

To extract the XH vectors of the backbone amide nitrogens where in the PDB file the backbone nitrogen is called '`N`' and the attached atom is called '`H`', assuming multiple types of spin have already been loaded, type one of:

```
relax> structure.vectors(spin_id='@N')

relax> structure.vectors('H', spin_id='@N')

relax> structure.vectors(attached='H',
spin_id='@N')
```

If the attached atom is called '`HN`', type:

```
relax> structure.vectors(attached='HN',
spin_id='@N')
```

For the '`CA`' spin bonded to the '`HA`' proton, type:

```
relax> structure.vectors(attached='HA',
spin_id='@CA')
```

If you are working with RNA, you can use the residue name identifier to calculate the vectors for each residue separately. For example to calculate the vectors for all possible spins in the bases, type:

```
relax> structure.vectors('H2', spin_id=':A')

relax> structure.vectors('H8', spin_id=':A')

relax> structure.vectors('H1', spin_id=':G')

relax> structure.vectors('H8', spin_id=':G')

relax> structure.vectors('H5', spin_id=':C')

relax> structure.vectors('H6', spin_id=':C')

relax> structure.vectors('H3', spin_id=':U')

relax> structure.vectors('H5', spin_id=':U')

relax> structure.vectors('H6', spin_id=':U')
```

Alternatively, assuming the desired spins have been loaded, regular expression can be used:

```
relax> structure.vectors('H*')
```

10.2.160 structure.write_pdb()

Synopsis

The PDB writing function.

Defaults

structure.write_pdb(self, file=None, dir=None, model_num=None, force=False)

Keyword Arguments

file: The name of the PDB file.

dir: The directory where the file is located.

model_num: The optional model to place in the PDB file.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

If the model_num argument is None, then all models will be written to a single file.

Example

To write all models and molecules to the PDB file '`ensemble.pdb`' within the directory '`~/pdb`', type one of:

```
relax> structure.write_pdb('ensemble.pdb',
'~/pdb')

relax> structure.write_pdb(file='ensemble.
pdb', dir='pdb')
```

To write model number 3 into the new file 'test.pdb', use one of:

```
relax> structure.write_pdb('test.pdb', model_num=3)
```

```
relax> structure.write_pdb(file='test.pdb', model_num=3)
```

10.2.161 system()

Synopsis

Function which executes the user supplied shell command.

Defaults

system(command)

10.2.162 temperature()

Synopsis

Specify the temperature of an experiment.

Defaults

temperature(self, id=None, temp=None)

Keyword arguments

id: The experiment identification string.

temp: The temperature of the experiment.

Description

This function allows the temperature of an experiment to be set. In certain analyses, for example those which use pseudocontact shift data, knowledge of the temperature is essential.

10.2.163 value.copy()

Synopsis

Copy spin specific data values from one data pipe to another.

Defaults

value.copy(self, pipe_from=None, pipe_to=None, param=None)

Keyword Arguments

pipe_from: The name of the pipe to copy from.

pipe_to: The name of the pipe to copy to.

param: The parameter to copy.

Description

Only one parameter may be selected, therefore the 'param' argument should be a string.

If this function is used to change values of previously minimised parameters, then the minimisation statistics (chi-squared value, iteration count, function count, gradient count, and Hessian count) will be reset.

Examples

To copy the CSA values from the data pipe 'm1' to 'm2', type:

```
relax> value.copy('m1', 'm2', 'CSA')
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Model-free set details

Setting a parameter value may have no effect depending on which model-free model is chosen, for example if S_f^2 values and S_s^2 values are set but the run corresponds to model-free model 'm4' then, because these data values are not parameters of the model, they will have no effect.

Note that the R_{ex} values are scaled quadratically with field strength and should be supplied as a field strength independent value. Use the following formula to get the correct value:

value = R_{ex} / (2.0 * π * frequency) ** 2

where:

R_{ex} is the chemical exchange value for the current frequency.

π is in the namespace of relax, ie just type 'pi'.

frequency is the proton frequency corresponding to the data.

Model-free data type string matching patterns

(see table 10.26)

Reduced spectral density mapping set details

In reduced spectral density mapping, three values must be set prior to the calculation of spectral density values: the bond length, CSA, and heteronucleus type.

Reduced spectral density mapping data type string matching patterns

(see table 10.27)

Relaxation curve fitting set details

Only three parameters can be set, the relaxation rate (Rx), the initial intensity (I0), and the intensity at infinity (Iinf). Setting the parameter Iinf has no effect if the chosen model is that of the exponential curve which decays to zero.

Relaxation curve fitting data type string matching patterns

(see table 10.28)

N-state model set details

Setting parameters for the N-state model is a little different from the other type of analyses as each state has a set of parameters with the same names as the other states. To set the parameters for a specific state c (ranging from 0 for the first to N-1 for the last, the number c should be added to the end of the parameter name. So the Euler angle γ of the third state is specified using the string 'gamma2'.

N-state model data type string matching patterns

(see table 10.29)

The objects corresponding to the object names are lists (or arrays) with each element corrsponding to each state.

10.2.164 value.display()

Synopsis

Display spin specific data values.

Defaults

value.display(self, param=None)

Keyword Arguments

param: The parameter to display.

Description

Only one parameter may be selected, therefore the 'param' argument should be a string.

Examples

To show all CSA values, type:

```
relax> value.display('CSA')
```

Table 10.26: First table for the value.copy() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.27: Second table for the value.copy() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.28: Third table for the value.copy() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.29: Fourth table for the value.copy() user function.

Data type	Object name	Patterns
Probabilities	'probs'	'p0', 'p1', 'p2', ..., 'pN'
Euler angle α	'alpha'	'alpha0', 'alpha1', ...
Euler angle β	'beta'	'beta0', 'beta1', ...
Euler angle γ	'gamma'	'gamma0', 'gamma1', ...
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Model-free data type string matching patterns

(see table 10.30)

Reduced spectral density mapping data type string matching patterns

(see table 10.31)

Relaxation curve fitting data type string matching patterns

(see table 10.32)

N-state model data type string matching patterns

(see table 10.33)

The objects corresponding to the object names are lists (or arrays) with each element corrsponding to each state.

10.2.165 value.read()

Synopsis

Read spin specific data values from a file.

Defaults

value.read(self, param=None, scaling=1.0, file=None, dir=None, spin_id_col=None, mol_name_col=None, res_num_col=None, res_name_col=None, spin_num_col=None, spin_name_col=None, data_col=None, error_col=None, sep=None, spin_id=None)

Keyword Arguments

param: The parameter.

scaling: The factor to scale parameters by.

file: The name of the file containing the values.

dir: The directory where the file is located.

Table 10.30: First table for the value.display() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.31: Second table for the value.display() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.32: Third table for the value.display() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.33: Fourth table for the value.display() user function.

Data type	Object name	Patterns
Probabilities	'probs'	'p0', 'p1', 'p2', ..., 'pN'
Euler angle α	'alpha'	'alpha0', 'alpha1', ...
Euler angle β	'beta'	'beta0', 'beta1', ...
Euler angle γ	'gamma'	'gamma0', 'gamma1', ...
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

spin_id_col: The spin ID string column (an alternative to the mol, res, and spin name and number columns).

mol_name_col: The molecule name column (alternative to the spin_id_col).

res_num_col: The residue number column (alternative to the spin_id_col).

res_name_col: The residue name column (alternative to the spin_id_col).

spin_num_col: The spin number column (alternative to the spin_id_col).

spin_name_col: The spin name column (alternative to the spin_id_col).

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

spin_id: The spin ID string to restrict the loading of data to certain spin subsets.

Description

The spin system can be identified in the file using two different formats. The first is the spin ID string column which can include the molecule name, the residue name and number, and the spin name and number. Alternatively the mol_name_col, res_num_col, res_name_col, spin_num_col, and/or spin_name_col arguments can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin_id argument can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Only one parameter may be selected, therefore the 'param' argument should be a string.

If this function is used to change values of previously minimised parameters, then the minimisation statistics (chi-squared value, iteration count, function count, gradient count, and Hessian count) will be reset.

Examples

To load 15N CSA values from the file 'csa_values' in the directory 'data', where spins are only identified by residue name and number, type one of the following:

```
relax> value.read('CSA', 'data/csa_value',
spin_id='@N')
```

```
relax> value.read('CSA', 'csa_value', dir=
'data', spin_id='@N')
```

```
relax> value.read(param='CSA', file=
'csa_value', dir='data', res_num_col=1,
res_name_col=2, data_col=3, error_col=4,
spin_id='@N')
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Model-free set details

Setting a parameter value may have no effect depending on which model-free model is chosen, for example if S_f^2 values and S_s^2 values are set but the run corresponds to model-free model 'm4' then, because these data values are not parameters of the model, they will have no effect.

Note that the R_{ex} values are scaled quadratically with field strength and should be supplied as a field strength independent value. Use the following formula to get the correct value:

value = R_{ex} / (2.0 * π * frequency) ** 2

where:

R_{ex} is the chemical exchange value for the current frequency.

π is in the namespace of relax, ie just type 'pi'.

frequency is the proton frequency corresponding to the data.

Model-free data type string matching patterns

(see table 10.34)

Reduced spectral density mapping set details

In reduced spectral density mapping, three values must be set prior to the calculation of spectral density values: the bond length, CSA, and heteronucleus type.

Reduced spectral density mapping data type string matching patterns

(see table 10.35)

Relaxation curve fitting set details

Only three parameters can be set, the relaxation rate (Rx), the initial intensity (I0), and the intensity at infinity (Iinf). Setting the parameter Iinf has no effect if the chosen model is that of the exponential curve which decays to zero.

Relaxation curve fitting data type string matching patterns

(see table 10.36)

N-state model set details

Setting parameters for the N-state model is a little different from the other type of analyses as each state has a set of parameters with the same names as the other states. To set the parameters for a specific state c (ranging from 0 for the first to N-1 for the last, the number c should be added to the end of the parameter name. So the Euler angle γ of the third state is specified using the string 'gamma2'.

N-state model data type string matching patterns

(see table 10.37)

The objects corresponding to the object names are lists (or arrays) with each element corrsponding to each state.

10.2.166 value.set()

Synopsis

Set spin specific data values.

Defaults

value.set(self, val=None, param=None, spin_id=None)

Keyword arguments

val: The value(s).

param: The parameter(s).

spin_id: The spin identifier.

Description

If this function is used to change values of previously minimised results, then the minimisation statistics (chi-squared value, iteration count, function count, gradient count, and Hessian count) will be reset to None.

The val argument can be None, a single value, or an array of values while the parameter argument can be None, a string, or array of strings. The choice of which combination determines the behaviour of this function. The

Table 10.34: First table for the value.read() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.35: Second table for the value.read() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.36: Third table for the value.read() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.37: Fourth table for the value.read() user function.

Data type	Object name	Patterns
Probabilities	'probs'	'p0', 'p1', 'p2', ..., 'pN'
Euler angle α	'alpha'	'alpha0', 'alpha1', ...
Euler angle β	'beta'	'beta0', 'beta1', ...
Euler angle γ	'gamma'	'gamma0', 'gamma1', ...
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

following table describes what occurs in each instance. The Value column refers to the 'val' argument while the Param column refers to the 'param' argument. In these columns, 'None' corresponds to None, '1' corresponds to either a single value or single string, and 'n' corresponds to either an array of values or an array of strings.

(see table 10.38)

Spin identification

If the 'spin_id' argument is left as the default of None, then the function will be applied to all spins. If the data is global non-spin specific data, such as diffusion tensor parameters, supplying the spin identifier will terminate the program with an error.

Examples

To set the parameter values for the current data pipe to the default values, for all spins, type:

```
relax> value.set()
```

To set the parameter values of residue 10, which is in the current model-free data pipe 'm4' and has the parameters $\{S^2, \tau_e, R_{ex}\}$, the following can be used. R_{ex} term is the value for the first given field strength.

```
relax> value.set([0.97, 2.048*1e-9, 0.149],
spin_id=':10')
```

```
relax> value.set(val=[0.97, 2.048*1e-9,
0.149], spin_id=':10')
```

To set the CSA value of all spins to the default value, type:

```
relax> value.set(param='csa')
```

To set the CSA value of all spins to -172 ppm, type:

```
relax> value.set(-172 * 1e-6, 'csa')
```

```
relax> value.set(val=-172 * 1e-6, param=
'csa')
```

To set the NH bond length of all spins to 1.02 Å, type:

```
relax> value.set(1.02 * 1e-10,
'bond_length')
```

```
relax> value.set(val=1.02 * 1e-10, param=
'r')
```

To set both the bond length and the CSA value to the default values, type:

```
relax> value.set(param=['bond length',
'csa'])
```

To set both τ_f and τ_s to 100 ps, type:

```
relax> value.set(100e-12, ['tf', 'ts'])
```

```
relax> value.set(val=100e-12, param=['tf',
'ts'])
```

To set the S^2 and τ_e parameter values of residue 126, Ca spins to 0.56 and 13 ps, type:

```
relax> value.set([0.56, 13e-12], ['S2',
'te'], ':126@Ca')
```

```
relax> value.set(val=[0.56, 13e-12], param=
['S2', 'te'], spin_id=':126@Ca')
```

```
relax> value.set(val=[0.56, 13e-12], param=
['S2', 'te'], spin_id=':126@Ca')
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

Table 10.38: First table for the value.set() user function.

Value	Param	Description
None	None	This case is used to set the model parameters prior to minimisation or calculation. The model parameters are set to the default values.
1	None	Invalid combination.
n	None	This case is used to set the model parameters prior to minimisation or calculation. The length of the val array must be equal to the number of model parameters. The parameters will be set to the corresponding number.
None	1	The parameter matching the string will be set to the default value.
1	1	The parameter matching the string will be set to the supplied number.
n	1	Invalid combination.
None	*n*	Each parameter matching the strings will be set to the default values.
1	*n*	Each parameter matching the strings will be set to the supplied number.
n	*n*	Each parameter matching the strings will be set to the corresponding number. Both arrays must be of equal length.

'`^`' – Match the start of the string.

'`$`' – Match the end of the string. For example, '`^[Ss]2$`' will match '`s2`' but not '`S2f`' or '`s2s`'.

'`.`' – Match any character.

'`x*`' – Match the character '`x`' any number of times, for example '`x`' will match, as will '`xxxxx`'

'`.*`' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Model-free set details

Setting a parameter value may have no effect depending on which model-free model is chosen, for example if S_f^2 values and S_s^2 values are set but the run corresponds to model-free model '`m4`' then, because these data values are not parameters of the model, they will have no effect.

Note that the R_{ex} values are scaled quadratically with field strength and should be supplied as a field strength independent value. Use the following formula to get the correct value:

$$\text{value} = R_{ex} \;/\; (2.0 * \pi * \text{frequency}) ** 2$$

where:

R_{ex} is the chemical exchange value for the current frequency.

π is in the namespace of relax, ie just type '`pi`'.

frequency is the proton frequency corresponding to the data.

Model-free data type string matching patterns

(see table 10.39)

Model-free default values

(see table 10.40)

Reduced spectral density mapping set details

In reduced spectral density mapping, three values must be set prior to the calculation of spectral density values: the bond length, CSA, and heteronucleus type.

Reduced spectral density mapping data type string matching patterns

(see table 10.41)

Table 10.39: Second table for the value.set() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.40: Third table for the value.set() user function.

Data type	Object name	Value
Local τ_m	'local_tm'	10 * 1e-9
Order parameters S^2, S_f^2, and S_s^2	's2', 's2f', 's2s'	0.8
Correlation time τ_e	'te'	100 * 1e-12
Correlation time τ_f	'tf'	10 * 1e-12
Correlation time τ_s	'ts'	1000 * 1e-12
Chemical exchange relaxation	'rex'	0.0
Bond length	'r'	1.02 * 1e-10
CSA	'csa'	-172 * 1e-6
Heteronucleus type	'heteronuc_type'	'15N'
Proton type	'proton_type'	'1H'

Table 10.41: Fourth table for the value.set() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.42: Fifth table for the value.set() user function.

Data type	Object name	Value
Bond length	'r'	1.02 * 1e-10
CSA	'csa'	-172 * 1e-6
Heteronucleus type	'heteronuc_type'	'15N'
Proton type	'proton_type'	'1H'

Reduced spectral density mapping default values

(see table 10.42)

Diffusion tensor set details

If the diffusion tensor has not been setup, use the more powerful function 'diffusion_tensor.init' to initialise the tensor parameters. This function cannot be used to initialise a diffusion tensor.

The units of the parameters are:

Inverse seconds for τ_m.

Seconds for $\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, $\mathfrak{D}_x$, $\mathfrak{D}_y$, $\mathfrak{D}_z$, $\mathfrak{D}_\parallel$, $\mathfrak{D}_\perp$.

Unitless for $\mathfrak{D}_{ratio}$ and $\mathfrak{D}_r$.

Radians for all angles (α, β, γ, θ, ϕ).

When setting a diffusion tensor parameter, the residue number has no effect. As the internal parameters of spherical diffusion are {tm}, spheroidal diffusion are {τ_m, $\mathfrak{D}_a$, θ, ϕ}, and ellipsoidal diffusion are {τ_m, $\mathfrak{D}_a$, $\mathfrak{D}_r$, α, β, γ}, supplying geometric parameters must be done in the following way. If a single geometric parameter is supplied, it must be one of τ_m, $\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, $\mathfrak{D}_r$, or $\mathfrak{D}_{ratio}$. For the parameters $\mathfrak{D}_\parallel$, $\mathfrak{D}_\perp$, $\mathfrak{D}_x$, $\mathfrak{D}_y$, and $\mathfrak{D}_x$, it is not possible to determine how to use the currently set values together with the supplied value to calculate the new internal parameters. For spheroidal diffusion, when supplying multiple geometric parameters, the set must belong to one of

{τ_m, $\mathfrak{D}_a$},

{$\mathfrak{D}_{iso}$, $\mathfrak{D}_a$},

{τ_m, $\mathfrak{D}_{ratio}$},

{$\mathfrak{D}_\parallel$, $\mathfrak{D}_\perp$},

{$\mathfrak{D}_{iso}$, $\mathfrak{D}_{ratio}$},

where either θ, ϕ, or both orientational parameters can be additionally supplied. For ellipsoidal diffusion, again when supplying multiple geometric parameters, the set must belong to one of

{τ_m, $\mathfrak{D}_a$, $\mathfrak{D}_r$},

{$\mathfrak{D}_{iso}$, $\mathfrak{D}_a$, $\mathfrak{D}_r$},

$\{\mathfrak{D}_x, \mathfrak{D}_y, \mathfrak{D}_z\}$,

where any number of the orientational parameters, α, β, or γ can be additionally supplied.

Diffusion tensor parameter string matching patterns

(see table 10.43)

Diffusion tensor parameter default values

(see table 10.44)

Relaxation curve fitting set details

Only three parameters can be set, the relaxation rate (Rx), the initial intensity (I0), and the intensity at infinity (Iinf). Setting the parameter Iinf has no effect if the chosen model is that of the exponential curve which decays to zero.

Relaxation curve fitting data type string matching patterns

(see table 10.45)

Relaxation curve fitting default values

These values are completely arbitrary as peak heights (or volumes) are extremely variable and the Rx value is a compensation for both the R_1 and R_2 values.

(see table 10.46)

N-state model set details

Setting parameters for the N-state model is a little different from the other type of analyses as each state has a set of parameters with the same names as the other states. To set the parameters for a specific state c (ranging from 0 for the first to N-1 for the last, the number c should be added to the end of the parameter name. So the Euler angle γ of the third state is specified using the string '`gamma2`'.

N-state model data type string matching patterns

(see table 10.47)

The objects corresponding to the object names are lists (or arrays) with each element corrsponding to each state.

N-state model default values

(see table 10.48)

In this table, N is the total number of states and c is the index of a given state ranging from 0 to N-1. The default probabilities are all set to be equal whereas the angles are given a range of values so that no 2 states are equal at the start of optimisation.

Note that setting the probability for state N will do nothing as it is equal to one minus all the other probabilities.

10.2.167 value.write()

Synopsis

Write spin specific data values to a file.

Defaults

value.write(self, param=None, file=None, dir=None, force=False)

Keyword Arguments

param: The parameter.

file: The name of the file.

dir: The directory name.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

If no directory name is given, the file will be placed in the current working directory.

The parameter argument should be a string.

Table 10.43: Sixth table for the value.set() user function.

Data type	Object name	Patterns
Global correlation time - τ_m	'tm'	'^tm$'
Isotropic component of the diffusion tensor - $\mathfrak{D}_{iso}$	'Diso'	'[Dd]iso'
Anisotropic component of the diffusion tensor - $\mathfrak{D}_a$	'Da'	'[Dd]a'
Rhombic component of the diffusion tensor - $\mathfrak{D}_r$	'Dr'	'[Dd]r$'
Eigenvalue associated with the x-axis of the diffusion diffusion tensor - $\mathfrak{D}_x$	'Dx'	'[Dd]x'
Eigenvalue associated with the y-axis of the diffusion diffusion tensor - $\mathfrak{D}_y$	'Dy'	'[Dd]y'
Eigenvalue associated with the z-axis of the diffusion diffusion tensor - $\mathfrak{D}_z$	'Dz'	'[Dd]z'
Diffusion coefficient parallel to the major axis of the spheroid diffusion tensor - $\mathfrak{D}_{\parallel}$	'Dpar'	'[Dd]par'
Diffusion coefficient perpendicular to the major axis of the spheroid diffusion tensor - $\mathfrak{D}_{\perp}$	'Dper'	'[Dd]per'
Ratio of the parallel and perpendicular components of the spheroid diffusion tensor - $\mathfrak{D}_{ratio}$	'Dratio'	'[Dd]ratio'
The first Euler angle of the ellipsoid diffusion tensor - α	'alpha'	'^a$' or 'alpha'
The second Euler angle of the ellipsoid diffusion tensor - β	'beta'	'^b$' or 'beta'
The third Euler angle of the ellipsoid diffusion tensor - γ	'gamma'	'^g$' or 'gamma'
The polar angle defining the major axis of the spheroid diffusion tensor - θ	'theta'	'theta'
The azimuthal angle defining the major axis of the spheroid diffusion tensor - ϕ	'phi'	'phi'

Table 10.44: Seventh table for the value.set() user function.

Data type	Object name	Value
τ_m	'tm'	10 * 1e-9
$\mathfrak{D}_{iso}$	'Diso'	1.666 * 1e7
$\mathfrak{D}_a$	'Da'	0.0
$\mathfrak{D}_r$	'Dr'	0.0
$\mathfrak{D}_x$	'Dx'	1.666 * 1e7
$\mathfrak{D}_y$	'Dy'	1.666 * 1e7
$\mathfrak{D}_z$	'Dz'	1.666 * 1e7
$\mathfrak{D}_\parallel$	'Dpar'	1.666 * 1e7
$\mathfrak{D}_\perp$	'Dper'	1.666 * 1e7
$\mathfrak{D}_{ratio}$	'Dratio'	1.0
α	'alpha'	0.0
β	'beta'	0.0
γ	'gamma'	0.0
θ	'theta'	0.0
ϕ	'phi'	0.0

Table 10.45: Eighth table for the value.set() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.46: Ninth table for the value.set() user function.

Data type	Object name	Value
Relaxation rate	'rx'	8.0
Initial intensity	'i0'	10000.0
Intensity at infinity	'iinf'	0.0

Table 10.47: Tenth table for the value.set() user function.

Data type	Object name	Patterns
Probabilities	'probs'	'p0', 'p1', 'p2', ..., 'pN'
Euler angle α	'alpha'	'alpha0', 'alpha1', ...
Euler angle β	'beta'	'beta0', 'beta1', ...
Euler angle γ	'gamma'	'gamma0', 'gamma1', ...
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.48: Eleventh table for the value.set() user function.

Data type	Object name	Value
Probabilities	'p0', 'p1', 'p2', ..., 'pN'	1/N
Euler angle α	'alpha0', 'alpha1', ...	(c+1) * π / (N+1)
Euler angle β	'beta0', 'beta1', ...	(c+1) * π / (N+1)
Euler angle γ	'gamma0', 'gamma1', ...	(c+1) * π / (N+1)

Examples

To write the CSA values to the file 'csa.txt', type one of:

```
relax> value.write('CSA', 'csa.txt')
```

```
relax> value.write(param='CSA', file='csa.
txt')
```

To write the NOE values to the file 'noe', type one of:

```
relax> value.write('noe', 'noe.out')
```

```
relax> value.write(param='noe', file='noe.
out')
```

```
relax> value.write(param='noe', file='noe.
out')
```

```
relax> value.write(param='noe', file='noe.
out', force=True)
```

Regular expression

The python function 'match', which uses regular expression, is used to determine which data type to set values to, therefore various data_type strings can be used to select the same data type. Patterns used for matching for specific data types are listed below.

This is a short description of python regular expression, for more information see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

'[]' – A sequence or set of characters to match to a single character. For example, '[Ss]2' will match both 'S2' and 's2'.

'^' – Match the start of the string.

'$' – Match the end of the string. For example, '^[Ss]2$' will match 's2' but not 'S2f' or 's2s'.

'.' – Match any character.

'x*' – Match the character 'x' any number of times, for example 'x' will match, as will 'xxxxx'

'.*' – Match any sequence of characters of any length.

Importantly, do not supply a string for the data type containing regular expression. The regular expression is implemented so that various strings can be supplied which all match the same data type.

Model-free data type string matching patterns

(see table 10.49)

Table 10.49: First table for the value.write() user function.

Data type	Object name	Patterns
Local τ_m	'local_tm'	'[Ll]ocal[-_]tm'
Order parameter S^2	's2'	'^[Ss]2$'
Order parameter S_f^2	's2f'	'^[Ss]2f$'
Order parameter S_s^2	's2s'	'^[Ss]2s$'
Correlation time τ_e	'te'	'^te$'
Correlation time τ_f	'tf'	'^tf$'
Correlation time τ_s	'ts'	'^ts$'
Chemical exchange	'rex'	'^[Rr]ex$' or '[Cc]emical[-_][Ee]xchange'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Table 10.50: Second table for the value.write() user function.

Data type	Object name	Patterns
$J(0)$	'j0'	'^[Jj]0$' or '[Jj]\(0\)'
$J(\omega_X)$	'jwx'	'^[Jj]w[Xx]$' or '[Jj]\(w[Xx]\)'
$J(\omega_H)$	'jwh'	'^[Jj]w[Hh]$' or '[Jj]\(w[Hh]\)'
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
CSA	'csa'	'^[Cc][Ss][Aa]$'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Reduced spectral density mapping data type string matching patterns

(see table 10.50)

NOE calculation data type string matching patterns

(see table 10.51)

Relaxation curve fitting data type string matching patterns

(see table 10.52)

N-state model data type string matching patterns

(see table 10.53)

The objects corresponding to the object names are lists (or arrays) with each element corrsponding to each state.

10.2.168 vmd.view()

Synopsis

Function for viewing the collection of molecules extracted from the PDB file.

Defaults

vmd.view(self)

Example

```
relax> vmd.view()
```

Table 10.51: Third table for the value.write() user function.

Data type	Object name	Patterns
Reference intensity	'ref'	'^[Rr]ef$' or '[Rr]ef[-_][Ii]nt'
Saturated intensity	'sat'	'^[Ss]at$' or '[Ss]at[-_][Ii]nt'
NOE	'noe'	'^[Nn][Oo][Ee]$'

Table 10.52: Fourth table for the value.write() user function.

Data type	Object name	Patterns
Relaxation rate	'rx'	'^[Rr]x$'
Peak intensities (series)	'intensities'	'^[Ii]nt$'
Initial intensity	'i0'	'^[Ii]0$'
Intensity at infinity	'iinf'	'^[Ii]inf$'
Relaxation period times (series)	'relax_times'	'^[Rr]elax[-_][Tt]imes$'

Table 10.53: Fifth table for the value.write() user function.

Data type	Object name	Patterns
Probabilities	'probs'	'p0', 'p1', 'p2', ..., 'pN'
Euler angle α	'alpha'	'alpha0', 'alpha1', ...
Euler angle β	'beta'	'beta0', 'beta1', ...
Euler angle γ	'gamma'	'gamma0', 'gamma1', ...
Bond length	'r'	'^r$' or '[Bb]ond[-_][Ll]ength'
Heteronucleus type	'heteronuc_type'	'^[Hh]eteronucleus$'
Proton type	'proton_type'	'^[Pp]roton$'

Chapter 11

Licence

11.1 Copying, modification, sublicencing, and distribution of relax

To ensure that the program relax, including all future versions, will remain legally available for perpetuity to anyone who wishes to use the program the code has been released under the GNU General Public Licence. The freedom of relax is guaranteed by the GPL. This is a licence which has been very carefully crafted to protect both the developers of the program as well as the users by means of copyright law. If the licence is violated by improper copying, modification, sublicencing, or distribution then the licence terminates – hence the violator is copying, modifying, sublicencing, or distributing the program illegally in full violation of copyright law. For a better understanding of the protections afforded by the GPL the licence is reprinted in whole within the next section.

11.2 The GPL

The following is a verbatim copy of the GNU General Public Licence. A text version is available in the relax 'docs' directory within the file 'COPYING'.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those

sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published

by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
```

```
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘`show w`’ and ‘`show c`’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

Abragam, A. (1961). *The Principles of Nuclear Magnetism*. Clarendon Press, Oxford.

Bloembergen, N., Purcell, E. M., and Pound, R. V. (1948). Relaxation effects in nuclear magnetic resonance absorption. *Phys. Rev.*, **73**(7), 679–712.

Broyden, C. G. (1970). The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *J. Inst. Maths. Applics.*, **6**(1), 76–90.

Chen, J., Brooks, 3rd, C. L., and Wright, P. E. (2004). Model-free analysis of protein dynamics: assessment of accuracy and model selection protocols based on molecular dynamics simulation. *J. Biomol. NMR*, **29**(3), 243–257.

Clore, G. M., Szabo, A., Bax, A., Kay, L. E., Driscoll, P. C., and Gronenborn, A. M. (1990). Deviations from the simple 2-parameter model-free approach to the interpretation of N-15 nuclear magnetic-relaxation of proteins. *J. Am. Chem. Soc.*, **112**(12), 4989–4991.

d'Auvergne, E. J. (2006). *Protein dynamics: a study of the model-free analysis of NMR relaxation data*. PhD thesis, Biochemistry and Molecular Biology, University of Melbourne. http://eprints.infodiv.unimelb.edu.au/archive/00002799/.

d'Auvergne, E. J. and Gooley, P. R. (2003). The use of model selection in the model-free analysis of protein dynamics. *J. Biomol. NMR*, **25**(1), 25–39.

d'Auvergne, E. J. and Gooley, P. R. (2006). Model-free model elimination: A new step in the model-free dynamic analysis of NMR relaxation data. *J. Biomol. NMR*, **35**(2), 117–135.

d'Auvergne, E. J. and Gooley, P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. **3**(7), 483–494.

d'Auvergne, E. J. and Gooley, P. R. (2008a). Optimisation of NMR dynamic models I. Minimisation algorithms and their performance within the model-free and Brownian rotational diffusion spaces. *J. Biomol. NMR*, **40**(2), 107–119.

d'Auvergne, E. J. and Gooley, P. R. (2008b). Optimisation of NMR dynamic models II. A new methodology for the dual optimisation of the model-free parameters and the Brownian rotational diffusion tensor. *J. Biomol. NMR*, **40**(2), 121–133.

Farrow, N. A., Zhang, O. W., Szabo, A., Torchia, D. A., and Kay, L. E. (1995). Spectral density-function mapping using N-15 relaxation data exclusively. *J. Biomol. NMR*, **6**(2), 153–162.

Fletcher, R. (1970). A new approach to variable metric algorithms. **13**(3), 317–322.

Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients. **7**(2), 149–154.

Fushman, D., Cahill, S., and Cowburn, D. (1997). The main-chain dynamics of the dynamin pleckstrin homology (PH) domain in solution: analysis of 15N relaxation with monomer/dimer equilibration. *J. Mol. Biol.*, **266**(1), 173–194.

Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical Optimization*. Academic Press.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Math. Comp.*, **24**(109), 23–26.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Res. Natn. Bur. Stand.*, **49**(6), 409–436.

Horne, J., d'Auvergne, E., Coles, M., Velkov, T., Chin, Y., Charman, W., Prankerd, R., Gooley, P., and Scanlon, M. (2007). Probing the flexibility of the DsbA oxidoreductase from Vibrio cholerae–a 15N - 1H heteronuclear NMR relaxation analysis of oxidized and reduced forms of DsbA. *J. Mol. Biol.*, **371**(3), 703–716.

Korzhnev, D. M., Billeter, M., Arseniev, A. S., and Orekhov, V. Y. (2001). NMR studies of Brownian tumbling and internal motions in proteins. *Prog. NMR Spectrosc.*, **38**(3), 197–266.

Lefevre, J., Dayie, K., Peng, J., and Wagner, G. (1996). Internal mobility in the partially folded DNA binding and dimerization domains of GAL4: NMR analysis of the N-H spectral density functions. *Biochemistry*, **35**(8), 2674–2686.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, **2**, 164–168.

Lipari, G. and Szabo, A. (1982a). Model-free approach to the interpretation of nuclear magnetic-resonance relaxation in macromolecules I. Theory and range of validity. *J. Am. Chem. Soc.*, **104**(17), 4546–4559.

Lipari, G. and Szabo, A. (1982b). Model-free approach to the interpretation of nuclear magnetic-resonance relaxation in macromolecules II. Analysis of experimental results. *J. Am. Chem. Soc.*, **104**(17), 4559–4570.

Mandel, A. M., Akke, M., and Palmer, 3rd, A. G. (1995). Backbone dynamics of *escherichia coli* ribonuclease HI: correlations with structure and function in an active enzyme. *J. Mol. Biol.*, **246**(1), 144–163.

Marquardt, D. W. (1963). An algorithm for least squares estimation of non-linear parameters. *SIAM J.*, **11**, 431–441.

Moré, J. J. and Thuente, D. J. (1994). Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Maths. Softw.*, **20**(3), 286–307.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York.

Orekhov, V. Y., Korzhnev, D. M., Diercks, T., Kessler, H., and Arseniev, A. S. (1999). H-1-N-15 NMR dynamic study of an isolated alpha-helical peptide (1-36)- bacteriorhodopsin reveals the equilibrium helix-coil transitions. *J. Biomol. NMR*, **14**(4), 345–356.

Polak, E. and Ribière, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, **16**, 35–43.

Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, **24**(111), 647–656.

Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, **20**(3), 626–637.

Tugarinov, V., Liang, Z., Shapiro, Y. E., Freed, J. H., and Meirovitch, E. (2001). A structural mode-coupling approach to 15N NMR relaxation in proteins. *J. Am. Chem. Soc.*, **123**(13), 3055–3063.

Zhuravleva, A. V., Korzhnev, D. M., Kupce, E., Arseniev, A. S., Billeter, M., and Orekhov, V. Y. (2004). Gated electron transfers and electron pathways in azurin: a NMR dynamic study at multiple fields and temperatures. *J. Mol. Biol.*, **342**(5), 1599–1611.

Index