

relax

Version 4.1.0



**Molecular dynamics by
NMR data analysis**

February 14, 2019

Copyright © 2001-2019 the relax development team

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License (GPL), Version 3 or any later version published by the Free Software Foundation.

The Oxygen Icons used herein are licensed under the terms of the GNU Lesser General Public License (GPL), Version 3 or any later version published by the Free Software Foundation.

Contents

Preface - citing relax	xxvii
I The basics	1
1 Introduction	3
1.1 Program features	4
1.1.1 Literature	4
1.1.2 Supported NMR theories	4
1.1.3 Data analysis tools	5
1.1.4 Data visualisation	5
1.1.5 Interfacing with other programs	6
1.1.6 The user interfaces (UI)	6
1.2 How to use relax	6
1.2.1 The prompt	6
1.2.2 Python	7
1.2.3 User functions	8
1.2.4 The help system	9
1.2.5 Tab completion	9
1.2.6 The data pipe	10
1.2.7 The spin and interatomic data containers	11
1.2.8 Scripting	12
1.2.9 The test suite	14
1.2.10 The GUI	14
1.2.11 Access to the internals of relax	16
1.3 The multi-processor framework	17
1.3.1 Introduction to the multi-processor	17
1.3.2 Usage of the multi-processor	19
1.3.3 Further details	22
1.4 Usage of the name relax	22
2 Installation instructions	23
2.1 Dependencies	23
2.2 Installation	23
2.2.1 The source releases	23
2.2.2 Installation on GNU/Linux	24
2.2.3 Installation on MS Windows	24
2.2.4 Installation on Mac OS X	25
2.2.5 Installation on your OS	26
2.2.6 Running a non-compiled version	26

2.3	Optional programs	26
2.3.1	Grace	26
2.3.2	OpenDX	26
2.3.3	Molmol	27
2.3.4	PyMOL	27
2.3.5	Dasha	27
2.3.6	Modelfree4	27
3	Free software infrastructure	29
3.1	History	29
3.2	The relax web sites	30
3.3	The mailing lists	30
3.3.1	relax-announce	30
3.3.2	relax-users	31
3.3.3	relax-devel	31
3.3.4	relax-commits	31
3.3.5	Replies to a message	31
3.4	Reporting bugs	31
3.5	Latest sources – the relax repositories	32
3.6	The relax distribution archives	32
4	The relax data model	35
4.1	The concept of the relax data model	35
4.2	The data model	35
4.2.1	The relax data store	35
4.2.2	Molecule, residue, and spin containers	36
4.3	Interatomic data containers	39
4.4	Setup in the prompt/script UI	39
4.4.1	Script mode – spins from structural data	39
4.4.2	Script mode – spins from a sequence file	40
4.4.3	Script mode – manual construction	41
4.5	Setup in the GUI	41
4.5.1	GUI mode – setting up the data pipe	41
4.5.2	GUI mode – spins from structural data	42
4.5.3	GUI mode – spins from a sequence file	45
4.5.4	GUI mode – manual construction	46
4.5.5	GUI mode – deselect spins	46
4.6	The next steps	48
II	The specific analyses	49
5	Relaxation curve-fitting	51
5.1	Introduction to relaxation curve-fitting	51
5.2	The exponential curve models	51
5.3	From spectra to peak intensities for the relaxation rates	52
5.3.1	Temperature control and calibration	52
5.3.2	Spectral processing	53
5.3.3	Measuring peak intensities	55
5.4	Relaxation curve-fitting in the prompt/script UI mode	56

5.4.1	Relax-fit script mode – the sample script	56
5.4.2	Relax-fit script mode – initialisation of the data pipe	58
5.4.3	Relax-fit script mode – setting up the spin systems	58
5.4.4	Relax-fit script mode – loading the data	59
5.4.5	Relax-fit script mode – the rest of the setup	60
5.4.6	Relax-fit script mode – optimisation of exponential curves	61
5.4.7	Relax-fit script mode – error analysis	61
5.4.8	Relax-fit script mode – finishing off	62
5.5	The relaxation curve-fitting auto-analysis in the GUI	63
5.5.1	Relax-fit GUI mode – initialisation of the data pipe	63
5.5.2	Relax-fit GUI mode – general setup	64
5.5.3	Relax-fit GUI mode – setting up the spin systems	65
5.5.4	Relax-fit GUI mode – unresolved spins	65
5.5.5	Relax-fit GUI mode – loading the data	65
5.5.6	Relax-fit GUI mode – optimisation and error analysis	68
5.6	Final checks of the curve-fitting	70
6	Calculating the NOE	71
6.1	Introduction to the steady-state NOE	71
6.2	From spectra to peak intensities for the NOE	71
6.3	Calculation of the NOE in the prompt/script UI mode	72
6.3.1	NOE script mode – the sample script	72
6.3.2	NOE script mode – initialisation of the data pipe	73
6.3.3	NOE script mode – setting up the spin systems	73
6.3.4	NOE script mode – loading the data	73
6.3.5	NOE script mode – setting the errors	74
6.3.6	NOE script mode – unresolved spins	74
6.3.7	NOE script mode – the NOE calculation	75
6.3.8	NOE script mode – viewing the results	75
6.4	The NOE auto-analysis in the GUI	77
6.4.1	NOE GUI mode – initialisation of the data pipe	77
6.4.2	NOE GUI mode – general setup	78
6.4.3	NOE GUI mode – setting up the spin systems	79
6.4.4	NOE GUI mode – unresolved spins	79
6.4.5	NOE GUI mode – loading the data	79
6.4.6	NOE GUI mode – the NOE calculation	82
7	Model-free analysis	85
7.1	Model-free theory	85
7.1.1	The chi-squared function – $\chi^2(\theta)$	85
7.1.2	The transformed relaxation equations – $R_i(\theta)$	86
7.1.3	The relaxation equations – $R'_i(\theta)$	86
7.1.4	The spectral density functions – $J(\omega)$	87
7.1.5	Brownian rotational diffusion	87
7.1.6	The model-free models	89
7.1.7	Model-free optimisation theory	90
7.2	Optimisation of a single model-free model	94
7.2.1	Single model-free model script mode – the sample script	94
7.2.2	Single model-free model script mode – explanation	95

7.3	Optimisation of all model-free models	96
7.3.1	All model-free models script mode – the sample script	96
7.3.2	All model-free models script mode – explanation	97
7.4	Model-free model selection	97
7.4.1	Model-free model selection script mode – the sample script	97
7.4.2	Model-free model selection script mode – explanation	98
7.5	The methodology of Mandel et al., 1995	98
7.6	The diffusion seeded paradigm	100
7.7	The new model-free optimisation protocol	100
7.7.1	The new protocol – model-free models	100
7.7.2	The new protocol – the diffusion tensor	102
7.7.3	The universal solution \mathfrak{U}	103
7.7.4	Model-free analysis in reverse	103
7.8	The new protocol in the prompt/script UI mode	107
7.8.1	d'Auvergne protocol script mode – the sample script	107
7.8.2	d'Auvergne protocol script mode – analysis variables	112
7.8.3	d'Auvergne protocol script mode – data pipe initialisation	112
7.8.4	d'Auvergne protocol script mode – setting up the spin systems	113
7.8.5	d'Auvergne protocol script mode – loading the data	114
7.8.6	d'Auvergne protocol script mode – deselection	114
7.8.7	d'Auvergne protocol script mode – relaxation interactions	114
7.8.8	d'Auvergne protocol script mode – execution	115
7.9	The new protocol in the GUI	116
7.9.1	d'Auvergne protocol GUI mode – data pipe initialisation	117
7.9.2	d'Auvergne protocol GUI mode – general setup	117
7.9.3	d'Auvergne protocol GUI mode – setting up the spin systems	118
7.9.4	d'Auvergne protocol GUI mode – unresolved spins	119
7.9.5	d'Auvergne protocol GUI mode – loading the data	119
7.9.6	d'Auvergne protocol GUI mode – relaxation interactions	122
7.9.7	d'Auvergne protocol GUI mode – spin isotopes	124
7.9.8	d'Auvergne protocol GUI mode – the rest of the setup	125
7.9.9	d'Auvergne protocol GUI mode – execution	125
7.9.10	d'Auvergne protocol GUI mode – completion	127
7.9.11	d'Auvergne protocol GUI mode – BMRB deposition	127
8	Reduced spectral density mapping	129
8.1	Introduction to reduced spectral density mapping	129
8.2	J(w) mapping script mode – the sample script	129
8.3	J(w) mapping script mode – data pipe and spin system setup	130
8.4	J(w) mapping script mode – relaxation data loading	131
8.5	J(w) mapping script mode – relaxation interactions	131
8.6	J(w) mapping script mode – calculation and error propagation	132
8.7	J(w) mapping script mode – visualisation and data output	132
9	Consistency testing	133
9.1	Introduction to the consistency testing of relaxation data	133
9.2	Consistency testing in the prompt/script UI mode	134
9.2.1	Consistency testing script mode – the sample script	134
9.3	Consistency testing script mode – data pipe and spin system setup	136

9.4	Consistency testing script mode – relaxation data loading	137
9.5	Consistency testing script mode – relaxation interactions	137
9.6	Consistency testing script mode – calculation and error propagation	138
9.7	Consistency testing script mode – visualisation and data output	138
10	The N-state model or ensemble analysis	141
10.1	Introduction to the N-state model	141
10.2	Experimental data support for the N-state model	142
10.2.1	RDCs in the N-state model	142
10.2.2	PCCs in the N-state model	142
10.2.3	NOEs in the N-state model	143
10.3	Determining stereochemistry in dynamic molecules	143
10.3.1	Stereochemistry – the auto-analysis	143
10.3.2	Stereochemistry – the sample script	144
11	Relaxation dispersion	147
11.1	Introduction to relaxation dispersion	147
11.1.1	The modelling of dispersion data	148
11.1.2	Implemented models	148
11.1.3	Dispersion model summary	151
11.2	The base dispersion models	156
11.2.1	The R _{2eff} model	156
11.2.2	The model for no chemical exchange relaxation	158
11.3	The analytic CPMG models	159
11.3.1	The LM63 2-site fast exchange CPMG model	159
11.3.2	The LM63 3-site fast exchange CPMG model	160
11.3.3	The full CR72 2-site CPMG model	161
11.3.4	The reduced CR72 2-site CPMG model	162
11.3.5	The IT99 2-site CPMG model	162
11.3.6	The TSMFK01 2-site CPMG model	163
11.3.7	The full B14 2-site CPMG model	164
11.3.8	The reduced B14 2-site CPMG model	166
11.4	The numeric CPMG models	166
11.4.1	The NS 2-site expanded CPMG model	166
11.4.2	The full NS 2-site 3D CPMG model	169
11.4.3	The reduced NS 2-site 3D CPMG model	169
11.4.4	The full NS 2-site star CPMG model	169
11.4.5	The reduced NS 2-site star CPMG model	170
11.5	The analytic MMQ CPMG models	170
11.5.1	The MMQ CR72 model	170
11.6	The numeric MMQ CPMG models	172
11.6.1	The NS MMQ 2-site model	172
11.6.2	The NS MMQ 3-site linear model	174
11.6.3	The NS MMQ 3-site model	175
11.7	The analytic R _{1ρ} models	176
11.7.1	The M61 2-site fast exchange R _{1ρ} model	177
11.7.2	The M61 skew 2-site fast exchange R _{1ρ} model	177
11.7.3	The DPL94 2-site fast exchange R _{1ρ} model	178
11.7.4	The TP02 2-site exchange R _{1ρ} model	178

11.7.5	The TAP03 2-site exchange $R_{1\rho}$ model	179
11.7.6	The MP05 2-site exchange $R_{1\rho}$ model	180
11.8	The numeric $R_{1\rho}$ models	180
11.8.1	The NS 2-site $R_{1\rho}$ model	181
11.8.2	The NS 3-site $R_{1\rho}$ model	181
11.8.3	The NS 3-site linear $R_{1\rho}$ model	183
11.9	Relaxation dispersion optimisation theory	185
11.9.1	The relaxation dispersion auto-analysis	185
11.9.2	Dispersion curve insignificance	190
11.9.3	The relaxation dispersion space	190
11.9.4	The clustered relaxation dispersion analysis	190
11.9.5	Dispersion parameter grid search	191
11.9.6	Dispersion parameter optimisation	192
11.9.7	Relaxation dispersion parameter constraints	192
11.9.8	Relaxation dispersion diagonal scaling	194
11.9.9	Relaxation dispersion model elimination	195
11.9.10	Monte Carlo simulation elimination	195
11.9.11	Relaxation dispersion on a computer cluster using OpenMPI	196
11.10	To do – dispersion features yet to be implemented	196
11.11	Tutorial for adding relaxation dispersion models	197
11.12	Comparison of dispersion analysis software	197
11.13	Analysing dispersion in the prompt/script UI mode	201
11.13.1	Dispersion script mode – the sample script	201
11.13.2	Dispersion script mode – imports	203
11.13.3	Dispersion script mode – analysis variables	204
11.13.4	Dispersion script mode – initialisation of the data pipe	205
11.13.5	Dispersion script mode – setting up the spin systems	206
11.13.6	Dispersion script mode – loading the data	206
11.13.7	Dispersion script mode – the rest of the setup	208
11.13.8	Dispersion script mode – execution	209
11.14	The relaxation dispersion auto-analysis in the GUI	210
11.14.1	Dispersion GUI mode – two analyses	210
11.14.2	Dispersion GUI mode – computation time	211
11.14.3	Dispersion GUI mode – initialisation of the data pipe	211
11.14.4	Dispersion GUI mode – general setup	212
11.14.5	Dispersion GUI mode – setting up the spin systems	213
11.14.6	Dispersion GUI mode – unresolved spins	214
11.14.7	Dispersion GUI mode – dispersion setup	214
11.14.8	Dispersion GUI mode – loading the data	215
11.14.9	Dispersion GUI mode – choosing the models to optimise	224
11.14.10	Dispersion GUI mode – optimisation settings	225
11.14.11	Dispersion GUI mode – execution of the non-clustered analysis	226
11.14.12	Dispersion GUI mode – inspection of the results	227
11.14.13	Dispersion GUI mode – comparing models	231
11.14.14	Dispersion GUI mode – the clustered analysis	232
11.14.15	Dispersion GUI mode – comparison of the analyses	234
12	Frame order	237
12.1	Introduction of frame ordering	237

12.1.1	Tensors of frame ordering	237
12.1.2	Ln^{3+} aligned RDC and PCS data	237
12.2	Frame order theory	238
12.2.1	Frame order introduction	238
12.2.2	Frame order and the alignment tensor	244
12.2.3	Single pivoted motions	247
12.2.4	Double pivoted motions	248
12.2.5	Frame order in rotational Brownian diffusion and NMR relaxation	251
12.3	Frame order modelling	252
12.3.1	Rigid body motions for a two domain system	252
12.3.2	Frame order axis permutations	254
12.3.3	Linear constraints for the frame order models	257
12.4	Computation time and the numerical integration of the PCS	258
12.4.1	Numerical integration techniques	258
12.4.2	Parallelization and running on a cluster	260
12.4.3	Frame order model nesting	260
12.4.4	PCS subset	263
12.4.5	Optimisation of the frame order models	263
12.4.6	Error analysis	264
12.5	The frame order data analysis	264
12.5.1	Introduction to frame order data analysis	264
12.5.2	The N-state model analysis scripts	265
12.5.3	The frame order analysis scripts	268
12.5.4	Computation times	273
III	Power users	275
13	relax development	277
13.1	The relax source code repositories	277
13.1.1	relax repositories	277
13.1.2	Primary relax repository	278
13.1.3	Mirrors of the relax repository	278
13.2	Coding conventions	279
13.2.1	Indentation	280
13.2.2	Doc strings	280
13.2.3	Variable, function, and class names	281
13.2.4	Whitespace	283
13.2.5	Comments	284
13.3	Committers	284
13.3.1	Becoming a committer	284
13.3.2	Register for a relax infrastructure account	285
13.3.3	Joining the relax project	285
13.3.4	Format of the commit logs	285
13.3.5	Discussing major changes	287
13.4	Submitting changes to the relax project	287
13.4.1	Development branches	287
13.4.2	Keeping the branch up to date	287
13.4.3	Submitting patches	288

13.4.4	Repository forks	289
13.4.5	Merging the branch back into the main line	289
13.5	The SCons build system	290
13.5.1	SCons help	290
13.5.2	C module compilation	290
13.5.3	Compilation of the user manual (PDF version)	290
13.5.4	Compilation of the user manual (HTML version)	291
13.5.5	Compilation of the API documentation (HTML version)	291
13.5.6	Making distribution archives	291
13.5.7	Cleaning up	291
13.6	The core design of relax	292
13.6.1	The divisions of relax's source code	292
13.6.2	The major components of relax	293
13.7	The mailing lists for development	295
13.7.1	Private vs. public messages	295
13.8	The bug, task, and support request trackers	296
13.8.1	Submitting a bug report	296
13.8.2	Assigning an issue to yourself	296
13.8.3	Closing an issue	297
13.9	Links, links, and more links	297
13.9.1	Navigation	297
13.9.2	Search engine indexing	298
IV	Advanced topics	299
14	Optimisation	301
14.1	Implementation	301
14.1.1	The interface	301
14.1.2	The minfx package	301
14.2	The optimisation space	303
14.3	Topology of the space	303
14.3.1	The function value	303
14.3.2	The gradient	304
14.3.3	The Hessian	304
14.4	Optimisation algorithms	305
14.4.1	Line search methods	305
14.4.2	Trust region methods	307
14.4.3	Conjugate gradient methods	308
14.4.4	Hessian modifications	309
14.4.5	Other methods	309
14.5	Constraint algorithms	310
14.5.1	Method of Multipliers algorithm	311
14.5.2	Logarithmic barrier constraint algorithm	312
14.6	Diagonal scaling	312
15	Optimisation of relaxation data – values, gradients, and Hessians	315
15.1	Introduction to the mathematics behind the optimisation of relaxation data	315
15.2	The four parameter combinations	315
15.2.1	Optimisation of the model-free models	315

15.2.2	Optimisation of the local τ_m models	316
15.2.3	Optimisation of the diffusion tensor parameters	316
15.2.4	Optimisation of the global model \mathbf{S}	317
15.3	Construction of the values, gradients, and Hessians	317
15.3.1	The sum of chi-squared values	317
15.3.2	Construction of the gradient	317
15.3.3	Construction of the Hessian	319
15.4	The value, gradient, and Hessian dependency chain	319
15.5	The χ^2 value, gradient, and Hessian	321
15.5.1	The χ^2 value	321
15.5.2	The χ^2 gradient	321
15.5.3	The χ^2 Hessian	321
15.6	The $R_i(\theta)$ values, gradients, and Hessians	322
15.6.1	The $R_i(\theta)$ values	322
15.6.2	The $R_i(\theta)$ gradients	322
15.6.3	The $R_i(\theta)$ Hessians	322
15.7	$R'_i(\theta)$ values, gradients, and Hessians	323
15.7.1	Components of the $R'_i(\theta)$ equations	323
15.7.2	$R'_i(\theta)$ values	326
15.7.3	$R'_i(\theta)$ gradients	326
15.7.4	$R'_i(\theta)$ Hessians	327
15.8	Optimisation equations for the model-free analysis	331
15.8.1	The model-free equations	331
15.8.2	The original model-free gradient	332
15.8.3	The original model-free Hessian	333
15.8.4	The extended model-free gradient	336
15.8.5	The extended model-free Hessian	338
15.8.6	The alternative extended model-free gradient	343
15.8.7	The alternative extended model-free Hessian	345
15.9	Ellipsoidal diffusion tensor	350
15.9.1	The diffusion equation of the ellipsoid	350
15.9.2	The weights of the ellipsoid	350
15.9.3	The weight gradients of the ellipsoid	351
15.9.4	The weight Hessians of the ellipsoid	353
15.9.5	The correlation times of the ellipsoid	359
15.9.6	The correlation time gradients of the ellipsoid	359
15.9.7	The correlation time Hessians of the ellipsoid	361
15.10	Spheroidal diffusion tensor	363
15.10.1	The diffusion equation of the spheroid	363
15.10.2	The weights of the spheroid	363
15.10.3	The weight gradients of the spheroid	364
15.10.4	The weight Hessians of the spheroid	364
15.10.5	The correlation times of the spheroid	365
15.10.6	The correlation time gradients of the spheroid	365
15.10.7	The correlation time Hessians of the spheroid	365
15.11	Spherical diffusion tensor	367
15.11.1	The diffusion equation of the sphere	367
15.11.2	The weight of the sphere	367
15.11.3	The weight gradient of the sphere	367

15.11.4	The weight Hessian of the sphere	368
15.11.5	The correlation time of the sphere	368
15.11.6	The correlation time gradient of the sphere	368
15.11.7	The correlation time Hessian of the sphere	368
15.12	Ellipsoidal dot product derivatives	369
15.12.1	The dot product of the ellipsoid	369
15.12.2	The dot product gradient of the ellipsoid	369
15.12.3	The dot product Hessian of the ellipsoid	371
15.13	Spheroidal dot product derivatives	373
15.13.1	The dot product of the spheroid	373
15.13.2	The dot product gradient of the spheroid	373
15.13.3	The dot product Hessian of the spheroid	373
16	The frame order models	375
16.1	The current frame order models	375
16.2	Simulation of the frame order models	375
16.3	Rigid frame order model	385
16.3.1	Rigid model parameterisation	385
16.3.2	Rigid model equations	385
16.4	Rotor frame order model	386
16.4.1	Rotor parameterisation	386
16.4.2	Rotor equations	387
16.5	Free rotor frame order model	391
16.5.1	Free rotor parameterisation	391
16.5.2	Free rotor equations	391
16.6	Isotropic cone frame order model	394
16.6.1	Isotropic cone parameterisation	394
16.6.2	Isotropic cone equations	395
16.7	Torsionless isotropic cone frame order model	400
16.7.1	Torsionless isotropic cone parameterisation	400
16.7.2	Torsionless isotropic cone equations	400
16.8	Free rotor isotropic cone frame order model	405
16.8.1	Free rotor isotropic cone parameterisation	405
16.8.2	Free rotor isotropic cone equations	407
16.9	Pseudo-ellipse frame order model	408
16.9.1	Pseudo-ellipse parameterisation	408
16.9.2	Derivation of a 2D trigonometric function - the pseudo-elliptic cosine	409
16.9.3	Pseudo-ellipse equations	411
16.10	Torsionless pseudo-ellipse frame order model	419
16.10.1	Torsionless pseudo-ellipse parameterisation	420
16.10.2	Torsionless pseudo-ellipse equations	420
16.11	Free rotor pseudo-ellipse frame order model	427
16.11.1	Free rotor pseudo-ellipse parameterisation	427
16.11.2	Free rotor pseudo-ellipse equations	427
16.12	Double rotor frame order model	433
16.12.1	Double rotor parameterisation	434
16.12.2	Double rotor equations	434

V Reference	441
17 Alphabetical listing of user functions	443
17.1 A warning about the formatting	443
17.2 The list of functions	443
17.2.1 The synopsis	443
17.2.2 Defaults	443
17.2.3 Docstring sectioning	444
17.2.4 align_tensor.copy	445
17.2.5 align_tensor.delete	446
17.2.6 align_tensor.display	446
17.2.7 align_tensor.fix	447
17.2.8 align_tensor.init	447
17.2.9 align_tensor.matrix_angles	448
17.2.10 align_tensor.reduction	449
17.2.11 align_tensor.set_domain	450
17.2.12 align_tensor.svd	450
17.2.13 angles.diff_frame	452
17.2.14 bmrb.citation	452
17.2.15 bmrb.display	454
17.2.16 bmrb.read	454
17.2.17 bmrb.script	455
17.2.18 bmrb.software	456
17.2.19 bmrb.software_select	457
17.2.20 bmrb.thiol_state	458
17.2.21 bmrb.write	458
17.2.22 bruker.read	459
17.2.23 chemical_shift.read	459
17.2.24 consistency_tests.set_frq	460
17.2.25 dasha.create	460
17.2.26 dasha.execute	461
17.2.27 dasha.extract	461
17.2.28 deselect.all	462
17.2.29 deselect.interatom	462
17.2.30 deselect.read	463
17.2.31 deselect.reverse	464
17.2.32 deselect.sn_ratio	465
17.2.33 deselect.spin	465
17.2.34 diffusion_tensor.copy	466
17.2.35 diffusion_tensor.delete	466
17.2.36 diffusion_tensor.display	467
17.2.37 diffusion_tensor.init	467
17.2.38 domain	470
17.2.39 dx.execute	471
17.2.40 dx.map	471
17.2.41 eliminate	475
17.2.42 error_analysis.covariance_matrix	476
17.2.43 fix	476
17.2.44 frame_order.count_sobol_points	477

17.2.45	frame_order.decompose	477
17.2.46	frame_order.distribute	478
17.2.47	frame_order.pdb_model	479
17.2.48	frame_order.permute_axes	480
17.2.49	frame_order.pivot	480
17.2.50	frame_order.quad_int	481
17.2.51	frame_order.ref_domain	481
17.2.52	frame_order.select_model	482
17.2.53	frame_order.simulate	483
17.2.54	frame_order.sobol_setup	484
17.2.55	grace.view	485
17.2.56	grace.write	485
17.2.57	interatom.copy	489
17.2.58	interatom.define	489
17.2.59	interatom.read_dist	490
17.2.60	interatom.set_dist	491
17.2.61	interatom.unit_vectors	492
17.2.62	j_coupling.copy	493
17.2.63	j_coupling.delete	493
17.2.64	j_coupling.display	494
17.2.65	j_coupling.read	494
17.2.66	j_coupling.write	495
17.2.67	jw_mapping.set_frq	496
17.2.68	minimise.calculate	496
17.2.69	minimise.execute	497
17.2.70	minimise.grid_search	500
17.2.71	minimise.grid_zoom	501
17.2.72	model_free.create_model	502
17.2.73	model_free.delete	503
17.2.74	model_free.remove_tm	504
17.2.75	model_free.select_model	504
17.2.76	model_selection	506
17.2.77	molecule.copy	507
17.2.78	molecule.create	508
17.2.79	molecule.delete	509
17.2.80	molecule.display	509
17.2.81	molecule.name	510
17.2.82	molecule.type	511
17.2.83	molmol.clear_history	512
17.2.84	molmol.command	512
17.2.85	molmol.macro_apply	513
17.2.86	molmol.macro_run	525
17.2.87	molmol.macro_write	526
17.2.88	molmol.ribbon	527
17.2.89	molmol.tensor_pdb	528
17.2.90	molmol.view	529
17.2.91	monte_carlo.create_data	529
17.2.92	monte_carlo.error_analysis	531
17.2.93	monte_carlo.initial_values	532

17.2.94	monte_carlo.off	533
17.2.95	monte_carlo.on	534
17.2.96	monte_carlo.setup	535
17.2.97	n_state_model.CoM	536
17.2.98	n_state_model.cone_pdb	537
17.2.99	n_state_model.elim_no_prob	538
17.2.100	n_state_model.number_of_states	539
17.2.101	n_state_model.ref_domain	539
17.2.102	n_state_model.select_model	540
17.2.103	noe.read_restraints	540
17.2.104	noe.spectrum_type	541
17.2.105	palmer.create	541
17.2.106	palmer.execute	542
17.2.107	palmer.extract	543
17.2.108	paramag.centre	543
17.2.109	pcs.back_calc	544
17.2.110	pcs.calc_q_factors	545
17.2.111	pcs.copy	545
17.2.112	pcs.corr_plot	546
17.2.113	pcs.delete	546
17.2.114	pcs.display	547
17.2.115	pcs.read	547
17.2.116	pcs.set_errors	548
17.2.117	pcs.structural_noise	549
17.2.118	pcs.weight	550
17.2.119	pcs.write	550
17.2.120	pipe.bundle	551
17.2.121	pipe.change_type	551
17.2.122	pipe.copy	552
17.2.123	pipe.create	552
17.2.124	pipe.current	553
17.2.125	pipe.delete	554
17.2.126	pipe.display	554
17.2.127	pipe.hybridise	555
17.2.128	pipe.switch	555
17.2.129	pymol.cartoon	556
17.2.130	pymol.clear_history	556
17.2.131	pymol.command	557
17.2.132	pymol.cone_pdb	557
17.2.133	pymol.frame_order	558
17.2.134	pymol.macro_apply	559
17.2.135	pymol.macro_run	560
17.2.136	pymol.macro_write	561
17.2.137	pymol.tensor_pdb	562
17.2.138	pymol.vector_dist	563
17.2.139	pymol.view	564
17.2.140	rdc.back_calc	564
17.2.141	rdc.calc_q_factors	565
17.2.142	rdc.copy	565

17.2.143 rdc.corr_plot	566
17.2.144 rdc.delete	566
17.2.145 rdc.display	567
17.2.146 rdc.read	567
17.2.147 rdc.set_errors	568
17.2.148 rdc.weight	569
17.2.149 rdc.write	569
17.2.150 relax_data.back_calc	570
17.2.151 relax_data.copy	570
17.2.152 relax_data.delete	571
17.2.153 relax_data.display	571
17.2.154 relax_data.peak_intensity_type	572
17.2.155 relax_data.read	572
17.2.156 relax_data.temp_calibration	573
17.2.157 relax_data.temp_control	574
17.2.158 relax_data.type	575
17.2.159 relax_data.write	575
17.2.160 relax_disp.catia_execute	576
17.2.161 relax_disp.catia_input	576
17.2.162 relax_disp.cluster	577
17.2.163 relax_disp.cpmg_setup	577
17.2.164 relax_disp.cpmgfit_execute	578
17.2.165 relax_disp.cpmgfit_input	578
17.2.166 relax_disp.exp_type	579
17.2.167 relax_disp.insignificance	580
17.2.168 relax_disp.nessy_input	580
17.2.169 relax_disp.parameter_copy	581
17.2.170 relax_disp.plot_disp_curves	581
17.2.171 relax_disp.plot_exp_curves	582
17.2.172 relax_disp.r1_fit	583
17.2.173 relax_disp.r20_from_min_r2eff	583
17.2.174 relax_disp.r2eff_err_estimate	584
17.2.175 relax_disp.r2eff_read	584
17.2.176 relax_disp.r2eff_read_spin	585
17.2.177 relax_disp.relax_time	586
17.2.178 relax_disp.select_model	586
17.2.179 relax_disp.sherekhan_input	588
17.2.180 relax_disp.spin_lock_field	589
17.2.181 relax_disp.spin_lock_offset	589
17.2.182 relax_disp.write_disp_curves	590
17.2.183 relax_fit.relax_time	590
17.2.184 relax_fit.select_model	591
17.2.185 reset	591
17.2.186 residue.copy	592
17.2.187 residue.create	592
17.2.188 residue.delete	593
17.2.189 residue.display	593
17.2.190 residue.name	594
17.2.191 residue.number	595

17.2.192	results.display	596
17.2.193	results.read	596
17.2.194	results.write	597
17.2.195	script	597
17.2.196	select.all	598
17.2.197	select.display	598
17.2.198	select.domain	599
17.2.199	select.interatom	600
17.2.200	select.read	601
17.2.201	select.reverse	602
17.2.202	select.sn_ratio	602
17.2.203	select.spin	603
17.2.204	sequence.attach_protons	603
17.2.205	sequence.copy	604
17.2.206	sequence.display	604
17.2.207	sequence.read	605
17.2.208	sequence.write	606
17.2.209	spectrometer.frequency	606
17.2.210	spectrometer.temperature	607
17.2.211	spectrum.baseplane_rmsd	607
17.2.212	spectrum.delete	608
17.2.213	spectrum.error_analysis	608
17.2.214	spectrum.error_analysis_per_field	610
17.2.215	spectrum.integration_points	611
17.2.216	spectrum.read_intensities	611
17.2.217	spectrum.read_spins	613
17.2.218	spectrum.replicated	614
17.2.219	spectrum.sn_ratio	614
17.2.220	spin.copy	615
17.2.221	spin.create	615
17.2.222	spin.create_pseudo	616
17.2.223	spin.delete	617
17.2.224	spin.display	617
17.2.225	spin.element	618
17.2.226	spin.isotope	619
17.2.227	spin.name	620
17.2.228	spin.number	621
17.2.229	state.load	622
17.2.230	state.save	622
17.2.231	statistics.aic	623
17.2.232	statistics.model	624
17.2.233	structure.add_atom	624
17.2.234	structure.add_helix	625
17.2.235	structure.add_model	625
17.2.236	structure.add_sheet	626
17.2.237	structure.atomic_fluctuations	626
17.2.238	structure.com	628
17.2.239	structure.connect_atom	628
17.2.240	structure.create_diff_tensor_pdb	629

17.2.241	structure.create_rotor_pdb	630
17.2.242	structure.create_vector_dist	631
17.2.243	structure.delete	631
17.2.244	structure.delete_ss	632
17.2.245	structure.displacement	632
17.2.246	structure.find_pivot	633
17.2.247	structure.get_pos	634
17.2.248	structure.load_spins	635
17.2.249	structure.mean	636
17.2.250	structure.pca	636
17.2.251	structure.read_gaussian	637
17.2.252	structure.read_pdb	638
17.2.253	structure.read_xyz	640
17.2.254	structure.rmsd	641
17.2.255	structure.rotate	642
17.2.256	structure.sequence_alignment	642
17.2.257	structure.superimpose	643
17.2.258	structure.translate	645
17.2.259	structure.web_of_motion	645
17.2.260	structure.write_pdb	646
17.2.261	system.cd	647
17.2.262	system.getcwd	648
17.2.263	system.sys_info	648
17.2.264	system.time	649
17.2.265	value.copy	649
17.2.266	value.display	653
17.2.267	value.read	654
17.2.268	value.set	656
17.2.269	value.write	660
17.2.270	vmd.view	661
18	Licence	663
18.1	Copying, modification, sublicencing, and distribution of relax	663
18.2	The GPL	663

List of Figures

1.1	Prompt screenshot	7
1.2	Scripting screenshot	8
1.3	GUI screenshot	10
1.4	GUI screenshot – Analysis wizard screenshot	12
1.5	GUI screenshot – NOE analysis	15
1.6	GUI screenshot – R_1 analysis	16
1.7	GUI screenshot – R_2 analysis	17
1.8	GUI screenshot – Model-free analysis	18
1.9	relax controller screenshot	19
1.10	Spin viewer window screenshot	20
1.11	Results viewer window screenshot	21
1.12	Pipe editor window screenshot	21
1.13	Prompt window screenshot	22
5.1	Peak intensity 2D plot xmgrace screenshot	70
6.1	NOE plot	76
7.1	A schematic of the model-free optimisation protocol of Mandel et al., 1995	99
7.2	Model-free analysis using the diffusion seeded paradigm	101
7.3	A schematic of the new model-free optimisation protocol	104
9.1	Example of consistency testing visual analysis	139
11.1	Comparison of relaxation dispersion errors	158
12.1	Frame order in the double pivot system.	249
12.2	Pseudo-ellipse axis permutations.	255
12.3	Isotropic cone axis permutations.	256
12.4	Structural noise and the PCS error.	267
13.1	The core design of relax.	294
15.1	The construction of the model-free gradient.	318
15.2	The model-free Hessian kite.	320
15.3	χ^2 dependencies of the values, gradients, and Hessians.	321
16.1	Rotor simulated and calculated in-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	388
16.2	Rotor simulated and calculated out-of-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	389
16.3	Free rotor simulated and calculated in-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	392
16.4	Free rotor simulated and calculated out-of-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	393
16.5	Isotropic cone simulated and calculated in-frame Daeg ⁽¹⁾ elements.	395
16.6	Isotropic cone simulated and calculated in-frame Daeg ⁽²⁾ elements.	396

16.7	Isotropic cone simulated and calculated out-of-frame Daeg ⁽¹⁾ elements. . .	397
16.8	Isotropic cone simulated and calculated out-of-frame Daeg ⁽²⁾ elements. . .	398
16.9	Torsionless isotropic cone simulated and calculated in-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	401
16.10	Torsionless isotropic cone simulated and calculated out-of-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	402
16.11	Free-rotor isotropic cone simulated and calculated in-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	405
16.12	Free-rotor isotropic cone simulated and calculated out-of-frame Daeg ⁽¹⁾ and Daeg ⁽²⁾ elements.	406
16.13	The pseudo-elliptic cone.	409
16.14	Pseudo-ellipse cosine 2D trigonometric function.	412
16.15	Pseudo-ellipse simulated and calculated in-frame Daeg ⁽¹⁾ elements.	413
16.16	Pseudo-ellipse simulated and calculated in-frame Daeg ⁽²⁾ elements.	414
16.17	Pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽¹⁾ elements. . . .	415
16.18	Pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽²⁾ elements. . . .	416
16.19	Torsionless pseudo-ellipse simulated and calculated in-frame Daeg ⁽¹⁾ elements.	421
16.20	Torsionless pseudo-ellipse simulated and calculated in-frame Daeg ⁽²⁾ elements.	422
16.21	Torsionless pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽¹⁾ elements.	423
16.22	Torsionless pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽²⁾ elements.	424
16.23	Free rotor pseudo-ellipse simulated and calculated in-frame Daeg ⁽¹⁾ elements.	428
16.24	Free rotor pseudo-ellipse simulated and calculated in-frame Daeg ⁽²⁾ elements.	429
16.25	Free rotor pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽¹⁾ elements.	430
16.26	Free rotor pseudo-ellipse simulated and calculated out-of-frame Daeg ⁽²⁾ elements.	431
16.27	Double rotor simulated and calculated in-frame Daeg ⁽¹⁾ elements.	435
16.28	Double rotor simulated and calculated in-frame Daeg ⁽²⁾ elements.	436
16.29	Double rotor simulated and calculated out-of-frame Daeg ⁽¹⁾ elements. . .	437
16.30	Double rotor simulated and calculated out-of-frame Daeg ⁽²⁾ elements. . .	438

List of Tables

5.1	Summary, First Point Scaling and Phase Correction	54
11.1	The dispersion models.	152
11.1	The dispersion models.	153
11.2	The parameters of relaxation dispersion.	154
11.2	The parameters of relaxation dispersion.	155
11.3	Model nesting for the relaxation dispersion auto-analysis.	188
11.3	Model nesting for the relaxation dispersion auto-analysis.	189
11.4	Dispersion software comparison.	199
11.4	Dispersion software comparison.	200
12.1	The pseudo-ellipse axis and half-angle permutations.	257
12.2	Frame order parameter nesting.	262
17.1	Boolean operators and their effects on selections	463
17.2	OpenDx mapping types.	473
17.3	Model-free parameters.	473
17.4	N-state model parameters.	473
17.5	Relaxation dispersion parameters.	474
17.6	Frame order parameters.	474
17.7	The frame order axis permutations.	481
17.8	Relaxation curve fitting parameters and minimisation statistics.	487
17.9	Steady-state NOE parameters.	487
17.10	Model-free parameters and minimisation statistics.	487
17.11	Reduced spectral density mapping parameters.	487
17.12	Consistency testing parameters.	488
17.13	Relaxation dispersion parameters and minimisation statistics.	488
17.14	Minimisation algorithms – unconstrained line search methods.	498
17.15	Minimisation algorithms – unconstrained trust-region methods.	498
17.16	Minimisation algorithms – unconstrained conjugate gradient methods.	499
17.17	Minimisation algorithms – miscellaneous unconstrained methods.	499
17.18	Minimisation algorithms – global minimisation methods.	499
17.19	Minimisation sub-algorithms – line search algorithms.	499
17.20	Minimisation sub-algorithms – Hessian modifications.	499
17.21	Minimisation sub-algorithms – Hessian type.	499
17.22	The model-free classic style for PyMOL and Molmol data mapping.	514
17.23	Molmol colour names and corresponding RGB colour values (from 0 to 1)	515
17.24	X11 colour names and corresponding RGB colour values	516
17.24	X11 colour names and corresponding RGB colour values	517
17.24	X11 colour names and corresponding RGB colour values	518
17.24	X11 colour names and corresponding RGB colour values	519

17.24	X11 colour names and corresponding RGB colour values	520
17.24	X11 colour names and corresponding RGB colour values	521
17.24	X11 colour names and corresponding RGB colour values	522
17.24	X11 colour names and corresponding RGB colour values	523
17.24	X11 colour names and corresponding RGB colour values	524
17.25	The six peak intensity error analysis types.	609
17.26	Diffusion tensor PDB scaling.	630
17.27	Relaxation curve fitting parameters.	650
17.28	Model-free parameters.	650
17.29	Reduced spectral density mapping parameters.	651
17.30	Consistency testing parameters.	651
17.31	N-state model parameters.	651
17.32	Relaxation dispersion parameters.	652
17.33	Relaxation curve fitting parameters.	654
17.34	Model-free parameters.	654
17.35	The value and parameter combinations for the value.set user function.	657
17.36	Relaxation curve fitting parameter value setting.	657
17.37	Model-free parameter value setting.	657
17.38	Reduced spectral density mapping parameter value setting.	657
17.39	Consistency testing parameter value setting.	658
17.40	N-state model parameter value setting.	658
17.41	Relaxation dispersion parameter value setting.	659

Abbreviations

AIC: Akaike's Information Criteria (model selection method)

AICc: small sample size corrected AIC (model selection method)

API: application programming interface

ANOVA: analysis of variance (field of statistics)

BC: back calculation

BIC: Bayesian Information Criteria (model selection method)

BFGS: Broyden-Fletcher-Goldfarb-Shanno (optimisation method)

$C(\tau)$: correlation function

χ^2 : chi-squared function

CG: conjugate gradient (optimisation)

CPMG: the Carr-Purcell-Meiboom-Gill pulse sequence

CR72: the [Carver and Richards \(1972\)](#) relaxation dispersion model

CSA: chemical shift anisotropy

CV: cross validation

CVS: Concurrent Versions System (free software version control system)

\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
- DPL94:** the [Davis et al. \(1994\)](#) relaxation dispersion model
- DQ:** double quantum
- ϵ_i : elimination value
- FSF:** Free Software Foundation
- GNU:** GNU's Not Unix!
- GPG:** GNU Privacy Guard (software)
- GPL:** GNU general public licence
- GUI:** graphical user interface
- ID string:** identification string
- IT99:** the [Ishima and Torchia \(1999\)](#) relaxation dispersion model
- $J(\omega)$: spectral density function
- LM63:** the [Luz and Meiboom \(1963\)](#) relaxation dispersion model
- M61:** the [Meiboom \(1961\)](#) relaxation dispersion model
- MC:** Monte Carlo (simulations)
- MD:** molecular dynamics (simulations)
- MMQ:** proton-heteronuclear SQ, ZQ, DQ, and MQ data (multi-multiple quantum)
- MP05:** the [Miloushev and Palmer \(2005\)](#) relaxation dispersion model
- MPI:** message passing interface
- MQ:** multiple quantum
- NMR:** if you do not know this one, do not read further
- NNTP:** network news transfer protocol
- NOE:** nuclear Overhauser effect
- NS:** numeric solution
- ORD:** optical rotatory dispersion
- OS:** operating system
- PCS:** pseudocontact shift
- PDB:** Protein Data Bank

pdf: probability distribution function

PRE: paramagnetic relaxation enhancement

r: bond length

R_1 : spin-lattice relaxation rate

R_2 : spin-spin relaxation rate

R_{ex} : chemical exchange relaxation rate

RDC: residual dipolar coupling

RMSD: root-mean-square deviation

ROE: rotating-frame Overhauser effect

RSDM: reduced spectral density mapping

RSS: rich site summary (web feed format)

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

SVN: Apache Subversion (free software version control system)

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

τ_m : global rotational correlation time

TP02: the [Trott and Palmer \(2002\)](#) relaxation dispersion model

TAP03: the [Trott et al. \(2003\)](#) relaxation dispersion model

TSMFK01: the [Tollinger et al. \(2001\)](#) relaxation dispersion model

UI: user interface

XML: extensible markup language

ZQ: zero quantum

Preface - citing relax

The relax project is a large collection of work created by diverse authors. It is a community driven project created by NMR spectroscopists which supports a broad range of dynamics analyses. Care must be taken to properly cite the parts of relax that you use so that the correct authors receive the citations and credit they deserve. The following is a breakdown of all of the citations relating to relax, including the basic citations for the various analysis types. Including a link to the relax website <http://www.nmr-relax.com> in publications and other forums would also be greatly appreciated.

The software relax

relax references

The primary citations for relax are:

- d'Auvergne, E. J. and Gooley, P. R. (2008b). 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. ([10.1007/s10858-007-9214-2](https://doi.org/10.1007/s10858-007-9214-2))
- d'Auvergne, E. J. and Gooley, P. R. (2008c). 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. ([10.1007/s10858-007-9213-3](https://doi.org/10.1007/s10858-007-9213-3))

If space is at a premium, the standard rules for concatenating back-to-back papers can be used:

- d'Auvergne, E. J. and Gooley, P. R. (2008a). Optimisation of NMR dynamic models. *J. Biomol. NMR*, **40**(2), 107–133

Graphical user interface reference

The primary citation for the GUI is:

- Bieri, M., d'Auvergne, E., and Gooley, P. (2011). relaxGUI: a new software for fast and simple NMR relaxation data analysis and calculation of ps-ns and μ s motion of proteins. *J. Biomol. NMR*, **50**, 147–155. ([10.1007/s10858-011-9509-1](https://doi.org/10.1007/s10858-011-9509-1))

The multi-processor reference

Although not published, if the multi-processor framework is used to run relax on multi-core systems, grids, or clusters, then please acknowledge the author of that code – Gary Thompson.

Specific analyses

The following subsections list the citations for the individual analysis specific parts of relax.

Model-free analysis references

If the automated analysis of the `dauvergne_protocol.py` sample script or the GUI model-free analysis which uses the same protocol has been used, then the following citations are all implicit:

- 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. ([10.1023/a:1021902006114](https://doi.org/10.1023/a:1021902006114))
- 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. ([10.1007/s10858-006-9007-z](https://doi.org/10.1007/s10858-006-9007-z))
- d'Auvergne, E. J. and Gooley, P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. *Mol. BioSyst.*, **3**(7), 483–494. ([10.1039/b702202f](https://doi.org/10.1039/b702202f))
- d'Auvergne, E. J. and Gooley, P. R. (2008b). 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. ([10.1007/s10858-007-9214-2](https://doi.org/10.1007/s10858-007-9214-2))
- d'Auvergne, E. J. and Gooley, P. R. (2008c). 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. ([10.1007/s10858-007-9213-3](https://doi.org/10.1007/s10858-007-9213-3))

Otherwise, if model-free analysis is used in relax but not via the inbuilt automated protocol, the first reference is for model selection, the second is for eliminating failed model-free models, and the forth is for the optimisation improvements (the third and fifth are for the automated protocol). All of the model-free implementation details of relax are covered by the PhD thesis (available as a PDF or as a printed version on Amazon.com) of:

- 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/>. ([10187/2281](#))

The reference for the hybridisation of different global diffusion models to analyse the residual inter-domain dynamics – a not very well documented feature of relax – is:

- Horne, J., d'Auvergne, E. J., Coles, M., Velkov, T., Chin, Y., Charman, W. N., Pranker, R., Gooley, P. R., and Scanlon, M. J. (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. ([10.1016/j.jmb.2007.05.067](#))

The base citations for model-free theory are [Lipari and Szabo \(1982a,b\)](#); [Clore et al. \(1990\)](#).

Consistency testing analysis references

The first is the main citation, whereas the next are the individual tests. The citation for the consistency testing of NMR relaxation as implemented in relax is:

- Morin, S. and Gagné, S. (2009a). Simple tests for the validation of multiple field spin relaxation data. *J. Biomol. NMR*, **45**, 361–372. ([10.1007/s10858-009-9381-4](#))

The base citations for the consistency testing of NMR relaxation are [Fushman et al. \(1999\)](#); [Farrow et al. \(1995\)](#); [Fushman et al. \(1998\)](#)

N-state model analysis references

Some citations demonstrating as well as presenting the use of the N-state model for diverse analyses types are:

- Sun, H., d'Auvergne, E. J., Reinscheid, U. M., Dias, L. C., Andrade, C. K. Z., Rocha, R. O., and Griesinger, C. (2011). Bijvoet in solution reveals unexpected stereoselectivity in a michael addition. *Chem. Eur. J.*, **17**(6), 1811–1817. ([10.1002/chem.201002520](#))
- Erdelyi, M., d'Auvergne, E., Navarro-Vazquez, A., Leonov, A., and Griesinger, C. (2011). Dynamics of the glycosidic bond: Conformational space of lactose. *Chem. Eur. J.*, **17**(34), 9368–9376. ([10.1002/chem.201100854](#))

Reduced spectral density mapping references

The base citations for reduced spectral density mapping are [Farrow et al. \(1995\)](#); [Lefevre et al. \(1996\)](#).

Relaxation dispersion references

For the base citations for relaxation dispersion, please see chapter 11 on page 147 for a listing of the individual models. The main citation is:

- Morin, S., Linnet, T. E., Lescanne, M., Schanda, P., Thompson, G. S., Tollinger, M., Teilmann, K., Gagne, S., Marion, D., Griesinger, C., Blackledge, M., and d'Auvergne, E. J. (2014). relax: the analysis of biomolecular kinetics and thermodynamics using NMR relaxation dispersion data. *Bioinformatics*, **30**(15), 2219–2220. ([10.1093/bioinformatics/btu166](https://doi.org/10.1093/bioinformatics/btu166))

Generic parts of relax

The following subsections will list the citations for the parts of relax independent of the specific analyses.

Model selection references

The citation for the model selection component of relax is:

- 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. ([10.1023/a:1021902006114](https://doi.org/10.1023/a:1021902006114))

The base citations for the specific model selection techniques of AIC, AICc, and BIC are respectively Akaike (1973); Hurvich and Tsai (1989); Schwarz (1978)

- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In: Petrov, B. N. and Csaki, F. (eds.): *Proceedings of the Second International Symposium on Information Theory*. Budapest, pages 267–281, Akademia Kiado
- Hurvich, C. M. and Tsai, C. L. (1989). Regression and time-series model selection in small samples. *Biometrika*, **76**(2), 297–307. ([10.1093/biomet/76.2.297](https://doi.org/10.1093/biomet/76.2.297))
- Schwarz, G. (1978). Estimating dimension of a model. *Ann. Stat.*, **6**(2), 461–464. ([10.1214/aos/1176344136](https://doi.org/10.1214/aos/1176344136))

Other citations

If you believe that other citations should be included in this chapter, please contact the relax users mailing list (nmr-relax-users at lists.sourceforge.net).

Part I

The basics

Chapter 1

Introduction

The program relax is designed for the study of molecular dynamics through the analysis of experimental NMR data. Organic molecules, proteins, RNA, DNA, sugars, and other biomolecules are all supported. It was originally written for the model-free analysis of protein dynamics, though its scope has been significantly expanded. It is a community driven project created by NMR spectroscopists for NMR spectroscopists. It supports many analysis types including:

Model-free analysis - the Lipari and Szabo model-free analysis of NMR relaxation data

R₁ and R₂ - the exponential curve fitting for the calculation of the R_x NMR relaxation rates.

NOE - the calculation of the steady-state NOE NMR relaxation data.

Data consistency - the consistency testing of multiple field NMR relaxation data.

RSDM - Reduced Spectral Density Mapping.

Frame order and N-state model - study of domain motions via the N-state model and frame order dynamics theories using anisotropic NMR parameters such as RDCs and PCSs.

Stereochemistry - investigations of absolute stereochemistry of flexible molecules.

Relaxation dispersion - the study of processes on the chemical exchange timescale.

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 relax's scripting capabilities, its Python prompt interface, or its graphical user interface (GUI). 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 be 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 \(2008b\)](#) and [d'Auvergne and Gooley \(2008c\)](#). To properly cite the various parts of relax used in your analysis, please see Chapter [on page xxvii](#).

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. \(1990\)](#) and the specific implementation of [d'Auvergne and Gooley \(2003, 2006, 2007, 2008b,c\)](#)). This includes the hybridisation of global diffusion models to study residual domain dynamics ([Horne et al., 2007](#)).
- Reduced spectral density mapping ([Farrow et al., 1995](#); [Lefevre et al., 1996](#)).
- Consistency testing – the validation of multiple field NMR relaxation data ([Morin and Gagné, 2009a](#); [Fushman et al., 1999](#)).
- Exponential curve fitting (to find the R_1 and R_2 relaxation rates).
- Steady-state NOE calculation.
- Determination of absolute stereochemistry of flexible molecules via the N-state model using isotropic and anisotropic NMR parameters such as NOE, ROE, and RDC combined with MD simulation or simulated annealing, and ORD ([Sun et al., 2011](#)).
- The N-state model for investigating domain motions.
- The frame order theory.
- Conformational analysis of paramagnetically tagged molecules via the N-state model ([Erdelyi et al., 2011](#)).
- Analysis and comparison of ensembles of structures using RDCs, PCSs, NOEs, etc. (the N-state model of dynamics).
- The analysis of relaxation dispersion.

The future

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 Free Software chapter on page [29](#) 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).

Partial support for relaxation dispersion software has been implemented as well, including [Catia](#), [CPMGFit](#), [NESSY](#), and [ShereKhan](#).

1.1.6 The user interfaces (UI)

relax can be used through the following UIs:

The prompt rather than reinventing a new command language, relax's prompt interface is the powerful Python prompt. This gives the power user full access to a proven programming language. See Figure 1.1 for a screenshot.

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. See Figure 1.2 for a screenshot.

GUI the graphical user interface provides a sub-set of relax's features - the automatic R_1 and R_2 relaxation rate curve-fitting, the NOE calculations, the automatic model-free analysis provided by the `dauvergne_protocol` module ([d'Auvergne and Gooley, 2008c](#)), and relaxation dispersion. See Figure 1.3 for a screenshot.

1.2 How to use relax

1.2.1 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 programming language to manipulate your data. You can for instance type

```
relax> print("Hello World")
```

the result being

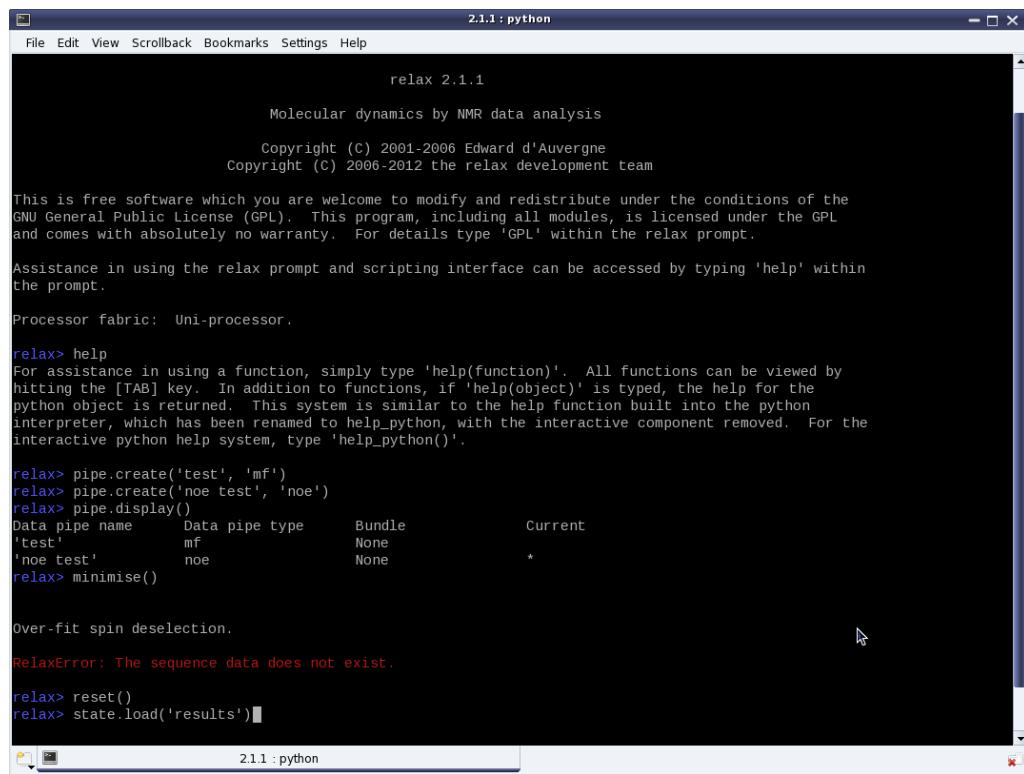


Figure 1.1: A screenshot of relax being run in prompt UI mode.

```
relax> print("Hello World")
Hello World
relax>
```

Or using relax as a calculator

```
relax> (1.0 + (2 * 3))/10
0.6999999999999996
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.execute()
```

```

1.3.15 : python
File Edit View Bookmarks Settings Help
[edward@localhost 1.3.15]$ ./relax simple_script.py

      relax 1.3.15
      Molecular dynamics by NMR data analysis
      Copyright (C) 2001-2006 Edward d'Auvergne
      Copyright (C) 2006-2012 the relax development team

This is free software which you are welcome to modify and redistribute under the conditions of the
GNU General Public License (GPL). This program, including all modules, is licensed under the GPL
and comes with absolutely no warranty. For details type 'GPL' within the relax prompt.

Assistance in using the relax prompt and scripting interface can be accessed by typing 'help' within
the prompt.

Processor fabric: Uni-processor.

script = 'simple_script.py'
-----
pipe.create('Dy test', 'N-state')
align_tensor.init(tensor='Dysprosium', params=(1.0278e-03, -1.4860e-03, 8.4778e-04, 5.7108e-04, 3.6500e-04), param_types=1)
rdc.calc_q_factors()
align_tensor.delete()
align_tensor.display()

relax> pipe.create(pipe_name='Dy test', pipe_type='N-state')
relax> align_tensor.init(tensor='Dysprosium', params=(0.0010278, -0.001486, 0.00084778, 0.00057108, 0.000365), scale=1.0, angle_units='deg',
', param_types=1, errors=False)
relax> rdc.calc_q_factors(spin_id=None)
RelaxWarning: No RDC data exists, Q factors cannot be calculated.

relax> align_tensor.delete(tensor=None)
Removing the 'Dysprosium' tensor.

relax> align_tensor.display(tensor=None)
RelaxError: No alignment tensor data exists.

[edward@localhost 1.3.15]$ 

```

Figure 1.2: A screenshot of relax being run in scripting mode.

will commence minimisation however

```
relax> minimise.execute
```

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(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
    res_num_col=1, data_col=3, error_col=4)
```

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

```
relax> help(function)
```

In addition to functions if

```
relax> 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

```
relax> 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.
```

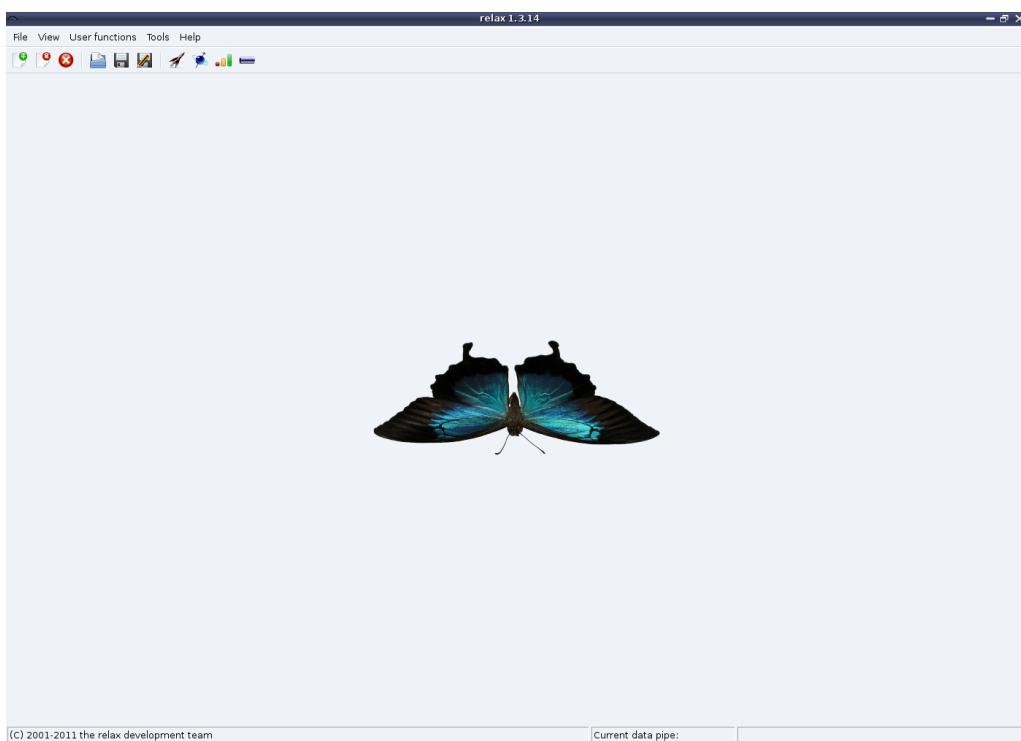


Figure 1.3: Screenshot of the relax GUI interface – the starting interface. To start one of the automated analyses, either the menu “File→New analysis” or the new analysis button in the toolbar should be selected.

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
“ <code>ct</code> ”	Consistency testing of relaxation data
“ <code>frame_order</code> ”	The Frame Order analyses of domain motions
“ <code>jw</code> ”	Reduced spectral density mapping
“ <code>hybrid</code> ”	A special hybridised data pipe
“ <code>mf</code> ”	Model-free data analysis
“ <code>N-state</code> ”	N-state model of domain motions
“ <code>noe</code> ”	Steady state NOE calculation
“ <code>relax_disp</code> ”	Relaxation dispersion curve fitting
“ <code>relax_fit</code> ”	Relaxation curve-fitting

1.2.7 The spin and interatomic data containers

Any data which is not considered global for the molecule, such as diffusion tensors, alignment tensors, global minimisation statistics, etc., are stored within two special structures of the data pipes. Any NMR data or information which is specific to an isolated spin system is stored within special spin containers. This includes for example relaxation data, CSA information, nuclear isotope type, chemical element type, model-free parameters, reduced spectral density mapping values, spin specific minimisation statistics and PCS data. NMR data or information which is defined as being between two spin systems, such as the magnetic dipole-dipole interaction involved in both NMR relaxation and RDC data, interatomic vectors and NOESY data, is stored within the interatomic data containers. The spin and interatomic data containers and their associated data can be manipulated using a multitude of the relax user functions.

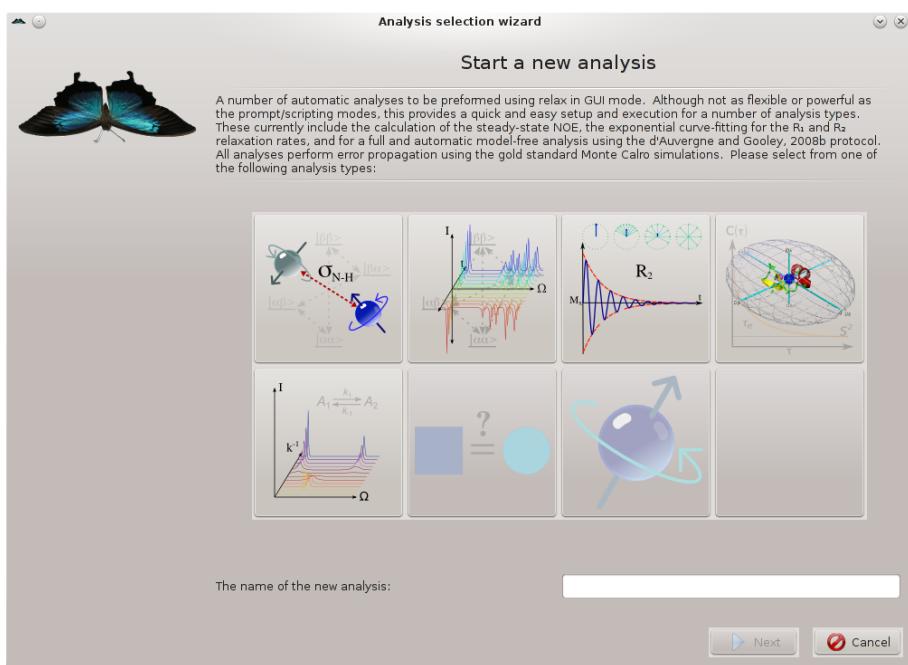


Figure 1.4: Screenshot of the relax GUI interface – the analysis selection wizard. From here, the steady-state NOE analysis, the R_1 and R_2 relaxation rates via exponential curve-fitting, and the automated model-free analysis can be selected.

1.2.8 Scripting

All operations that can be performed within the prompt UI are also accessible through scripting (Figure 1.2). First type your commands into a text file ending in `*.py` – a relax script is a Python script (loaded and executed as a Python module). Note that scripts can also be run through the GUI.

To use this mode of relax, you will need to open up a terminal in your respective operating system:

GNU/Linux: Here you have an incredible number of choices. If you don't have a preferred shell already, you could try one of `Konsole`, `GNOME Terminal` or even `XTerm` if you are a masochist.

Mac OS X: This is as simple as in GNU/Linux – just launch `Terminal.app` from the `Utilities` folder.

MS Windows: If your system supports it, you should install and use `Windows PowerShell`. The alternative is the nasty `cmd` command line terminal program which comes installed by default on all Windows versions. The `PowerShell`, although no where near as powerful as the GNU/Linux and Mac terminals, is a huge improvement on the ancient `cmd` program and will make relax much better to use on MS Windows.

Once your terminal is running, go to the directory containing your script using the `cd` command (if you do not know what this is, please see the documentation for your terminal

program to understand some of its basic usage). Once you are in the correct directory, within the terminal type:

```
$ relax your_script.py
```

You will need to replace `your_script.py` with the name of your script. In most cases you would probably like to keep a log of all of the messages, warnings and errors relax produces for future reference. To active logging within relax, type:

```
$ relax --log log your_script.py
```

This will place all output (both STDOUT and STDERR) into the `log` file (you can choose any name for this log file). Alternatively you can both log the output and simultaneously see the messages in your terminal by typing:

```
$ relax --tee log your_script.py
```

These command line arguments could be replaced by IO redirection if this is a familiar concept to you, but note that these arguments are active also in the GUI mode whereby IO redirection in the terminal will have no effect. An example of a simple script which will minimise the model-free model “m4” after loading six relaxation data sets is

```

1 # Create the data pipe.
2 name = 'm4'
3 pipe.create(name, 'mf')
4
5 # Load the PDB file.
6 structure.read_pdb('1f3y.pdb')
7
8 # Set up the 15N and 1H spins.
9 structure.load_spins('@N', ave_pos=True)
10 structure.load_spins('@H', ave_pos=True)
11 spin.isotope('15N', spin_id='@N')
12 spin.isotope('1H', spin_id='@H')
13
14 # Load the relaxation data.
15 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
16     res_num_col=1, data_col=3, error_col=4)
16 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
17     res_num_col=1, data_col=3, error_col=4)
17 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
18     res_num_col=1, data_col=3, error_col=4)
18 relax_data.read(ri_id='R1_500', ri_type='R1', frq=500.0*1e6, file='r1.500.out',
19     res_num_col=1, data_col=3, error_col=4)
19 relax_data.read(ri_id='R2_500', ri_type='R2', frq=500.0*1e6, file='r2.500.out',
20     res_num_col=1, data_col=3, error_col=4)
20 relax_data.read(ri_id='NOE_500', ri_type='NOE', frq=500.0*1e6, file='noe.500.out',
21     res_num_col=1, data_col=3, error_col=4)
21
22 # Initialise the diffusion tensor.
23 diffusion_tensor.init((2e-8, 1.3, 60, 290), spheroid_type='prolate', param_types=2, fixed=
24     True)
24
25 # Create all attached protons.
26 sequence.attach_protons()
27
28 # Define the magnetic dipole-dipole relaxation interaction.
29 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
30 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)
31 interatom.unit_vectors()
```

```

32
33 # Define the CSA relaxation interaction.
34 value.set(-172 * 1e-6, 'csa')
35
36 # Select a preset model-free model.
37 model_free.select_model(model=name)
38
39 # Grid search.
40 minimise.grid_search(inc=11)
41
42 # Minimise.
43 minimise.execute('newton')
44
45 # Finish.
46 results.write(file='results', force=True)
47 state.save('save', force=True)

```

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

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
```

Alternatively the three components of the test suite – system tests, unit tests, and GUI tests – can be run separately with

```

$ relax --system-tests
$ relax --unit-tests
$ relax --gui-tests

```

1.2.10 The GUI

If the wxPython module is installed on your system, you will have access to the GUI interface of relax. To launch relax in GUI mode, type either

```
$ relax -g
```

or

```
$ relax --gui
```

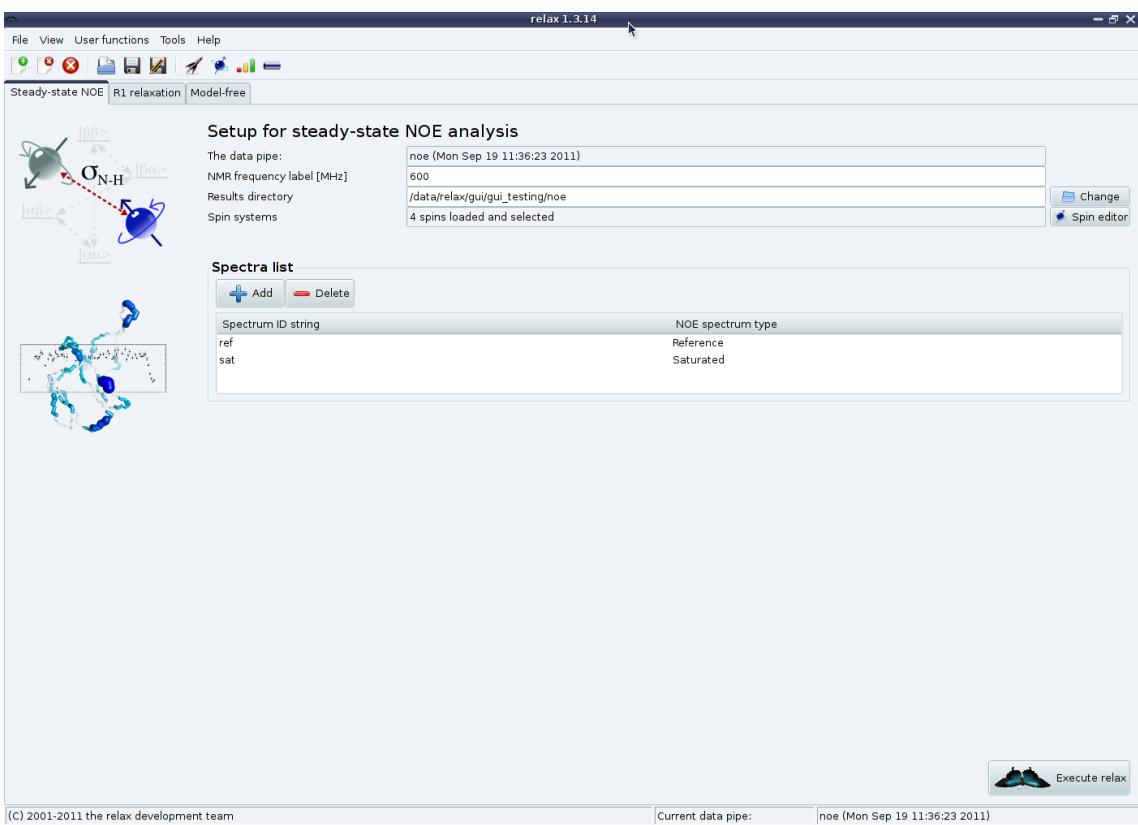


Figure 1.5: Screenshot of the relax GUI interface – the steady-state NOE analysis.

In most cases you will probably like to have a permanent copy of all the messages, warnings, and errors relax produces for future reference. In such a case you could run the GUI with:

```
$ relax --gui --log log
```

This will place all of the output into the `log` file.

The GUI is currently an interface to the automated analyses, providing an easy way to perform quick analyses. The interface consists of a tab for each analysis. By clicking on the “File→New analysis” menu entry or the “New analysis” toolbar button, the analysis wizard will appear (see Figure 1.4). The following analyses can be set up using this wizard:

Steady-state NOE: this provides access to the steady-state NOE calculation with pseudo Monte Carlo simulations for error analysis (this falls back to bootstrapping as this is a calculation rather than optimisation). See Figure 1.5 on page 15.

R₁ and R₂ : these provide easy access to optimisations and error analysis for the R₁ and R₂ relaxation rates via exponential curve-fitting (see Figures 1.6 and 1.7 on pages 16 and 17).

Model-free analysis : A fully automatic model-free protocol is provided in another tab. This operates via the `dauvergne_protocol` module which implements the protocol of d’Auvergne and Gooley (2008c) (see Figure 1.8 on page 18).

A number of windows in the GUI provide user feedback or allow for the viewing and editing of data. These include:

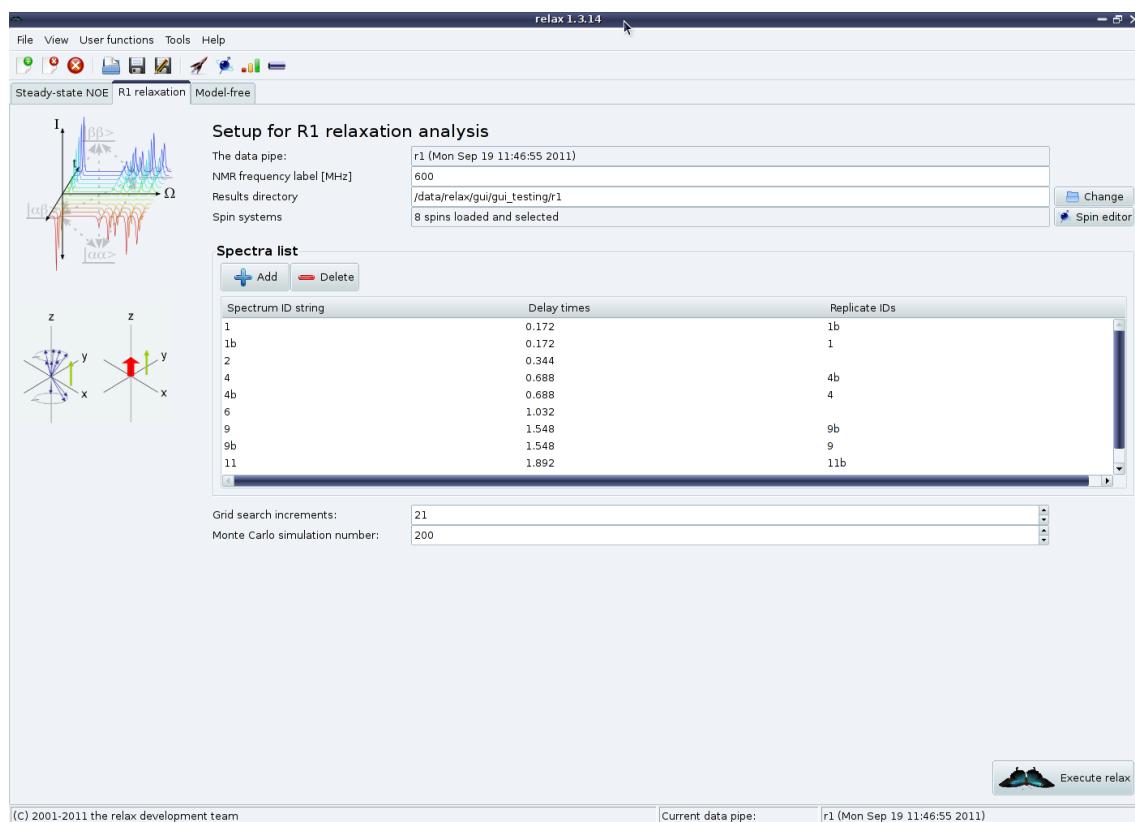


Figure 1.6: Screenshot of the relax GUI interface – the R_1 analysis.

The relax controller : This window shows the progress of relax's execution and displays relax's text output for checking if the analysis has been performed correctly and has completed successfully (see Figure 1.9).

Spin viewer window : This is used to load spins system information into the relax data store and to see the contents of the spin containers (see Figure 1.10).

Results viewer window : This presents a list of the results files which can be opened by double clicking for visualisation using a text editor, Grace, PyMOL, MOLMOL, etc (see Figure 1.11).

Data pipe editor : This window allows for easy manipulation of the data pipes of the relax data store (see Figure 1.12).

The relax prompt : This window gives access to the relax prompt (see Figure 1.13).

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_store import Relax_data_store; ds = Relax_data_store()
```

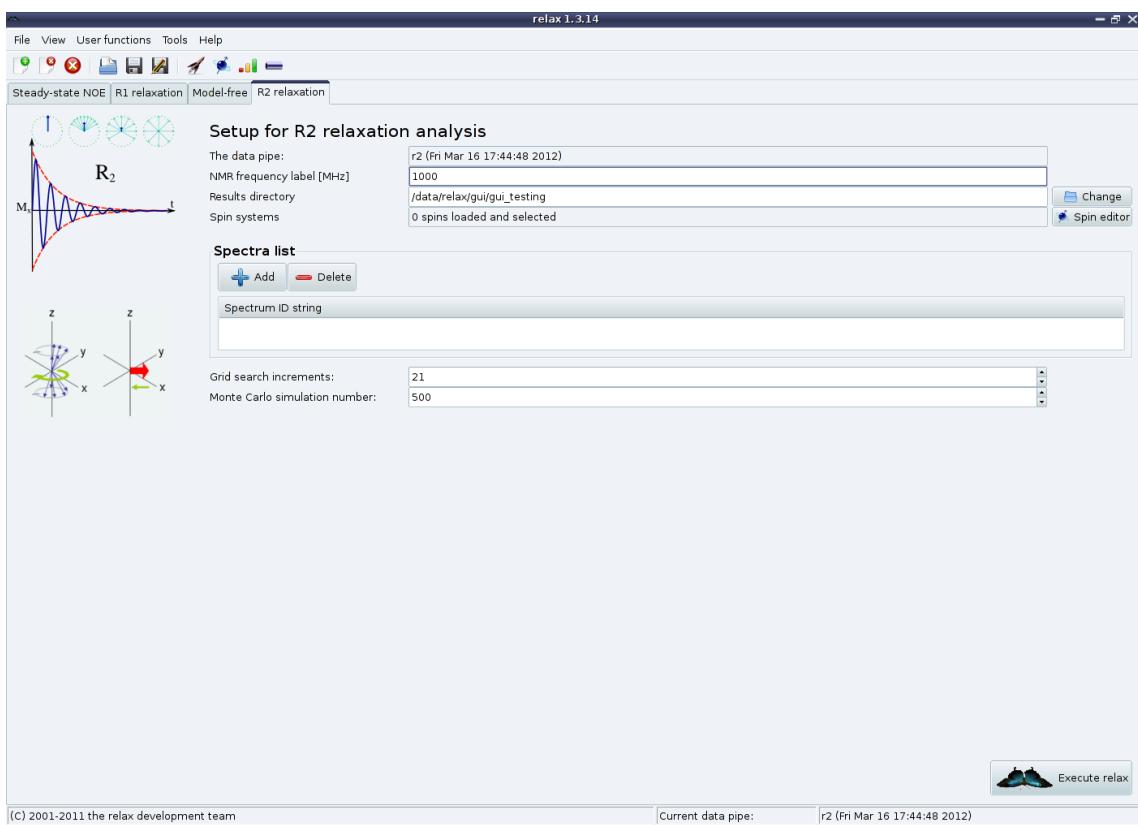


Figure 1.7: Screenshot of the relax GUI interface – the R_2 analysis.

The `ds` object is a dictionary type 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 lib.geometry.rotations import euler_to_R_zyz
relax> from numpy import float64, zeros
relax> R = zeros((3,3), float64)
relax> euler_to_R_zyz(alpha, beta, gamma, R)
relax> print(R)
[[ -0.494666415429033 -0.557373756841289 -0.666813041737502]
 [ 0.219125193028791 -0.822460914570202  0.524921131013452]
 [-0.84100492699311   0.113545317776532  0.528978424497956]]
relax>
```

1.3 The multi-processor framework

1.3.1 Introduction to the multi-processor

Thanks to Gary Thompson's multi-processor framework, relax can be run on multi-core/multi-CPU systems or on clusters to speed up calculations. As most analyses are

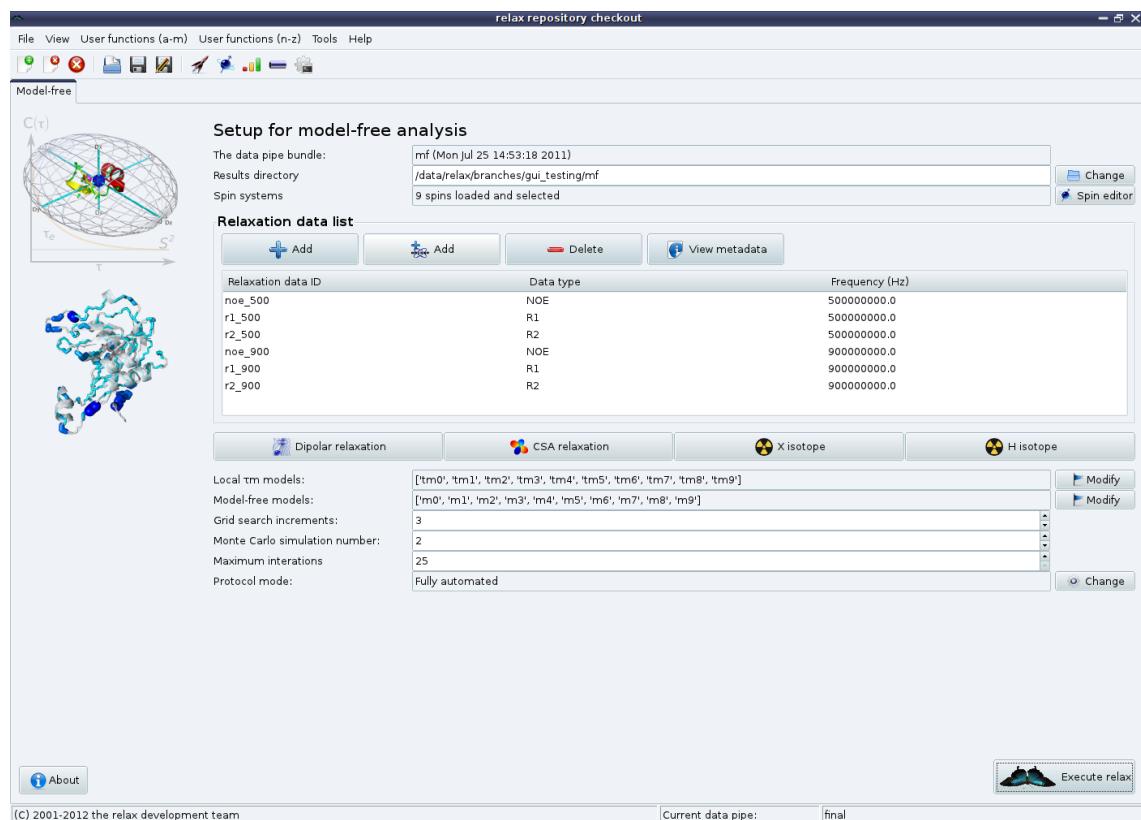


Figure 1.8: Screenshot of the relax GUI interface – the automated model-free analysis. The analysis is fully automated via a new model-free protocol as described in detail in Chapter 7. Clicking on the “About” button in the bottom left hand corner will give a full description of the protocol. For using this interface or any of the modern-day model-free protocols, data from at least two magnetic field strengths must be without question collected.

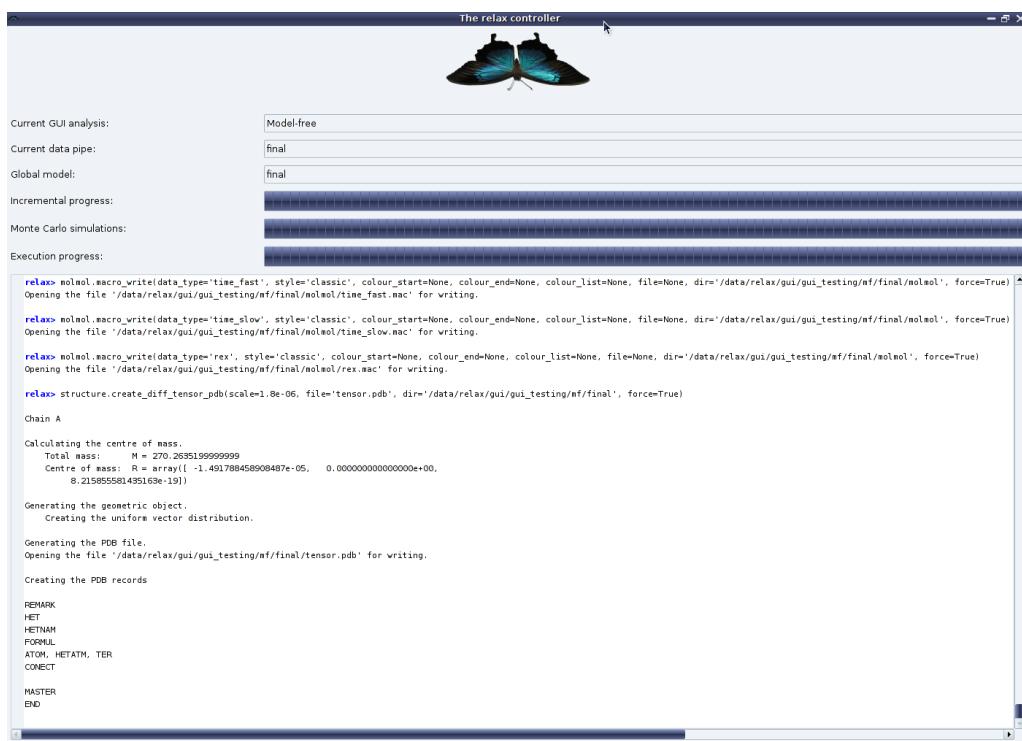


Figure 1.9: Screenshot of the relax GUI interface – the relax controller window. The purpose of the controller is for feedback. It shows the current analysis and current data pipe, a number of progress gauges, and the relax text output.

relatively quick and would not benefit from the multi-processor framework, only the model-free and frame order analyses have currently been parallelised to run within this framework. To use the multi-processor framework, the following should be installed:

OpenMPI: This is the most commonly used Message Passing Interface (MPI) protocol software. The rest of this manual will assume that this is the implementation in use. If another implementation is used, please see the specific documentation for that software for how to set up a program to run via MPI.

mpi4py: This dependency is essential for running in MPI mode in relax. If you would like to use another Python implementation to access the MPI protocol, please consider becoming a relax developer.

1.3.2 Usage of the multi-processor

If you have access to a 256 node cluster and can run calculations on all nodes, assuming that the `dauvergne_protocol.py` automated model-free analysis sample script will be used (after modification for the system under study), relax can be executed by typing:

```
$ mpirun -np 257 /usr/local/bin/relax --multi='mpi4py' --tee log dauvergne_protocol.py
```

Note that the argument `-np` value is one more than the number of slaves you would like to run. You should then see the following text in the initial relax printout:

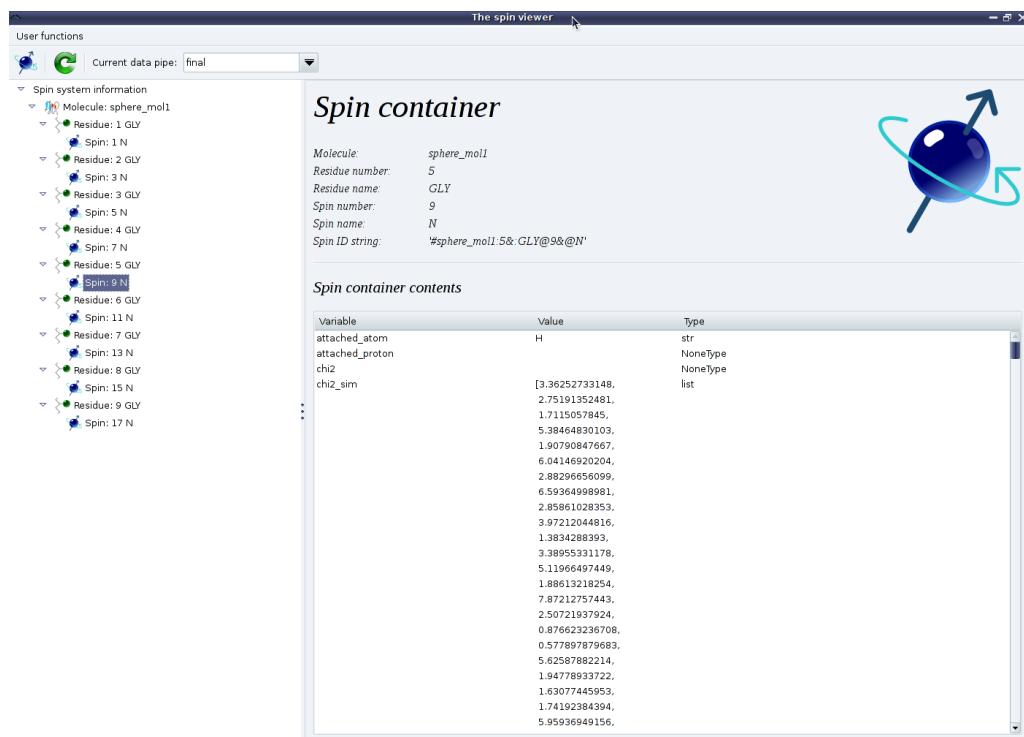


Figure 1.10: Screenshot of the relax GUI interface – the spin viewer window. This viewer is designed for easy addition and manipulation of spin systems within the relax data store. The window is accessible via the “View→Spin viewer” menu entry, typing “[Ctrl-T]”, the spin viewer button in the toolbar, or the “spin editor” button within the auto-analysis tabs.

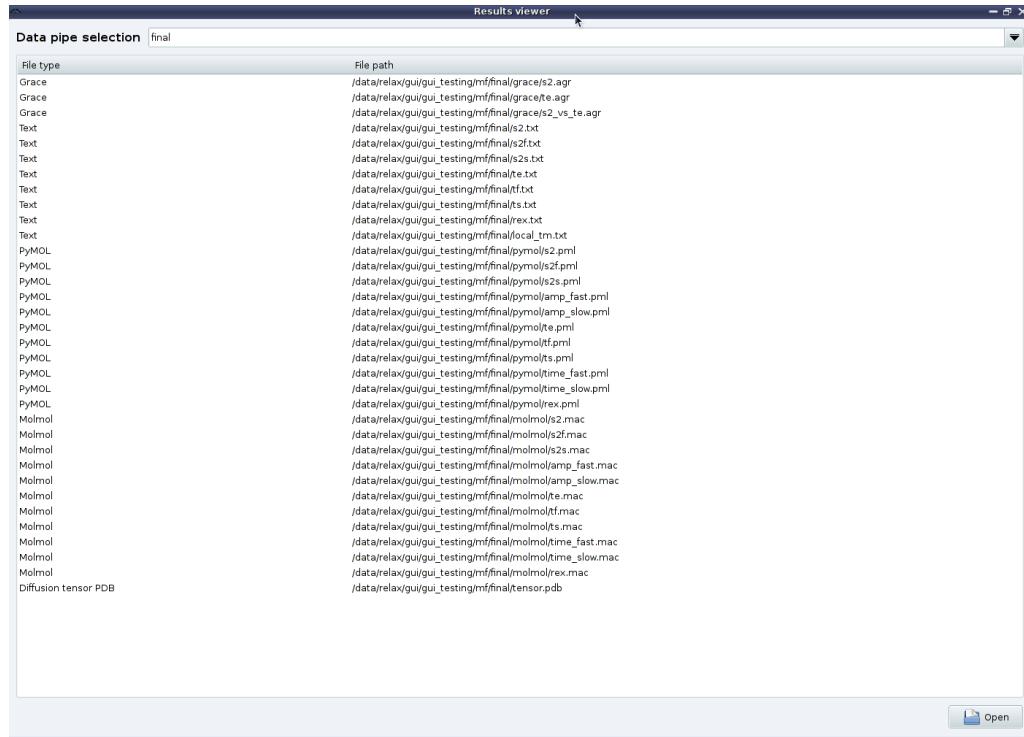


Figure 1.11: Screenshot of the relax GUI interface – the results viewer window. At the end of one of the automated analyses, a number of results files will be created. This can include text files containing the results, 2D Grace plots of the results, PyMOL and MOLMOL macros plotting the results onto the structure, diffusion tensor objects for viewing in PyMOL, etc. This window allows for easy opening of these results files.

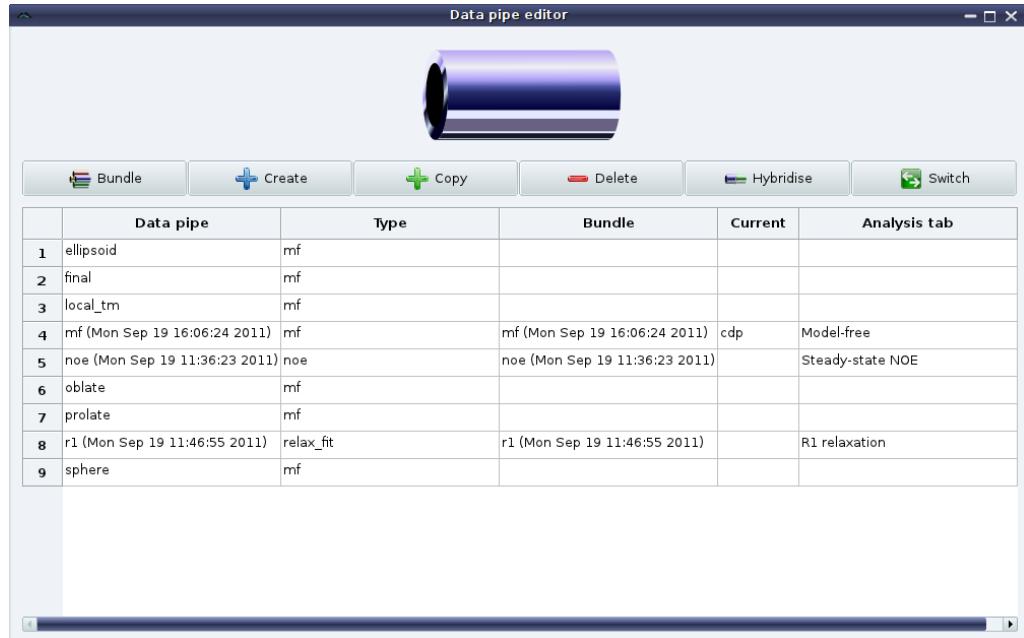


Figure 1.12: Screenshot of the relax GUI interface – the pipe editor window. One analysis may consist of one or more data pipes. And each analysis has its own unique set of data pipes. This editor allows for the easy manipulation of data pipes for advanced users.

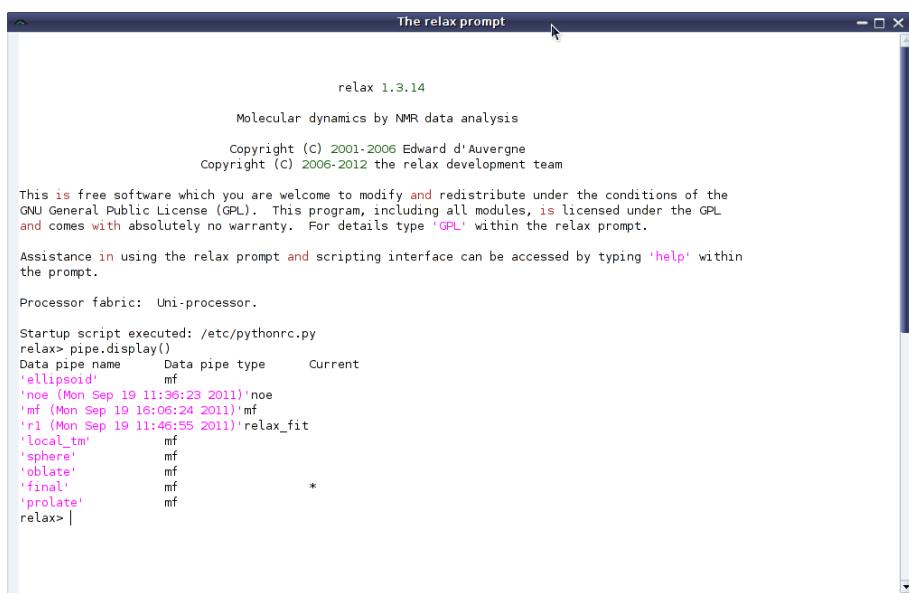


Figure 1.13: Screenshot of the relax GUI interface – the prompt window. This window mimics relax in the prompt user interface mode, and provides the full power of the prompt/script UI modes within the GUI.

Processor fabric: MPI 2.1 running via mpi4py with 256 slave processors & 1 master. Using Open MPI 1.4.3.

1.3.3 Further details

For a full description of the multi-processor framework and how to use it, please see Gary Thompson's official archived announcement to the [relax-devel mailing list](#).

1.4 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 2.5 or higher.

NumPy: Version 1.6 or higher. This package is used for most of the numerical calculations within relax.

SciPy: Version 0.7.1 or higher. This package is optional. It is required only for the frame order theory analyses.

wxPython: Version 2.9 or higher. This package is also optional. It is required for the operation of the graphical user interface (GUI).

mpi4py: Version 1.2 or higher. This optional dependency is essential for running relax in MPI multi-processor mode.

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](#). That way a solution may be created for the next relax release.

Note that only the official Python distribution from <http://python.org> is supported. If you use the Enthought Python Distribution (EPD) or other non-official distributions you may encounter problems with the relax C modules, the graphical user interface, or other issues. These alternative distributions are to be used at your own risk. Any issues encountered will not be considered as relax bugs.

2.2 Installation

2.2.1 The source releases

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`, `*.pyd` or `*.dylib` which are loaded as modules into relax. To build these modules the Scons system from <http://scons.org/> is required. This software requires the Python and numpy header files installed. Once Scons 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](#).

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 possibly creating the byte-compiled Python `*.pyc` files to speed up the start time of relax by typing

```
$ python -m compileall .
```

in the relax base directory. Alternatively if the Scons system is installed, by typing as the root user

```
$ scons install
```

in the relax base directory, a directory in `/usr/local/` called `relax` will be created, all the uncompressed and untarred files will be copied into this directory, a symbolic link in `/usr/local/bin` to the file `/usr/local/relax/relax` will be created, and then finally the Python `*.pyc` files will be byte-compiled. To change the installation path to a non-standard location the Scons 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: Any version.

ctypes: The pyreadline package requires ctypes.

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 “Environment 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:\Python27`”, 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
```

Note that the pre-compiled binary distribution was built using a specific Python version and that that version may need to be installed for the modules to be loaded. More details are given on the [download](#) webpage.

2.2.4 Installation on Mac OS X

There are three ways of installing relax on a Mac. These are described at <http://www.nmr-relax.com/download.html> and are the pre-compiled relax application, the Fink or the source releases.

The relax application

The stand-alone relax application requires none of the dependencies listed above to be installed. It is a universal binary compiled for the i386, x86-64 and PPC CPU architectures (fat3) using the Mac OS X 10.5 framework. It should therefore run on Leopard, Snow Leopard, and Lion. This very large bundle does not require system administrator access to run.

Fink

Certain relax versions are available for Mac OS X within the Fink project. These can be installed for Python 2.7 with the command:

```
> fink install relax-py27
```

The relax releases packaged within Fink can be browsed at <http://pdb.finkproject.org/pdb/browse.php?name=relax>. If the desired version is not available, please download the relevant source package below or contact the fink project using the “Maintainer” email address given in the relax fink pages.

Note that when installing via fink, all the dependencies will be automatically selected and installed as well. Although automatic, when starting from scratch that there could be well over 250 source packages that need to be compiled (to set up the full GNU compilation chain and other libraries which are then required to build Python, numpy, scipy, etc.). This may take anywhere between 2 days to over a week (don’t forget to mention this fact to your poor sys-admin).

The fink relax packages for different Python versions are [relax-py27](#), [relax-py26](#), [relax-py25](#) and [relax-py24](#).

Source release

See Section [2.2.1](#) on page [23](#).

2.2.5 Installation on your OS

For all others systems, please use the source distribution files and the Scons software to build the C modules.

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 R₁ and R₂ 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. This program can also be used to display geometric objects generated by relax for representing physical concepts such as the diffusion tensor and certain cone diffusion models.

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

Free software infrastructure

3.1 History

Starting with the initial code in November 2001, the relax sources were not stored within a version control repository. Instead version control was performed by creating a log-less `*.tar.gz` file backup after each change. In June 2005, these backup files were imported into a new Subversion (SVN) repository.

Corresponding with the switch to the SVN version control repository, relax was made public by shifting development onto the [Gna! free software infrastructure](#). This allowed for the following new infrastructure to be set up:

- Hosting for the relax website.
- File download services for the relax distribution files (and PDF manual).
- The relax mailing lists.
- Access to the relax source code.
- Bug, support request, task, and patch trackers.
- News feed.

An archive of relax's old Gna! website can be found on the Internet Archive.

In May 2017, without much warning, the [Gna! infrastructure](#) that relax relied upon was permanently shut down. This lead to a long period with no open source infrastructure. During this time, the relax SVN repository was painstakingly converted into a git version control repository with all branches and history preserved. To ensure that relax would be accessible for a long time into the future, this new git repository was mirrored to a number of free software/open source infrastructures:

- relax at Bitbucket
- relax at GitHub

- [relax at GitLab](#)
- [relax at SourceForge](#)

The webpages were also migrated with history from the SVN repository where they were located alongside the relax source code to a separate git repository.

From January 2019, relax moved to the [SourceForge open source infrastructure](#). This provides the following relax infrastructure.

- Hosting for the relax website.
- File download services for the relax distribution files (and PDF manual).
- The relax mailing lists.
- Access to the relax source code, web pages, and relax demo files.
- Bug, support request, and task trackers.
- SVN support for hosting the old and archived SVN repository.
- Backend shell log in (shell services).
- MySQL and PHP support (possibly allowing for the relax Mediawiki to be migrated here in the future).

3.2 The relax web sites

The main web site for relax is <http://www.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.

3.3 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.

3.3.1 relax-announce

The relax announcement list “nmr-relax-announce at lists.sourceforge.net” 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. The mailing list links are: [Subscribe](#), [Archive](#), [Search](#).

3.3.2 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 “nmr-relax-users at lists.sourceforge.net” is the place to post your message. The mailing list links are: [Subscribe](#), [Archive](#), [Search](#).

3.3.3 relax-devel

A second mailing list exists for posts relating to the development of relax. Feature requests, program design, or any other posts relating to relax’s structure or code should be sent to this list instead. The list is “nmr-relax-devel at lists.sourceforge.net” and the relevant links are: [Subscribe](#), [Archive](#), [Search](#).

3.3.4 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, please subscribe to this list. The mailing list links are: [Subscribe](#), [Archive](#), [Search](#).

3.3.5 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.

3.4 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](#). Please [submit a relax bug here](#) rather than reporting 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 through the bugs here](#).

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 (Chapter 13 on page 277), 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.

3.5 Latest sources – the relax repositories

relax’s source code is kept within a version control system called [git](#). This system allows for 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 [git Reference Manual](#) is a good place to start. The contents of the relax repository can be viewed online at <https://sourceforge.net/p/nmr-relax/code/ci/master/tree/>. The current sources can be downloaded using the git protocol by typing

```
$ git clone git://git.code.sf.net/p/nmr-relax/code relax
```

In addition there is a git repository for the [relax website](#) and a git repository for the relax [demonstration files](#).

3.6 The relax distribution archives

The relax distribution archives are the files to [download to install 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 a git repository ‘tag’. To check out version 4.0.3, for example, within an existing git repository clone type

```
$ git checkout 4.0.3
```

The binary distribution can then be created for your architecture by typing

```
$ 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 (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 <https://sourceforge.net/projects/nmr-relax/files/>.

Chapter 4

The relax data model

4.1 The concept of the relax data model

To begin to understand how to use relax, a basic comprehension of the relax data model is needed. The data model includes the concepts of the relax data store, the data pipes, the molecule, residue and spin data structures and the interatomic data containers. These concepts are independent of the specific analyses presented in the next chapters and are important for setting up relax.

4.2 The data model

4.2.1 The relax data store

All permanent data handled by relax is kept in a structure known as the relax data store. This structure is initialised when relax is launched. The data store is primarily organised into a series of objects known as data pipes, and all usage of relax revolves around the flow of information in these data pipes.

Data pipes



The first thing one must do when relax is launched is to create a data pipe. When using the GUI, a base data pipe will be created when opening one of the automatic analyses via the analysis selection window (see figure 1.4 on page 12). This will also create a data pipe bundle for the analysis (*vide infra*). Alternatively the data pipe editor window can be

used to create data pipes (see figure 1.12 on page 21). For the prompt/scripting modes, or the “User functions→pipe→create” menu entry, a data pipe can be initialised by specifying the unique name of the data pipe and the data pipe type:

```
1 pipe.create(pipe_name='NOE 1200 MHz', pipe_type='noe')
```

A number of relax operations will also create data pipes by merging a group of pipes or branching pre-existing pipes. See section 1.2.6 on page 10 for additional details.

All data not associated with spin systems will be stored in the base data pipe. This includes information such as global optimisation statistics, diffusion tensors, alignment tensors, 3D structural data, the molecule, residue and spin container data structure and the interatomic data containers. One data pipe from the set will be defined as being the current data pipe, and all operations in relax will effect data from this pipe. The `pipe.switch` user function in all UI modes can be used to change which pipe is the current data pipe. In the GUI, switching between analysis tabs will automatically switch the current data pipe to match the analysis being displayed.

Data pipe bundles



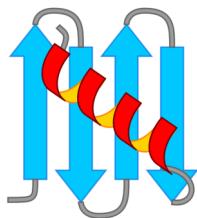
Related data pipes can be grouped into a ‘bundle’. For example if the data pipes “sphere”, “oblate spheroid”, “prolate spheroid”, and “ellipsoid” preexist, these can be grouped into a bundle called “diffusion tensors” with the following series of user function calls:

```
1 pipe.bundle(bundle='diffusion tensors', pipe='sphere')
2 pipe.bundle(bundle='diffusion tensors', pipe='oblate spheroid')
3 pipe.bundle(bundle='diffusion tensors', pipe='prolate spheroid')
4 pipe.bundle(bundle='diffusion tensors', pipe='ellipsoid')
```

The data pipe editor window of the GUI can also be used to bundle pipes together (see figure 1.12 on page 21).

4.2.2 Molecule, residue, and spin containers

Within a data pipe is the molecule, residue, and spin container data structure. Data which is specific to a given nucleus is stored in a special spin container structure. This includes relaxation data, model-free parameters, reduced spectral density mapping values, spin specific optimisation parameters, chemical shift tensor information, pseudo-contact shift values, etc. The spin containers can be created from 3D structural data or a sequence file, as described in the next two sections, or manually built.



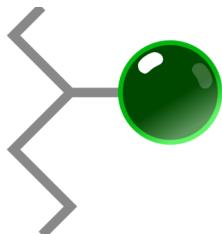
Molecule containers

The spin containers are part of a nested set of containers, and are graphically depicted in the spin viewer window of the GUI in figure 1.10 on page 20. As can be seen from the figure, the top level holds a single molecular container. Multiple molecular containers can be present if the study is of a molecular complex. Using the GUI menus or the prompt/scripting mode, molecule containers can be manually created with the user function:

```
1 molecule.create(mol_name='glycerol', mol_type='organic molecule')
```

In the spin viewer window of the GUI, right clicking on the “Spin system information” element will pop up a menu with an entry for adding molecule containers. Right clicking on molecule containers will show a pop up menu with an entry for permanently deleting the container.

Residue containers



Nested within the molecule containers are residue containers. These are graphically depicted in the spin viewer window (see figure 1.10 on page 20). Each molecule container can possess multiple residues. These require either a unique residue number or unique residue name. For organic molecules where the residue concept is meaningless, all spin containers can be held within a single unnamed and unnumbered residue container. Using the GUI menus or the prompt/scripting mode, residue containers can be manually created with the user function:

```
1 residue.create(res_num='-5', res_name='ASP')
```

Alternatively residues can be added in the spin viewer window from the pop up menu when right clicking on molecule containers, and can be deleted via the pop up menu when right clicking on the residue to delete.

Spin containers



Spin containers are nested within a residue container (again graphically depicted in the spin viewer window in figure 1.10 on page 20). Multiple spin containers can exist per residue. This allows, for example, a single model-free analysis simultaneously on the backbone nitrogen spins, side-chain tryptophan indole nitrogen spins and alpha carbon spins. Or, for example, studying the pseudocontact shifts for all nitrogen, carbon and proton spins in the molecule simultaneously.

Spin containers can be manually added via the `spin.create` user function in the GUI or prompt/scripting mode:

```
1 spin.create(spin_num='200', spin_name='NE1')
```

The spin viewer window can also be used by right clicking on residue containers.

Spin ID strings

Spins are often identified in relax using their ID strings. The spin ID strings follow the basic construct found in a number of other NMR software such as MOLMOL. The identification string is composed of three components:

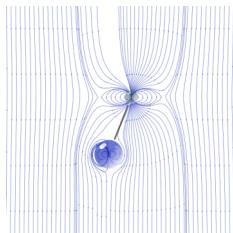
- The molecule ID token beginning with the “#” character,
- The residue ID token beginning with the “:” character,
- The atom or spin system ID token beginning with the “@” character.

Each token can be composed of multiple elements – one per spin – 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can, in some instances, be used to select spins. For example the string “@H*” will select the protons ‘H’, ‘H2’ and ‘H98’.

4.3 Interatomic data containers



Separate from the spin containers, yet strongly linked to them, are the interatomic data containers. These containers are grouped together within the same data pipe as the spins they point to. These define interactions between two spins located anywhere within the molecule, residue and spin nested data structure. These are automatically created when reading in data defined between two spins such as RDCs and NOE distance constraints. They can also be created using the `interatom.define` user function:

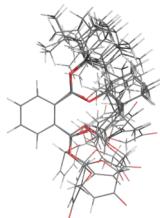
```
1 interatom.define(spin_id1=':2@N', spin_id2=':2@H')
```

As the interatomic data container concept is relatively new, how they are created and handled is likely to evolve and change in the future.

4.4 Setup in the prompt/script UI

Below are three different examples showing how to set up the relax data model for any analysis type requiring spin specific data.

4.4.1 Script mode – spins from structural data



3D structural data is stored at the level of the current data pipe. This data is completely separate from the molecule, residue and spin data structure. However the structural data can be used to generate the spin containers. For example for the nitrogen relaxation in a model-free analysis where both the nitrogen and proton are needed to define the magnetic dipole-dipole relaxation:

```
1 # Create a data pipe.
2 pipe.create(pipe_name='ellipsoid', pipe_type='mf')
3
4 # Load the PDB file.
5 structure.read_pdb('1f3y.pdb')
6
7 # Set up the 15N and 1H backbone spins.
```

```

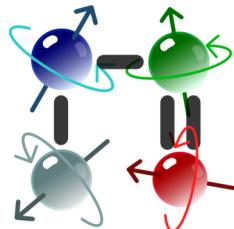
8  structure.load_spins('@N', ave_pos=True)
9  structure.load_spins('@H', ave_pos=True)
10
11 # Set up the 15N and 1H for the tryptophan indole ring.
12 structure.load_spins('@NE1', ave_pos=True)
13 structure.load_spins('@HE1', ave_pos=True)
14
15 # Define the spin isotopes.
16 spin.isotope('15N', spin_id='@N*')
17 spin.isotope('1H', spin_id='@H*')

```

The `structure.read_pdb` user function will load the structural data into the current data pipe, and the `structure.load_spins` user function will create the molecule, residue, and spin containers as needed. This will also load atomic position information into the matching spin containers. The `spin.isotope` user function is required to define the magnetic dipole-dipole interaction and is information not present in the PDB file.

Note that if structural data from the PDB is used to generate the spin containers, then all subsequent data loaded into relax must follow the exact naming convention from the PDB file. Automatic residue name matching (i.e. ‘GLY’ = ‘Gly’ = ‘gly’ = ‘G’) is currently not supported.

4.4.2 Script mode – spins from a sequence file



Alternatively to setting up the molecule, residue, and spin containers via 3D structural data, a plain text columnar formatted file can be used. This is useful for when no 3D structure exists for the molecule. It also has the advantage that the residue and atom names need not conform to the PDB standard. An example for reading sequence data is:

```

1 # Create a data pipe.
2 pipe.create(pipe_name='R1_1200', pipe_type='relax_fit')
3
4 # Set up the 15N spins.
5 sequence.read(file='noe.500.out', mol_name_col=1, res_num_col=2, res_name_col=3,
   spin_num_col=4, spin_name_col=5)
6 spin.element(element='N', spin_id='@N*')
7 spin.isotope('15N', spin_id='@N')

```

Here the molecule, residue, and spin information is extracted from the “`noe.500.out`” file which could look like:

# mol_name	res_num	res_name	spin_num	spin_name	value	error
Ap4Aase_new_3_mol1	1	GLY	1	N	None	None
Ap4Aase_new_3_mol1	2	PRO	11	N	None	None
Ap4Aase_new_3_mol1	3	LEU	28	N	None	None
Ap4Aase_new_3_mol1	4	GLY	51	N	0.03892194698453	0.01903177024613

Ap4Aase_new_3_mol1	5	SER	59	N	0.31240422567912	0.01859693729836
Ap4Aase_new_3_mol1	6	MET	71	N	0.42850831873249	0.0252585632304
Ap4Aase_new_3_mol1	7	ASP	91	N	0.53054928103134	0.02799062314416
Ap4Aase_new_3_mol1	8	SER	104	N	0.56528429775819	0.02170612146773
Ap4Aase_new_3_mol1	9	PRO	116	N	None	None
Ap4Aase_new_3_mol1	40	TRP	685	N	0.65394813490548	0.03830061886537
Ap4Aase_new_3_mol1	40	TRP	698	NE1	0.67073879732046	0.01426066343831

The file can contain columns for the molecule name, the residue name and number, and the spin name and number in any order though not all are needed. For example for a single protein system, the molecule name, residue name and spin number are nonessential. Or for an organic molecule, the molecule name, residue name and number and spin number could be nonessential. The subsequent user functions in the above example are used to set up the spin containers appropriately for a model-free analysis. These are not required in the automatic analysis of GUI as these user functions will be presented to you when adding relaxation data, or when clicking on the heteronucleus and proton buttons (“X isotope” and “H isotope”).

In the GUI, the creation of molecule, residue, and spin containers from a sequence file is also available via the “Load spins” wizard within the spin viewer window (*vide supra*).

4.4.3 Script mode – manual construction

For the masochists out there, the full molecule, residue and spin data structure can be manually constructed. For example:

```

1 # Manually create the molecule, residue, and spin containers.
2 molecule.create(mol_name='Ap4Aase', mol_type='protein')
3 residue.create(res_num=1, res_name='GLY')
4 residue.create(res_num=3, res_name='LEU')
5 residue.create(res_num=96, res_name='TRP')
6 spin.create(res_num=1, spin_name='N')
7 spin.create(res_num=3, spin_name='N')
8 spin.create(res_num=96, spin_name='N')
9 spin.create(res_num=96, spin_name='NE1')
```

These user functions can be repeated until the full sequence has been constructed.

4.5 Setup in the GUI

4.5.1 GUI mode – setting up the data pipe

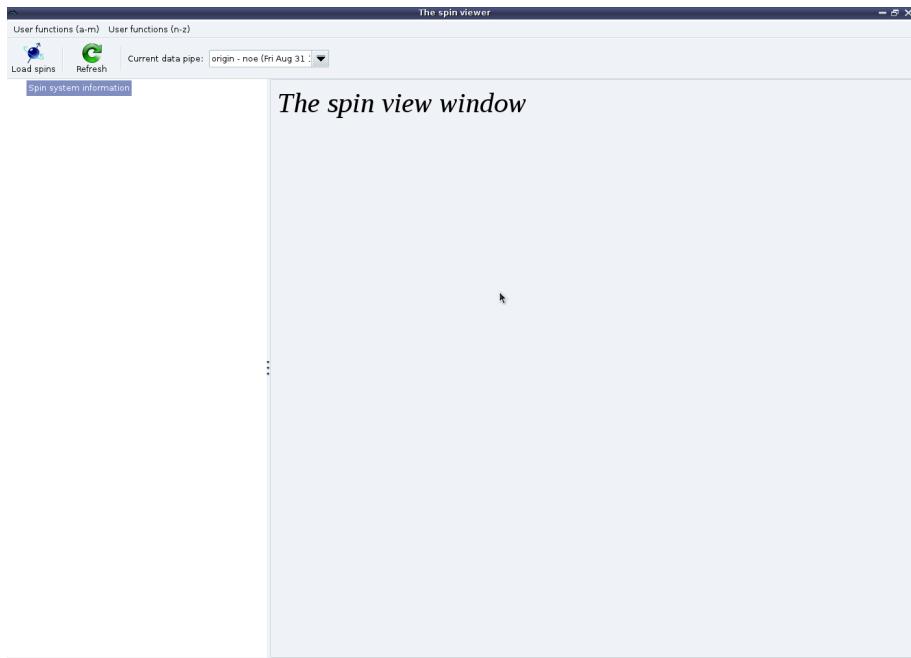
In the GUI, the most common way to create the data pipe is to initialise one of the auto-analyses via the analysis selection wizard (see Figure 1.4 on page 12). The initialisation will create the appropriate starting data pipe. Alternatively the data pipe editor can be used (see Figure 1.12 on page 21). Or the “User functions→pipe→create” menu item can be selected for graphical access to the `pipe.create` user function.

4.5.2 GUI mode – spins from structural data

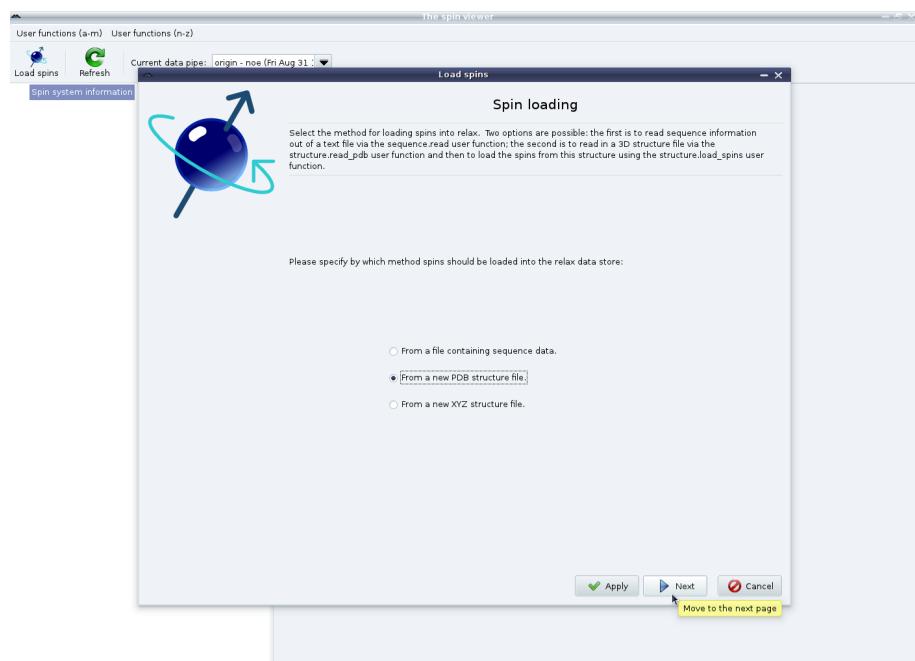
For this section, the example of protein ^{15}N relaxation data will be used to illustrate how to set up the data structures. To manipulate the molecule, residue and spin data structures in the GUI, the most convenient option is to use the spin viewer window (see Figure 1.10 on page 20). This window can be opened in four ways:

- The “View→Spin viewer” menu item,
- The “[Ctrl+T]” key combination,
- The spin viewer icon in the toolbar (represented by the blue spin icon),
- The “Spin editor” button part of the “Spin systems” GUI element in the specific analysis tabs.

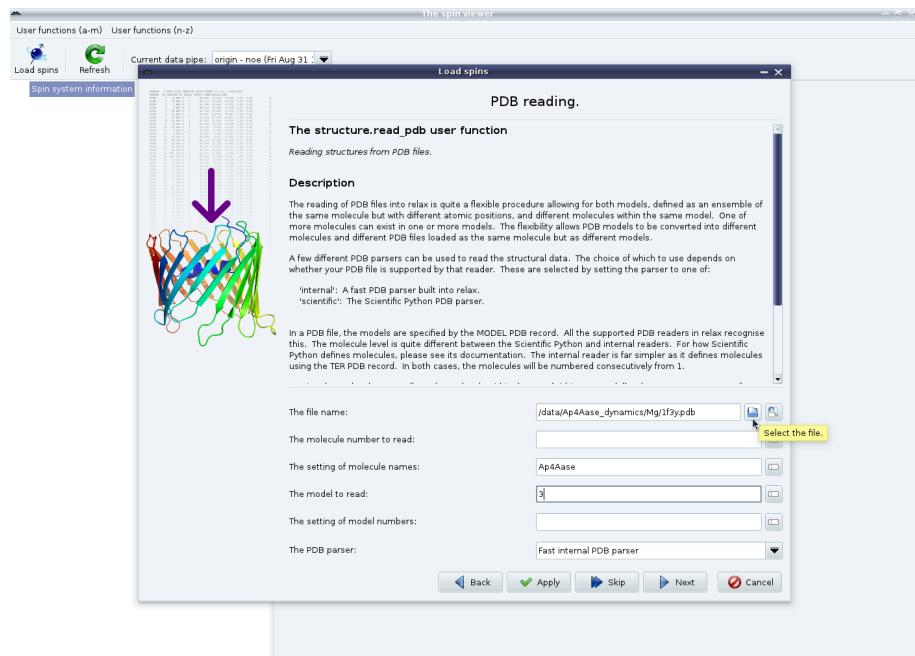
You will then see:



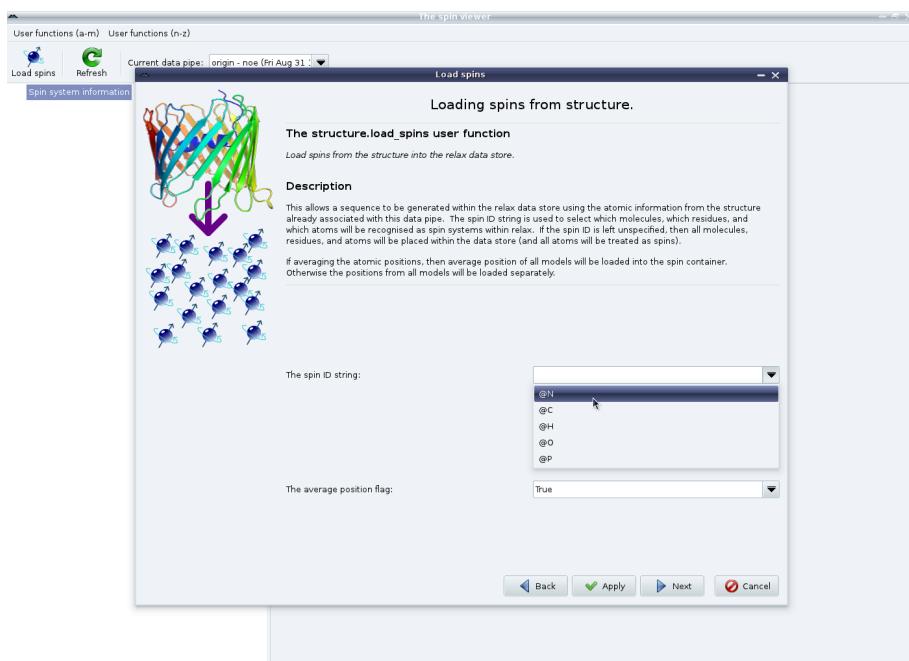
At this point, click on the “Load spins” button (or the “Load spins” menu entry from the right click pop up menu) to launch the spin loading wizard. A number of options will be presented to you:



Here the spins will be loaded from a PDB file. If you do not have a 3D structure file, please see the next section. After selecting “From a new PDB structure file” and clicking on “Next”, you will see:



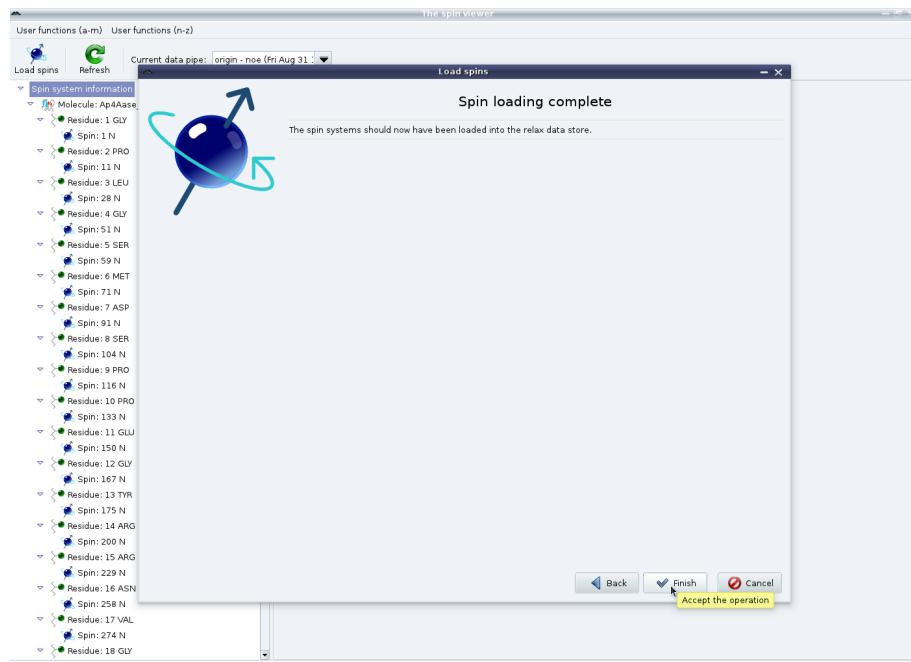
Now select the PDB file you wish to use. The other options in this screen allow you to handle NMR models and multiple molecules within a single PDB file. These options are explained in the window. Hovering the mouse over the options will give additional hints. In this example, the 3rd model from the 1F3Y PDB file will be read and the single molecule will be named “Ap4Aase” to override the default naming of “1f3y_mol1”. Now click on “Next” to bring up the spin loading page:



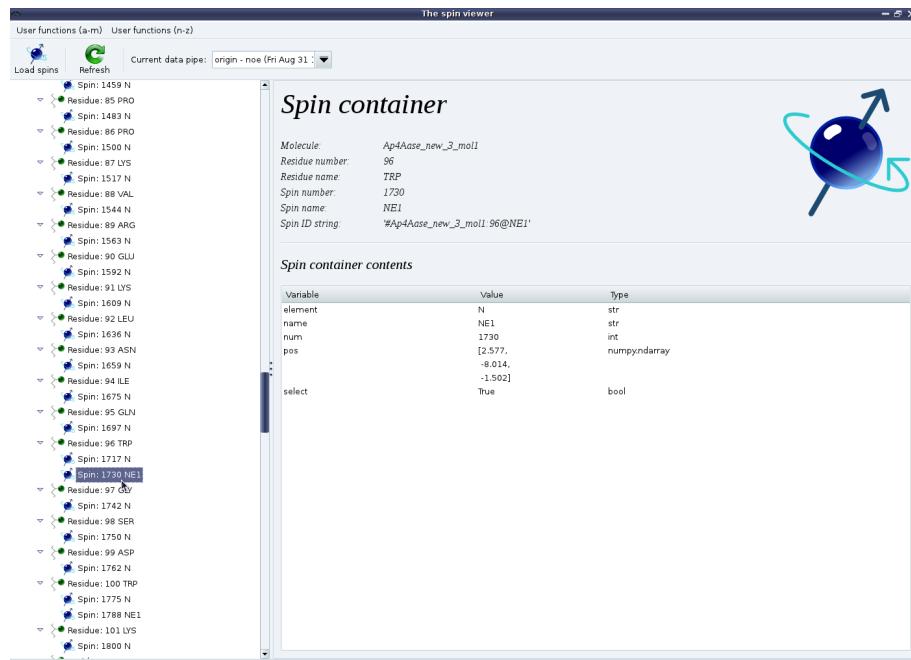
This is a bit more complicated. In this example we are studying the backbone dynamics of ^{15}N spins of a protein. Therefore first set the spin ID string to “@N” (which can be selected from the pull down) and click on “Apply” to set up the backbone spins. Do not click on “Next” yet. If the current study requires the specification of the dipole-dipole interaction (for example if it involves relaxation data – model-free analyses, consistency testing, reduced spectral density mapping; or the dipolar coupling – the N-state model or ensemble analyses, the Frame Order theory) you will also need to load the ^1H spins as well. Therefore set the spin ID string to “@H” and click on “Apply” again.



Now change the spin ID string to “@NE1” and then click on “Next” (or “Apply” if the Trp protons “@HE1” need to be loaded as well). This will add spin containers for the tryptophan indole ^{15}N spins. Finally click on “Finish” to exit the wizard:



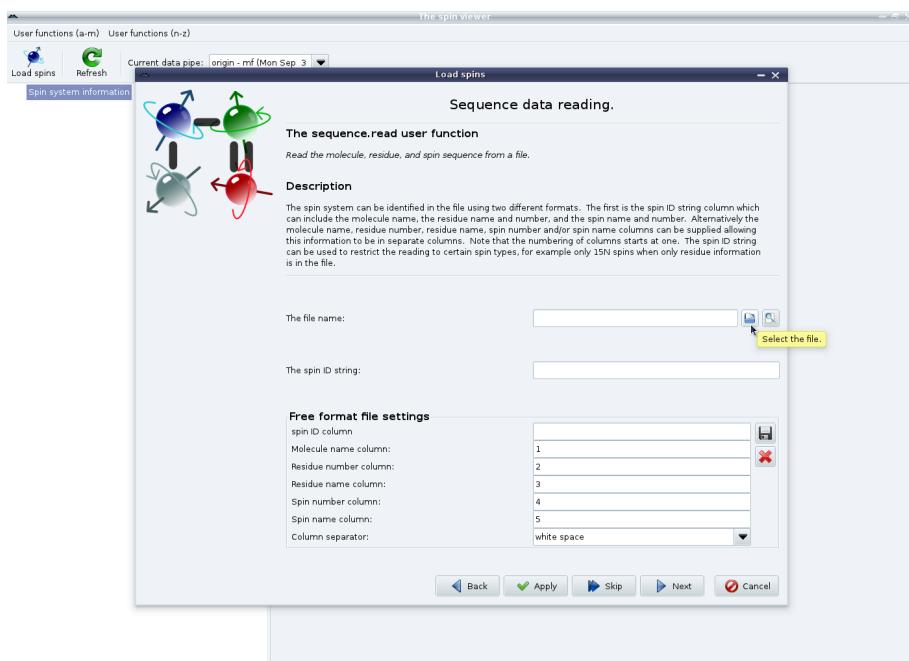
You should now see something such as:



If the ^1H spins have been loaded as well, then you should see exactly twice as many spin containers as shown above.

4.5.3 GUI mode – spins from a sequence file

Starting from the empty spin viewer window on page 42), click on the “Load spins” button. You will then see the spin loading wizard (see page 43). Select the option for reading data from a sequence file. You should then see:



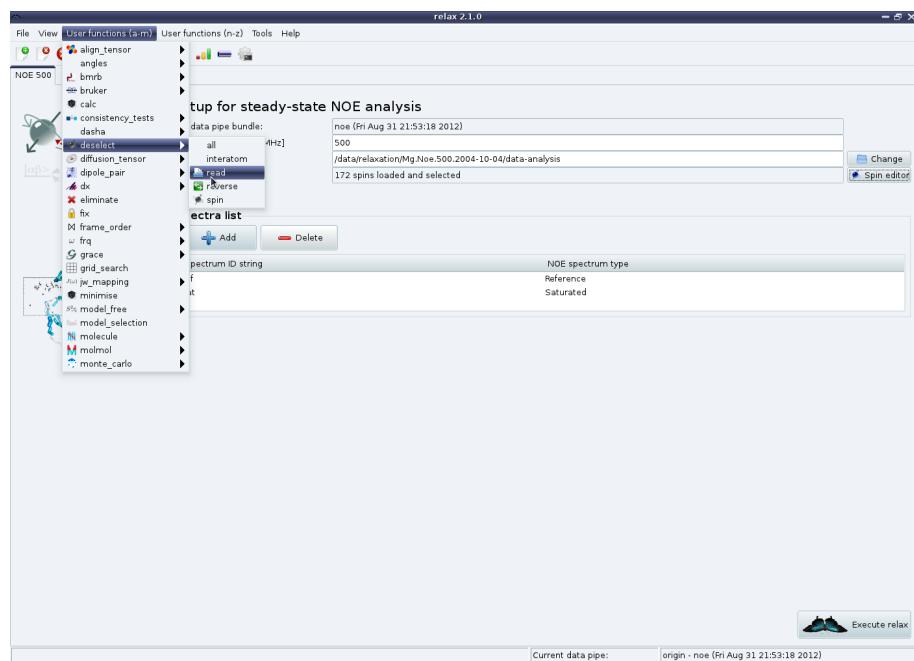
Select the file to load and change the “Free format file settings” as needed. An example of a suitable format is given on page [41](#). Click on “Next” to reach the wizard ending page (see [45](#)). Finally click on “Finish” to exit the wizard.

4.5.4 GUI mode – manual construction

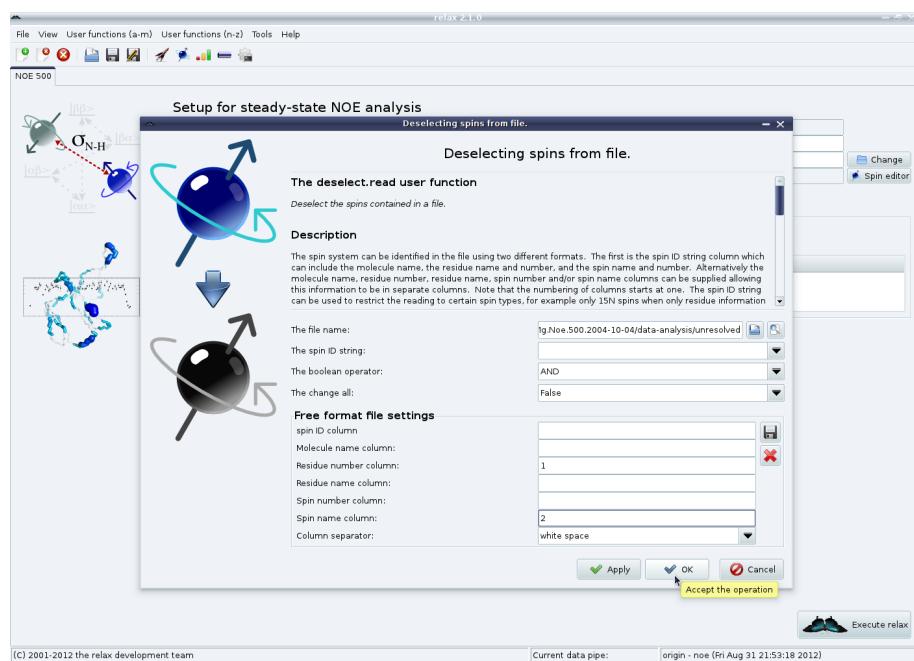
Just as in the prompt/script UI mode, the molecules, residues and spins can be manually added. First add a molecule by right clicking on the “Spin system information” element and selecting the relevant entry in the popup menu. Then right click on the newly created molecule container to add residues, and right click on residue containers to add spins.

4.5.5 GUI mode – deselect spins

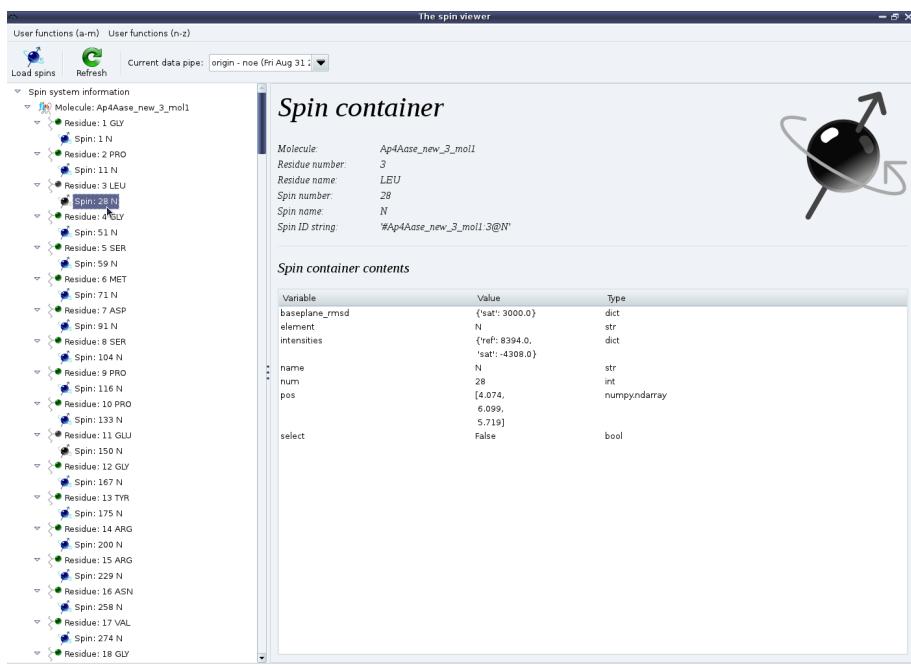
To deselect spins (for example if they are unresolved, overlapping peaks), click on the “User functions→deselect→read” menu item from the main relax window or the spin viewer window:



Select the file listing the unresolved spins and change the column numbers in the “Free format file settings” GUI element as needed:



Alternatively the spin editor window can be reopened and the spins manually deselected by right clicking on them and selecting “Deselect”. Returning to the spin editor window, you should now see certain spins coloured grey:



4.6 The next steps

This chapter presented the basics of setting up the relax data store, concepts which are needed for all analysis types built into relax. The next chapters will introduce specific analyses types – the steady-state NOE, R₁ and R₂ relaxation curve-fitting, and the automated full model-free analysis protocol of [d'Auvergne and Gooley \(2007, 2008c\)](#) – which build on the ideas introduced here.

Part II

The specific analyses

Chapter 5

The R_1 and R_2 relaxation rates – relaxation curve-fitting



5.1 Introduction to relaxation curve-fitting

The fitting of exponentials to relaxation curves (relaxation curve-fitting or as used throughout this chapter abbreviated simply as relax-fit) 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 exponential curves during the fit for each observed spin, 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 exponential curve models

A number of different models are supported in this analysis. These include the two parameter exponential decay to zero, the inversion recovery experiment, and the saturation recovery experiment. These can be selected using the `relax_fit.select_model` user function.

The default is the two parameter exponential decay whereby the magnetisation starts at I_0 and decays to zero. It has the parameters $\{R_x, I_0\}$. The formula of this function is

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

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).

In the inversion recovery experiment, the magnetisation starts at a negative value at $-I_0$ and relaxes to a positive I_∞ value. This curve consists of three parameters $\{R_x, I_0, I_\infty\}$. The formula is

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

In the saturation recovery experiment, the magnetisation starts at zero and relaxes to a positive I_∞ value. The model consists of the two parameters $\{R_x, I_\infty\}$ and has the formula

$$I(t) = I_\infty (1 - e^{-R_x \cdot t}). \quad (5.3)$$

5.3 From spectra to peak intensities for the relaxation rates

The following subsections simply contain advice on how to go from the recorded FIDs to the peak lists ready to be input into relax. This need not be followed – it is simply a set of recommendations for obtaining the highest quality relaxation rates.

5.3.1 Temperature control and calibration



Before starting with the spectral processing, it should be noted that proper temperature control and calibration are essential for relaxation data. Small temperature changes can have an effect on the viscosity and hence global tumbling of the molecule being studied and, as the molecular diffusion tensor is the major contributor to relaxation, any non-consistent data will likely lead to artificial motions appearing in subsequent model-free analyses.

Per-experiment temperature calibration is essential and the technique used will need to be specified for BMRB data deposition. Note that the standard MeOH/ethylene glycol calibration of a spectrometer is of no use when you are running experiments which pump in large amounts of power into the probe head. Although the R1 experiment should be about the same temperature as a HSQC and hence be close to the standard MeOH/ethylene glycol spectrometer calibration, the R2 CPMG or spin lock and, to a lesser extent, the NOE pre-saturation pump a lot more power into the probe head. The power differences can either cause the temperature in the sample to be too high or too low. This is unpredictable as the thermometer used by the VT unit is next to the coils in the probe head and not inside the NMR sample. So the VT unit tries to control the temperature inside the probe head rather than in the NMR sample. However between the thermometer and the sample is the water of the sample, the glass of the NMR tube, the air gap where the VT unit controls

air flow and the outside components of the probe head protecting the electronics. If the sample, the probe head or the VT unit is changed, this will have a different affect on the per-experiment temperature. The VT unit responds differently under different conditions and may sometimes over or under compensate by a couple of degrees. Therefore each relaxation data set from each spectrometer requires a per-experiment calibration.

Explicit temperature control techniques are also essential for relaxation data collection. Again the technique used will be asked for by relax for BMRB data deposition. A number of factors can cause significant temperature fluctuations between individual relaxation experiments. This includes the daily temperature cycle of the room housing the spectrometer, different amounts of power for the individual experiments, etc. The best methods for eliminating such problems are single scan interleaving and temperature compensation block. Single scan interleaving is the most powerful technique for averaging the temperature fluctuations not only across different experiments, but also across the entire measurement time. The application of off-resonance temperature compensation blocks at the start of the experiment is useful for the R2 and will normalise the temperature between the individual experiments, but single scan or single fid interleaving is nevertheless required for normalising the temperature across the entire measurement.

5.3.2 Spectral processing

For the best measurement of peak heights across the myriad of NMR spectral analysis software, it is recommend to zero fill a lot – 8k to 16k would give the best results. This does not increase the information content of the spectrum or decrease the errors, it simply interpolates. Even if the NMR spectral software performs 3-point quadratic interpolation between the highest points to determine the peak height, the additional free interpolation will make the estimation more accurate.

Additionally, care must be taken to properly scale the first point as this can cause a baseline roll which will affect peak heights. A very useful description comes directly from the [NMRPipe manual](#):

Depending on the delay, the first point of the FID should be adjusted before Fourier Transform. The first point scaling factor is selected by the window function argument `-c`.

If the required first order phase P1 for the given dimension is 0.0, the first point scaling factor should be 0.5. This is because the discrete Fourier transform does the equivalent of counting the point at $t=0$ twice. If the first point is not scaled properly in this case, ridge-line baseline offsets in the spectrum will result.

In all other cases (P1 is not zero), this scale factor should be 1.0. This is because the first point of the FID no longer corresponds to $t=0$, and so it shouldn't be scaled. If the scale factor is not set correctly, it will introduce a baseline distortion which is either zero-order or sinusoidal, depending on what first-order phase is required. When possible, it is best to set up experiments with either exactly 0, 1/2, or 1-point delay. There are several reasons:

- Phase correction values can be determined easily.

Table 5.1: Summary, First Point Scaling and Phase Correction

Delay	P1	FID	Spectrum
0 point	0	Scale -c 0.5	
1/2 point	180	Scale -c 1.0	Folded peaks have opposite sign
1 point	360	Scale -c 1.0	Use “POLY -auto -ord 0”

- If the delay is not a multiple of 1/2 point, the phase of folded peaks will be distorted.
- The Hilbert transform (HT) is used, sometimes automatically, to reconstruct previously deleted imaginary data for interactive rephasing or inverse processing. But, the HT can only reconstruct imaginary data perfectly if the phase is a multiple of 1/2 point.
- Data with P1 = 360 have the first point t=0 missing (i.e. 1 point delay). Since the first point of the FID corresponds to the sum of points in the corresponding spectrum, this missing first point can be “restored” by adding a constant to the phased spectrum. This can be done conveniently by automated zero-order baseline correction, as shown in table 5.1.

Here is an example NMRPipe script designed for optimal relaxation rate extraction:

```

1 #!/bin/csh
2
3 setenv FILEROOT $1
4 set PHASE=81.4
5
6 echo "\n# Fourier Transform (nmrPipe fid/*.fid to ft/*.dat)"
7 echo "# t2 phase is set to $PHASE"
8 echo "# t1 phase is set to 0.0\n"
9
10 nmrPipe -in fid/$FILEROOT.fid \
11 | nmrPipe -fn SOL \
12 | nmrPipe -fn GM -g1 15 -g2 20 -c 0.5 \
13 | nmrPipe -fn ZF -size 8192 \
14 | nmrPipe -fn FT -auto \
15 | nmrPipe -fn PS -p0 $PHASE -p1 0.0 -di -verb \
16 | nmrPipe -fn TP \
17 | nmrPipe -fn SP -off 0.5 -end 0.98 -pow 2 -c 0.5 \
18 | nmrPipe -fn ZF -size 8192 \
19 | nmrPipe -fn FT -auto \
20 | nmrPipe -fn PS -p0 0.0 -p1 0.0 -di -verb \
21 | nmrPipe -fn TP \
22 | nmrPipe -fn POLY -auto \
23 | nmrPipe -fn EXT -left -sw \
24 | nmrPipe -out ft/$FILEROOT.dat -ov

```

The script is run by supplying the FILEROOT value as a command line option so if the script is called `nmrpipe.sh` and the var2pipe or bruk2pipe processed file `R1_ncyc4.fid` is in the `fid` directory, you would run:

```
$ ./nmrpipe.sh R1_ncyc4
```

The `ft` directory must exist for this script to execute. Different experiment specific options may be needed such as:

```
| nmrPipe -fn REV \
| nmrPipe -fn FT -neg \
| nmrPipe -fn PS -rs 2.5 \
```

The script should be changed for different phasing, first point scaling, a polynomial baseline correction added in the direct dimension or removed from the indirect dimension, solvent suppression removed or changed, and the window functions modified for optimal spectral quality. Each system and spectrum is different, so it is recommended that to find the optimal processing that each part of the script be removed and re-added one-by-one between processing and checking of the resultant spectrum. Note that the extraction at the end after the polynomial baseline correction in the indirect dimension is important as the baseline correction often displays a much better performance when the empty part of the spectrum is used in the calculation.

5.3.3 Measuring peak intensities

For the measurement of peak intensities, again care must be taken. A read of the paper:

- Viles, J., Duggan, B., Zaborowski, E., Schwarzinger, S., Huntley, J., Kroon, G., Dyson, H. J., and Wright, P. (2001). Potential bias in NMR relaxation data introduced by peak intensity analysis and curve fitting methods. *J. Biomol. NMR*, **21**, 1–9. ([10.1023/A:1011966718826](https://doi.org/10.1023/A:1011966718826))

is highly recommended. Despite the recommendations in the discussion of this paper, a different methodology using peak heights can be used to solve the same problems. This will be discussed in a paper which is currently in preparation from the Gooley group. The steps involved are:

- For the first spectrum in the time series, shift the peak list to the tops of the peaks (for example using “pc” in Sparky on subsets of peaks).
- Copy this 1st spectrum list onto all spectra, shifting the peaks to the top as in the previous step.
- When the peak disappears into the noise, leave it at its current position and do not type “pc” or equivalent. This will add weight to the first point in the subsequent step.
- Once all spectra are shifted, calculate an average peak list.
- Copy this average peak list onto fresh copies of all spectra.
- Measure peak heights using this averaged peak list.

This will produce the most accurate peak intensity measurements until better, more robust peak shape integration comes along. This is a special technique which is designed to minimise the white-noise bias talked about in the [Viles et al. \(2001\)](https://doi.org/10.1023/A:1011966718826) paper. As the noise

often decreases with the decrease in total spectral power, using the tops of the peaks means that you are actually measuring the real peak height plus positive noise in all cases. This non-constant additional positive noise contribution can result in a double exponential in the measured data. The technique above eliminates this as you then measure close to real peak height with the addition of white noise centred at zero – it is both negative and positive to equal amounts – rather than the peak high with noise contribution strongly biased towards the positive. Where the peaks disappear, you then are measuring the pure baseplane noise. This is fine as these white-noise data points centred at zero will help in the subsequent exponential fit in relax.

If using Sparky then, to be sure that the peak heights are properly updated, for each spectrum type “pa” to select all peaks, “ph” to update all selected peak heights, “lt” to show the spectrum peaks window, make sure “data height” is selected in the options, and then save the peak list.

5.4 Relaxation curve-fitting in the prompt/script UI mode

5.4.1 Relax-fit script mode – the sample script

The following is a verbatim copy of the contents of the `sample_scripts/relax_fit.py` file. If your copy of the sample script is different than that below, please send an email to the relax-devel mailing list to tell the relax developers that the manual is out of date (see section 3.3.3 on page 31). You will need to first copy this script to a dedicated analysis directory containing peak lists, a PDB file and a file listing unresolved spin systems, and then modify its contents to suit your specific analysis. The script contents are:

```

1 # Script for relaxation curve-fitting.
2
3 # Create the 'rx' data pipe.
4 pipe.create('rx', 'relax_fit')
5
6 # Load the backbone amide 15N spins from a PDB file.
7 structure.read_pdb('Ap4Aase_new_3.pdb')
8 structure.load_spins(spin_id='@N')
9 structure.load_spins(spin_id='@NE1')
10
11 # Spectrum names.
12 names = [
13     'T2_ncyc1_ave',
14     'T2_ncyc1b_ave',
15     'T2_ncyc2_ave',
16     'T2_ncyc4_ave',
17     'T2_ncyc4b_ave',
18     'T2_ncyc6_ave',
19     'T2_ncyc9_ave',
20     'T2_ncyc9b_ave',
21     'T2_ncyc11_ave',
22     'T2_ncyc11b_ave'
23 ]
24
25 # Relaxation times (in seconds).
26 times = [
27     0.0176,
28     0.0176,
```

```

29      0.0352,
30      0.0704,
31      0.0704,
32      0.1056,
33      0.1584,
34      0.1584,
35      0.1936,
36      0.1936
37  ]
38
39 # Loop over the spectra.
40 for i in range(len(names)):
41     # Load the peak intensities.
42     spectrum.read_intensities(file=names[i]+'.list', dir=data_path, spectrum_id=names[i],
43                               int_method='height')
44
45     # Set the relaxation times.
46     relax_fit.relax_time(time=times[i], spectrum_id=names[i])
47
48     # Specify the duplicated spectra.
49     spectrum.replicated(spectrum_ids=['T2_ncyc1_ave', 'T2_ncyc1b_ave'])
50     spectrum.replicated(spectrum_ids=['T2_ncyc4_ave', 'T2_ncyc4b_ave'])
51     spectrum.replicated(spectrum_ids=['T2_ncyc9_ave', 'T2_ncyc9b_ave'])
52     spectrum.replicated(spectrum_ids=['T2_ncyc11_ave', 'T2_ncyc11b_ave'])
53
54     # Peak intensity error analysis.
55     spectrum.error_analysis()
56
57     # Deselect unresolved spins.
58     deselect.read(file='unresolved', mol_name_col=1, res_num_col=2, res_name_col=3,
59                   spin_num_col=4, spin_name_col=5)
60
61     # Set the relaxation curve type.
62     relax_fit.select_model('exp')
63
64     # Grid search.
65     minimise.grid_search(inc=11)
66
67     # Minimise.
68     minimise.execute('newton', constraints=False)
69
70     # Monte Carlo simulations.
71     monte_carlo.setup(number=500)
72     monte_carlo.create_data()
73     monte_carlo.initial_values()
74     minimise.execute('newton', constraints=False)
75     monte_carlo.error_analysis()
76
77     # Save the relaxation rates.
78     value.write(param='rx', file='rx.out', force=True)
79
80     # Save the results.
81     results.write(file='results', force=True)
82
83     # Create Grace plots of the data.
84     grace.write(y_data_type='chi2', file='chi2.agr', force=True)      # Minimised chi-squared
85                               value.
86     grace.write(y_data_type='i0', file='i0.agr', force=True)      # Initial peak intensity.
87     grace.write(y_data_type='rx', file='rx.agr', force=True)      # Relaxation rate.
88     grace.write(x_data_type='relax_times', y_data_type='peak_intensity', file='intensities.agr
89                               ', force=True)      # Average peak intensities.

```

```

86 grace.write(x_data_type='relax_times', y_data_type='peak_intensity', norm=True, file='
     intensities_norm.agr', force=True)      # Average peak intensities (normalised).
87
88 # Display the Grace plots.
89 grace.view(file='chi2.agr')
90 grace.view(file='i0.agr')
91 grace.view(file='rx.agr')
92 grace.view(file='intensities.agr')
93 grace.view(file='intensities_norm.agr')
94
95 # Save the program state.
96 state.save('rx.save', force=True)

```

The next sections will break this script down into its logical components and explain how these parts will be interpreted by relax. To execute this script, please see section [1.2.8](#) on page [12](#) for details.

5.4.2 Relax-fit script mode – initialisation of the data pipe

The data pipe is simply created by the command

```

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

```

This user function will then create a relaxation exponential curve-fitting specific data pipe labelled “rx”. The second argument sets the pipe type to that of the relaxation curve-fitting. Setting the pipe type is important so that the program knows which user functions are compatible with the data pipe, for example in the steady-state NOE analysis the function `minimise.execute` (see page [497](#)) is meaningless as the NOE values are calculated directly rather than optimised.

5.4.3 Relax-fit script mode – setting up the spin systems

The first thing which needs to be completed prior to any spin specific command is to generate the molecule, residue and spin data structures for storing the spin specific data. In the sample script above this is generated from a PDB file, however a plain text file with the sequence information can be used instead (see the `sequence.read` user function on page [605](#) for more details). In the case of the sample script, the command

```

6 # Load the backbone amide 15N spins from a PDB file.
7 structure.read_pdb(name, 'Ap4Aase_new_3.pdb')

```

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

```

8 structure.load_spins(spin_id='@N')
9 structure.load_spins(spin_id='@NE1')

```

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 as well as ^{15}N spins set up for the tryptophan indole nitrogens. Although the 3D coordinates have been loaded into the program from the PDB file, this structural information serves no purpose when calculating R_1 and R_2 values.

5.4.4 Relax-fit script mode – loading the data

To load the peak intensities into relax the `spectrum.read_intensities` and `relax_fit.relax_time` user functions are executed. Important keyword arguments for these user functions are the file name and directory, the spectrum identification string and the relaxation time period of the experiment in seconds. By default the file format will be automatically detected. Currently Sparky, XEasy, NMRView, and generic columnar formatted peak lists are supported. 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 – an expansion of the `for` loop in the sample script – will load peak intensities from six different relaxation periods, four of which have been duplicated, from Sparky peak lists with the peak heights in the 10th column.

```
spectrum.read_intensities('T2_ncyc1.list', spectrum_id='1', int_col=10)
relax_fit.relax_time(spectrum_id='1', time=0.0176)
spectrum.read_intensities('T2_ncyc1b.list', spectrum_id='1b', int_col=10)
relax_fit.relax_time(spectrum_id='1b', time=0.0176)
spectrum.read_intensities('T2_ncyc2.list', spectrum_id='2', int_col=10)
relax_fit.relax_time(spectrum_id='2', time=0.0352)
spectrum.read_intensities('T2_ncyc4.list', spectrum_id='4', int_col=10)
relax_fit.relax_time(spectrum_id='4', time=0.0704)
spectrum.read_intensities('T2_ncyc4b.list', spectrum_id='4b', int_col=10)
relax_fit.relax_time(spectrum_id='4b', time=0.0704)
spectrum.read_intensities('T2_ncyc6.list', spectrum_id='6', int_col=10)
relax_fit.relax_time(spectrum_id='6', time=0.1056)
spectrum.read_intensities('T2_ncyc9.list', spectrum_id='9', int_col=10)
relax_fit.relax_time(spectrum_id='9', time=0.1584)
spectrum.read_intensities('T2_ncyc9b.list', spectrum_id='9b', int_col=10)
relax_fit.relax_time(spectrum_id='9b', time=0.1584)
spectrum.read_intensities('T2_ncyc11.list', spectrum_id='11', int_col=10)
relax_fit.relax_time(spectrum_id='11', time=0.1936)
spectrum.read_intensities('T2_ncyc11b.list', spectrum_id='11b', int_col=10)
relax_fit.relax_time(spectrum_id='11b', time=0.1936)
```

The replicated spectra a set up with the commands

```
47 # Specify the duplicated spectra.
48 spectrum.replicated(spectrum_ids=['T2_ncyc1_ave', 'T2_ncyc1b_ave'])
49 spectrum.replicated(spectrum_ids=['T2_ncyc4_ave', 'T2_ncyc4b_ave'])
50 spectrum.replicated(spectrum_ids=['T2_ncyc9_ave', 'T2_ncyc9b_ave'])
51 spectrum.replicated(spectrum_ids=['T2_ncyc11_ave', 'T2_ncyc11b_ave'])
```

Note that the relaxation time period should be calculated directly from the pulse sequence (as the sum of delays and pulses for the period), as the estimated time may not match the real time. For the Sparky peak lists, by default relax assumes that the intensity value is in the 4th column. A typical file looks like:

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

By supplying the `int_col` argument to the `spectrum.read_intensities` user function, this can be changed. A typical XEasy file will look like:

No.	Color	w1	w2	ass. in w1	ass. in w2	Volume	Vol. Err.	Method	Comment
2	2	10.014	134.221	HN 21 LEU	N 21 LEU	7.919e+03	0.00e+00	m	
3	2	10.481	132.592	HE1 79 TRP	NE1 79 TRP	1.532e+04	0.00e+00	m	
17	2	9.882	129.041	HN 110 PHE	N 110 PHE	9.962e+03	0.00e+00	m	
18	2	8.757	128.278	HN 52 ASP	N 52 ASP	2.041e+04	0.00e+00	m	
19	2	10.086	128.297	HN 69 SER	N 69 SER	9.305e+03	0.00e+00	m	
20	3	9.111	127.707	HN 15 ARG	N 15 ARG	9.714e+03	0.00e+00	m	

where the peak height is in the `Volume` column. And for an NMRView file:

```
label dataset sw sf
H1 N15
cNTnC_noe0.nv
2505.63354492 1369.33557129
499.875 50.658000946
H1.L H1.P H1.W H1.B H1.E H1.J H1.U N15.L N15.P N15.W N15.B N15.E N15.J N15.U vol int stat comment flag0
0 {70.HN} 10.75274 0.02954 0.05379 ++ 0.0 {} {70.N} 116.37241 0.23155 0.35387 ++ 0.0 {} -6.88333129883 -0.1694 0 {} 0
1 {72.HN} 9.67752 0.03308 0.05448 ++ 0.0 {} {72.N} 126.41302 0.27417 0.37217 ++ 0.0 {} -5.49038267136 -0.1142 0 {} 0
2 {} 8.4532 0.02331 0.05439 ++ 0.0 {} {} 122.20137 0.38205 0.33221 ++ 0.0 {} -2.58034267191 -0.1320 0 {} 0
```

5.4.5 Relax-fit script mode – 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 spins 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

```
53 # Peak intensity error analysis.
54 spectrum.error_analysis()
```

Any spins which cannot be resolved due to peak overlap were included in a file called `unresolved`. This file can consist of optional columns of the molecule name, the residue name and number, and the spin name and number. The matching spins are excluded from the analysis by the user function

```
56 # Deselect unresolved spins.
57 deselect.read(file='unresolved', mol_name_col=1, res_num_col=2, res_name_col=3,
    spin_num_col=4, spin_name_col=5)
```

Finally the experiment type is specified by the command

```
59 # Set the relaxation curve type.
60 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. Changing the user function argument to “`inv`” will select the inversion recovery experiment, and changing it to “`sat`” will select the saturation recovery experiment (see section 5.2 on page 51).

5.4.6 Relax-fit script mode – optimisation of exponential curves

Now that everything has been setup minimisation 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

```
62 # Grid search.
63 minimise.grid_search(inc=11)
```

The next step is to select one of the minimisation algorithms to optimise the model parameters

```
65 # Minimise.
66 minimise.execute('newton', constraints=False)
```

5.4.7 Relax-fit script mode – 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

```
68 # Monte Carlo simulations.
69 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

```
70 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

```
71 monte_carlo.initial_values()
```

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

```
72 minimise.execute('newton', constraints=False)
```

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

```
73 monte_carlo.error_analysis()
```

5.4.8 Relax-fit script mode – finishing off

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

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

Grace plots are created and viewed

```
81 # Create Grace plots of the data.
82 grace.write(y_data_type='chi2', file='chi2.agr', force=True)      # Minimised chi-squared
83             value.
84 grace.write(y_data_type='i0', file='i0.agr', force=True)      # Initial peak intensity.
85 grace.write(y_data_type='rx', file='rx.agr', force=True)      # Relaxation rate.
86 grace.write(x_data_type='relax_times', y_data_type='peak_intensity', file='intensities.agr
87             ', force=True)      # Average peak intensities.
88
89 grace.write(x_data_type='relax_times', y_data_type='peak_intensity', norm=True, file='
90             intensities_norm.agr', force=True)      # Average peak intensities (normalised).
```

and viewed

```
91
92
93 # Display the Grace plots.
94 grace.view(file='chi2.agr')
95 grace.view(file='i0.agr')
96 grace.view(file='rx.agr')
97 grace.view(file='intensities.agr')
98 grace.view(file='intensities_norm.agr')
```

and finally the program state is saved for future reference

```
99
100 # Save the program state.
101 state.save(file='rx.save', force=True)
```

5.5 The relaxation curve-fitting auto-analysis in the GUI

The R_1 and R_2 relaxation rates can be calculated using the relax GUI (see Figures 1.6 and 1.7). These auto-analyses can be selected using the analysis selection wizard (Figure 1.4 on page 12). Just as with the steady-state NOE in the next chapter, these auto-analyses are very similar in spirit to the sample script described in this chapter, though the Grace 2D visualisation is more advanced. If you have read this chapter, the usage of these analyses should be self explanatory.

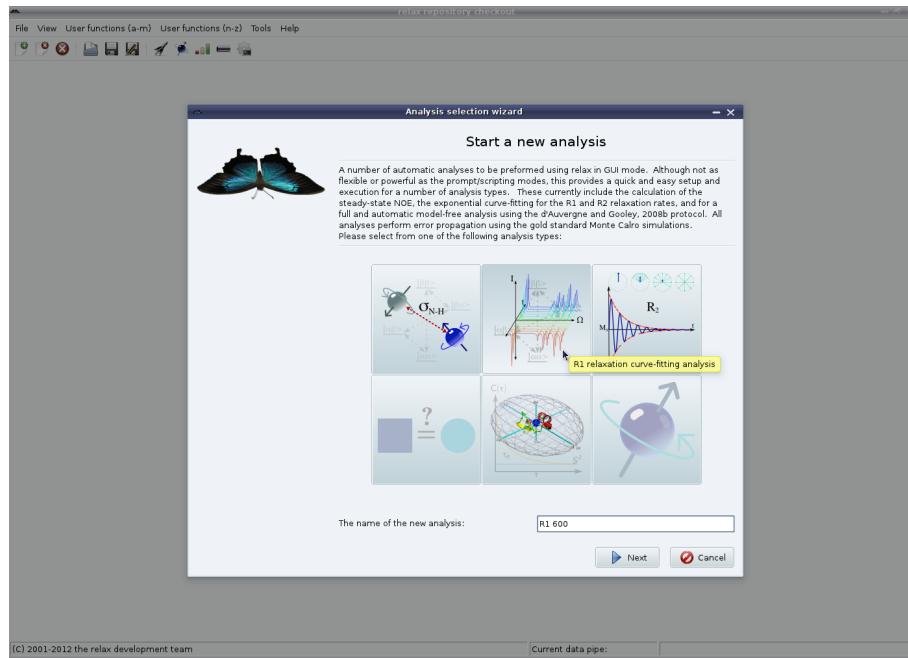
As in the script/prompt UI section above, the example of protein ^{15}N R_1 relaxation analysis will be performed in the following sections. To keep track of all the messages relax produces for future reference, you can run the relax GUI with the following command line arguments:

```
$ relax --log log --gui
```

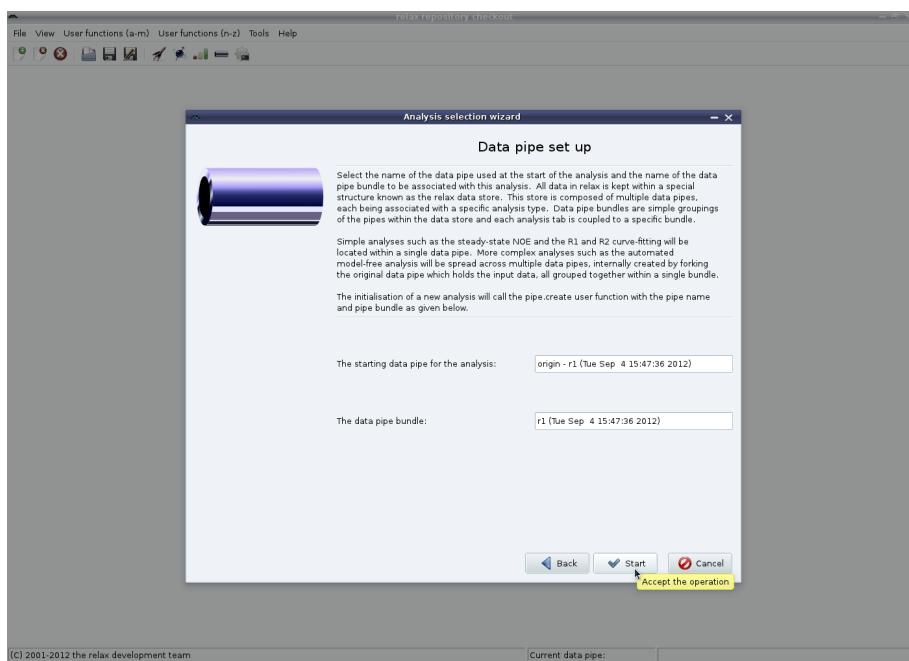
The messages will then appear both in the relax controller window (see Figure 1.9 on page 19) and in the `log` file.

5.5.1 Relax-fit GUI mode – initialisation of the data pipe

To begin the analysis, launch the analysis selection wizard (see Figure 1.4 on page 12). Select either the R_1 or R_2 analyses, and change the name of the analysis if you plan on running multiple analyses from different field strengths in one relax instance.

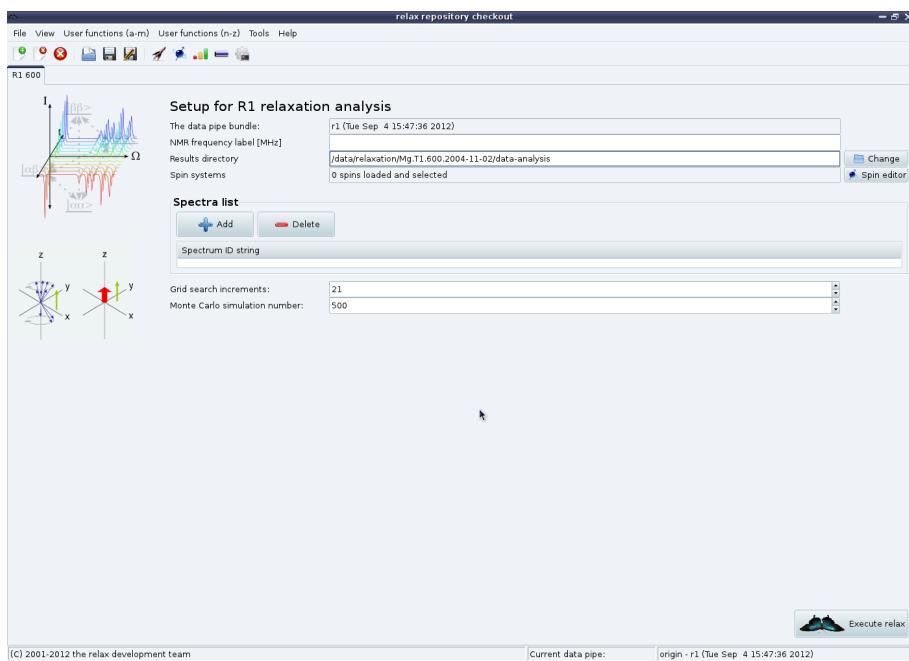


Then click on the “Next” button. On the second page click on “Start” to commence the analysis – this second part of the wizard does not need to be changed. For the R_1 and R_2 analyses in the GUI, a data pipe bundle containing only a single data pipe for that analysis will be created. This data pipe bundle can be safely ignored.



5.5.2 Relax-fit GUI mode – general setup

You will now be presented with a blank analysis tab:



Here there are two things unique to the GUI which need to be preformed:

NMR frequency label: First set the NMR frequency label. This is only used for the name of the output file. For example if you set the label to “1200”, the file `r1.1200.out` will be created at the end of the analysis.

Results directory: All of the automatically created results and Grace files will be placed into this directory. The “Results directory” can now be changed.

5.5.3 Relax-fit GUI mode – setting up the spin systems

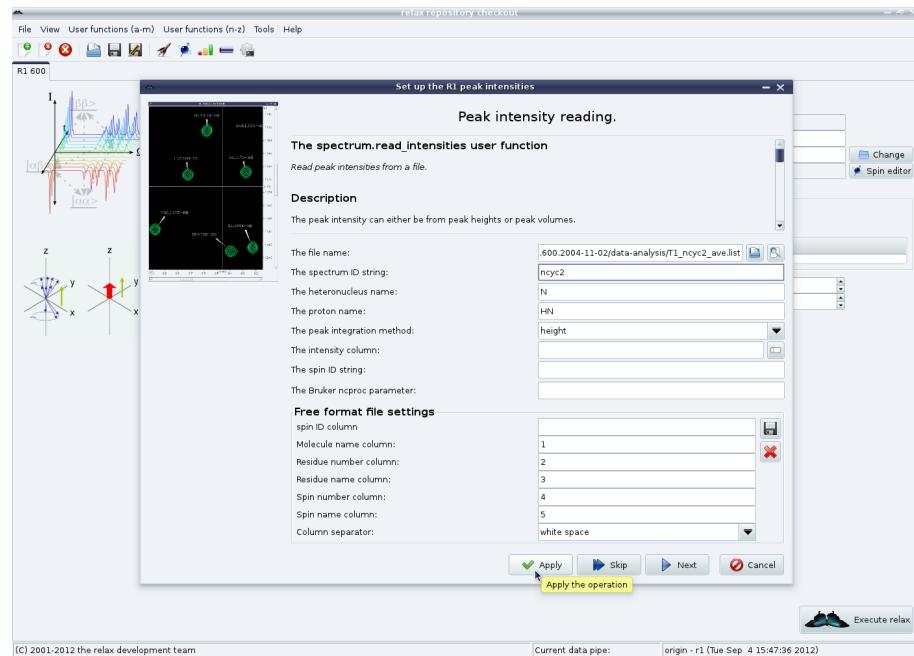
As the relaxation data is at the level of the spins, the molecule, residue and spin data structures need to be set up. In the R₁ and R₂ GUI analysis tabs, there is a special “Spin systems” GUI element designed for this. This will initially say “0 spins loaded and selected”. Click on the “Spin editor” button to launch the spin viewer window. The steps for setting up the spin containers using PDB files are described in section 4.5.2 on page 42 or for sequence files in section 4.5.3 on page 45.

5.5.4 Relax-fit GUI mode – unresolved spins

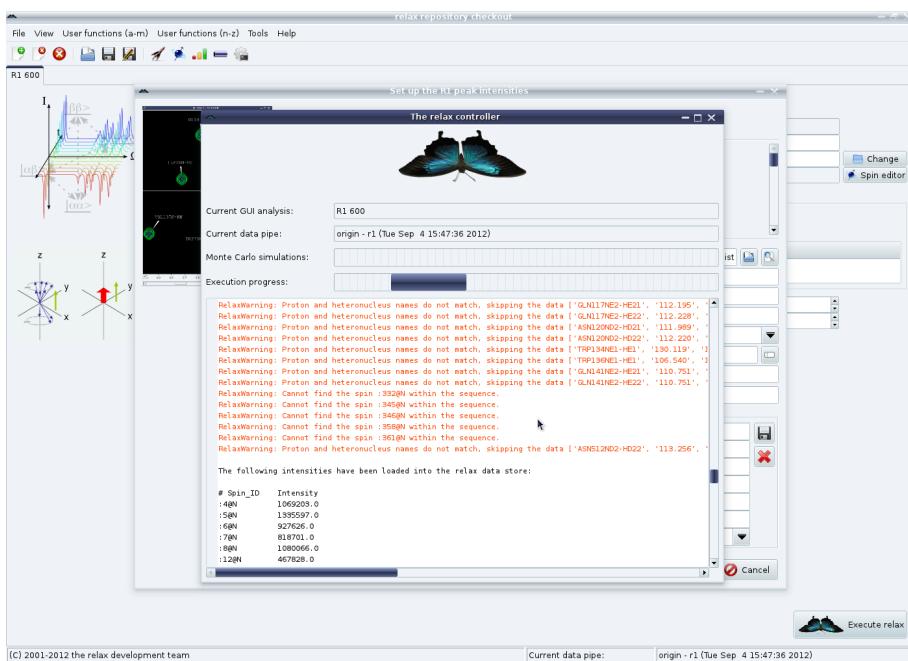
As in the prompt/script UI section 5.4.5, the spins can be deselected at this point using the same **unresolved** file. This is described in detail in section 4.5.5 on page 46.

5.5.5 Relax-fit GUI mode – loading the data

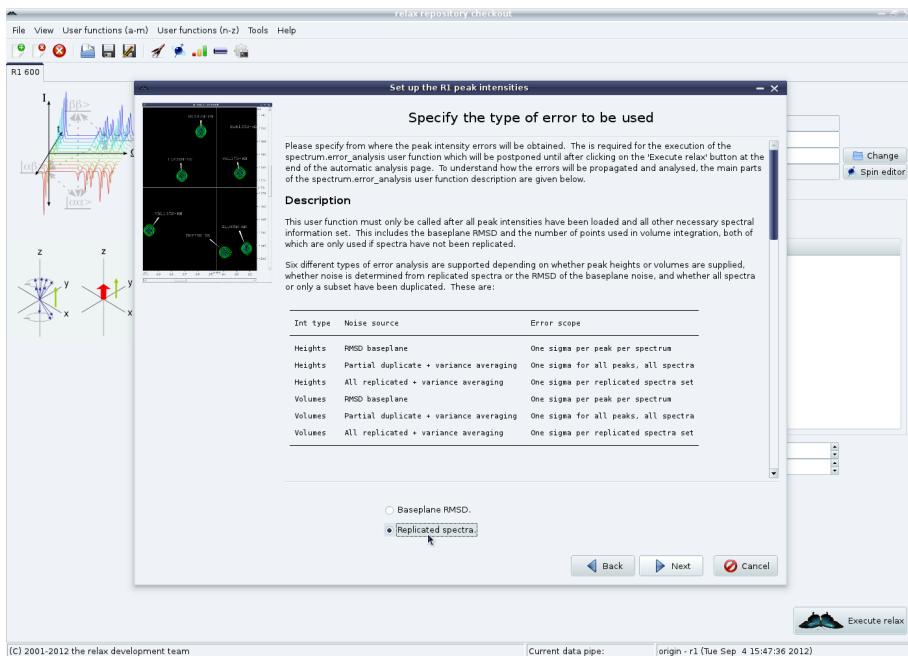
At this stage, the peak intensity data needs to be loaded. In both the R₁ and R₂ analysis tabs is a “Spectra list” GUI element. Click on the “Add” button to launch the peak intensity loading wizard:



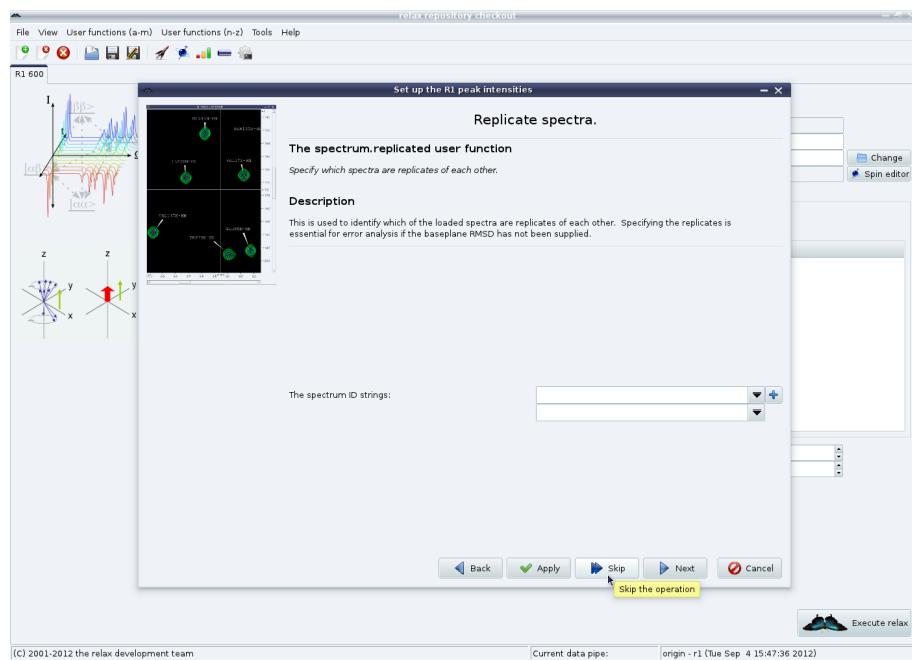
In this example, a Sparky peak list containing the peak heights determined from the averaged chemical shift positions for all spectra will be loaded. Set the spectrum ID string to a unique value. Click on “Next”. This will most likely cause a **RelaxWarning** message to appear for all peak list elements which do not correspond to any spins loaded into the relax data store:



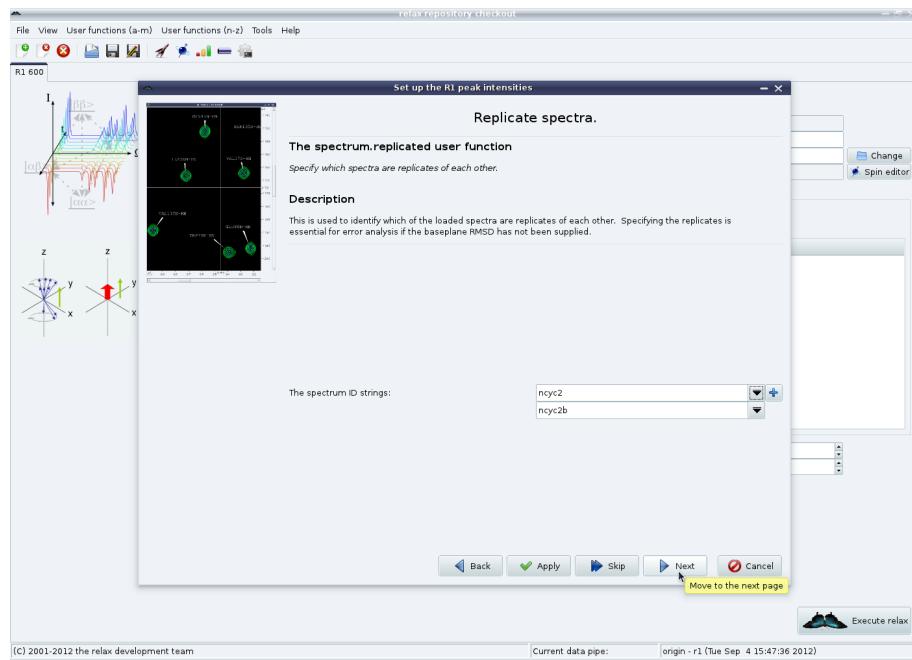
These messages must be carefully checked to be sure that the correct data has been loaded. A `RelaxError` might be thrown if the peak list is corrupted or if the dimension has been incorrectly given. In this case check the message, go “Back”, fix the problem, and click on “Next” again. Then click on “Next”. You should now see the error type page:



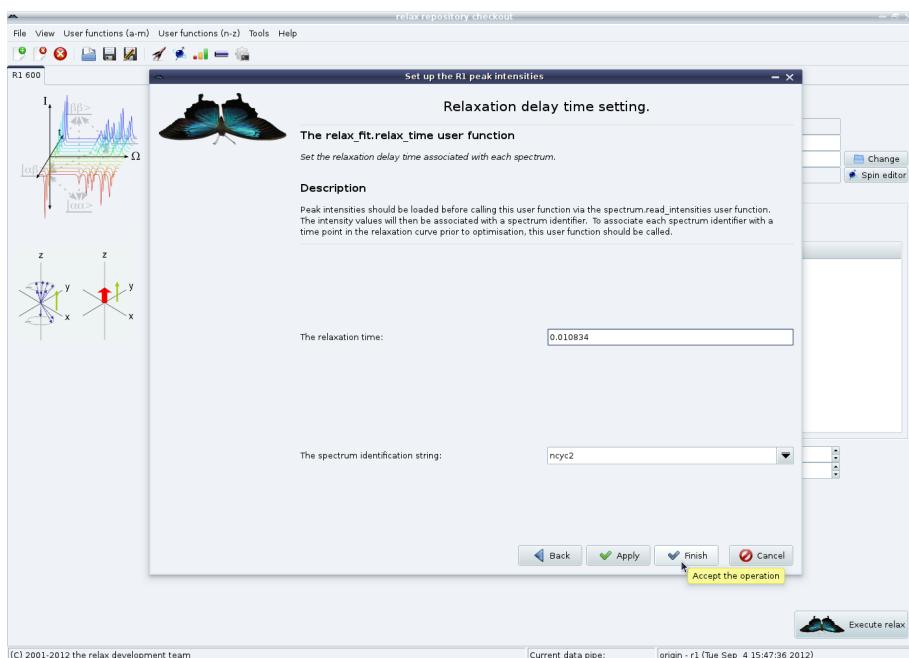
The description for this wizard page should be very carefully read – it will tell you about all of the error analysis options available and how these are implemented in relax. For the protein relaxation example, replicated spectra have been collected. Therefore the option “Replicated spectra” will be chosen. The “Baseplane RMSD” option is documented in the NOE chapter. After clicking on “Next” you will see:



For the first of the duplicate spectra, or any spectrum without a duplicate, you can click on the “Skip” button. If this is the second spectrum you have loaded from a duplicated set, select the two replicated spectra and then click on “Next”:

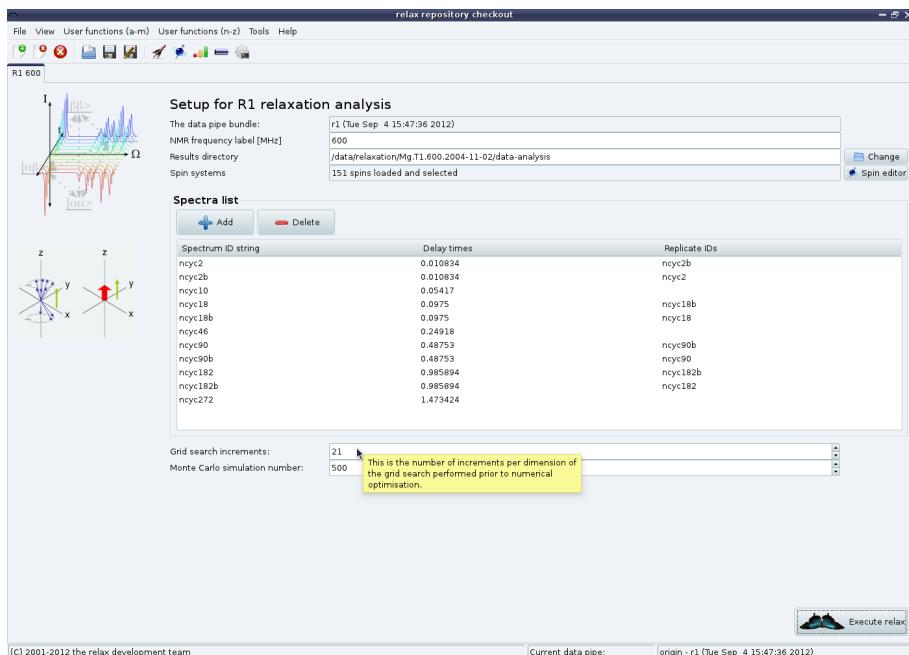


Finally set the relaxation time period for this experiment in seconds:



All delays and pulse lengths in the pulse sequence should be carefully checked to be sure that the time is exactly what you would expect – the estimated time may not match the real time. To set the time and close the wizard, click on the “Finish” button.

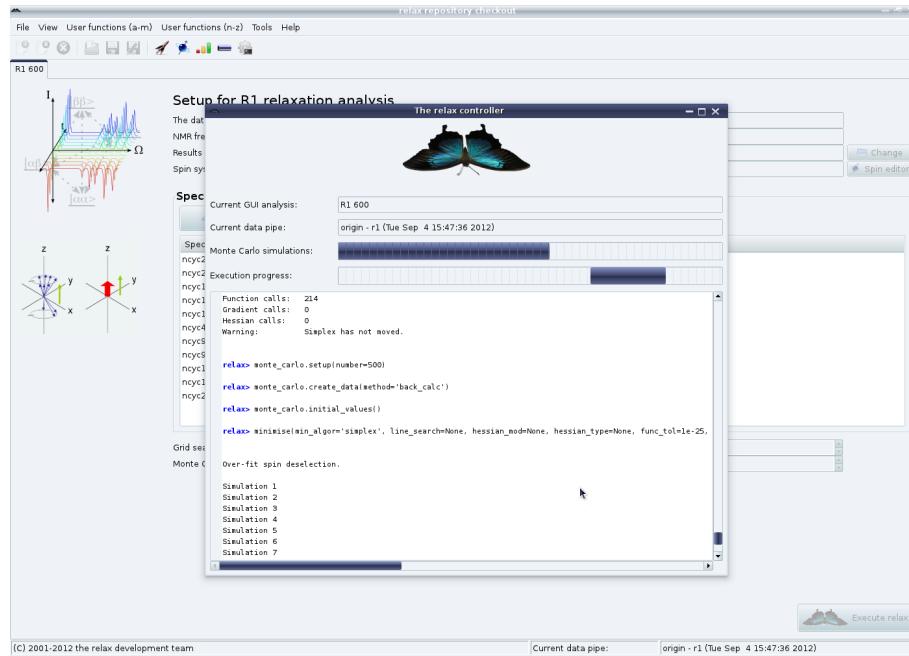
This procedure should be repeated for every experiment you have collected (you could, as an alternative, load all at the same time using the “Apply” button at each stage). In the end you should see something such as:



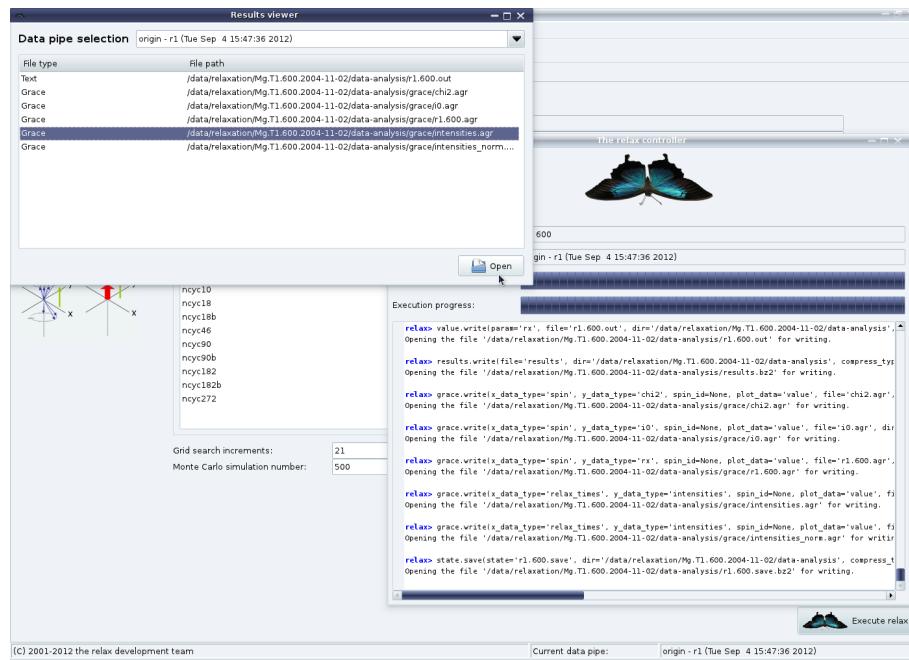
5.5.6 Relax-fit GUI mode – optimisation and error analysis

Back in the main R_1 analysis tab, the grid search increments and number of Monte Carlo simulations can be changed. The default values of 21 grid search increments and 500 MC

simulations are optimal – lower values are not recommended. To perform the optimisation and error analysis, click on the “Execute relax” button. The relax controller will open to show you the progress of the optimisation and simulations:



Once finished, the “Results viewer” window will also appear:



This window can be used to open the text files in the default text editor for your operating system or the 2D Grace plots in **xmgrace** if available on your system.

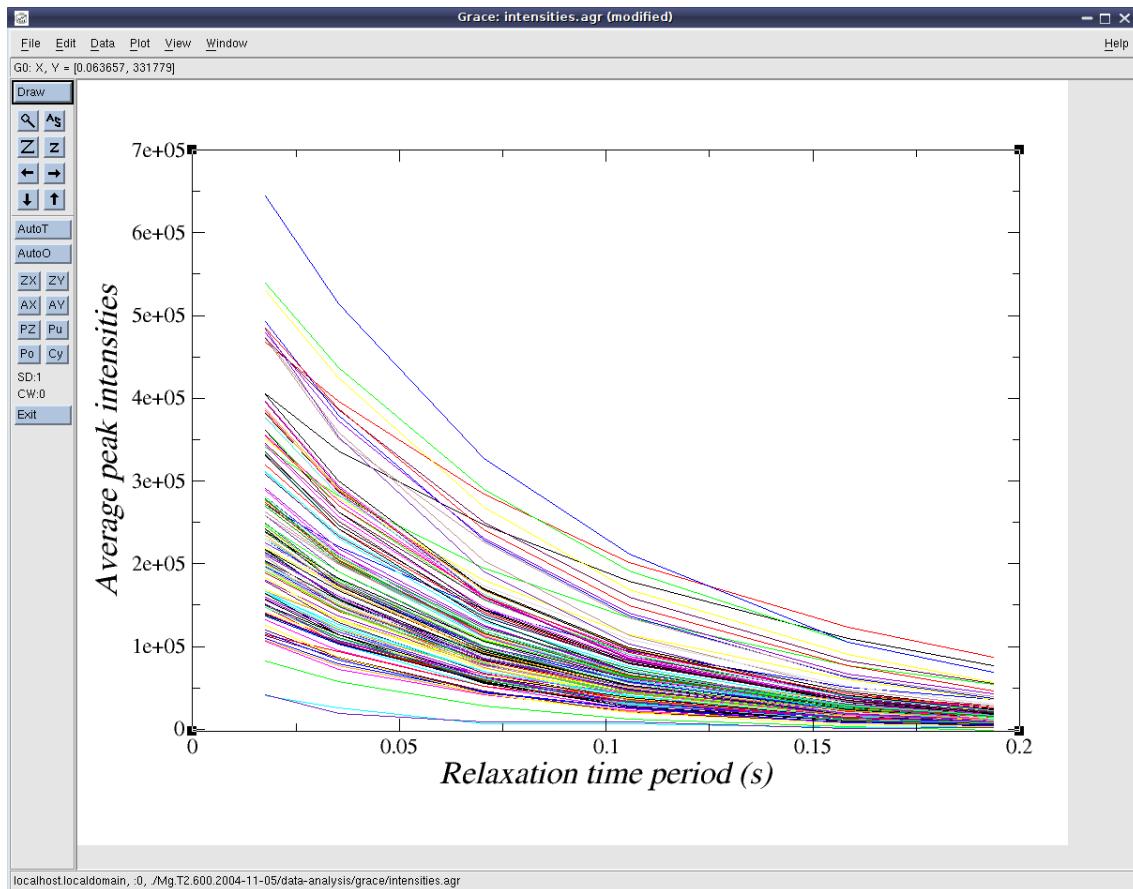


Figure 5.1: Screenshot of the 2D peak intensity plots for the exponential relaxation curves in Xmgrace.

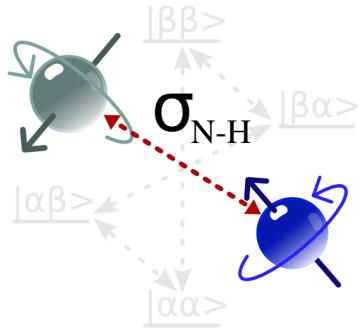
5.6 Final checks of the curve-fitting

To be sure that the data has been properly collected and that no instrumentation or pulse sequence timing errors have occurred, it is essential to carefully check the `intensities.agr` and `intensities_norm.agr` 2D Grace files. These are plots of the decay curves for each spin system analysed, and any non-exponential behaviour should be clearly visible (see Figure 5.1). If Xmgrace or a compatible program is not available for your operating system, the Grace files contain text representations of the curves at the end which can be opened, edited and visualised in any another 2D graphing software package.

Note that errors resulting in systematic bias in the data – for example if temperature control (single-scan interleaving or temperature compensation blocks) or per-experiment/per-spectrometer temperature calibration on MeOH or ethylene glycol have not been performed – will not be detected by looking at the decay curves. See section 5.3.1 or the `relax_data.temp_calibration` user function documentation on page 573 and the `relax_data.temp_control` user function documentation on page 574 for more details.

Chapter 6

Calculating the NOE



6.1 Introduction to the steady-state NOE

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. Then the same operations will be presented for the perspective of the graphical user interface.

6.2 From spectra to peak intensities for the NOE

For a set of recommendations for how to obtain the best quality relaxation rates, please see section 5.3 on page 52. In summary the following are important – temperature control (though the standard steady-state NOE single FID interleaved pulse sequences are fine), per-experiment temperature calibration, spectral processing with massive zero-filling and no baseplane rolling, and using an averaged peak list for determining the peak heights.

6.3 Calculation of the NOE in the prompt/script UI mode

6.3.1 NOE script mode – the sample script

This sample script can be found in the `sample_scripts` directory and will be used as the template for the next sections describing how to use relax.

```

1 # Script for calculating NOEs.
2
3 # Create the data pipe.
4 pipe.create('NOE', 'noe')
5
6 # Load the sequence from a PDB file.
7 structure.read_pdb('Ap4Aase_new_3.pdb')
8 structure.load_spins(spin_id='@N')
9 structure.load_spins(spin_id='@NE1')
10
11 # Load the reference spectrum and saturated spectrum peak intensities.
12 spectrum.read_intensities(file='ref.list', spectrum_id='ref_ave')
13 spectrum.read_intensities(file='sat.list', spectrum_id='sat_ave')
14
15 # Set the spectrum types.
16 noe.spectrum_type('ref', 'ref_ave')
17 noe.spectrum_type('sat', 'sat_ave')
18
19 # Set the errors.
20 spectrum.baseplane_rmsd(error=3600, spectrum_id='ref_ave')
21 spectrum.baseplane_rmsd(error=3000, spectrum_id='sat_ave')
22
23 # Individual residue errors.
24 spectrum.baseplane_rmsd(error=122000, spectrum_type='ref', res_num=114)
25 spectrum.baseplane_rmsd(error=8500, spectrum_type='sat', res_num=114)
26
27 # Peak intensity error analysis.
28 spectrum.error_analysis()
29
30 # Deselect unresolved spins.
31 deselect.read(file='unresolved', res_num_col=1, spin_name_col=2)
32
33 # Calculate the NOEs.
34 minimise.calculate()
35
36 # Save the NOEs.
37 value.write(param='noe', file='noe.out', force=True)
38
39 # Create Grace files.
40 grace.write(y_data_type='peak_intensity', file='intensities.agr', force=True)
41 grace.write(y_data_type='noe', file='noe.agr', force=True)
42
43 # View the Grace files.
44 grace.view(file='intensities.agr')
45 grace.view(file='noe.agr')
46
47 # Write the results.
48 results.write(file='results', dir=None, force=True)
49
50 # Save the program state.
51 state.save('save', force=True)

```

6.3.2 NOE script mode – initialisation of the data pipe

The start of this sample script is very similar to that of the relaxation curve-fitting calculation on page 58. The command

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

initialises the data pipe labelled “NOE”. The data pipe type is set to the NOE calculation by the argument “noe”.

6.3.3 NOE script mode – setting up the spin systems

The backbone amide nitrogen sequence is extracted from a PDB file using the same commands as the relaxation curve-fitting script (Chapter 5). The command

```
6 # Load the sequence from a PDB file.
7 structure.read_pdb('Ap4Aase_new_3.pdb')
```

will load the PDB file Ap4Aase_new_3.pdb into relax. Then the following commands will generate both the backbone amide and tryptophan indole ^{15}N spins

```
8 structure.load_spins(spin_id='@N')
9 structure.load_spins(spin_id='@NE1')
```

6.3.4 NOE script mode – loading the data

The commands

```
11 # Load the reference spectrum and saturated spectrum peak intensities.
12 spectrum.read_intensities(file='ref.list', spectrum_id='ref_ave')
13 spectrum.read_intensities(file='sat.list', spectrum_id='sat_ave')
```

will load the peak heights of the reference and saturated NOE experiments (although the volume could be used instead). relax will automatically determine the format of the peak list. Currently only Sparky, XEasy, NMRView and a generic columnar formatted text file are supported.

In this example, relax will determine from the file contents that these are Sparky peak lists (saved after typing “lt”). The first column of the file should be the Sparky assignment string and it is assumed that the 4th column contains either the peak height or peak volume (though this can be in any column – the int_col argument is used to specify where the data is). Without specifying the int_method argument, peak heights will be assumed. See page 611 for a description of all the spectrum.read_intensities user function arguments. In this example, the peak list looks like:

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

For subsequent usage of the data in relax, assuming a 3D structure exists, it is currently advisable to use the same residue and atom numbering as found in the PDB file.

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.

6.3.5 NOE script mode – setting the errors

In this example the errors were 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

```
19 # Set the errors.
20 spectrum.baseplane_rmsd(error=3600, spectrum_id='ref_ave')
21 spectrum.baseplane_rmsd(error=3000, spectrum_id='sat_ave')
```

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

```
23 # Individual residue errors.
24 spectrum.baseplane_rmsd(error=122000, spectrum_type='ref', res_num=114)
25 spectrum.baseplane_rmsd(error=8500, spectrum_type='sat', res_num=114)
```

There are many other ways of setting the errors, for example via spectrum duplication, triplication, etc. See the documentation for the `spectrum.error_analysis` user function on page [608](#) for all possible options. This user function needs to be executed at this stage to correctly set up the errors for all spin systems:

```
27 # Peak intensity error analysis.
28 spectrum.error_analysis()
```

6.3.6 NOE script mode – unresolved spins

As the peaks of certain spins overlap to such an extent that the heights or volumes cannot be resolved, a simple text file was created called “unresolved” in which each line consists of the residue number followed by the atom name. By using the command

```
30 # Deselect unresolved spins.
31 deselect.read(name, file='unresolved', res_num_col=1, spin_name_col=2)
```

all spins in the file “unresolved” are excluded from the analysis.

6.3.7 NOE script mode – the NOE calculation

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

```
33 # Calculate the NOEs.
34 minimise.calculate()
```

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

$$NOE = \frac{I_{sat}}{I_{ref}}, \quad (6.1)$$

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

$$\sigma_{NOE} = \sqrt{\frac{(\sigma_{sat} \cdot I_{ref})^2 + (\sigma_{ref} \cdot I_{sat})^2}{I_{ref}}}, \quad (6.2)$$

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

```
36 # Save the NOEs.
37 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

# mol_name	res_num	res_name	spin_num	spin_name	value	error
Ap4Aase_new_3_mol1	1	GLY	1	N	None	None
Ap4Aase_new_3_mol1	2	PRO	11	N	None	None
Ap4Aase_new_3_mol1	3	LEU	28	N	None	None
Ap4Aase_new_3_mol1	4	GLY	51	N	-0.038921946984531344	0.019031770246176943
Ap4Aase_new_3_mol1	5	SER	59	N	-0.312404225679127	0.018596937298386886
Ap4Aase_new_3_mol1	6	MET	71	N	-0.42850831873249773	0.02525856323041225
Ap4Aase_new_3_mol1	7	ASP	91	N	-0.5305492810313481	0.027990623144176396
Ap4Aase_new_3_mol1	8	SER	104	N	-0.5652842977581912	0.021706121467731133
Ap4Aase_new_3_mol1	9	PRO	116	N	None	None
Ap4Aase_new_3_mol1	10	PRO	133	N	None	None
Ap4Aase_new_3_mol1	11	GLU	150	N	None	None
Ap4Aase_new_3_mol1	12	GLY	167	N	-0.7036626368123614	0.04681370194503697
Ap4Aase_new_3_mol1	13	TYR	175	N	-0.747464566367261	0.03594640051809186
Ap4Aase_new_3_mol1	14	ARG	200	N	-0.7524129557634996	0.04957018638401278

6.3.8 NOE script mode – 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 Xmgr. The highly flexible relax user function `grace.write` is capable of producing 2D plots of any x-y data sets. The two commands

```
39 # Create Grace files.
40 grace.write(y_data_type='peak_intensity', file='intensities.agr', force=True)
41 grace.write(y_data_type='noe', file='noe.agr', force=True)
```

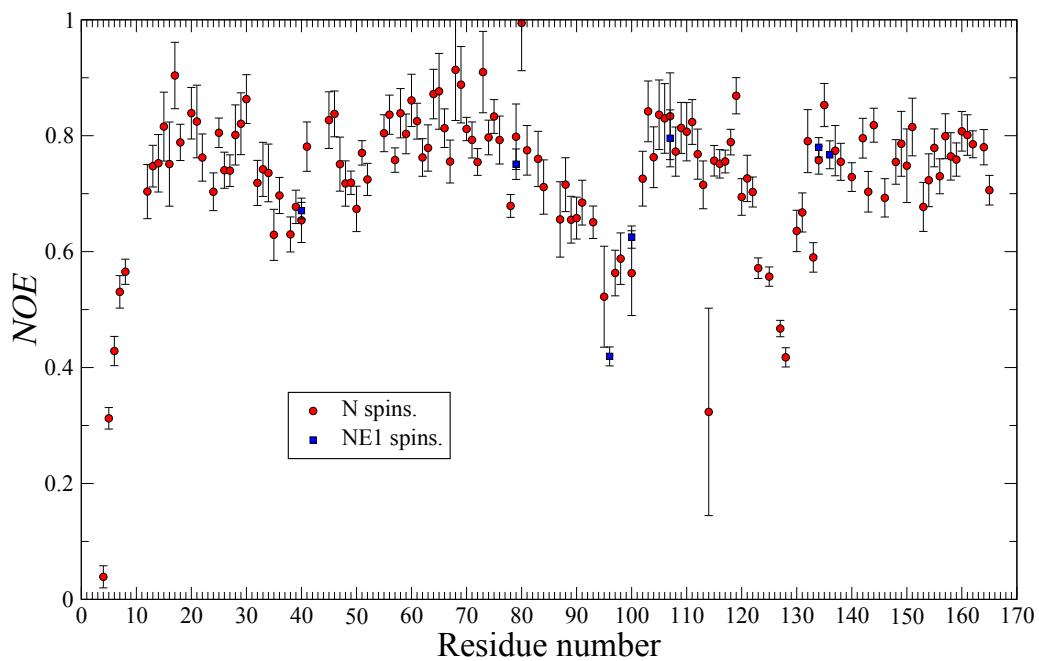


Figure 6.1: A Grace plot of the NOE value and error against the residue number. This is an example of the output of the user function `grace.write`.

will create one plot of the peak intensity of the reference and saturated spectra as different graph sets in the same plot as well as one plot for the NOE and its error. The x-axis in all three defaults to the residue number. Returning to the sample script three Grace data files are created `intensities.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

```
43 # View the Grace files.
44 grace.view(file='intensities.agr')
45 grace.view(file='noe.agr')
```

An example of the output after modifying the axes is shown in figure 6.1.

6.4 The NOE auto-analysis in the GUI

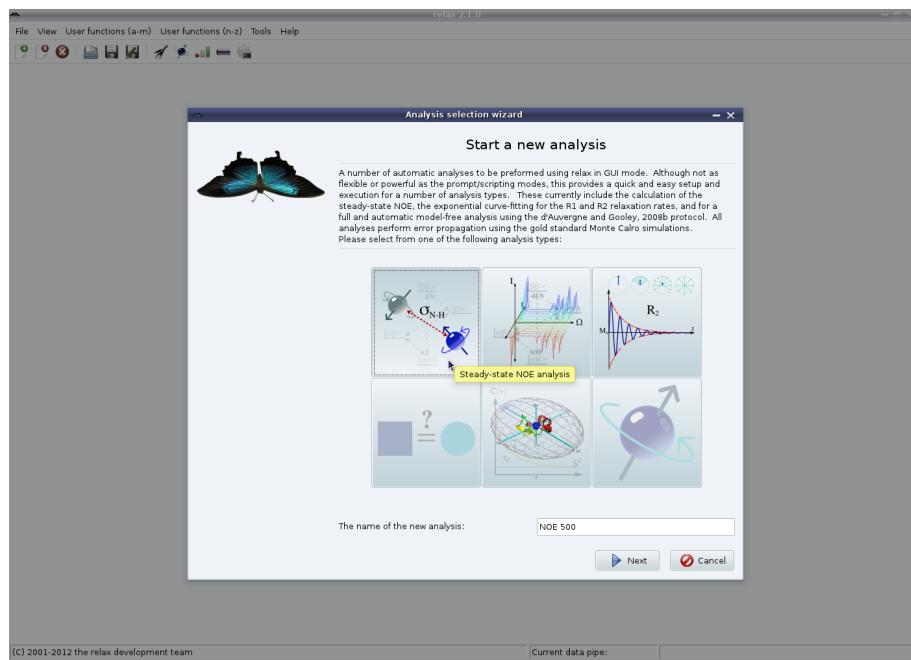
The relax graphical user interface provides access to an automated steady-state NOE analysis. This auto-analysis operates in the same way as the sample script described earlier in this chapter. In this example, relax will be launched with:

```
$ relax --log log --gui
```

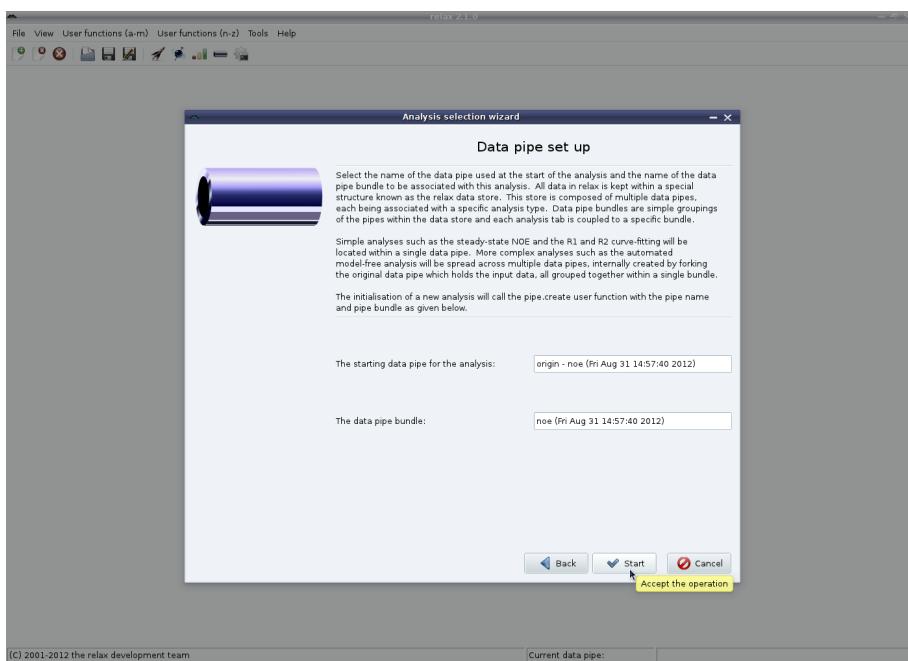
The `--log` command line argument will cause all of relax's text printouts to be placed into the `log` file which can serve as a record for later reference (the `--tee` command line argument could be used as well).

6.4.1 NOE GUI mode – initialisation of the data pipe

First launch the analysis selection wizard (see Figure 1.4 on page 12). Select the NOE analysis and, if you plan on running steady-state NOE analyses from multiple fields in one relax instance, change the name of the analysis:

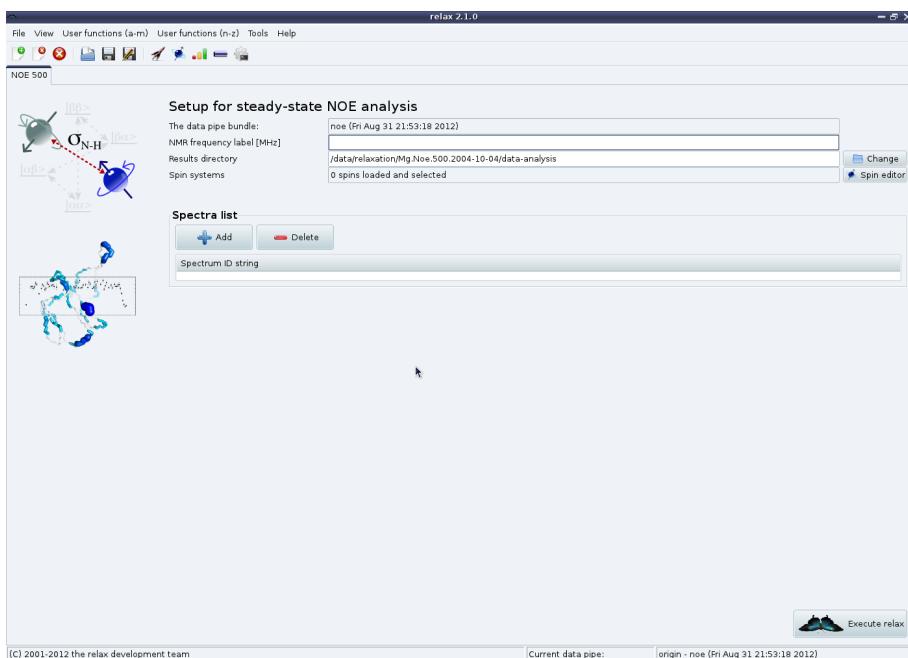


The second part of the wizard need not be modified, just click on “Start” to begin. This will create a dedicated data pipe for the analysis. A data pipe bundle will also be created, but for the steady-state NOE will only contain a single data throughout the analysis.



6.4.2 NOE GUI mode – general setup

You should then see the blank analysis tab:



The first thing to do now is to set the NMR frequency label. This is only used for the name of the NOE output file. For example if you set the label to “500”, the file `noe.500.out` will be created at the end of the analysis.

You can also choose to change the “Results directory” where all of the automatically created results files will be placed. These two steps are unique to the GUI mode.

6.4.3 NOE GUI mode – setting up the spin systems

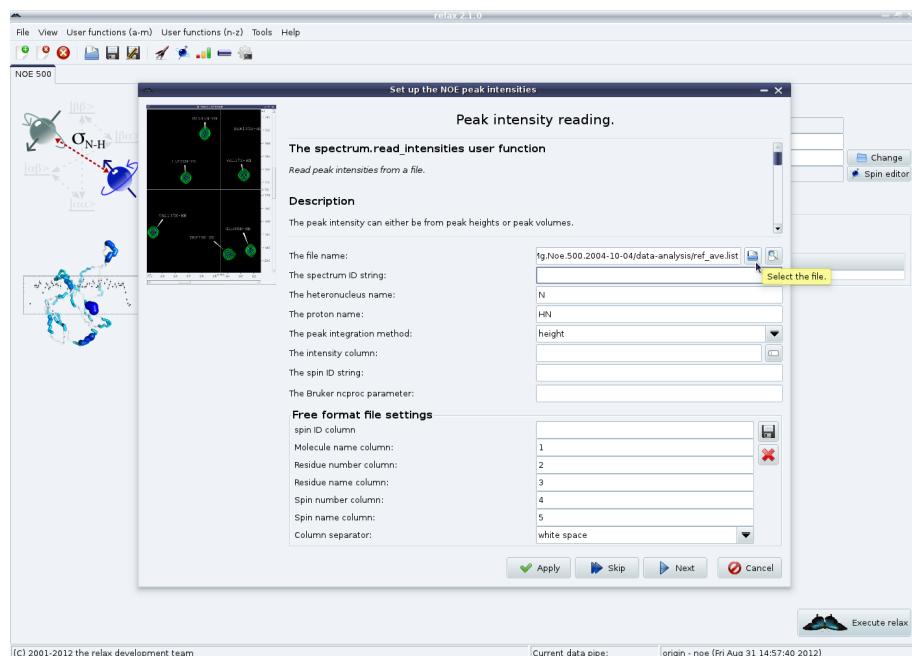
Just as in the prompt and scripting UI modes, the molecule, residue and spin data structures need to be set up prior to the loading of any spin specific data. The “Spin systems” GUI element is used for this purpose. Before any spin systems have been set up, this should say something like “0 spins loaded and selected”. To fix this, click on the “Spin editor” button and you should then see the spin viewer window. The next steps are fully described in section 4.5.2 on page 42 for PDB files or section 4.5.3 on page 45 for a sequence file. The spin viewer window can now be closed.

6.4.4 NOE GUI mode – unresolved spins

Using the unresolved spins file as described in the prompt/script UI sections, the same spins can be deselected at this point. See Section 4.5.5 on page 46 for the details of how to deselect the spins in the GUI.

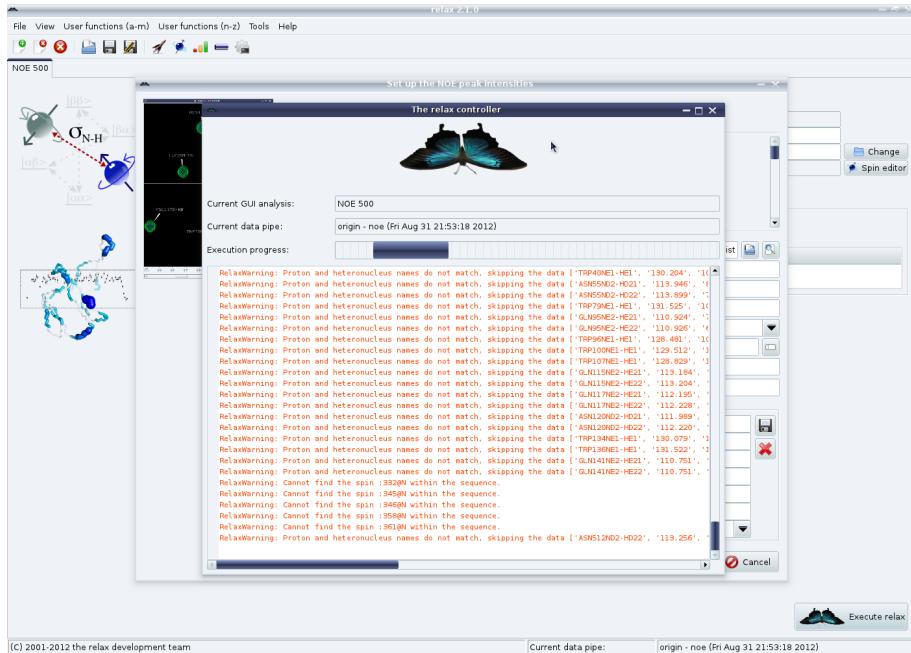
6.4.5 NOE GUI mode – loading the data

The next step is to load the saturated and reference NOE peak lists. From the main NOE auto-analysis tab, click on the “Add” button in the “Spectra list” GUI element. This will launch the NOE peak intensity loading wizard. From the first wizard page, select the peak list file containing the reference intensities (from the averaged shift list):

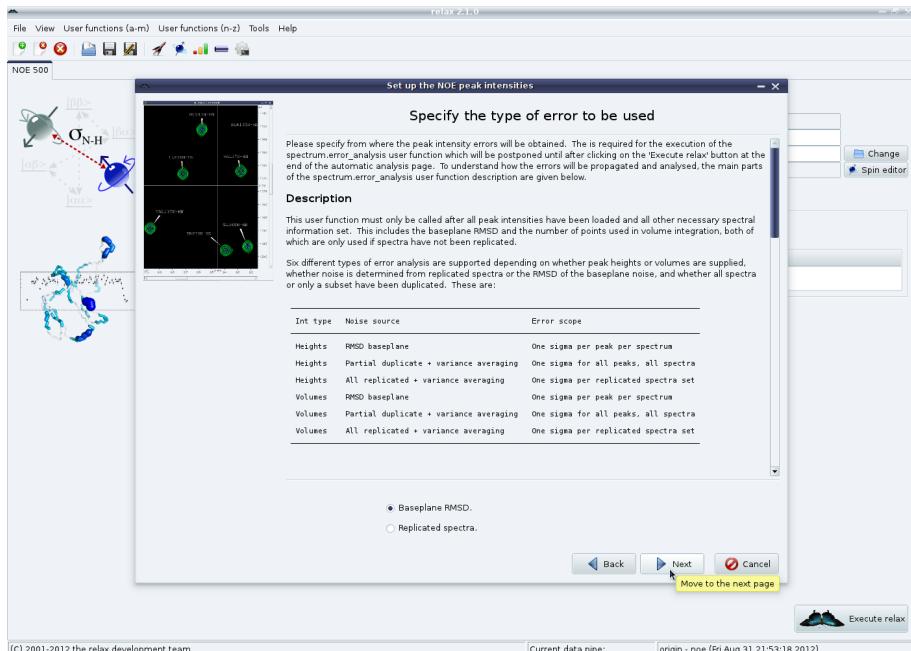


Then set the obligatory spectrum ID string to a unique value (in this case “ref”). The spectral dimension may need to be changed so that the peak intensities are associated with the correct atom of the pair. In case you have forgotten the spin names or the format of the peak list next to the file name selection button is a preview button which can be used to open the peak list in the default text editor. Set the other fields as needed. Click on “Next”. Note that a `RelaxWarning` will be thrown for all peak list entries which do

not match a spin system within the relax data store. This will cause the relax controller window to appear:

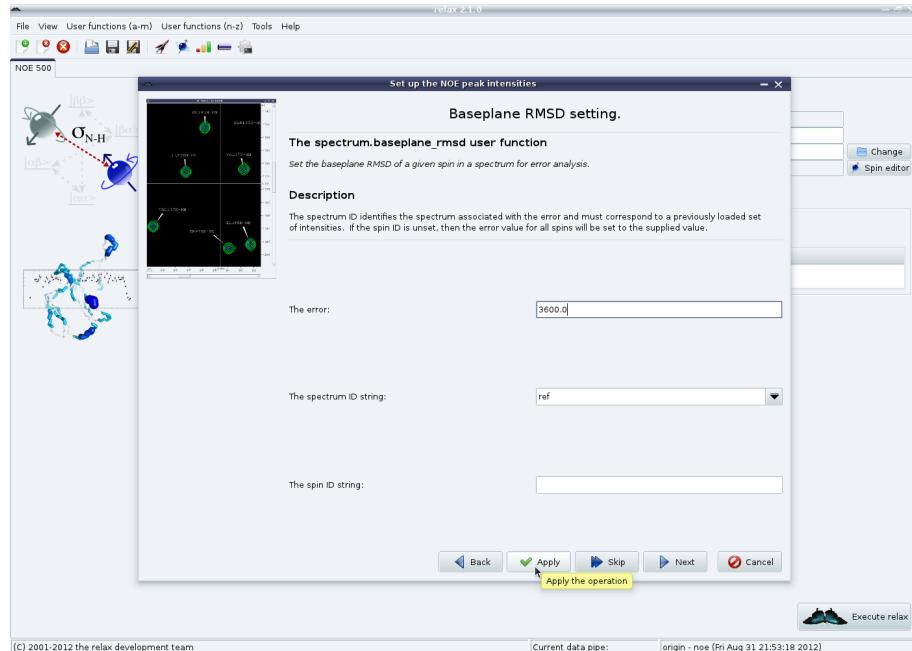


Carefully check these warnings to be sure that the data is correctly loaded and, if everything is fine, the relax controller window can be closed. If the dimension has been wrongly specified or some other setting is incorrect a **RelaxError** might appear saying that no data was loaded – you will then need to fix the settings and click on “Apply” again. The error type page should now appear.

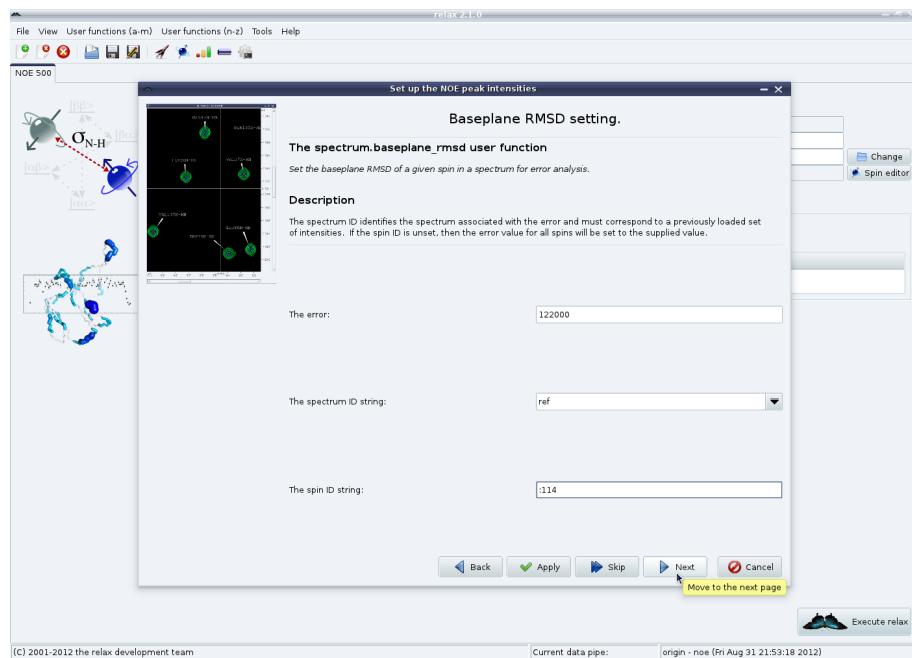


Please read the description in this window very carefully to know what to do next. In this example, we will choose “Baseplane RMSD”. For this specific example, Sparky’s “Extensions→Spectrum→Spectrum baseplane RMSD” option in the “F1” selection mode was used to measure empty regions of the spectrum (mainly in the random coil region) to determine an average

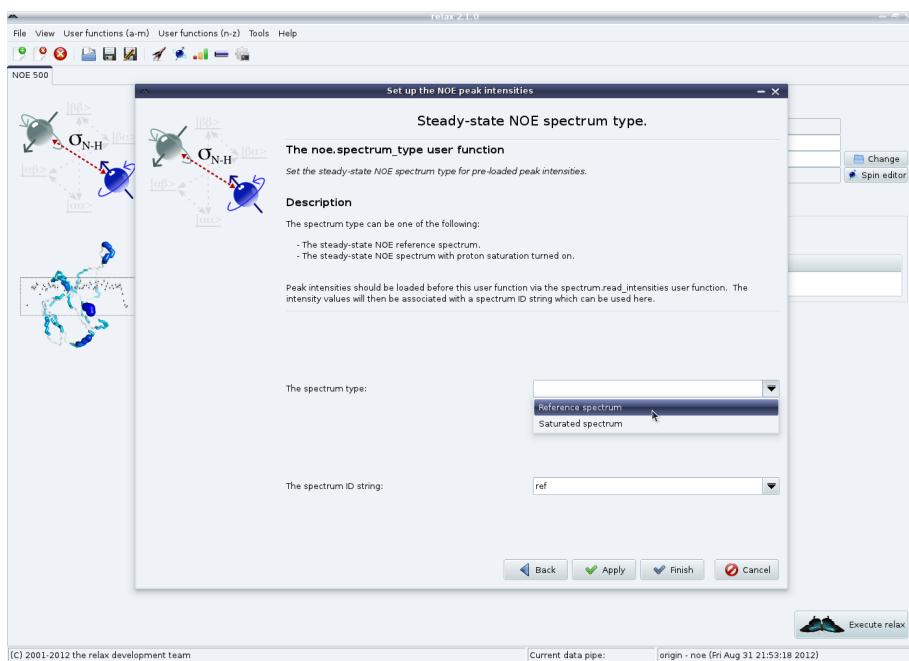
RMSD of approximately 3600. Set the value and click on “Apply”.



As glycine 114 is located close to the noise signal, its error was much higher at 122000. Individual spin errors can be set via the spin ID string (see section 4.2.2 on page 38 for information about spin IDs):

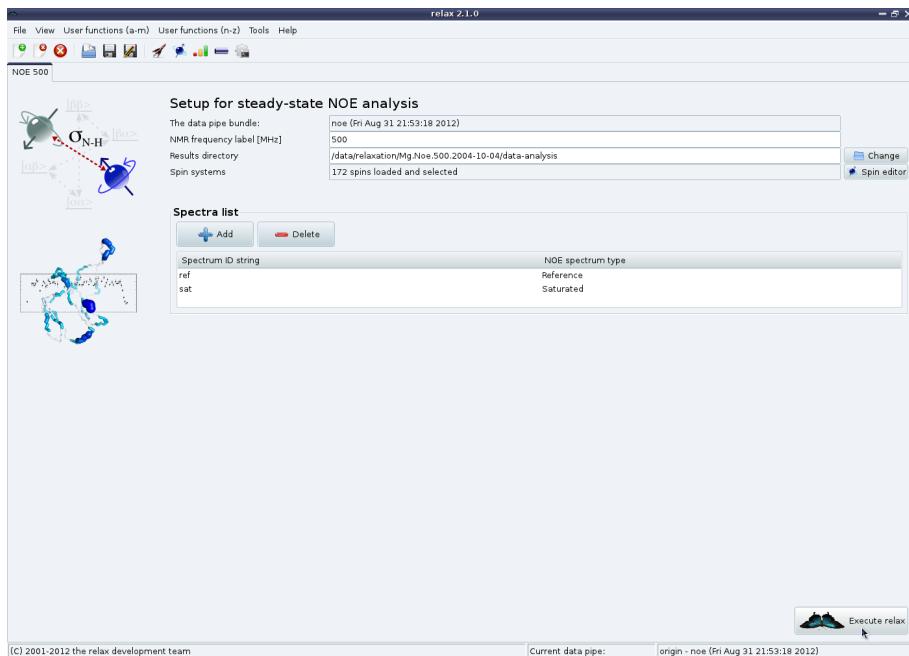


Finally select which type of spectrum this is and click on “Finish”:



The entire procedure should be repeated for the saturated spectrum (or you may have worked out that both can be loaded simultaneously by using the “Apply” button more often). For this example, the spectrum ID was set to “sat” and the baseplane RMSD to 3000 for all spins (except for G114 which had an error of 8500).

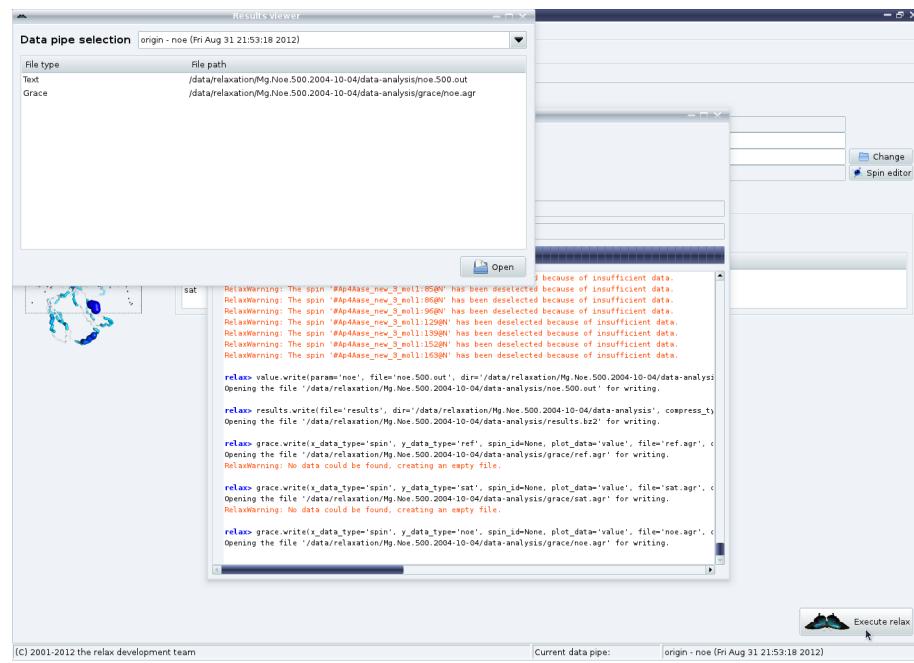
The NOE analysis tab should now look like:



6.4.6 NOE GUI mode – the NOE calculation

Now that everything is set up, simply click on “Execute relax” in the NOE analysis tab. The relax controller window will appear displaying many messages. These should all be

checked very carefully to make sure that everything has executed as you expected. The “Results viewer” window will also appear:

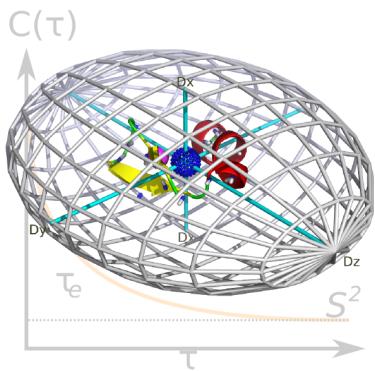


The results viewer window can be used to launch a text editor to see the NOE values and error or Grace to visualise the results (see Figure 6.1 on page 76).

As a last step, the relax state can be saved (via the “File” menu) and relax closed. Take one last look at the `noe.out` log file to be certain that there are no strange warnings or errors.

Chapter 7

Model-free analysis



7.1 Model-free theory

7.1.1 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

$$\chi^2(\theta) = \sum_{i=1}^n \frac{(R_i - R_i(\theta))^2}{\sigma_i^2}, \quad (7.1)$$

where the index i is the summation index ranging over all the experimentally collected relaxation data of all spins used in the analysis; R_i belongs to the relaxation data set R for an individual spin, a collection of spins, or the entire macromolecule and includes the R_1 , R_2 , and NOE data at all field strengths; $R_i(\theta)$ is the back-calculated relaxation value belonging to the set $R(\theta)$; 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 (7.1) 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.

7.1.2 The transformed relaxation equations – $R_i(\theta)$

The chi-squared equation is itself dependent on the relaxation equations through the back-calculated relaxation data $R(\theta)$. Letting the relaxation values of the set $R(\theta)$ be the $R_1(\theta)$, $R_2(\theta)$, and $\text{NOE}(\theta)$ 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 $\sigma_{\text{NOE}}(\theta)$ and the auto relaxation rate $R_1(\theta)$. Taking equation (7.6) below the transformed relaxation equations are

$$R_1(\theta) = R'_1(\theta), \quad (7.2a)$$

$$R_2(\theta) = R'_2(\theta), \quad (7.2b)$$

$$\text{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X} \frac{\sigma_{\text{NOE}}(\theta)}{R_1(\theta)}. \quad (7.2c)$$

whereas the relaxation equations are the $R_1(\theta)$, $R_2(\theta)$, $\sigma_{\text{NOE}}(\theta)$.

7.1.3 The relaxation equations – $R'_i(\theta)$

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

$$R_1(\theta) = d \left(J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H + \omega_X) \right) + cJ(\omega_X), \quad (7.3a)$$

$$R_2(\theta) = \frac{d}{2} \left(4J(0) + J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H) + 6J(\omega_H + \omega_X) \right) + \frac{c}{6} \left(4J(0) + 3J(\omega_X) \right) + R_{ex}, \quad (7.3b)$$

$$\sigma_{\text{NOE}}(\theta) = d \left(6J(\omega_H + \omega_X) - J(\omega_H - \omega_X) \right), \quad (7.3c)$$

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

$$d = \frac{1}{4} \left(\frac{\mu_0}{4\pi} \right)^2 \frac{(\gamma_H \gamma_X \hbar)^2}{\langle r^6 \rangle}, \quad (7.4)$$

$$c = \frac{(\omega_X \Delta\sigma)^2}{3}, \quad (7.5)$$

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

$$\text{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X} \frac{\sigma_{\text{NOE}}(\theta)}{R_1(\theta)}. \quad (7.6)$$

7.1.4 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](#))

$$J(\omega) = \frac{2}{5} \sum_{i=-k}^k c_i \cdot \tau_i \left(\frac{S^2}{1 + (\omega\tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega\tau_e\tau_i)^2} \right), \quad (7.7)$$

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

$$\begin{aligned} J(\omega) = \frac{2}{5} \sum_{i=-k}^k c_i \cdot \tau_i & \left(\frac{S^2}{1 + (\omega\tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega\tau_f\tau_i)^2} \right. \\ & \left. + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega\tau_s\tau_i)^2} \right). \end{aligned} \quad (7.8)$$

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 $S^2 = S_f^2 \cdot S_s^2$.

If these forms of the model-free spectral density functions are unfamiliar, that is because these are the numerically stabilised forms presented in [d'Auvergne and Gooley \(2008b\)](#). The original model-free spectral density functions presented in [Lipari and Szabo \(1982a\)](#) and [Clore et al. \(1990\)](#) are not the most numerically stable form of these equations. An important problem encountered in optimisation is round-off error in which machine precision influences the result of mathematical operations. The double reciprocal $\tau^{-1} = \tau_m^{-1} + \tau_e^{-1}$ used in the equations are operations which are particularly susceptible to round-off error, especially when $\tau_e \ll \tau_m$. By incorporating these reciprocals into the model-free spectral density functions and then simplifying the equations this source of round-off error can be eliminated, giving relax an edge over other model-free optimisation software.

7.1.5 Brownian rotational diffusion

In equations (7.7) and (7.8) the generic Brownian diffusion NMR correlation function presented in [d'Auvergne \(2006\)](#) has been used. This function is

$$C(\tau) = \frac{1}{5} \sum_{i=-k}^k c_i \cdot e^{-\tau/\tau_i}, \quad (7.9)$$

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

$$\mathfrak{D}_{iso} = \frac{1}{3}(\mathfrak{D}_x + \mathfrak{D}_y + \mathfrak{D}_z), \quad (7.10a)$$

$$\mathfrak{D}_a = \mathfrak{D}_z - \frac{1}{2}(\mathfrak{D}_x + \mathfrak{D}_y), \quad (7.10b)$$

$$\mathfrak{D}_r = \frac{\mathfrak{D}_y - \mathfrak{D}_x}{2\mathfrak{D}_a}, \quad (7.10c)$$

and are constrained by

$$0 < \mathfrak{D}_{iso} < \infty, \quad (7.11a)$$

$$0 \leq \mathfrak{D}_a < \frac{\mathfrak{D}_{iso}}{\frac{1}{3} + \mathfrak{D}_r} \leq 3\mathfrak{D}_{iso}, \quad (7.11b)$$

$$0 \leq \mathfrak{D}_r \leq 1. \quad (7.11c)$$

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

$$c_{-2} = \frac{1}{4}(d - e), \quad (7.12a)$$

$$c_{-1} = 3\delta_y^2\delta_z^2, \quad (7.12b)$$

$$c_0 = 3\delta_x^2\delta_z^2, \quad (7.12c)$$

$$c_1 = 3\delta_x^2\delta_y^2, \quad (7.12d)$$

$$c_2 = \frac{1}{4}(d + e), \quad (7.12e)$$

where

$$d = 3(\delta_x^4 + \delta_y^4 + \delta_z^4) - 1, \quad (7.13)$$

$$e = \frac{1}{\mathfrak{R}} \left[(1 + 3\mathfrak{D}_r)(\delta_x^4 + 2\delta_y^2\delta_z^2) + (1 - 3\mathfrak{D}_r)(\delta_y^4 + 2\delta_x^2\delta_z^2) - 2(\delta_z^4 + 2\delta_x^2\delta_y^2) \right], \quad (7.14)$$

and where

$$\mathfrak{R} = \sqrt{1 + 3\mathfrak{D}_r^2}. \quad (7.15)$$

The five correlation times τ_i are

$$1/\tau_{-2} = 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R}, \quad (7.16a)$$

$$1/\tau_{-1} = 6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r), \quad (7.16b)$$

$$1/\tau_0 = 6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r), \quad (7.16c)$$

$$1/\tau_1 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a, \quad (7.16d)$$

$$1/\tau_2 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R}. \quad (7.16e)$$

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

$$\mathfrak{D}_{iso} = \frac{1}{3}(\mathfrak{D}_{\parallel} + 2\mathfrak{D}_{\perp}), \quad (7.17a)$$

$$\mathfrak{D}_a = \mathfrak{D}_{\parallel} - \mathfrak{D}_{\perp}. \quad (7.17b)$$

and are constrained by

$$0 < \mathfrak{D}_{iso} < \infty, \quad (7.18a)$$

$$-\frac{3}{2}\mathfrak{D}_{iso} < \mathfrak{D}_a < 3\mathfrak{D}_{iso}. \quad (7.18b)$$

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

$$c_{-1} = \frac{1}{4}(3\delta_z^2 - 1)^2, \quad (7.19a)$$

$$c_0 = 3\delta_z^2(1 - \delta_z^2), \quad (7.19b)$$

$$c_1 = \frac{3}{4}(\delta_z^2 - 1)^2. \quad (7.19c)$$

The five correlation times τ_i are

$$1/\tau_{-1} = 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a, \quad (7.20a)$$

$$1/\tau_0 = 6\mathfrak{D}_{iso} - \mathfrak{D}_a, \quad (7.20b)$$

$$1/\tau_1 = 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a. \quad (7.20c)$$

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

$$1/\tau_m = 6\mathfrak{D}_{iso}. \quad (7.21)$$

This is diffusion equation presented in [Bloembergen et al. \(1948\)](#).

7.1.6 The model-free models

Extending the list of models given in [Mandel et al. \(1995\)](#); [Fushman et al. \(1997\)](#); [Orekhov et al. \(1999a\)](#); [Korzhnev et al. \(2001\)](#); [Zhuravleva et al. \(2004\)](#), the models built

into relax include

$$m0 = \{\}, \quad (7.22.0)$$

$$m1 = \{S^2\}, \quad (7.22.1)$$

$$m2 = \{S^2, \tau_e\}, \quad (7.22.2)$$

$$m3 = \{S^2, R_{ex}\}, \quad (7.22.3)$$

$$m4 = \{S^2, \tau_e, R_{ex}\}, \quad (7.22.4)$$

$$m5 = \{S^2, S_f^2, \tau_s\}, \quad (7.22.5)$$

$$m6 = \{S^2, \tau_f, S_f^2, \tau_s\}, \quad (7.22.6)$$

$$m7 = \{S^2, S_f^2, \tau_s, R_{ex}\}, \quad (7.22.7)$$

$$m8 = \{S^2, \tau_f, S_f^2, \tau_s, R_{ex}\}, \quad (7.22.8)$$

$$m9 = \{R_{ex}\}. \quad (7.22.9)$$

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

$$tm0 = \{\tau_m\}, \quad (7.23.0)$$

$$tm1 = \{\tau_m, S^2\}, \quad (7.23.1)$$

$$tm2 = \{\tau_m, S^2, \tau_e\}, \quad (7.23.2)$$

$$tm3 = \{\tau_m, S^2, R_{ex}\}, \quad (7.23.3)$$

$$tm4 = \{\tau_m, S^2, \tau_e, R_{ex}\}, \quad (7.23.4)$$

$$tm5 = \{\tau_m, S^2, S_f^2, \tau_s\}, \quad (7.23.5)$$

$$tm6 = \{\tau_m, S^2, \tau_f, S_f^2, \tau_s\}, \quad (7.23.6)$$

$$tm7 = \{\tau_m, S^2, S_f^2, \tau_s, R_{ex}\}, \quad (7.23.7)$$

$$tm8 = \{\tau_m, S^2, \tau_f, S_f^2, \tau_s, R_{ex}\}, \quad (7.23.8)$$

$$tm9 = \{\tau_m, R_{ex}\}. \quad (7.23.9)$$

7.1.7 Model-free optimisation theory

The implementation of optimisation in relax is discussed in detail in Chapter 14. To understand the concepts in this subsection, it is best to look at that chapter first.

The model-free space

In model-free analysis the target function $f(\theta)$ is the chi-squared equation

$$\chi^2(\theta) = \sum_{i=1}^n \frac{(R_i - R_i(\theta))^2}{\sigma_i^2}, \quad (7.24)$$

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, $R_i(\theta)$ is the back calculated relaxation data belonging to the set $R(\theta)$, 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 spin. If the diffusion parameters are optimised simultaneously with the model-free parameters the summation index ranges over all relaxation data of all selected spins 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 (7.7) and (7.8). The theoretical R_1 , R_2 , and NOE values are then back-calculated using Equations (7.3a), (7.3b), (7.3c), and (7.6). Finally, the chi-squared value is calculated using Equation (7.24).

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 (7.7) and (7.8) are necessary. Then the partial derivatives of the relaxation equations (7.3a) to (7.3c) followed by the NOE equation (7.6) are needed. Finally the partial derivative of the chi-squared formula (7.24) 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 as Chapter 15.

Grid search

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. Therefore the model-free space renders many optimisation algorithms ineffective (d'Auvergne and Gooley, 2008b).

Parameter constraints

To understand this section, please see Section 14.5 on page 310. 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 $\tau_s - \tau_f \geq 0$.

For the parameters specific to individual spins the linear constraints in the notation of

(14.18) are

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} S^2 \\ S_f^2 \\ S_s^2 \\ \tau_e \\ \tau_f \\ \tau_s \\ R_{ex} \\ r \\ CSA \end{pmatrix} \geq \begin{pmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.9e^{-10} \\ 2e^{-10} \\ 300e^{-6} \\ 0 \end{pmatrix}. \quad (7.25)$$

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

$$0 \leq S^2 \leq 1, \quad (7.26a)$$

$$0 \leq S_f^2 \leq 1, \quad (7.26b)$$

$$0 \leq S_s^2 \leq 1, \quad (7.26c)$$

$$S^2 \leq S_f^2, \quad (7.26d)$$

$$S^2 \leq S_s^2, \quad (7.26e)$$

$$\tau_e \geqslant 0, \quad (7.26f)$$

$$\tau_f \geqslant 0, \quad (7.26g)$$

$$\gamma_s \geq 0, \quad (7.26\text{ii})$$

$$\tau_s \geq 0, \quad \tau_1 < \tau_2 \quad (7.26i)$$

$$B_{\perp} \geq 0 \quad (7.26k)$$

$$0.9e^{-10} \leq r \leq 2e^{-10} \quad (7.26)$$

$$-300e^{-6} \leq CSA \leq 0. \quad (7.26m)$$

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 $\tau_e, \tau_f, \tau_s \leq 2\tau_m$ was implemented. When the global correlation time is fixed the constraints in the matrix notation of (14.18) are

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_e \\ \tau_f \\ \tau_s \end{pmatrix} \geqslant \begin{pmatrix} -2\tau_m \\ -2\tau_m \\ -2\tau_m \end{pmatrix}. \quad (7.27)$$

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

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ 2 & 0 & -1 & 0 \\ 2 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_m \\ \tau_e \\ \tau_f \\ \tau_s \end{pmatrix} \geqslant \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \quad (7.28)$$

For the parameters of the diffusion tensor the constraints utilised are

$$0 \leq \tau_m \leq 200.0e^{-9}, \quad (7.29a)$$

$$\mathfrak{D}_a \geqslant 0, \quad (7.29b)$$

$$0 \leq \mathfrak{D}_r \leq 1, \quad (7.29c)$$

which in the matrix notation of (14.18) become

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \tau_m \\ \mathfrak{D}_a \\ \mathfrak{D}_r \end{pmatrix} \geqslant \begin{pmatrix} 0 \\ -200.0e^{-9} \\ 0 \\ 0 \\ -1 \end{pmatrix}. \quad (7.30)$$

The upper limit of 200 ns on τ_m prevents the parameter from heading towards infinity when model failure occurs (see [d'Auvergne and Gooley \(2006\)](#)). This can significantly decrease the computation time. To isolate the prolate spheroid the constraint

$$(1) \cdot (\mathfrak{D}_a) \geqslant (0), \quad (7.31)$$

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

$$(-1) \cdot (\mathfrak{D}_a) \geq (0). \quad (7.32)$$

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

Diagonal scaling

The concept of diagonal scaling is explained in Section 14.6 on page 312.

For the model-free analysis the scaling factor of one is used for the order parameter and a scaling factor of $1e^{-12}$ is used for the correlation times. The R_{ex} parameter is 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 spins is

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

$$\begin{pmatrix} 1e^{-12} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e^7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (7.34)$$

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

$$\begin{pmatrix} 1e^{-12} & 0 & 0 & 0 \\ 0 & 1e^7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (7.35)$$

7.2 Optimisation of a single model-free model

7.2.1 Single model-free model script mode – 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/single_model.py`. The text of the script is:

```

1 # Script for model-free analysis.
2
3 # Create the data pipe.
4 name = 'm4'
5 pipe.create(name, 'mf')
6
7 # Set up the 15N spins.
8 sequence.read('noe.500.out', res_num_col=1, res_name_col=2)
9 spin.name('N')
10 spin.element(element='N', spin_id='@N')
11 spin.isotope('15N', spin_id='@N')
12
13 # Load the relaxation data.
14 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
15   res_num_col=1, data_col=3, error_col=4)
15 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
16   res_num_col=1, data_col=3, error_col=4)
16 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
17   res_num_col=1, data_col=3, error_col=4)
17 relax_data.read(ri_id='R1_500', ri_type='R1', frq=500.0*1e6, file='r1.500.out',
18   res_num_col=1, data_col=3, error_col=4)
18 relax_data.read(ri_id='R2_500', ri_type='R2', frq=500.0*1e6, file='r2.500.out',
19   res_num_col=1, data_col=3, error_col=4)
19 relax_data.read(ri_id='NOE_500', ri_type='NOE', frq=500.0*1e6, file='noe.500.out',
20   res_num_col=1, data_col=3, error_col=4)
20
21 # Initialise the diffusion tensor.
22 diffusion_tensor.init(10e-9, fixed=True)
23
24 # Create all attached protons.
```

```

25 sequence.attach_protons()
26
27 # Define the magnetic dipole-dipole relaxation interaction.
28 interatom.define(spinid1='15N', spin_id2='1H', direct_bond=True)
29 interatom.set_dist(spin_id1='15N', spin_id2='1H', ave_dist=1.02 * 1e-10)
30 #interatom.unit_vectors()
31
32 # Define the CSA relaxation interaction.
33 value.set(-172 * 1e-6, 'csa')
34
35 # Select the model-free model.
36 model_free.select_model(model=name)
37
38 # Grid search.
39 minimise.grid_search(inc=11)
40
41 # Minimise.
42 minimise.execute('newton')
43
44 # Monte Carlo simulations.
45 monte_carlo.setup(number=100)
46 monte_carlo.create_data()
47 monte_carlo.initial_values()
48 minimise.execute('newton')
49 eliminate()
50 monte_carlo.error_analysis()
51
52 # Finish.
53 results.write(file='results', force=True)
54 state.save('save', force=True)

```

7.2.2 Single model-free model script mode – explanation

The above script consists of three major sections:

Loading of data Firstly a data pipe called “m4” is created to hold all of the analysis data.

Then the ¹⁵N spin system data consisting of molecule, residue, and spin information is loaded into relax from the columns of the noe.500.out file, assuming that only residue numbers and names are present and are in the first and second columns respectively. The options of this `sequence.read` user function allow the molecule name, residue number, residue name, spin number, or spin name columns to be specified if desired. The ¹⁵N spin is then set up using the `spin` user functions. The next part is to load all of the relaxation data, to set up the initial diffusion tensor, create the ¹H spins required for the magnetic dipole-dipole interaction, and to set up the magnetic dipole-dipole and CSA relaxation mechanisms. Finally the model-free model “m4” is chosen.

Optimisation The optimisation of model-free models requires an initial grid search to find a position close to the minimum, followed by the high precision Newton optimisation together with the Method of Multipliers constraint algorithm ([d'Auvergne and Gooley, 2008b](#)). Errors are propagated from the relaxation data to the model-free parameters via Monte Carlo simulations which is a multi-step process in relax (designed for flexibility and to teach how the simulations are constructed and carried out).

Data output The last stage consists of writing out the XML formatted results file which contains all of the data in the current data pipe, as well as the XML formatted save file which contains not only the current data pipe data but all of the relax data store data. Both files can be loaded back into relax later on.

7.3 Optimisation of all model-free models

7.3.1 All model-free models script mode – the sample script

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

```

1 # Set the data pipe names (also the names of preset model-free models).
2 pipes = ['m0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9']
3
4 # Loop over the pipes.
5 for name in pipes:
6     # Create the data pipe.
7     pipe.create(name, 'mf')
8
9     # Set up the 15N spins.
10    sequence.read('noe.500.out', res_num_col=1)
11    spin.name('N')
12    spin.element(element='N', spin_id='@N')
13    spin.isotope('15N', spin_id='@N')
14
15    # Load a PDB file.
16    structure.read_pdb('example.pdb')
17
18    # Load the relaxation data.
19    relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
20    res_num_col=1, data_col=3, error_col=4)
21    relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
22    res_num_col=1, data_col=3, error_col=4)
23    relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
24    res_num_col=1, data_col=3, error_col=4)
25    relax_data.read(ri_id='R1_500', ri_type='R1', frq=500.0*1e6, file='r1.500.out',
26    res_num_col=1, data_col=3, error_col=4)
27    relax_data.read(ri_id='R2_500', ri_type='R2', frq=500.0*1e6, file='r2.500.out',
28    res_num_col=1, data_col=3, error_col=4)
29    relax_data.read(ri_id='NOE_500', ri_type='NOE', frq=500.0*1e6, file='noe.500.out',
30    res_num_col=1, data_col=3, error_col=4)
31
32    # Set up the diffusion tensor.
33    diffusion_tensor.init(1e-8, fixed=True)
34
35    # Generate the 1H spins for the magnetic dipole-dipole relaxation interaction.
36    sequence.attach_protons()
37
38    # Define the magnetic dipole-dipole relaxation interaction.
39    interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
40    interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)
41    structure.get_pos('@N')
42    structure.get_pos('@H')
43    interatom.unit_vectors()
```

```

38
39      # Define the chemical shift relaxation interaction.
40      value.set(-172 * 1e-6, 'csa', spin_id='@N')
41
42      # Select the model-free model.
43      model_free.select_model(model=name)
44
45      # Minimise.
46      minimise.grid_search(inc=11)
47      minimise.execute('newton')
48
49      # Write the results.
50      results.write(file='results', force=True)
51
52      # Save the program state.
53      state.save('save', force=True)

```

7.3.2 All model-free models script mode – explanation

The above script is very similar in spirit to the previous single model script in section 7.2 on page 94. The major difference is that this script loops over all of the model-free models, saving all of the results in the `save.bz2` file.

7.4 Model-free model selection

7.4.1 Model-free model selection script mode – the sample script

The sample script which demonstrates both model-free model elimination and model-free model selection between models from m_0 to m_9 is `model_free/modsel.py`. The text of the script is:

```

1  # Set the data pipe names.
2  pipes = ['m0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9']
3
4  # Loop over the data pipe names.
5  for name in pipes:
6      print("\n\n# " + name + " #")
7
8      # Create the data pipe.
9      pipe.create(name, 'mf')
10
11     # Reload precalculated results from the file 'm1/results', etc.
12     results.read(file='results', dir=name)
13
14     # Model elimination.
15     eliminate()
16
17     # Model selection.
18     model_selection(method='AIC', modsel_pipe='aic')
19
20     # Write the results.
21     state.save('save', force=True)
22     results.write(file='results', force=True)

```

7.4.2 Model-free model selection script mode – explanation

This script is designed to be used in conjunction with the `model_free/mf_multimodel.py` script in the previous section. It will load all of the results files from the previous script and then perform the following:

Model-free model elimination The optimisation of model-free models performed by the previous script will fail for certain data sets together with certain models. To ensure that these models are never selected, they are removed from the analysis (see [d'Auvergne and Gooley \(2006\)](#)).

Model-free model selection The AIC model selection as described in [d'Auvergne and Gooley \(2003\)](#) will be used to determine which model-free model best describes the relaxation data.

Data output Finally both a save state and result file will be created.

These three sample scripts describe the basic components of model-free analysis. However a full analysis requires the construction of a much more complex iterative procedure. The following sections will describe both the original diffusion seeded approaches as well as the new model-free protocol built into relax.

7.5 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 7.1.

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.

To implement this analysis, a number of scripts would need to be written. There is no sample script in relax for performing this analysis. The simple sample scripts from above would need to be extended. For example a starting script for determining the initial diffusion tensor estimates based on the R1/R2 ratio of [Kay et al. \(1989\)](#) would have to be written. The tensor from this script could then be feed into the `model_free/mf_multimodel.py` script, followed by the `model_free/modsel.py` script, and then a third script written to optimise the diffusion tensor. A master script could be written first run the initial diffusion tensor script, then to iteratively execute the last three scripts until convergence, and finally to select the best diffusion model (see Figure 7.1). Alternatively, these could all be combined into one super script.

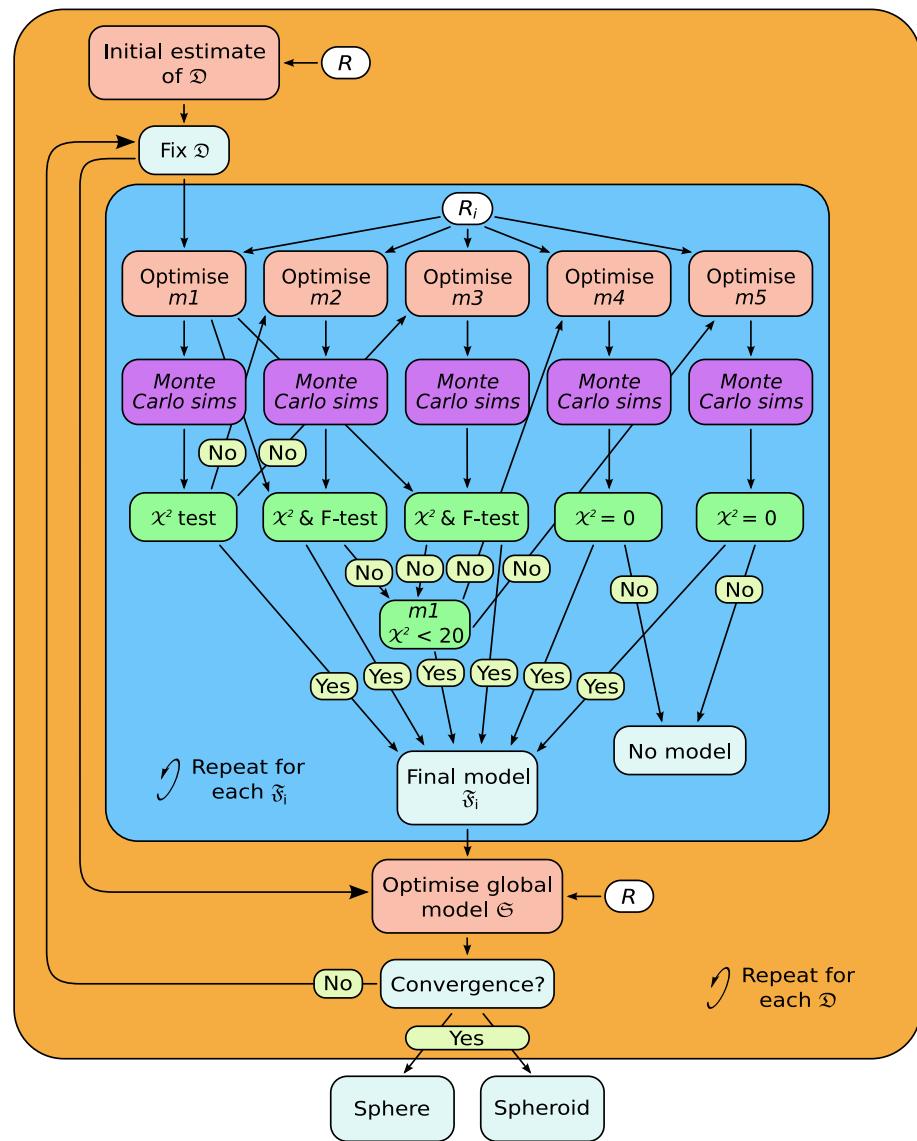


Figure 7.1: A schematic of the model-free optimisation protocol of Mandel et al. (1995). This specific protocol is for single field strength data. The initial diffusion tensor estimate is calculated using the R_2/R_1 ratio. The diffusion parameters of \mathfrak{D} are held constant while model-free models $m1$ to $m5$ (7.22.1–7.22.5) of the set \mathfrak{F}_i for each spin 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 spins of the molecule. The global model \mathfrak{G} , 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).

7.6 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, when combined with AIC model selection ([d'Auvergne and Gooley, 2003](#)), is illustrated in Figure 7.2.

Again this protocol is not implemented in the relax sample scripts. This would have to be implemented in exactly the same manner as described in the previous section, but using the AIC model selection build into relax. Constructing this set of scripts, or a single master script, would be much easier than the [Mandel et al. \(1995\)](#) protocol as Modelfree4 would not need to be used, and the handling of F-tests and chi-squared tests is avoided.

7.7 The new model-free optimisation protocol

Here a new, fully automated model-free optimisation protocol will be presented. This protocol, defined in [d'Auvergne and Gooley \(2007\)](#) and [d'Auvergne and Gooley \(2008c\)](#), is significantly different from all those that came before, reversing the diffusion seeded paradigm as detailed below. Within relax it is referred to as the “new protocol” or the “d’Auvergne protocol”. The later name is to allow for more advanced protocols to be developed and added to relax by adventurous users in the future. Note that for advanced model-free analysis protocols, such as this one, that multiple field relaxation data is essential.

7.7.1 The new protocol – model-free models

The study of the dynamics of a macromolecule using model-free analysis to interpret the R_1 and R_2 relaxation rates together with the steady-state heteronuclear NOE brings two distinct, yet linked physical theories into play. The Brownian rotational diffusion of the molecule is the major contributor to relaxation. Although having less of an influence on relaxation the internal dynamics of individual nuclei within the molecule is nevertheless significant. The model-free description of the internal motion and the global diffusion of the entire molecule are theories which are linked due to their dependence on the same relaxation data. The model-free models for individual spin system constructed from the original and extended model-free theories ([Lipari and Szabo, 1982a,b](#); [Clore et al., 1990](#)) are assembled using parametric restrictions, the dropping of insignificant parameters, and the addition of the chemical exchange parameter R_{ex} . Labelled as $m0$ to $m9$ (Models 7.22.0–7.22.9 on page 90) these models are an extended list of those in ([Fushman et al., 1997](#); [Orekhov et al., 1999a](#); [Korzhnev et al., 2001](#); [Zhuravleva et al., 2004](#)).

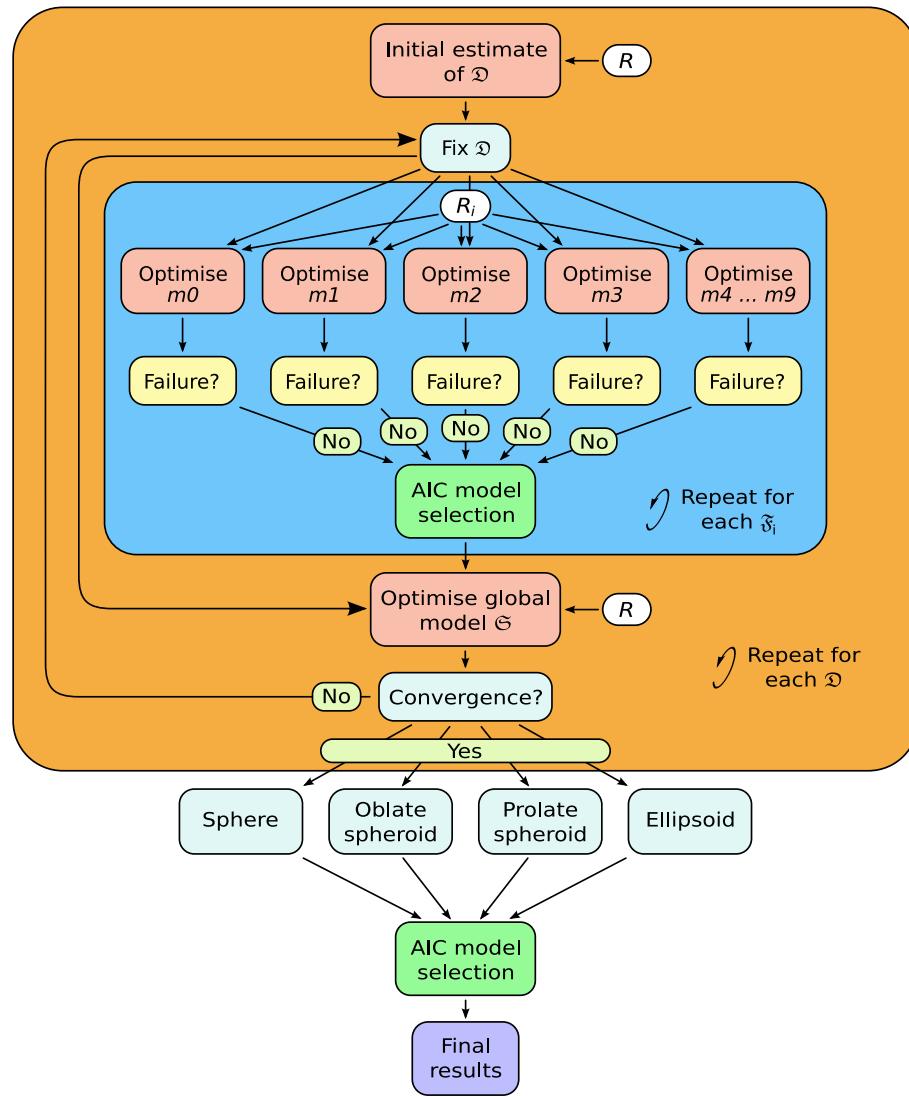


Figure 7.2: A schematic of model-free analysis using the diffusion seeded paradigm – the initial diffusion tensor estimate – together with AIC model selection and model elimination. The initial estimates of the parameters of Ω are held constant while model-free models m_0 to m_9 (7.22.0–7.22.9) of the 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 Ω 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.

7.7.2 The new protocol – the diffusion tensor

The ellipsoid

The most general form of Brownian rotational diffusion of macromolecules is the diffusion of an ellipsoid, a diffusion also labelled as asymmetric or fully anisotropic. This diffusion tensor can be fully specified by the geometric parameters \mathfrak{D}_x , \mathfrak{D}_y , and \mathfrak{D}_z , the eigenvalues of the tensor, as well as three orientational parameters, the Euler angles α , β , and γ . The diffusion equation for an ellipsoid was derived using the reasoning of [Einstein \(1905\)](#) in the two papers of [Perrin \(1934\)](#) and [Perrin \(1936\)](#). Following this, [Favro \(1960\)](#) unknowingly derived the same equations as presented in [Perrin \(1936\)](#) using a pseudo quantum mechanical approach. Borrowing heavily from [Perrin \(1936\)](#), [Woessner \(1962\)](#) derived the correlation function relevant for NMR relaxation of a bond vector rigidly attached to an ellipsoid. However these equations are not fully simplified and the parameter set $\{\mathfrak{D}_x, \mathfrak{D}_y, \mathfrak{D}_z, \alpha, \beta, \gamma\}$, the eigenvalues and Euler angles defining the tensor, is not optimally constructed for minimisation. A parameter shift to the set $\{\mathfrak{D}_{iso}, \mathfrak{D}_a, \mathfrak{D}_r, \alpha, \beta, \gamma\}$, whereby the three geometric parameters are respectively the isotropic, anisotropic, and rhombic components of the diffusion tensor, drastically simplifies optimisation and is how the diffusion tensor is implemented within relax.

The spheroid

When two of the eigenvalues of the diffusion tensor are equal the molecule diffuses as a spheroid. This is also called axially symmetric anisotropic diffusion and can be described by the two geometric parameters \mathfrak{D}_{iso} and \mathfrak{D}_a together with the polar angle θ and azimuthal angle ϕ which define the unique axis of the diffusion tensor. Two classes of spheroid can be distinguished dependent on the relative values of the eigenvalues – the prolate and oblate spheroids. By using parametric constraints, both tensor types can be optimised within relax.

The sphere

The simplest form of diffusion occurs when all three eigenvalues are equal and the molecule diffuses as a sphere. This isotropic rotation can be characterised by the single parameter \mathfrak{D}_{iso} which is related to the global correlation time by the formula $1/\tau_m = 6\mathfrak{D}_{iso}$ ([Bloembergen et al., 1948](#)).

The local τ_m model-free models

Not only can the diffusion tensor be optimised as a global model affecting all spins of the molecule but a set of model-free models can be constructed in which each spin is assumed to diffuse independently. In these models a single local τ_m parameter approximates the true, multiexponential description of the Brownian rotational diffusion of the molecule. Each spin of the macromolecule is treated independently. Another set of model-free models which include the local τ_m parameter can be created and include $tm0$ to $tm9$ ([Models 7.23.0–7.23.9 on page 90](#)). These are simply models $m0$ to $m9$ with the local τ_m

parameter added. These models are an extension of the ideas introduced in [Barbato et al. \(1992\)](#) and [Schurr et al. \(1994\)](#) whereby the model *tm2*, the original Lipari and Szabo model-free equation with a local τ_m parameter, is optimised to avoid issues with inaccurate diffusion tensor approximations.

Determination of the diffusion tensor from the local τ_m parameter

In [Brüschweiler et al. \(1995\)](#) and further investigated in [Lee et al. \(1997\)](#), a methodology for determining the diffusion tensor from the local τ_m parameter together with the orientation of the XH bond represented by the unit vector μ_i was presented. A local τ_m value was obtained for each spin i by optimising model *tm2*. The $\tau_{m,i}$ values were approximated using the quadric model

$$(6\tau_{m,i})^{-1} = \mu_i^T Q \mu_i, \quad (7.36)$$

where the eigenvalues of the matrix Q are defined as $Q_x = (\mathfrak{D}_y + \mathfrak{D}_z)/2$, $Q_y = (\mathfrak{D}_x + \mathfrak{D}_z)/2$, and $Q_z = (\mathfrak{D}_x + \mathfrak{D}_y)/2$. The diffusion tensor is then found by linear least-squares fitting.

7.7.3 The universal solution \mathfrak{U}

The complex model-free problem, in which the motions of each spin are both mathematically and statistically dependent on the diffusion tensor and vice versa, was formulated using set theory in [d'Auvergne and Gooley \(2007\)](#). This paper is important for understanding the entire concept of the new protocol in relax and for truly grasping the complexity of the model-free problem. The solution $\widehat{\mathfrak{U}}$ to the model-free problem was derived as an element of the universal set \mathfrak{U} , the union of the diverse model-free parameter spaces \mathfrak{S} . Each set \mathfrak{S} was constructed from the union of the model-free models \mathfrak{F} for all spins and the diffusion parameter set \mathfrak{D} . A single parameter loss on a single spin shifts optimisation to a different space \mathfrak{S} . Ever since the seminal work of [Kay et al. \(1989\)](#) the model-free problem has been tackled by first finding an initial estimate of the diffusion tensor and then determining the model-free dynamics of the system (see Sections 7.5 on page 98 and 7.6 on page 100). This diffusion seeded paradigm is now highly evolved and much theory has emerged to improve this path to the solution $\widehat{\mathfrak{U}}$. The technique can, at times, suffer from a number of issues including the two minima problem of the spheroid diffusion tensor parameter space, the appearance of artificial chemical exchange ([Tjandra et al., 1996](#)), the appearance of artificial nanosecond motions ([Schurr et al., 1994](#)), and the hiding of internal nanosecond motions caused by the violation of the rigidity assumption ([Orekhov et al., 1995, 1999a,b](#)).

7.7.4 Model-free analysis in reverse

A different approach was proposed in [d'Auvergne and Gooley \(2008c\)](#) for finding the universal solution $\widehat{\mathfrak{U}}$ of the extremely complex, convoluted model-free optimisation and modelling problem ([d'Auvergne and Gooley, 2007](#)), defined as

$$\widehat{\mathfrak{U}} = \hat{\theta} \in \left\{ \mathfrak{S} : \min_{\hat{\theta} \in \mathfrak{U}} \Delta_{K-L}(\hat{\theta}) \right\}, \quad \text{s.t. } \hat{\theta} = \arg \min \left\{ \chi^2(\theta) : \theta \in \mathfrak{S} \right\}. \quad (7.37)$$

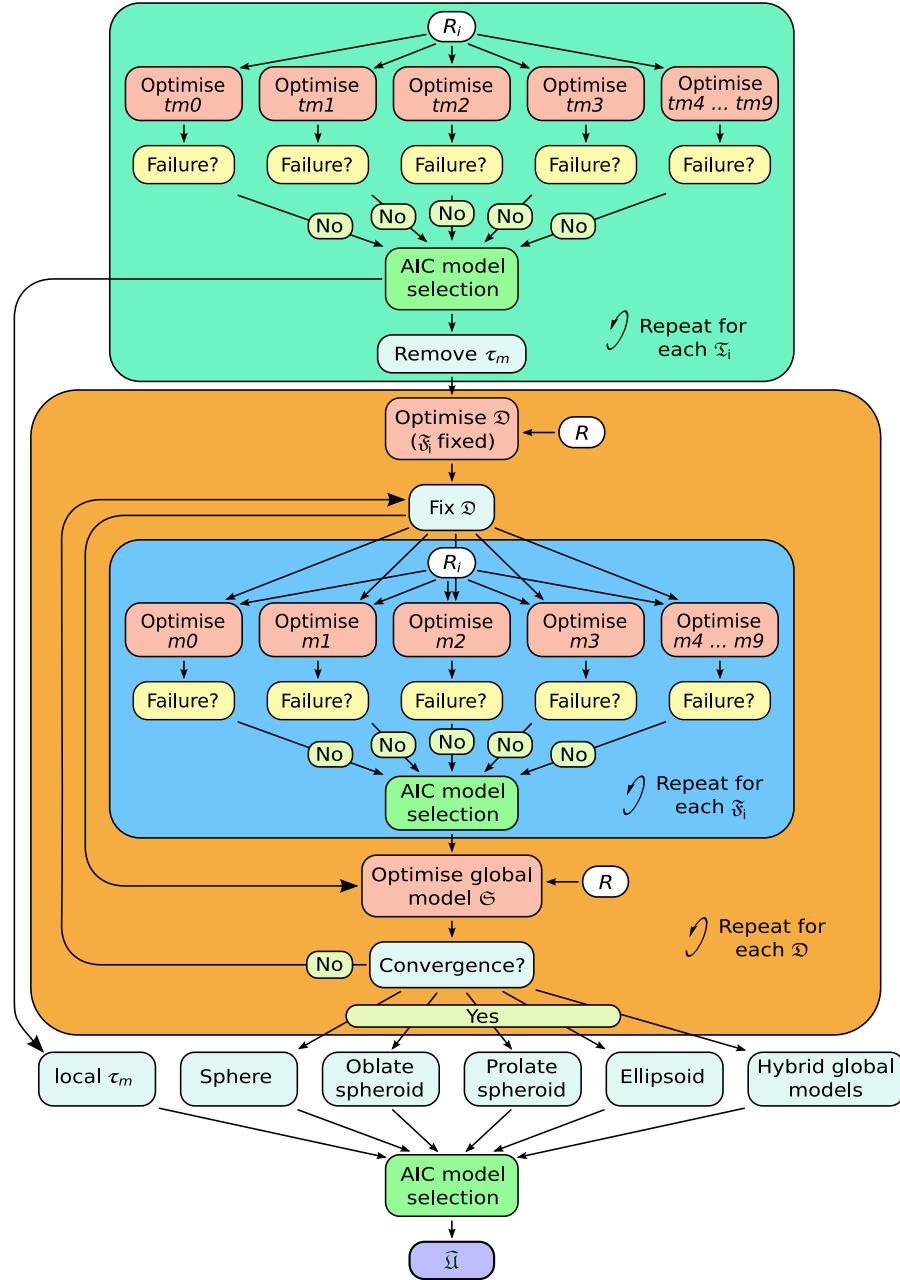


Figure 7.3: A schematic of the new model-free optimisation protocol. Initially models tm_0 to tm_9 (7.23.0–7.23.9) of the set \mathfrak{T}_i for each spin system i are optimised, model elimination used to remove failed models, and AIC model selection used to pick the best model. Once all the \mathfrak{T}_i 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 7.2. Convergence is however precisely defined as identical models \mathfrak{S} , identical χ^2 values, and identical parameters θ between two iterations. The universal solution $\hat{\mathfrak{U}}$, 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.

This notation says that the minimised parameter vector within the space \mathfrak{S} which minimises the common Kullback-Leibler discrepancy Δ_{K-L} is selected from the universal set \mathfrak{U} as the universal solution $\hat{\mathfrak{U}}$. The discrepancy of [Kullback and Leibler \(1951\)](#) is a measure of how well the model fits the data, in this case how well the global model \mathfrak{S} of the diffusion tensor together with the model-free models of all residues fits the relaxation data. This selection is subject to the condition that $\hat{\theta}$ is the argument or specific parameter vector which minimises the chi-squared function $\chi^2(\theta)$ such that θ is an element of the space \mathfrak{S} . Whereas the minimisation of the continuous chi-squared function within the single space \mathfrak{S} belongs to the mathematical field of optimisation ([Nocedal and Wright, 1999](#)), the selection of the universe \mathfrak{S} which minimises the discrepancy belongs to the statistical field of model selection ([Akaike, 1973](#); [Schwarz, 1978](#); [Linhart and Zucchini, 1986](#); [Zucchini, 2000](#); [d'Auvergne and Gooley, 2003](#)).

This new model-free optimisation protocol incorporates the ideas of the local τ_m model-free model ([Barbato et al., 1992](#); [Schurr et al., 1994](#)) and the optimisation of the diffusion tensor using information from these models, analogously to the linear least-squares fitting of the quadric model ([Brüschweiler et al., 1995](#); [Lee et al., 1997](#)). The protocol also follows the lead of the model-free optimisation protocol presented in [Butterwick et al. \(2004\)](#) whereby the diffusion seeded paradigm was reversed. Rather than starting with an initial estimation of the global diffusion tensor from the set \mathfrak{D} the protocol starts with the model-free parameters from \mathfrak{F} .

The first step of the [Butterwick et al. \(2004\)](#) protocol is the reduced spectral density mapping of [Farrow et al. \(1995\)](#). As R_{ex} has been eliminated from the analysis, three model-free models corresponding to $tm1$, $tm2$, and $tm5$ (Models [7.23.1](#), [7.23.2](#), and [7.23.5](#) on page [90](#)) are employed. The model-free parameters are optimised using the reduced spectral density values and the best model is selected using F-tests. The spherical, spheroidal, and ellipsoidal diffusion tensors are obtained by linear least-squares fitting of the quadric model of Equation [\(7.36\)](#) using the local τ_m values ([Brüschweiler et al., 1995](#); [Lee et al., 1997](#)). The best diffusion model is selected via F-tests and refined by iterative elimination of spins systems with high chi-squared values. This tensor is used to calculate local τ_m values for each spin system, approximating the multiexponential sum of the Brownian rotational diffusion correlation function with a single exponential, using the quadric model of Equation [\(7.36\)](#). In the final step of the protocol these τ_m values are fixed and $m1$, $m2$, and $m5$ (Models [7.22.1](#), [7.22.2](#), and [7.22.5](#) on page [90](#)) are optimised and the best model-free model selected using F-tests.

The new model-free protocol built into relax utilises the core foundation of the [Butterwick et al. \(2004\)](#) protocol yet its divergent implementation is designed to solve the universal equation of [d'Auvergne and Gooley \(2007\)](#) to find $\hat{\mathfrak{U}}$ (Equation [7.37](#)). Models $tm0$ to $tm9$ ([7.23.0–7.23.9](#) on page [90](#)) in which no global diffusion parameters exist are employed to significantly collapse the complexity of the problem. Model-free minimisation ([d'Auvergne and Gooley, 2008b](#)), model elimination ([d'Auvergne and Gooley, 2006](#)), and then AIC model selection ([Akaike, 1973](#); [d'Auvergne and Gooley, 2003](#)) can be carried out in the absence of the influence of global parameters. By removing the local τ_m parameter and holding the model-free parameter values constant these models can then be used to optimise the diffusion parameters of \mathfrak{D} . Model-free optimisation, model elimination, AIC model selection, and optimisation of the global model \mathfrak{S} is iterated until convergence. The iterations allow for sliding between different universes \mathfrak{S} to enable the collapse of model complexity, to refine the diffusion tensor, and to find the solution within the universal set

¶. The last step is the AIC model selection between the different diffusion models. Because the AIC criterion approximates the Kullback-Leibler discrepancy ([Kullback and Leibler, 1951](#)), central to the universal solution of Equation (7.37), it was chosen for all three model selection steps over BIC model selection ([Schwarz, 1978](#); [d'Auvergne and Gooley, 2003](#); [Chen et al., 2004](#)). The new protocol avoids the problem of under-fitting whereby artificial motions appear, avoids the problems involved in finding the initial diffusion tensor within \mathfrak{D} , and avoids the problem of hidden internal nanosecond motions and the inability to slide between universes to get to $\hat{\mathfrak{U}}$ (see [d'Auvergne and Gooley \(2007\)](#) for more details). The full protocol is summarised in Figure 7.3.

7.8 The new protocol in the prompt/script UI mode

7.8.1 d'Auvergne protocol script mode – the sample script

The sample script for performing this new analysis is `sample_scripts/model_free/dauvergne_protocol.py`. The full script is replicated below. The docstring at the start of the script explains the practical implementation of the full protocol. If your copy of the `dauvergne_protocol.py` script taken from the same relax version as this manual does not match the text below, please contact the relax developers via the relax-devel mailing list (see section 3.3.3 on page 31). To use this script, copy it to a dedicated directory containing your PDB file and relaxation data files. The protocol will produce many files and directories, so it is best that these are placed within a dedicated and results directory. The contents of the script are:

```
1  """Script for black-box model-free analysis.
2
3  This script is designed for those who appreciate black-boxes or those who appreciate
4  complex code. Importantly data at multiple magnetic field strengths is essential for
5  this analysis. The script will need to be heavily tailored to the molecule in
6  question by changing the variables just below this documentation. If you would like
7  to change how model-free analysis is performed, the code in the class Main can be
8  changed as needed. For a description of object-oriented coding in python using
9  classes, functions/methods, self, etc., see the python tutorial.
10
11 If you have obtained this script without the program relax, please visit http://www.nmr-relax.com.
12
13 References
14 =====
15
16 The model-free optimisation methodology herein is that of:
17
18 d'Auvergne, E. J. and Gooley, P. R. (2008b). Optimisation of NMR dynamic models II. A
19 new methodology for the dual optimisation of the model-free parameters and the
20 Brownian rotational diffusion tensor. J. Biomol. NMR, 40(2), 121-133
21
22 Other references for features of this script include model-free model selection using
23 Akaike's Information Criterion:
24
25 d'Auvergne, E. J. and Gooley, P. R. (2003). The use of model selection in the model-
26 free analysis of protein dynamics. J. Biomol. NMR, 25(1), 25-39.
27
28 The elimination of failed model-free models and Monte Carlo simulations:
29
30 d'Auvergne, E. J. and Gooley, P. R. (2006). Model-free model elimination: A new step
31 in the model-free dynamic analysis of NMR relaxation data. J. Biomol. NMR, 35(2),
32 117-135.
33
34 Significant model-free optimisation improvements:
35
36 d'Auvergne, E. J. and Gooley, P. R. (2008a). Optimisation of NMR dynamic models I.
37 Minimisation algorithms and their performance within the model-free and Brownian
38 rotational diffusion spaces. J. Biomol. NMR, 40(2), 107-109.
39
40 Rather than searching for the lowest chi-squared value, this script searches for the model
41 with the lowest AIC criterion. This complex multi-universe, multi-dimensional search
42 is formulated using set theory as the universal solution:
```

```

28
29      d'Auvergne, E. J. and Gooley, P. R. (2007). Set theory formulation of the model-free
29      problem and the diffusion seeded model-free paradigm. 3(7), 483-494.
30
31      The basic three references for the original and extended model-free theories are:
32
33      Lipari, G. and Szabo, A. (1982a). Model-free approach to the interpretation of nuclear
33      magnetic-resonance relaxation in macromolecules I. Theory and range of validity. J.
33      Am. Chem. Soc., 104(17), 4546-4559.
34
35      Lipari, G. and Szabo, A. (1982b). Model-free approach to the interpretation of nuclear
35      magnetic-resonance relaxation in macromolecules II. Analysis of experimental results.
35      J. Am. Chem. Soc., 104(17), 4559-4570.
36
37      Clore, G. M., Szabo, A., Bax, A., Kay, L. E., Driscoll, P. C., and Gronenborn, A.M.
37      (1990). Deviations from the simple 2-parameter model-free approach to the
37      interpretation of N-15 nuclear magnetic-relaxation of proteins. J. Am. Chem. Soc.,
37      112(12), 4989-4991.

38
39
40      How to use this script
41      =====
42
43      The value of the variable DIFF_MODEL will determine the behaviour of this script. The
43      five diffusion models used in this script are:
44
45      Model I (MI) - Local tm.
46      Model II (MII) - Sphere.
47      Model III (MIII) - Prolate spheroid.
48      Model IV (MIV) - Oblate spheroid.
49      Model V (MV) - Ellipsoid.

50
51      Model I must be optimised prior to any of the other diffusion models, while the Models II
51      to V can be optimised in any order. To select the various models, set the variable
51      DIFF_MODEL to the following strings:
52
53      MI - 'local_tm'
54      MII - 'sphere'
55      MIII - 'prolate'
56      MIV - 'oblate'
57      MV - 'ellipsoid'

58
59      This approach has the advantage of eliminating the need for an initial estimate of a
59      global diffusion tensor and removing all the problems associated with the initial
59      estimate.

60
61      It is important that the number of parameters in a model does not exceed the number of
61      relaxation data sets for that spin. If this is the case, the list of models in the
61      MF_MODELS and LOCAL_TM_MODELS variables will need to be trimmed.

62
63
64      Model I - Local tm
64      =====
65
66
67      This will optimise the diffusion model whereby all spin of the molecule have a local tm
67      value, i.e. there is no global diffusion tensor. This model needs to be optimised
67      prior to optimising any of the other diffusion models. Each spin is fitted to the
67      multiple model-free models separately, where the parameter tm is included in each
67      model.

68
69      AIC model selection is used to select the models for each spin.

```

```

70
71
72 Model II - Sphere
~~~~~
73
74
75 This will optimise the isotropic diffusion model. Multiple steps are required, an initial
    optimisation of the diffusion tensor, followed by a repetitive optimisation until
    convergence of the diffusion tensor. Each of these steps requires this script to be
    rerun. For the initial optimisation, which will be placed in the directory './sphere/
    init/', the following steps are used:
76
77 The model-free models and parameter values for each spin are set to those of diffusion
    model MI.
78
79 The local tm parameter is removed from the models.
80
81 The model-free parameters are fixed and a global spherical diffusion tensor is minimised.
82
83
84 For the repetitive optimisation, each minimisation is named from 'round_1' onwards. The
    initial 'round_1' optimisation will extract the diffusion tensor from the results file
    in './sphere/init/', and the results will be placed in the directory './sphere/
    round_1/'. Each successive round will take the diffusion tensor from the previous
    round. The following steps are used:
85
86 The global diffusion tensor is fixed and the multiple model-free models are fitted to each
    spin.
87
88 AIC model selection is used to select the models for each spin.
89
90 All model-free and diffusion parameters are allowed to vary and a global optimisation of
    all parameters is carried out.
91
92
93 Model III - Prolate spheroid
~~~~~
94
95
96 The methods used are identical to those of diffusion model MII, except that an axially
    symmetric diffusion tensor with  $Da \geq 0$  is used. The base directory containing all
    the results is './prolate/'.
97
98
99 Model IV - Oblate spheroid
~~~~~
100
101
102 The methods used are identical to those of diffusion model MII, except that an axially
    symmetric diffusion tensor with  $Da \leq 0$  is used. The base directory containing all
    the results is './oblade/'.
103
104
105 Model V - Ellipsoid
~~~~~
106
107
108 The methods used are identical to those of diffusion model MII, except that a fully
    anisotropic diffusion tensor is used (also known as rhombic or asymmetric diffusion).
    The base directory is './ellipsoid/'.
109
110
111
112 Final run
~~~~~
113

```

```

114
115 Once all the diffusion models have converged, the final run can be executed. This is done
116 by setting the variable DIFF_MODEL to 'final'. This consists of two steps, diffusion
117 tensor model selection, and Monte Carlo simulations. Firstly AIC model selection is
118 used to select between the diffusion tensor models. Monte Carlo simulations are then
119 run solely on this selected diffusion model. Minimisation of the model is bypassed as
120 it is assumed that the model is already fully optimised (if this is not the case the
121 final run is not yet appropriate).

122
123 The final black-box model-free results will be placed in the file 'final/results'.
124 """
125
126
127 # Python module imports.
128 from time import asctime, localtime
129
130 # relax module imports.
131 from auto_analyses.dauvergne_protocol import dAuvergne_protocol
132
133
134 # Analysis variables.
135 #####
136
137 # The diffusion model.
138 DIFF_MODEL = 'local_tm'
139
140 # The model-free models. Do not change these unless absolutely necessary, the protocol is
141 # likely to fail if these are changed.
142 MF_MODELS = ['m0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9']
143 LOCAL_TM_MODELS = ['tm0', 'tm1', 'tm2', 'tm3', 'tm4', 'tm5', 'tm6', 'tm7', 'tm8', 'tm9']
144
145
146 # The grid search size (the number of increments per dimension).
147 GRID_INC = 11
148
149
150 # The optimisation technique.
151 MIN_ALGOR = 'newton'
152
153 # The number of Monte Carlo simulations to be used for error analysis at the end of the
154 # analysis.
155 MC_NUM = 500
156
157 # Automatic looping over all rounds until convergence (must be a boolean value of True or
158 # False).
159 CONV_LOOP = True
160
161
162 # Set up the data pipe.
163 #####
164
165 # The following sequence of user function calls can be changed as needed.
166
167 # Create the data pipe.
168 pipe_bundle = "mf (%s)" % asctime(localtime())
169 name = "origin - " + pipe_bundle
170 pipe.create(name, 'mf', bundle=pipe_bundle)
171
172 # Load the PDB file.
173 structure.read_pdb('1f3y.pdb', set_mol_name='Ap4Aase', read_model=3)
174
175 # Set up the 15N and 1H spins (both backbone and Trp indole sidechains).
176 structure.load_spins('@N', ave_pos=True)

```

```

166 structure.load_spins('@NE1', ave_pos=True)
167 structure.load_spins('@H', ave_pos=True)
168 structure.load_spins('@HE1', ave_pos=True)
169 spin.isotope('15N', spin_id='@N*')
170 spin.isotope('1H', spin_id='@H*')

171 # Set up the 15N spins (alternative to the structure-based approach).
172 #sequence.read(file='noe.500.out', dir=None, mol_name_col=1, res_num_col=2, res_name_col
173     =3, spin_num_col=4, spin_name_col=5)
174 #spin.element(element='N', spin_id='@N*')
175 #spin.isotope('15N', spin_id='@N*')

176 # Generate the 1H spins for the magnetic dipole-dipole relaxation interaction (alternative
177     to the structure-based approach).
178 #sequence.attach_protons()

179 # Load the relaxation data.
180 relax_data.read(ri_id='R1_600', ri_type='R1', frq=599.719*1e6, file='r1.600.out',
181     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
182     data_col=6, error_col=7)
182 relax_data.read(ri_id='R2_600', ri_type='R2', frq=599.719*1e6, file='r2.600.out',
183     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
183     data_col=6, error_col=7)
183 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=599.719*1e6, file='noe.600.out',
184     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
184     data_col=6, error_col=7)
184 relax_data.read(ri_id='R1_500', ri_type='R1', frq=500.208*1e6, file='r1.500.out',
185     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
185     data_col=6, error_col=7)
185 relax_data.read(ri_id='R2_500', ri_type='R2', frq=500.208*1e6, file='r2.500.out',
186     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
186     data_col=6, error_col=7)
186 relax_data.read(ri_id='NOE_500', ri_type='NOE', frq=500.208*1e6, file='noe.500.out',
187     mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
187     data_col=6, error_col=7)

187 # Deselect spins to be excluded (including unresolved and specifically excluded spins).
188 deselect.read(file='unresolved', dir=None, spin_id_col=None, mol_name_col=1, res_num_col
189     =2, res_name_col=3, spin_num_col=4, spin_name_col=5, sep=None, spin_id=None, boolean='
189     AND', change_all=False)
190 deselect.read(file='exclude', spin_id_col=1)

191 # Define the magnetic dipole-dipole relaxation interaction.
192 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
193 interatom.define(spin_id1='@NE1', spin_id2='@HE1', direct_bond=True)
194 interatom.set_dist(spin_id1='@N*', spin_id2='@H*', ave_dist=1.02 * 1e-10)
195 interatom.unit_vectors()

196 # Define the chemical shift relaxation interaction.
197 value.set(-172 * 1e-6, 'csa', spin_id='@N*')

198 # Execution.
199 ######
200
201
202
203 # Do not change!
204 #####
205
206
207 dAuvergne_protocol(pipe_name=name, pipe_bundle=pipe_bundle, diff_model=DIFF_MODEL,
207     mf_models=MF_MODELS, local_tm_models=LOCAL_TM_MODELS, grid_inc=GRID_INC, min_algor=
207     MIN_ALGOR, mc_sim_num=MC_NUM, conv_loop=CONV_LOOP)

```

7.8.2 d'Auvergne protocol script mode – analysis variables

At the start of the script you will notice a number of **Analysis variables**. Unless you know what you are doing, you should only change the DIFF_MODEL variable to the following:

'local_tm': This is the first diffusion model which must be optimised prior to optimising any of the other diffusion models. It consists of the local τ_m models (equations 7.23.0 to 7.23.9 on page 90).

'sphere': This second diffusion model is that of isotropic Brownian diffusion.

'prolate': This third diffusion model is that of the prolate axially-symmetric rotor.

'oblate': This fourth diffusion model is that of the oblate axially-symmetric rotor.

'ellipsoid': This fifth diffusion model is that of fully rhombic Brownian diffusion (see [Perrin \(1934, 1936\)](#) for the original theory).

'final': This is a special value which will finalise the analysis by selecting the best diffusion model to describe your system and to perform Monte Carlo simulations for error propagation.

The MF_MODELS and LOCAL_TM_MODELS variables specify which model-free models will be used in the analysis. But, as the full protocol behind this script which is designed to find the solution of the universal set \mathfrak{U} (see section 7.7.2 on page 103) expects that all these models are present, you should not change these variables. If you do remove some model-free models, you should fully expect to see artificial motions which you will not be able to distinguish from the real molecular motions.

The next variables GRID_INC and MIN_ALGOR are related to the optimisation of the model-free models. These should also not be touched unless you fully understand the consequences (and have read [d'Auvergne and Gooley \(2008b\)](#)). The variable MC_NUM specifies the number of Monte Carlo simulations. This number can be increased but, for realistic parameter errors in your publication, it should not set lower than 500 simulations.

Finally the CONV_LOOP variable is designed to make your life easier. If left at the value of **True**, the script will iterate until convergence (see Figure 2 in [d'Auvergne and Gooley \(2008c\)](#) to understand this concept). If changed to **False**, then you will need to run the script manually for the 15 or so iterations of each diffusion model, and then repeat this for all diffusion models II to V.

7.8.3 d'Auvergne protocol script mode – data pipe initialisation

The next part of the script between the **Analysis variables** and execution sets up a data pipe with all of the spin information and relaxation data to pass into the automated protocol. The data pipe is created in the lines:

```

156 # Create the data pipe.
157 pipe_bundle = "mf (%s)" % asctime(localtime())
158 name = "origin - " + pipe_bundle
159 pipe.create(name, 'mf', bundle=pipe_bundle)

```

Firstly a data pipe bundle name is created containing the date and time at the point the script is first executed. This pipe bundle is used to group together all of the data pipes created automatically by the protocol. See section 4.2.1 on page 36 for more details.

The data pipe name used for this initial setup is set to `origin - mf (x)` where `x` is the data and time again. This name is unique and will not clash with the data pipes created within the protocol. The `pipe.create` command will create the data pipe and add it to a new pipe bundle.

7.8.4 d'Auvergne protocol script mode – setting up the spin systems

To see how to set up the spin system data in all possible situations, please see Chapter 4 for a thorough description. Here two different methods are presented. The first is by extracting the spins from a PDB file which is first loaded with:

```
161 # Load the PDB file.
162 structure.read_pdb('1f3y.pdb', set_mol_name='Ap4Aase', read_model=3)
```

This will read the 3rd model from the 1F3Y PDB file and name the single molecule as ‘Ap4Aase’. The ¹⁵N and ¹H spins for the backbone and tryptophan indole sidechain are extracted from the structure with the user functions:

```
164 # Set up the 15N and 1H spins (both backbone and Trp indole sidechains).
165 structure.load_spins('@N', ave_pos=True)
166 structure.load_spins('@NE1', ave_pos=True)
167 structure.load_spins('@H', ave_pos=True)
168 structure.load_spins('@HE1', ave_pos=True)
```

As the PDB file does not contain isotope information, this is set with the user functions:

```
169 spin.isotope('15N', spin_id='@N*')
170 spin.isotope('1H', spin_id='@H*')
```

The spin ID ‘@N*’ uses regular expression and will match both the ‘N’ and ‘NE1’ spins.

The alternative approach is if a structure is missing. This is the commented out code:

```
172 # Set up the 15N spins (alternative to the structure-based approach).
173 sequence.read(file='noe.500.out', dir=None, mol_name_col=1, res_num_col=2, res_name_col=3,
    spin_num_col=4, spin_name_col=5)
174 spin.element(element='N', spin_id='@N*')
175 spin.isotope('15N', spin_id='@N*')
176
177 # Generate the 1H spins for the magnetic dipole-dipole relaxation interaction (alternative
    to the structure-based approach).
178 sequence.attach_protons()
```

To use this, you will need to place comments (the # character) in front of the previous `structure.read_pdb`, `structure.load_spins` and `spin.isotope` user functions. Then uncomment the `sequence.read`, `spin.element`, `spin.isotope` and `sequence.attach_protons` user functions. The ¹⁵N spins will be extracted from the `noe.500.out` file. The `spin.element` and `spin.isotope` user functions set the information required for relax to understand which relaxation mechanisms are active. Finally the `sequence.attach_protons` user function will automatically attach protons to all nitrogen spin systems. As

this method is devoid of atomic positional information, the N-H bonds are absent and the diffusion models requiring structural information (the spheroids and ellipsoid) must be skipped.

7.8.5 d'Auvergne protocol script mode – loading the data

The next step is to load the relaxation data for each spin system. The sample script assumes that the NOE, R₁ and R₂ data was generated using relax. One of the six user function calls is:

```
180 # Load the relaxation data.
181 relax_data.read(ri_id='R1_600', ri_type='R1', frq=599.719*1e6, file='r1.600.out',
    mol_name_col=1, res_num_col=2, res_name_col=3, spin_num_col=4, spin_name_col=5,
    data_col=6, error_col=7)
```

This pattern is repeated for all of the relaxation data files loaded. The important points are that each relaxation data set must have its own unique identification string (`ri_id`), the relaxation data type specified (`ri_type`) and the frequency in Hertz (not MHz) specified. Note that the frequency must be the exact value – see the `sfrq` parameter in the Varian `procpar` file or the `SF01` parameter in the Bruker `acqus` file.

7.8.6 d'Auvergne protocol script mode – deselection

The sample script now presents the deselection of spins using two different files:

```
188 # Deselect spins to be excluded (including unresolved and specifically excluded spins).
189 deselect.read(file='unresolved', dir=None, spin_id_col=None, mol_name_col=1, res_num_col
    =2, res_name_col=3, spin_num_col=4, spin_name_col=5, sep=None, spin_id=None, boolean='
        AND', change_all=False)
190 deselect.read(file='exclude', spin_id_col=1)
```

The `unresolved` file contains a list of spins which are unresolved in all spectra. If relax has been used for calculating the NOE and fitting the relaxation curves, then this step is not needed as the relaxation data files will not have any data for the spins deselected in those analyses. The second file `exclude` is a list of spin ID strings (see section 4.2.2 on page 38) of spins that for whatever reason are to be excluded from the analysis.

7.8.7 d'Auvergne protocol script mode – relaxation interactions

The next step is to fully specify all of the relaxation interactions active on the spins of interest. Firstly the magnetic dipole-dipole interaction is defined between directly bonded nitrogens and protons:

```
192 # Define the magnetic dipole-dipole relaxation interaction.
193 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
194 interatom.define(spin_id1='@NE1', spin_id2='@HE1', direct_bond=True)
195 interatom.set_dist(spin_id1='@N*', spin_id2='@H*', ave_dist=1.02 * 1e-10)
196 interatom.unit_vectors()
```

The regular expression ‘@N*’ and ‘@H*’ cannot be used with the `dipole_pair`. define user function as otherwise @N spins will be connected to @HE1 spins of the same tryptophan residue and @H spins to @NE1 spins. The average interatomic distance is set to 1.02 Ångstrom (though the `dipole_pair.set_dist` user function expects the units of meters). The `dipole_pair.unit_vectors` is used to calculate the averaged unit vector between the two atoms.

Secondly the chemical shift anisotropy (CSA) relaxation mechanism is defined via the single command:

```
198 # Define the chemical shift relaxation interaction.
199 value.set(-172 * 1e-6, 'csa', spin_id='@N*')
```

If your system does not experience CSA relaxation, the value can be set to zero.

7.8.8 d'Auvergne protocol script mode – execution

Once the data is set up and you have modified your script to match your analysis needs, then the data pipe, pipe bundle and analysis variables are passed into the `dAuvergne_protocol` class. This is the final line of the script:

```
203 # Execution.
204 #####
205
206 # Do not change!
207 dAuvergne_protocol(pipe_name=name, pipe_bundle=pipe_bundle, diff_model=DIFF_MODEL,
    mf_models=MF_MODELS, local_tm_models=LOCAL_TM_MODELS, grid_inc=GRID_INC, min_algor=
    MIN_ALGOR, mc_sim_num=MC_NUM, conv_loop=CONV_LOOP)
```

This script needs to be executed multiple times, once for each of the diffusion models. For example if the `DIFF_MODEL` variable is set to ‘ellipsoid’, you can run relax with:

```
$ relax --tee log.ellipsoid dauvergne_protocol.py
```

You should use a different log file for each diffusion model, though relax will prevent you from overwriting an old log file. Note that the `log.*` files for each diffusion model may end up being a few gigabytes in size.

For a full analysis of a protein system, the analysis may require between one to two weeks to complete. This can be speed up using Gary Thompson's multi-processor code (see section 1.3 on page 17). The analysis is performed as described in the previous sections and summarised in Figure 7.3. If you are curious, the implementation is within a very large relax script called `auto_analyses/dauvergne_protocol.py` (which must never be changed). This automatic analysis script hides all of the complexity of the full protocol from the sample script.

7.9 The new protocol in the GUI

A model-free analysis can be performed within the GUI (see Figure 1.8 on page 18). This analysis is that of the fully automated d’Auvergne protocol which can be chosen via the analysis selection wizard (Figure 1.4 on page 12). Please see Section 7.7 on page 100 for a description of this new model-free protocol. As mentioned previously, please note that this protocol requires multiple field relaxation data.

The GUI is designed to be robust – you should be able to set up all the input data and parameters in any order with relax returning you warnings if something is missing. The analysis will only execute once everything is correctly set up. If this is not the case, clicking on the “Execute relax” button will display a warning window explaining what the issue is rather than initialising the analysis. Despite the self-explanatory nature of the GUI a tutorial on how to use the GUI, with screenshots, will be presented below.

If the “Protocol mode” field is left to the “Fully automated” setting then, after clicking on “Execute relax”, the calculation can be left to complete. It is highly recommended to check the log messages in the relax controller window, at least at the start of the analysis, to make sure that all the data is being read correctly and everything is set up as desired. All warnings should be carefully checked as these can indicate a fatal problem. If you would like to log all the messages into a file, relax can be run with:

```
$ relax -g --log log
```

Note that the size of this `log` file could end up being in the gigabyte range for a model-free analysis.

For the full analysis to complete, for a protein system this may take about a week. Depending on the nature of the problem and the speed of the computer, the calculation time may be significantly shorter or longer. To speed up the calculations, if you have access to multiple cores and/or hyper-threading, the GUI can be run using Gary Thompson’s multi-processor framework (see section 1.3 on page 17). For example on a dual-core, dual-CPU system, four calculations can be run simultaneously. In this case, the GUI can be launched with:

```
$ mpirun -np 5 /usr/local/bin/relax --multi='mpi4py' --gui --log log
```

This assumes that OpenMPI and the Python mpi4py module have been installed on your system, and relax is installed into the `/usr/local/bin/` directory. If this is successful, you should only see a single relax GUI window (and not five windows) and in the relax controller, you should see text similar to:

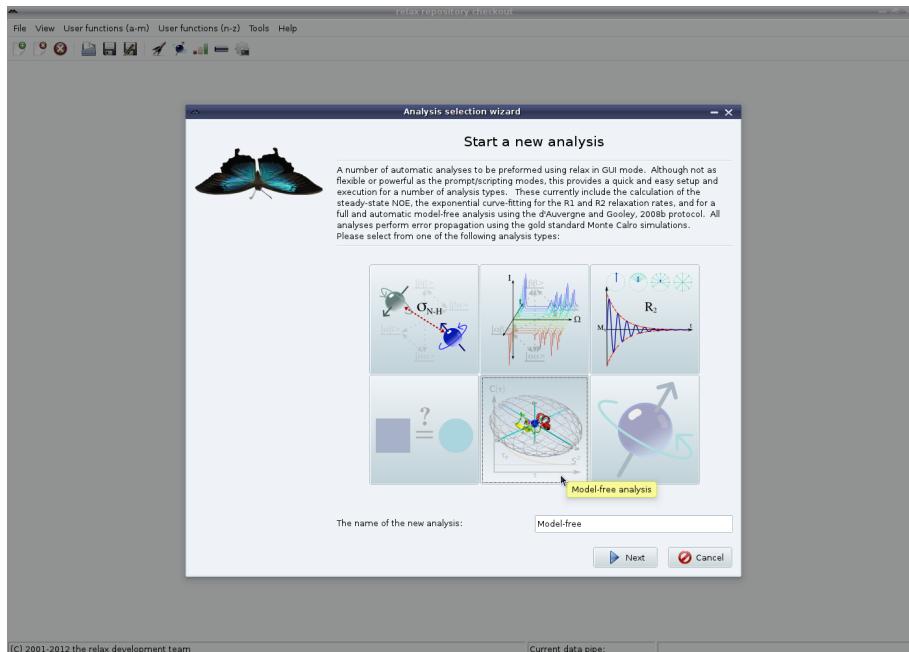
```
Processor fabric: MPI 2.1 running via mpi4py with 4 slave processors & 1 master. Using
Open MPI 1.4.3.
```

If you are using a different MPI implementation, please see the documentation of that implementation to see how to launch a program in MPI mode. Finally as the calculation takes so long, we will run the calculations at a lower priority so that the computer is not slowed down too much and remains responsive. Therefore this model-free GUI analysis tutorial will be launched with the full command:

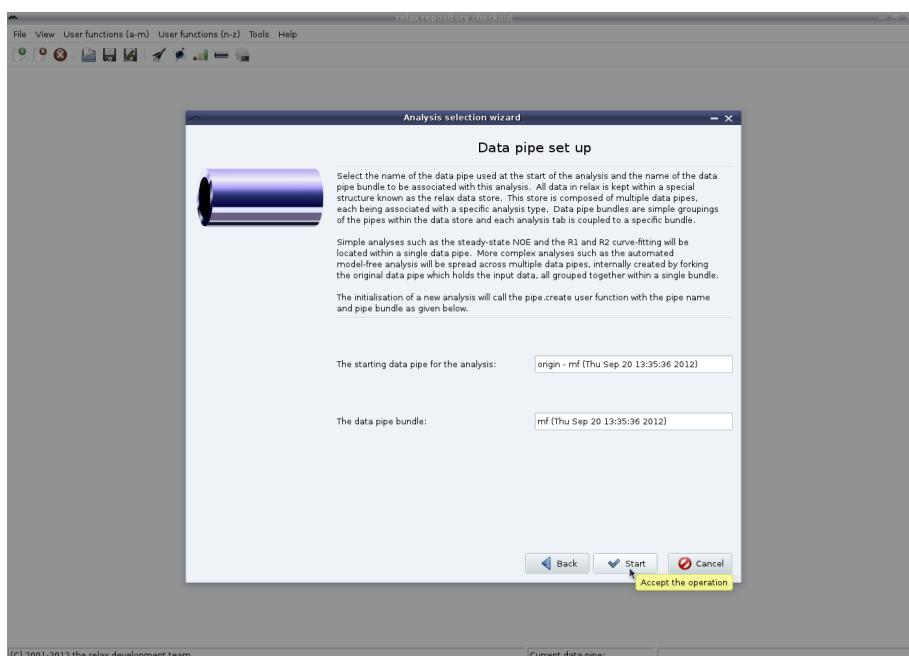
```
$ nice -n 15 mpirun -np 5 /usr/local/bin/relax --multi='mpi4py' --gui --log log
```

7.9.1 d'Auvergne protocol GUI mode – data pipe initialisation

First launch the analysis selection wizard (see Figure 1.4 on page 12) and click on the model-free analysis button.

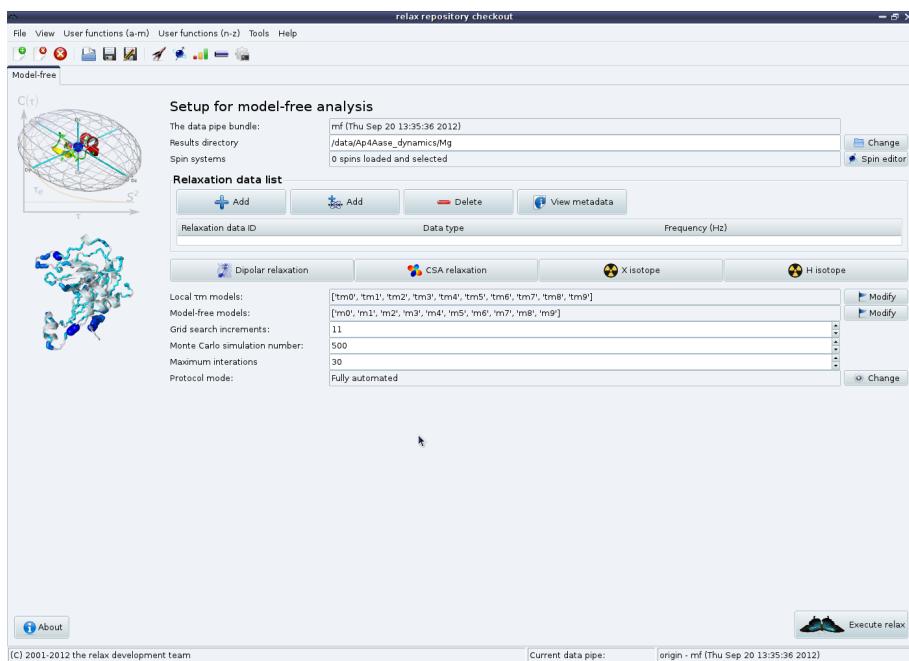


Click on the “Next” button and on the second page click on the “Start” button. The text in the second page need not be changed.

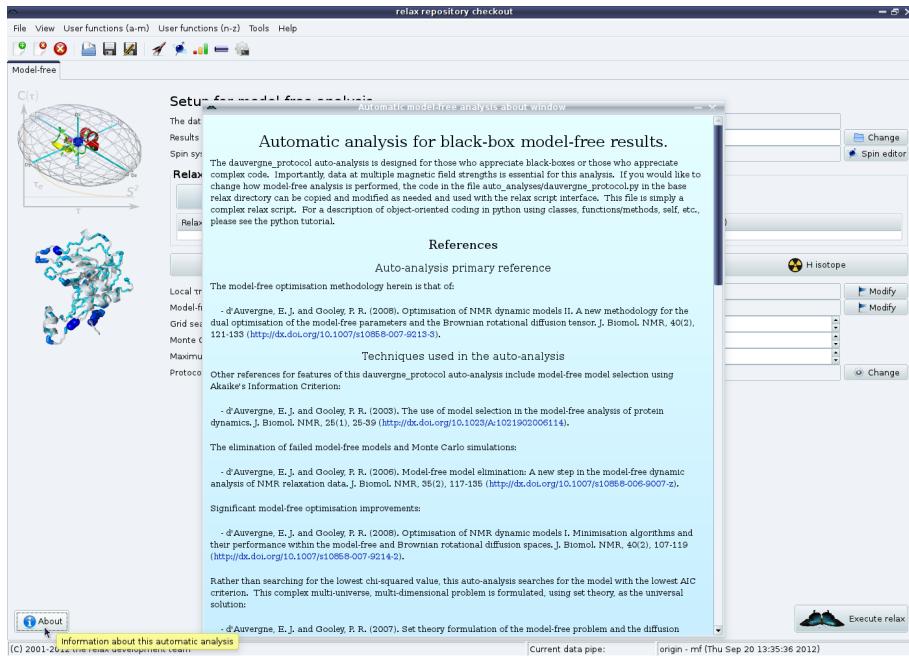


7.9.2 d'Auvergne protocol GUI mode – general setup

Once the analysis is initialised, the screen should look like:



The “About” button in the bottom left will bring up a window with the same description as given in the sample script:



At this point, back in the main relax window, the results directory where all of the output files and directories will be saved can be changed.

7.9.3 d'Auvergne protocol GUI mode – setting up the spin systems

The model-free dynamics is at the level of the spins – relaxation affects individual nuclei. In the main model-free tab you will see the “Spin systems” GUI element. Clicking on the “Spin editor” button to the right of this element will launch the spin editor window.

In this tutorial, the 3rd model of the PDB file 1f3y.pdb will be used to extract the spin system information. The molecule will be named “Ap4Aase”. For details on how to create the spin containers necessary for this analysis, please see section 4.5.2 on page 42 (or analyses lacking structural data in section 4.5.3 on page 45 for sequence files).

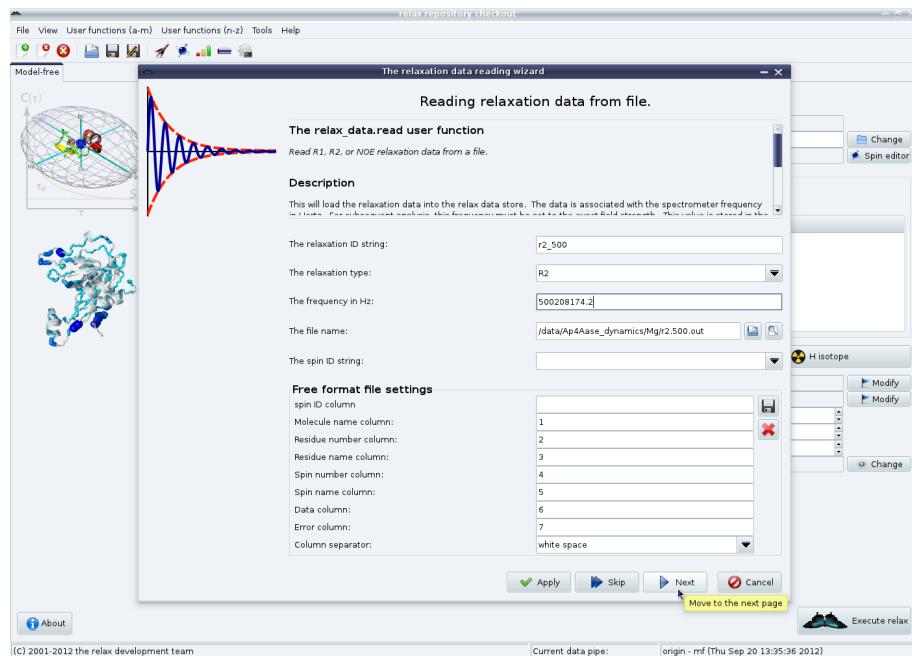
Note that for this tutorial, the protein backbone spins “@N” and “@H” as well as the tryptophan sidechain indole “@NE1” and “@HE1” spins should be loaded in the spin viewer window.

7.9.4 d’Auvergne protocol GUI mode – unresolved spins

To deselect all unwanted spins, please read section 4.5.5 on page 46 for all the necessary instructions for how to do this in the GUI.

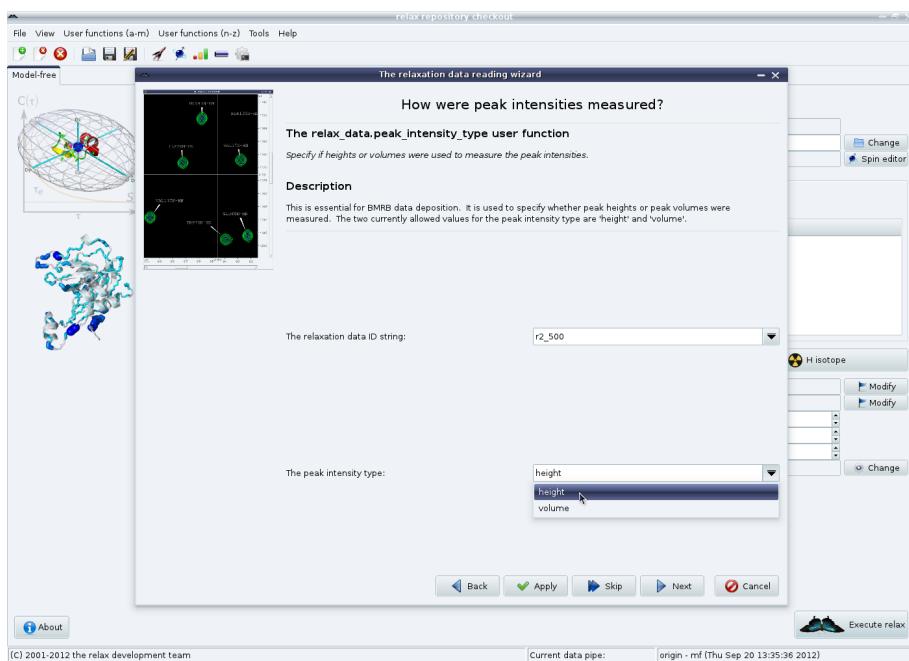
7.9.5 d’Auvergne protocol GUI mode – loading the data

The relaxation data can either come from plain columnar formatted text files (such as if relax was used for the NOE, R₁ and R₂ analyses) or from the Bruker Dynamics Centre. For the former, click on the “Add” button in the “Relaxation data list” GUI element. This route will be used for this tutorial. For the later, click on the “Add Bruker” button. After clicking on “Add”, you will see the relaxation data loading wizard:

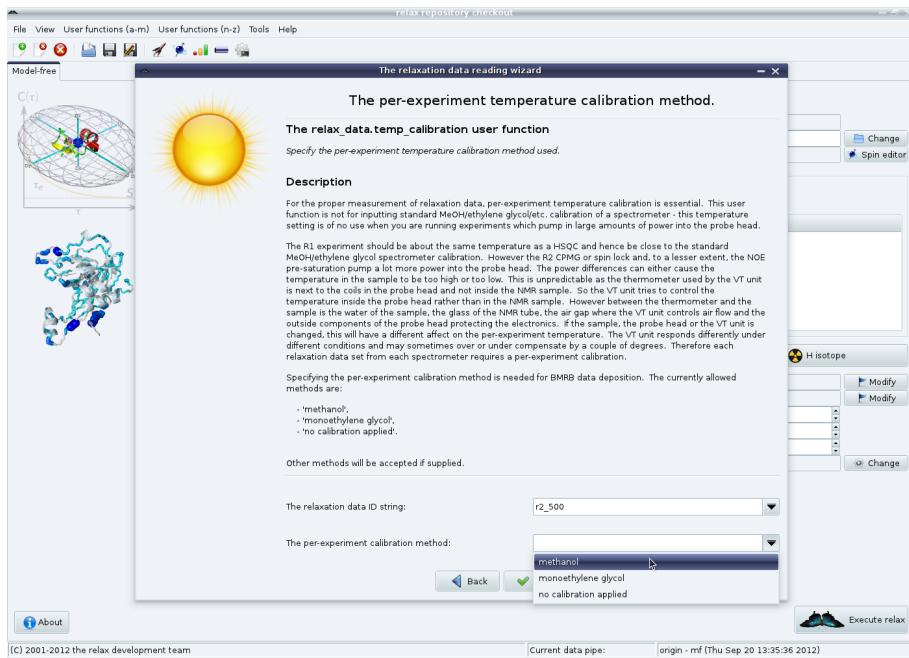


In this first page, the unique relaxation data identification string (“r2_500”), the relaxation data type (“R2”), the frequency in Hertz (“500208174.2”) and the file (“r2.500.out”) are specified. If your data comes from another program, you many need to change the values in the “Free format file settings” element. Click on “Next” to load the data from the file.

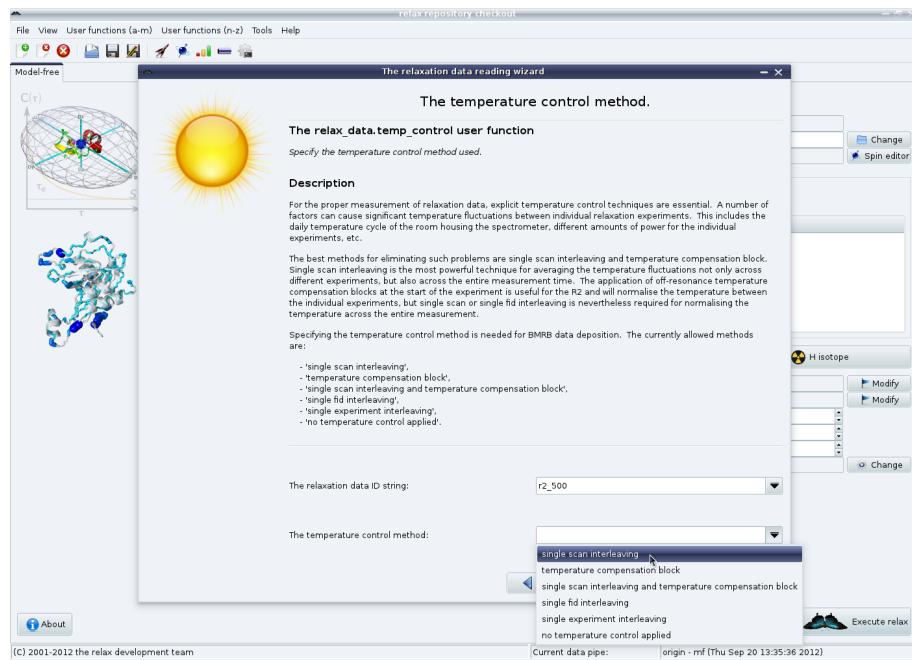
The next wizard pages are for loading the metadata which is used in the BioMagResBank deposition of your final results. The first is how the peak intensities were measured, either peak heights or volumes. Select the appropriate value, then click on “Next”.



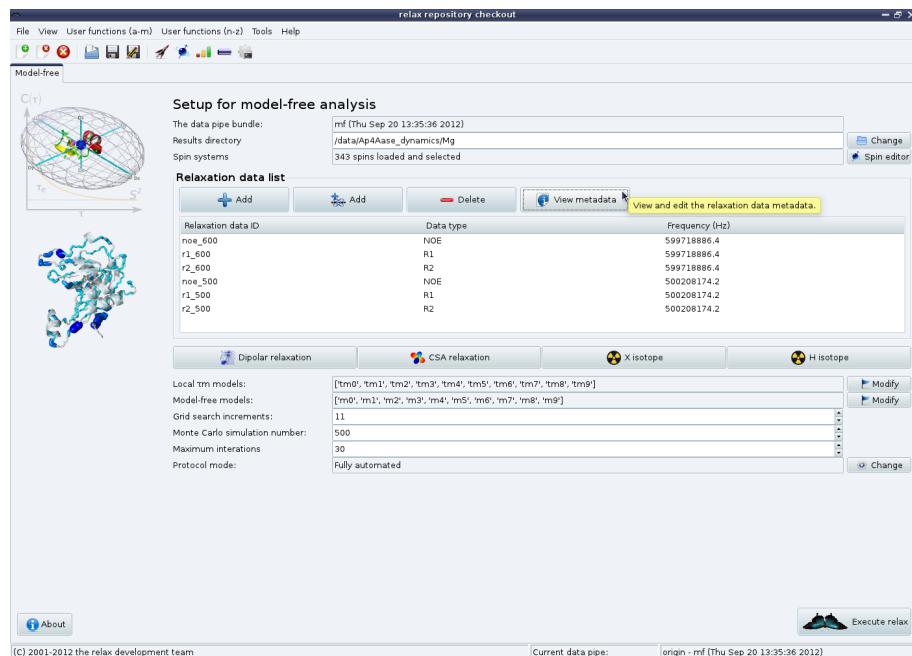
Then the temperature control method is given. For more details, please read the documentation provided in the wizard and see section 5.3.1 on page 52. Click on “Next” to continue.



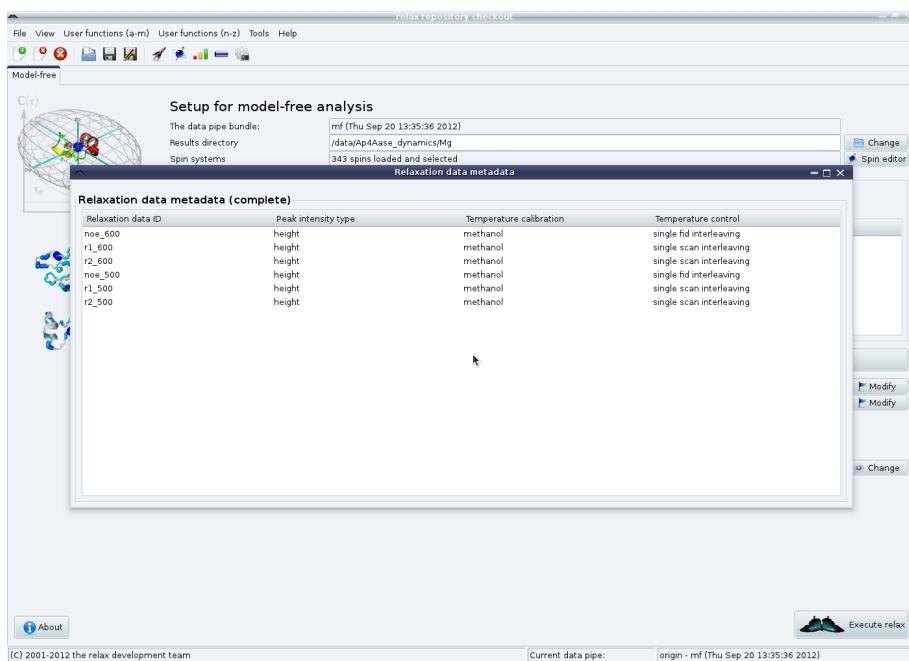
The temperature calibration method can finally be specified. Again, see section 5.3.1 on page 52 for the full details. Click on the “Finish” button to close the wizard.



After you have repeated this for the NOE, R₁ and R₂ at both 500 and 600 MHz, you should now see:

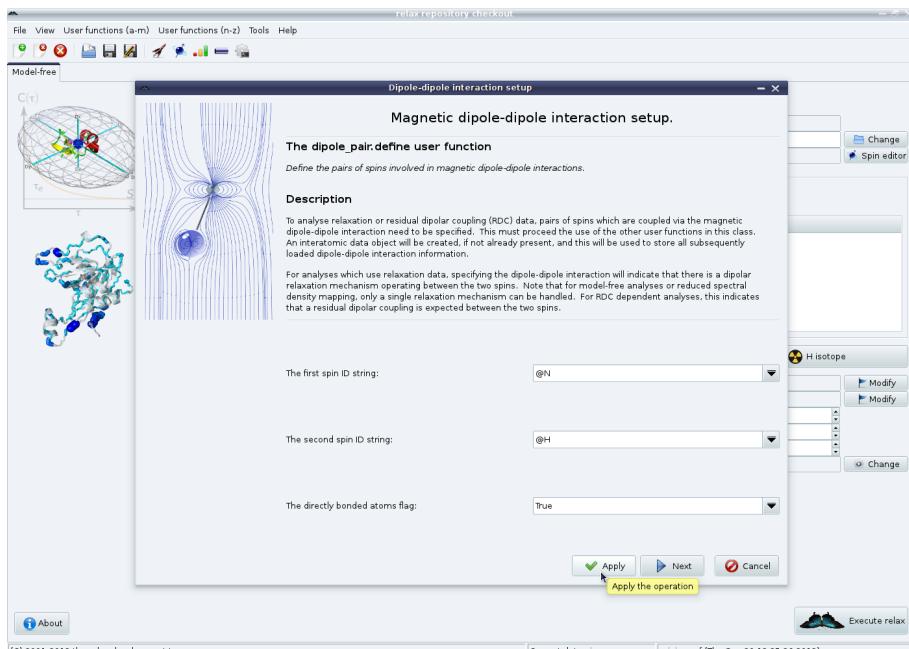


Check that the metadata has been properly entered by clicking on the “View metadata” button in the “Relaxation data list” GUI element:



7.9.6 d'Auvergne protocol GUI mode – relaxation interactions

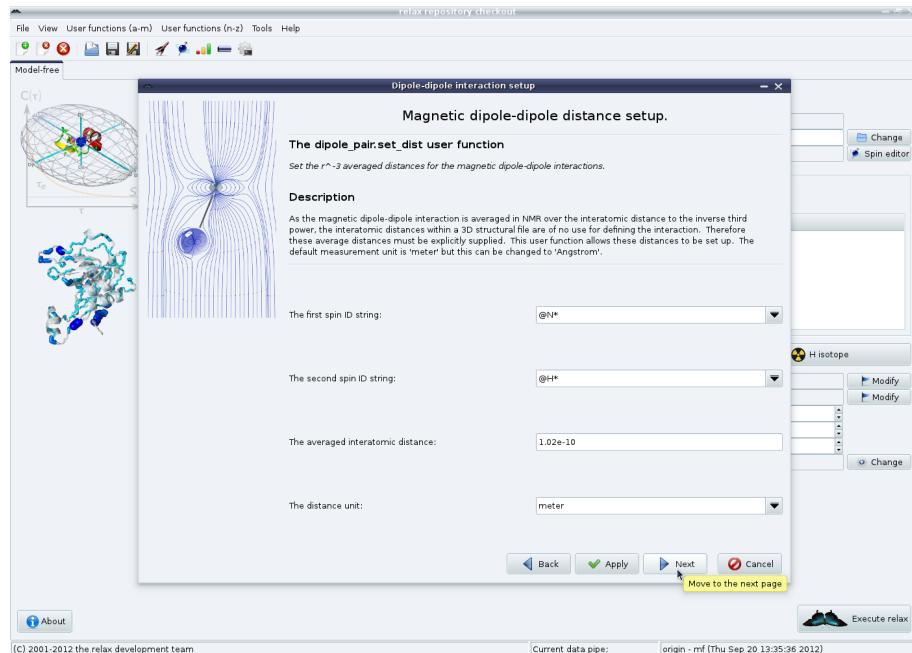
Just as in the scripting mode, the relaxation interactions need to now be defined. The first is the magnetic dipole-dipole interaction. All coupled nitrogen and proton spins should already be loaded at this point. Click on the “Dipolar relaxation” button in the model-free tab in the main relax window to launch the magnetic dipole-dipole interaction wizard:



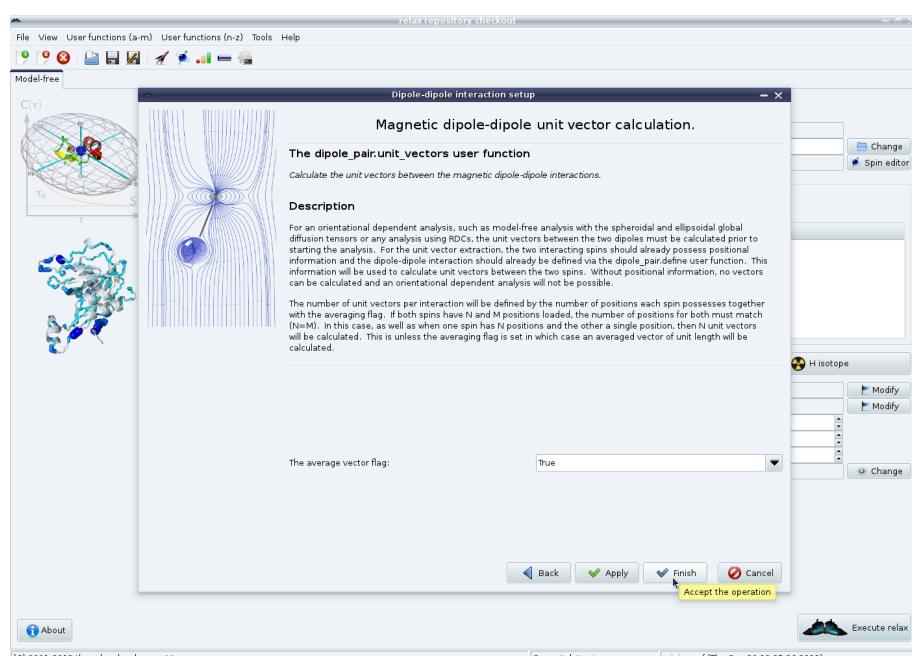
For this example, directly bonded nitrogens and protons will be analysed. To start with, the backbone NH pairs will be defined. Leave the values at “@N” and “@H” and click on the “Apply” button. Then change the two spin ID strings to “@NE1” and “@HE1” to set up the tryptophan sidechain indole NH pairs and click on the “Next” button. Note that

the regular expression “@N*” and “@H*” should not be used in this first wizard page as otherwise @N spins will be connected to @HE1 spins of the same tryptophan residue and @H spins to @NE1 spins.

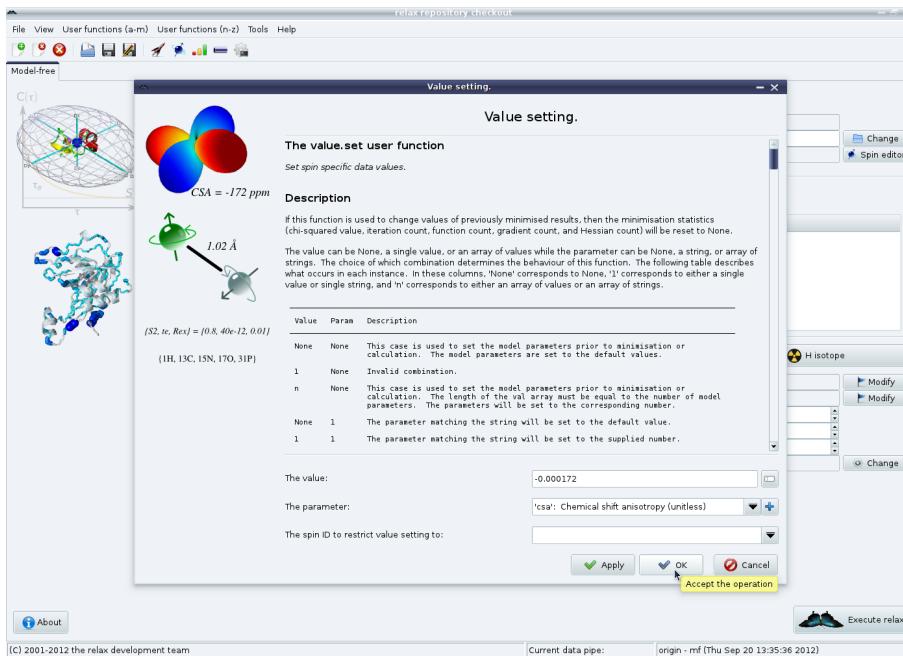
Now the $\langle r^{-3} \rangle$ averaged distance of 1.02 Å will be set. Leave all settings as they are and click on “Next”:



If multiple models have been loaded in the previous steps, then the unit vectors between each model need to be calculated. For a model-free analysis multiple unit vectors must be averaged to a single vector – current model-free theory is based on the assumption of a single vector orientation. Therefore the averaged vector flag must be left on “True”. Click on “Finish” to terminate the set up of the magnetic dipole-dipole interactions:

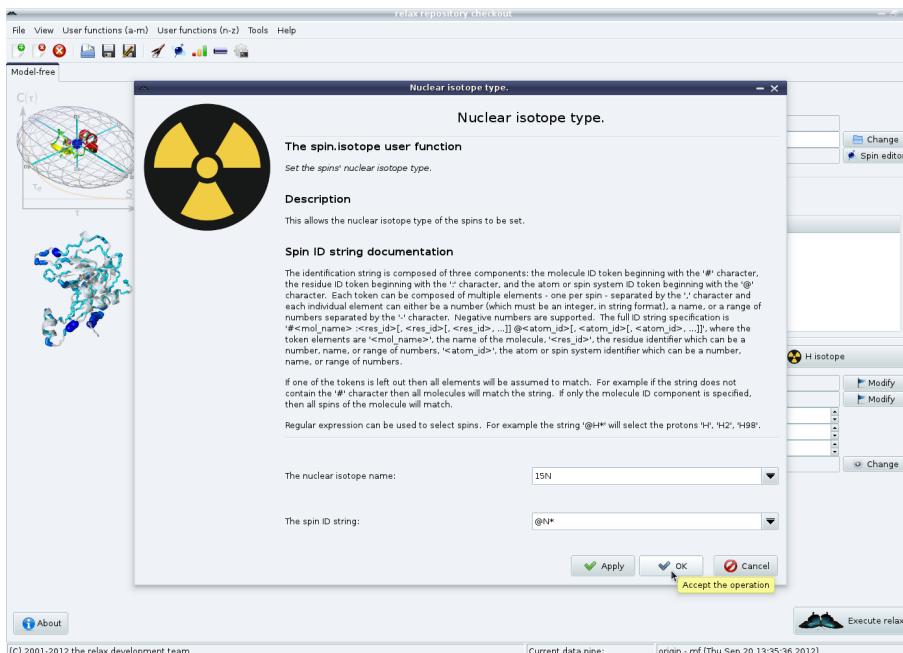


Secondly the chemical shift anisotropy (CSA) relaxation mechanism needs to be defined. Click on the “CSA relaxation” button in the model-free tab in the main relax window. An averaged CSA value of -172 ppm will be used for all spins, so simply click on “Ok” to finish.



7.9.7 d'Auvergne protocol GUI mode – spin isotopes

As the PDB file contains no isotope information, this needs to now be specified. First click on the “X isotope” button to set the nuclear isotope type of the heteronuclei:



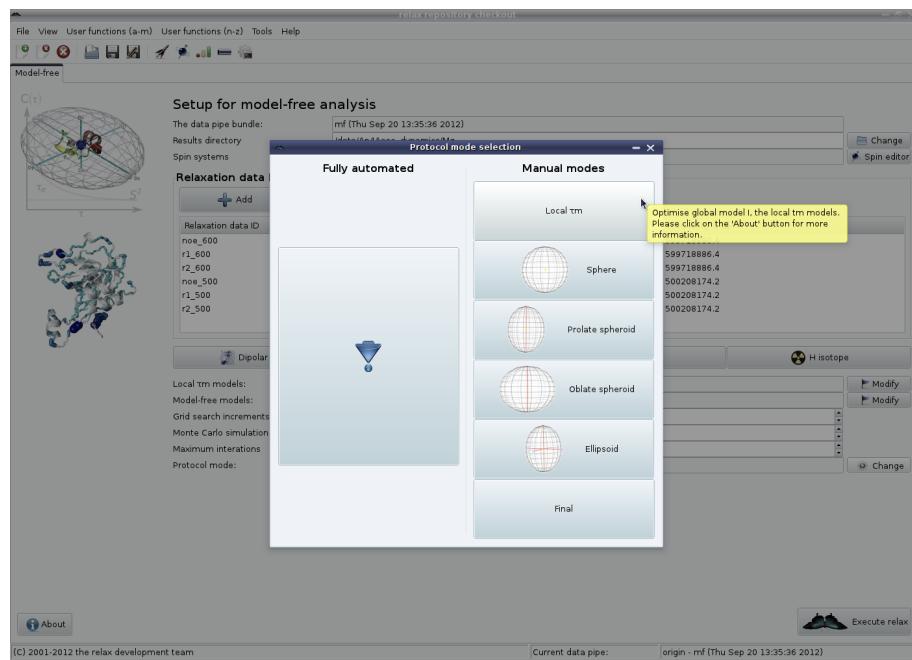
As nitrogen relaxation is being studied, the nuclear isotope name can be left as “15N” and the spin ID string to “@N*”. Therefore simply click on the “Ok” button. Exactly the same

procedure can be used for the proton with the “H isotope” button.

7.9.8 d’Auvergne protocol GUI mode – the rest of the setup

The local τ_m models and model-free models should not be modified, the reason for this is explained in section 7.8.2 on page 112. The grid search increments defaults to “11”. This is used in the optimisation of the individual model-free models for each spin. This value should also not be touched unless you know what you are doing (and have read d’Auvergne and Gooley (2008b)). The number of Monte Carlo simulations can be increased but, for accurate error estimates, it should not be less than 500 simulations. One additional setting is the “Maximum iterations”. This is a maximum number of times the protocol will iterate before terminating. This allows infinite loops to be broken. The value of 30 iterations should be fine for most analyses.

The “Protocol mode” GUI element setting of “Fully automated” will not be changed for the analysis of this tutorial. However if you are studying a system without a 3D structure, you can execute each individual component of the analysis by clicking on the “Change” button. This will make the protocol mode selection window appear:



From this you can first select the “Local τ_m ” model, then the “Sphere” and finally the “Final” mode, clicking on “Execute relax” between each selection.

7.9.9 d’Auvergne protocol GUI mode – execution

Prior to executing relax, you should very carefully check the relax controller window for any strange messages, warnings or errors. You can open this window in three ways:

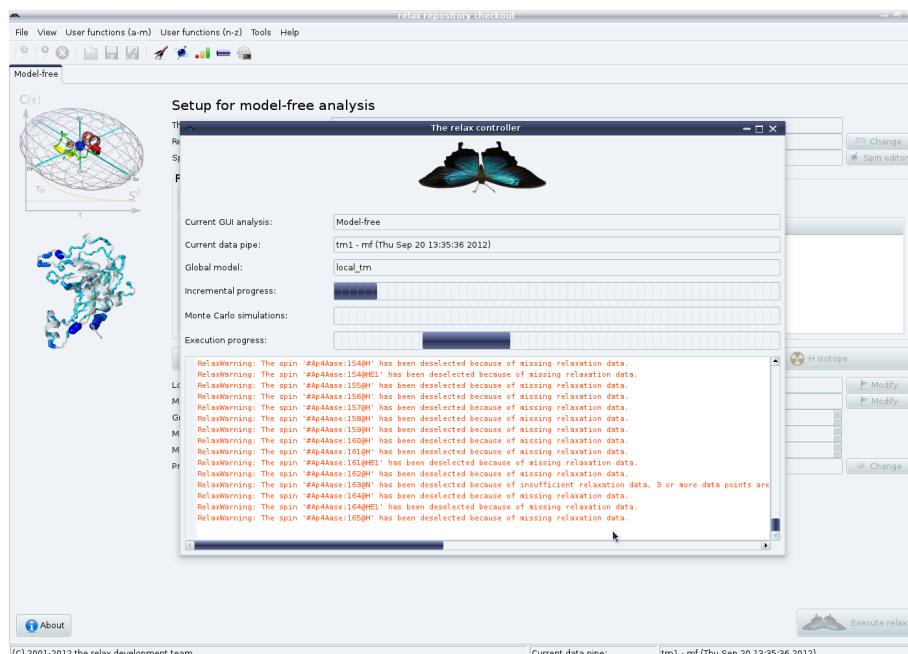
- Selecting the “View→Controller” menu item.
- Typing “[Ctrl+Z]” within the main relax window.

- Clicking on the “relax controller” button on the toolbar.

These messages are very important and will indicate to you if there are any problems prior to starting the very long model-free calculation. This information should be stored in the `log` file as well. As the execution of a fully iterative and complete model-free protocol takes a very long time to finish, it is advisable to save the current relax state. This will allow you restart the calculation without performing all of the steps detailed above. Just in case you cannot work out how to do this yourself, here is a list of the different ways you can do this (if this is not enough for you, please email the relax-users mailing list with your suggestions):

- Selecting the “File→Save relax state” menu item.
- Typing “[**Ctrl+S**]” within the main relax window.
- Clicking on the “Save relax state” button on the toolbar.
- Selecting the “File→Save as...” menu item.
- Typing “[**Shift+Ctrl+S**]” within the main relax window.
- Clicking on the “Save as” button on the toolbar.
- Selecting the “User functions→state→save” menu item.
- Opening up the relax prompt window with “View→relax prompt” or “[**Ctrl+P**]” and using the `state.save` user function.

If all the messages in the relax controller or `log` file appear to be fine and you have saved the current relax state, then click on “Execute relax”. This will start the calculations, freeze most of the GUI and open up the relax controller to give you feedback on the progress of the calculations:



At the start of the protocol, you should again check the messages carefully to be sure that relax is operating as you would expect. There may be very important `RelaxWarnings` that will require you to quit relax and start the analysis all over again.

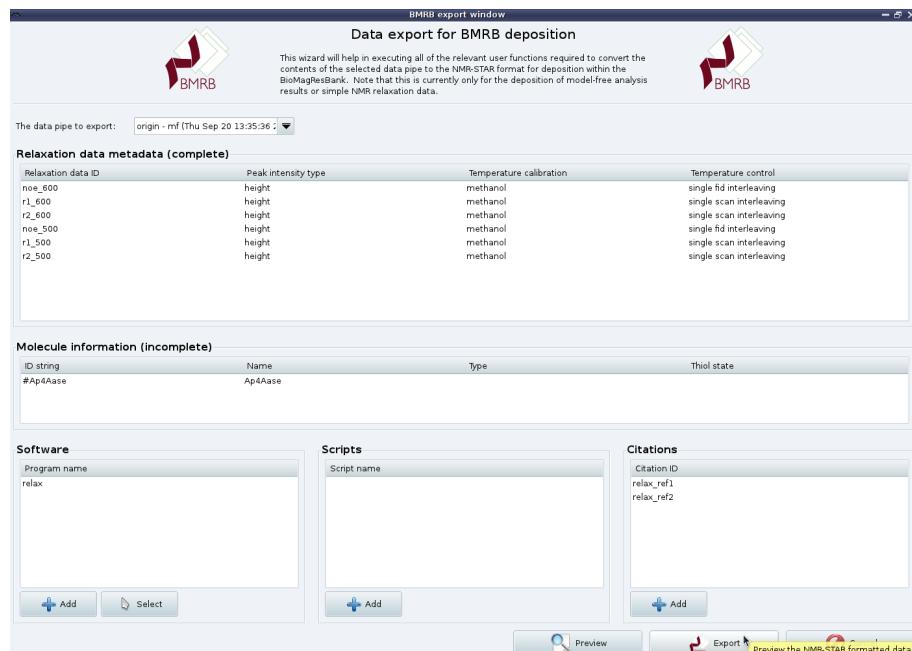
7.9.10 d'Auvergne protocol GUI mode – completion

Upon completion of the analysis, the save and results files for the final result will be located in the `final` directory within the selected results directory. The results files will consist of text files for each of the spin specific model-free parameters, 2D Grace plots of the model-free parameters, PyMOL and MOLMOL macros for superimposing the model-free parameter values onto the 3D structure of the molecule, and a PDB representation of the final diffusion tensor.

Further visualisations of the results are possible via the “User functions” menu entry. For example to generate a 2D plot of order parameters for one of the other diffusion tensor results, the pipe editor window can be used to switch data pipes to the other diffusion models and then the “User functions→grace→write” menu item can be selected to create the plot.

7.9.11 d'Auvergne protocol GUI mode – BMRB deposition

Once you are ready to publish your results, the very last step of the model-free analysis is to create a NMR-STAR formatted file for [BioMagResBank](#) submission for each model-free analysis you perform. This can be accomplished using the BMRB export window. Simply select the “File→Export for BMRB deposition” menu item. You will then see the BMRB export window:



From here you can complete the relaxation data metadata if needed, set up all the molecule information needed for a BMRB deposition, specify the software you have used running up

to the model-free analysis and any spectral processing or relax scripts you have used. You can also add as many citations relevant to your analysis as you wish. The NMR-STAR formatted file can be previewed in the relax controller window via the “Preview” button and the final file created using the “Export” button.

Once you are in the stage of writing up, simply go to the ADIT-NMR webpage at <http://deposit.bmrb.wisc.edu/bmrb-adit/>, create a new BMRB deposition, upload the file you have created, complete the deposition as needed, and add the BMRB deposition number to your paper.

Chapter 8

Reduced spectral density mapping

8.1 Introduction to reduced spectral density mapping

The reduced spectral density mapping analysis is often performed when the system under study is not suitable for model-free analysis, or as a last resort if a model-free analysis fails. The aim is to convert the relaxation data into three $J(\omega)$ values for the given field strength. Interpretation of this data, although slightly less convoluted than the relaxation data, is still plagued by problems related to non-spherical diffusion and much care must be taken when making conclusions. A full understanding of the model-free analysis and the effect of diffusion tensor anisotropy and rhombicity allows for better interpretation of the raw numbers.

To understand how reduced spectral density mapping is implemented in relax, the sample script will be worked through. This analysis type is not implemented in the GUI yet, though it shouldn't be too hard if anyone would like to contribute this and have a reference added to Chapter , the citations chapter.

8.2 $J(w)$ mapping script mode – the sample script

```
1 """Script for reduced spectral density mapping."""
2
3
4 # Create the data pipe.
5 pipe.create(pipe_name='my_protein', pipe_type='jw')
6
7 # Set up the 15N spins.
8 sequence.read(file='noe.600.out', res_num_col=1, res_name_col=2)
9 spin.name(name='N')
10 spin.element(element='N')
11 spin.isotope(isotope='15N', spin_id='@N')
12
13 # Load the 15N relaxation data.
14 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
15             res_num_col=1, data_col=3, error_col=4)
15 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
16             res_num_col=1, data_col=3, error_col=4)
```

```

16 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
17   res_num_col=1, data_col=3, error_col=4)
18
19 # Generate 1H spins for the magnetic dipole-dipole relaxation interaction.
20 sequence.attach_protons()
21
22 # Define the magnetic dipole-dipole relaxation interaction.
23 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
24 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)
25
26 # Define the chemical shift relaxation interaction.
27 value.set(val=-172 * 1e-6, param='csa')
28
29 # Select the frequency.
30 jw_mapping.set_frq(frq=600.0 * 1e6)
31
32 # Reduced spectral density mapping.
33 minimise.calculate()
34
35 # Monte Carlo simulations (well, bootstrapping as this is a calculation and not a fit!).
36 monte_carlo.setup(number=500)
37 monte_carlo.create_data()
38 minimise.calculate()
39 monte_carlo.error_analysis()
40
41 # Create grace files.
42 grace.write(y_data_type='j0', file='j0.agr', force=True)
43 grace.write(y_data_type='jwx', file='jwx.agr', force=True)
44 grace.write(y_data_type='jwh', file='jwh.agr', force=True)
45
46 # View the grace files.
47 grace.view(file='j0.agr')
48 grace.view(file='jwx.agr')
49 grace.view(file='jwh.agr')
50
51 # Write out the values.
52 value.write(param='j0', file='j0.txt', force=True)
53 value.write(param='jwx', file='jwx.txt', force=True)
54 value.write(param='jwh', file='jwh.txt', force=True)
55
56 # Finish.
57 results.write(file='results', force=True)
state.save('save', force=True)

```

8.3 J(w) mapping script mode – data pipe and spin system setup

The steps for setting up relax and the data model concept are described in full detail in Chapter 4. The first step, as for all analyses in relax, is to create a data pipe for storing all the data:

```

4 # Create the data pipe.
5 pipe.create(pipe_name='my_protein', pipe_type='jw')

```

Then, in this example, the ^{15}N spins are created from one of the NOE relaxation data files (Chapter 6):

```

7 # Set up the 15N spins.

```

```

8 sequence.read(file='noe.600.out', res_num_col=1, res_name_col=2)
9 spin.name(name='N')
10 spin.element(element='N')
11 spin.isotope(isotope='15N', spin_id='@N')

```

Skipping the relaxation data loading, the next part of the analysis is to create protons attached to the nitrogens for the magnetic dipole-dipole relaxation interaction:

```

18 # Generate 1H spins for the magnetic dipole-dipole relaxation interaction.
19 sequence.attach_protons()

```

This is needed to define the magnetic dipole-dipole interaction which governs relaxation.

8.4 J(w) mapping script mode – relaxation data loading

The loading of relaxation data is straight forward. This is performed prior to the creation of the proton spins so that the data is loaded only into the ^{15}N spin containers and not both spins for each residue. Only data for a single field strength can be loaded:

```

13 # Load the 15N relaxation data.
14 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
15     res_num_col=1, data_col=3, error_col=4)
15 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
16     res_num_col=1, data_col=3, error_col=4)
16 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
17     res_num_col=1, data_col=3, error_col=4)

```

The frequency of the data must also be explicitly specified:

```

28 # Select the frequency.
29 jw_mapping.set_frq(frq=600.0 * 1e6)

```

8.5 J(w) mapping script mode – relaxation interactions

Prior to calculating the $J(\omega)$ values, the physical interactions which govern relaxation of the spins must be defined. For the magnetic dipole-dipole relaxation interaction, the user functions are:

```

21 # Define the magnetic dipole-dipole relaxation interaction.
22 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
23 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)

```

For the chemical shift relaxation interaction, the user function call is:

```

25 # Define the chemical shift relaxation interaction.
26 value.set(val=-172 * 1e-6, param='csa')

```

8.6 J(w) mapping script mode – calculation and error propagation

Optimisation for this analysis is not needed as this is a direct calculation. Therefore the $J(\omega)$ values are simply calculated with the call:

```
31 # Reduced spectral density mapping.
32 minimise.calculate()
```

The propagation of errors is more complicated. The Monte Carlo simulation framework of relax can be used to propagate the relaxation data errors to the spectral density errors. As this is a direct calculation, this collapses into the standard bootstrapping method. The normal Monte Carlo user functions can be called:

```
34 # Monte Carlo simulations (well, bootstrapping as this is a calculation and not a fit!).
35 monte_carlo.setup(number=500)
36 monte_carlo.create_data()
37 minimise.calculate()
38 monte_carlo.error_analysis()
```

In this case, the `monte_carlo.initial_values` user function call is not required.

8.7 J(w) mapping script mode – visualisation and data output

The rest of the script is used to output the results to 2D Grace files for visualisation (the `grace.view` user function calls will launch Grace with the created files), and the output of the values into plain text files.

Chapter 9

Consistency testing

9.1 Introduction to the consistency testing of relaxation data

In spin relaxation, datasets are often recorded at different magnetic fields. This is especially important when R_2 values are to be used since μ s-ms motions contribute to R_2 . This contribution being scaled quadratically with the strength of the magnetic field, recording at multiple magnetic fields helps extract it. Also, acquiring data at multiple magnetic fields allows over-determination of the mathematical problems, e.g. in the model-free approach.

Recording at multiple magnetic fields is a good practice. However, it can cause artifacts if those different datasets are inconsistent. Inconsistencies can originate from, inter alia, the sample or the acquisition. Sample variations can be linked to changes in temperature, concentration, pH, etc. Water suppression is the main cause of acquisition variations as it affect relaxation parameters (especially NOE) of exposed and exchangeable moieties (e.g. the NH moiety).

It is thus a good idea to assess consistency of datasets acquired at different magnetic fields. For this purpose, three tests are implemented in relax. They are all based on the same principle – calculate a field independent value and compare it from one field to another.

The three tests are:

$J(0)$ The spectral density at the zero frequency calculated using the reduced spectral density approach.

F_η A consistency function proposed by [Fushman et al. \(1998\)](#).

F_{R_2} A consistency function proposed by [Fushman et al. \(1998\)](#).

These three tests are very similar (all probing consistency of R_2 data and all suffering from the same limitations) and any of them can be used for consistency testing. In the example below, the $J(0)$ values are used for consistency testing.

Different methods exist to compare tests values calculated from one field to another. These include correlation plots and histograms, and calculation of correlation, skewness and

kurtosis coefficients. The details of how to interpret such analyses are available at the end of this chapter in Section 9.7.

For more details on the tests and their implementation within relax, see:

- Morin, S. and Gagné, S. (2009a). Simple tests for the validation of multiple field spin relaxation data. *J. Biomol. NMR*, **45**, 361–372. ([10.1007/s10858-009-9381-4](https://doi.org/10.1007/s10858-009-9381-4))

Or for the origin of the tests themselves:

- Fushman, D., Tjandra, N., and Cowburn, D. (1999). An approach to direct determination of protein dynamics from ^{15}N NMR relaxation at multiple fields, independent of variable ^{15}N chemical shift anisotropy and chemical exchange contributions. *J. Am. Chem. Soc.*, **121**(37), 8577–8582. ([10.1021/ja9904991](https://doi.org/10.1021/ja9904991))

In addition, see the following review which includes a discussion on how to evaluate the reliability of recorded relaxation data:

- Morin, S. (2011). A practical guide to protein dynamics from ^{15}N spin relaxation in solution. *Prog. NMR Spectrosc.*, **59**(3), 245–262. ([10.1016/j.pnmrs.2010.12.003](https://doi.org/10.1016/j.pnmrs.2010.12.003))

9.2 Consistency testing in the prompt/script UI mode

The consistency testing analysis is only available via the prompt/script UI modes – no GUI auto-analysis has yet been built by a relax power-user.

9.2.1 Consistency testing script mode – the sample script

The following script can be found in the `sample_scripts` directory.

```

1  """ Script for consistency testing.
2
3  Severe artifacts can be introduced if model-free analysis is performed from inconsistent
   multiple magnetic field datasets. The use of simple tests as validation tools for the
   consistency assessment can help avoid such problems in order to extract more reliable
   information from spin relaxation experiments. In particular, these tests are useful
   for detecting inconsistencies arising from R2 data. Since such inconsistencies can
   yield artificial Rex parameters within model-free analysis, these tests should be used
   routinely prior to any analysis such as model-free calculations.
4
5  This script will allow one to calculate values for the three consistency tests J(0), F_eta
   and F_R2. Once this is done, qualitative analysis can be performed by comparing
   values obtained at different magnetic fields. Correlation plots and histograms are
   useful tools for such comparison, such as presented in Morin & Gagné (2009a) J.
   Biomol. NMR, 45: 361-372.
6
7
8  References
9  =====
10
```

```

11 The description of the consistency testing approach:
12
13 Morin & Gagne (2009a) Simple tests for the validation of multiple field spin
14 relaxation data. J. Biomol. NMR, 45: 361-372. U{http://dx.doi.org/10.1007/s10858-009-9381-4}
15 The origins of the equations used in the approach:
16
17 J(0):
18 Farrow et al. (1995) Spectral density function mapping using 15N relaxation data
19 exclusively. J. Biomol. NMR, 6: 153-162. U{http://dx.doi.org/10.1007/BF00211779}
20
21 F_eta:
22 Fushman et al. (1998) Direct measurement of 15N chemical shift anisotropy in
23 solution. J. Am. Chem. Soc., 120: 10947-10952. U{http://dx.doi.org/10.1021/ja981686m}
24
25 F_R2:
26 Fushman et al. (1998) Direct measurement of 15N chemical shift anisotropy in
27 solution. J. Am. Chem. Soc., 120: 10947-10952. U{http://dx.doi.org/10.1021/ja981686m}
28
29 A study where consistency tests were used:
30
31 Morin & Gagne (2009) NMR dynamics of PSE-4 beta-lactamase: An interplay of ps-ns order
32 and us-ms motions in the active site. Biophys. J., 96: 4681-4691. U{http://dx.doi.org/10.1016/j.bpj.2009.02.068}
33 """
34
35 # Create the data pipe.
36 name = 'consistency'
37 pipe.create(name, 'ct')
38
39 # Set up the 15N spins.
40 sequence.read('noe.600.out', res_num_col=1)
41 spin.name(name='N')
42 spin.element(element='N')
43 spin.isotope(isotope='15N', spin_id='@N')
44
45 # Load the relaxation data.
46 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
47 res_num_col=1, data_col=3, error_col=4)
48 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
49 res_num_col=1, data_col=3, error_col=4)
50 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
51 res_num_col=1, data_col=3, error_col=4)
52
53 # Generate the 1H spins for the magnetic dipole-dipole interaction.
54 sequence.attach_protons()
55
56 # Define the magnetic dipole-dipole relaxation interaction.
57 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
58 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)
59
60 # Define the chemical shift relaxation interaction.
61 value.set(val=-172 * 1e-6, param='csa')
62
63 # Set the angle between the 15N-1H vector and the principal axis of the 15N chemical shift
64 tensor
65 value.set(val=15.7, param='orientation')
66
67 # Set the approximate correlation time.
68 value.set(val=13 * 1e-9, param='tc')

```

```

61
62 # Set the frequency.
63 consistency_tests.set_frq(frq=600.0 * 1e6)
64
65 # Consistency tests.
66 minimise.calculate()
67
68 # Monte Carlo simulations.
69 monte_carlo.setup(number=500)
70 monte_carlo.create_data()
71 minimise.calculate()
72 monte_carlo.error_analysis()
73
74 # Create grace files.
75 grace.write(y_data_type='j0', file='j0.agr', force=True)
76 grace.write(y_data_type='f_eta', file='f_eta.agr', force=True)
77 grace.write(y_data_type='f_r2', file='f_r2.agr', force=True)
78
79 # View the grace files.
80 grace.view(file='j0.agr')
81 grace.view(file='f_eta.agr')
82 grace.view(file='f_r2.agr')
83
84 # Finish.
85 results.write(file='results', force=True)
86 state.save('save', force=True)

```

This is similar in spirit to the reduced spectral density mapping sample script (Chapter 8 on page [129](#)).

9.3 Consistency testing script mode – data pipe and spin system setup

The steps for setting up relax and the data model concept are described in full detail in Chapter 4. The first step, as for all analyses in relax, is to create a data pipe for storing all the data:

```

31 # Create the data pipe.
32 name = 'consistency'
33 pipe.create(name, 'ct')

```

Then, in this example, the ^{15}N spins are created from one of the NOE relaxation data files (Chapter 6):

```

35 # Set up the  $^{15}\text{N}$  spins.
36 sequence.read('noe.600.out', res_num_col=1)
37 spin.name(name='N')
38 spin.element(element='N')
39 spin.isotope(isotope=' $^{15}\text{N}$ ', spin_id='@N')

```

Skipping the relaxation data loading, the next part of the analysis is to create protons attached to the nitrogens for the magnetic dipole-dipole relaxation interaction:

```

46 # Generate the  $^1\text{H}$  spins for the magnetic dipole-dipole interaction.
47 sequence.attach_protons()

```

This is needed to define the magnetic dipole-dipole interaction which governs relaxation.

9.4 Consistency testing script mode – relaxation data loading

The loading of relaxation data is straight forward. This is performed prior to the creation of the proton spins so that the data is loaded only into the ^{15}N spin containers and not both spins for each spin system. Note that if the relaxation data files contain spin information, then this order is not important. For this analysis, only data for a single field strength can be loaded:

```
41 # Load the relaxation data.
42 relax_data.read(ri_id='R1_600', ri_type='R1', frq=600.0*1e6, file='r1.600.out',
43     res_num_col=1, data_col=3, error_col=4)
43 relax_data.read(ri_id='R2_600', ri_type='R2', frq=600.0*1e6, file='r2.600.out',
44     res_num_col=1, data_col=3, error_col=4)
44 relax_data.read(ri_id='NOE_600', ri_type='NOE', frq=600.0*1e6, file='noe.600.out',
    res_num_col=1, data_col=3, error_col=4)
```

The frequency of the data must also be explicitly specified:

```
62 # Set the frequency.
63 consistency_tests.set_frq(frq=600.0 * 1e6)
```

9.5 Consistency testing script mode – relaxation interactions

Prior to calculating the $J(0)$, F_η , and F_{R_2} values, the physical interactions which govern relaxation of the spins must be defined. For the magnetic dipole-dipole relaxation interaction, the user functions are:

```
49 # Define the magnetic dipole-dipole relaxation interaction.
50 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
51 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.02 * 1e-10)
```

For the chemical shift relaxation interaction, the user function call is:

```
53 # Define the chemical shift relaxation interaction.
54 value.set(val=-172 * 1e-6, param='csa')
```

For the angle in degrees between the $^{15}\text{N}-^1\text{H}$ vector and the principal axis of the ^{15}N chemical shift tensor, the user function call is:

```
56 # Set the angle between the 15N-1H vector and the principal axis of the 15N chemical shift
      tensor
57 value.set(val=15.7, param='orientation')
```

9.6 Consistency testing script mode – calculation and error propagation

Optimisation for this analysis is not needed as this is a direct calculation. Therefore the $J(0)$, F_η , and F_{R_2} values are simply calculated with the call:

```
65 # Consistency tests.
66 minimise.calculate()
```

The propagation of errors is more complicated. The Monte Carlo simulation framework of relax can be used to propagate the relaxation data errors to the spectral density errors. As this is a direct calculation, this collapses into the standard bootstrapping method. The normal Monte Carlo user functions can be called:

```
68 # Monte Carlo simulations.
69 monte_carlo.setup(number=500)
70 monte_carlo.create_data()
71 minimise.calculate()
72 monte_carlo.error_analysis()
```

In this case, the `monte_carlo.initial_values` user function call is not required.

9.7 Consistency testing script mode – visualisation and data output

The rest of the script is used to output the results to 2D Grace files for visualisation (the `grace.view` user function calls will launch Grace with the created files), and the output of the values into plain text files.

However, simply visualizing the calculated $J(0)$, F_η , and F_{R_2} values this way does not allow proper consistency testing. Indeed, for assessing the consistency of relaxation data using these tests, different methods exist to compare values calculated from one field to another. These include correlation plots and histograms, and calculation of correlation, skewness and kurtosis coefficients.

To complete the consistency testing analysis, the following steps are needed:

- Extract the $J(0)$ values at multiple magnetic fields.
- Join together the data from a pair of magnetic fields either by pasting them as two columns of one file (approach A), or by dividing values from a first magnetic field by values from a second magnetic field (approach B).
- Make either a correlation plot (approach A), or an histogram of the ratios (approach B).
- See if the correlation plot is centered around a perfect correlation or skewed away (approach A), or if the values are centered around 1 in the histogram (approach B). If yes, data from multiple magnetic fields is consistent from one magnetic field to another. If no, data is inconsistent. In the case where inconsistency arises, if data

from more than two magnetic fields is available, more than one pair of data can be checked and the inconsistent magnetic field data can be identified.

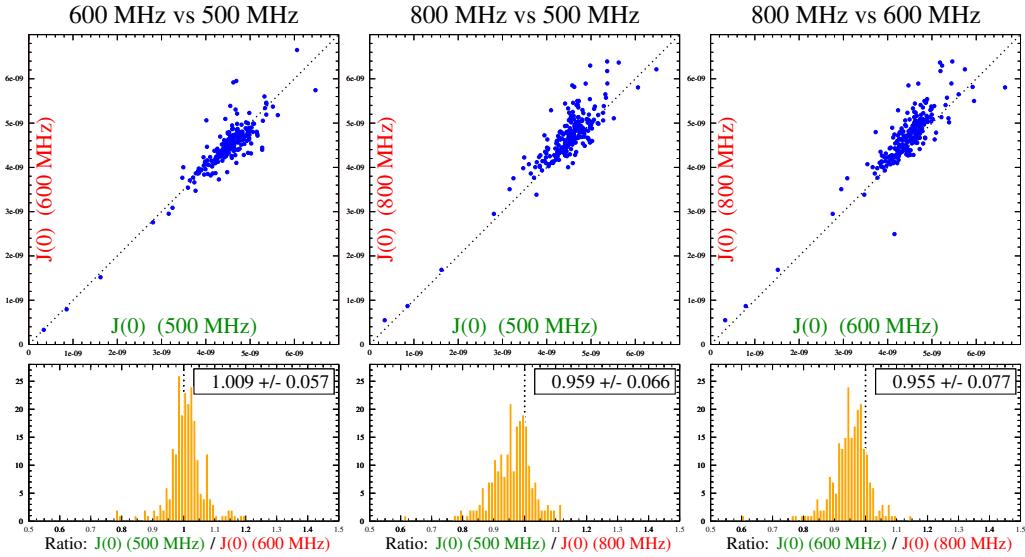


Figure 9.1: Example of consistency testing visual analysis. Relaxation data from three different magnetic fields are compared. For each pair of magnetic field, a correlation plot of the calculated $J(0)$ values (approach A, top) as well as an histogram of the ratio of calculated $J(0)$ values (approach B, bottom) are shown. These graphs must be manually created from the output of the sample script shown in section 9.2.1. The PSE-4 data, as published in Morin and Gagné (2009b), has been reused for the purpose of this example.

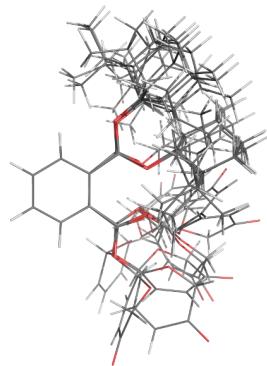
An example of such an analysis is shown in Figure 9.1. This example displays both consistent and inconsistent data. As the figure shows, the data recorded at 500 MHz and 600 MHz are consistent with each other whereas the data recorded at 800 MHz is consistent with the neither the 500 MHz nor 600 MHz data. Since more than two magnetic fields were used, this allowed the identification of the 800 MHz data as being inconsistent allowing the authors to take special care with this data set.

The 800 MHz data inconsistency is seen in the correlation plots (top) by a deviation from the dotted line (which represents the theoretical situation when equal $J(0)$ values are extracted from both magnetic fields. It is also observable in the histograms (bottom) where the ratio of the data from two magnetic fields is not centered at 1.0. In fact, there seems to be a systematic shift of the calculated $J(0)$ values at 800 MHz when compared to the two other magnetic fields. This is caused by a similar shift in the experimental R_2 (transversal relaxation rate) data.

For the 500 MHz and 600 MHz data pair, the data are centered around the dotted line in the correlation plot (approach A, top left) as well as centered around a value of 1.0 in the histogram comparing the ratios of values from both magnetic fields (approach B, bottom left). Of course, there are some outlier values even in the case of consistent data. There are caused by specific dynamic characteristics of these spins and are different from systematic inconsistencies such as depicted in the example above with the data recorded at 800 MHz.

Chapter 10

The N-state model or ensemble analysis



10.1 Introduction to the N-state model

The modelling of motion in molecules using experimental data consists of either continuous or discrete distributions. These can be visualised respectively as either an infinite number of states or a limited set of N states. The N -state model analysis in relax models the molecular dynamics using an ensemble of static structures.

This analysis supports a number of data types including:

- Residual dipolar couplings (RDCs)
- Pseudo-contact shifts (PCSs)
- NOEs

The main idea is to evaluate the quality of a fixed ensemble of structures. relax will not perform structural optimisations. The evaluation includes:

- Alignment tensor optimisation for the RDCs and PCSs.

- Optional optimisation of the position of the paramagnetic centre for the PCSs.
- Calculation of NOE constraint violations.
- Q factor calculation for the RDC, PCS, and NOE.

Note that this analysis will also handle single structures. Hence you can use the N-state model in relax with N set to 1 to find, for example, a single alignment tensor for a single structure using RDCs, PCSs, or both together. This is useful for comparing a ensemble to a single structure to determine if any statistically significant motions are present.

The primary references for the N-state model analysis in relax are:

- Sun, H., d'Auvergne, E. J., Reinscheid, U. M., Dias, L. C., Andrade, C. K. Z., Rocha, R. O., and Griesinger, C. (2011). Bijvoet in solution reveals unexpected stereoselectivity in a michael addition. *Chem. Eur. J.*, **17**(6), 1811–1817. ([10.1002/chem.201002520](https://doi.org/10.1002/chem.201002520))
- Erdelyi, M., d'Auvergne, E., Navarro-Vazquez, A., Leonov, A., and Griesinger, C. (2011). Dynamics of the glycosidic bond: Conformational space of lactose. *Chem. Eur. J.*, **17**(34), 9368–9376. ([10.1002/chem.201100854](https://doi.org/10.1002/chem.201100854))

10.2 Experimental data support for the N-state model

10.2.1 RDCs in the N-state model

Residual dipolar couplings (RDCs) can be used to evaluate ensembles. The ensemble interconversion is assumed to be fast relative to timescale of the alignment process, hence a single tensor for all members of the ensemble will be used. As such, precise superimposition of structures using a logical frame of reference is very important. This can be performed in relax using the `structure.superimpose` user function. The RDCs can either be from external or internal alignment.

10.2.2 PCSs in the N-state model

Pseudo-contact shifts (PCSs) can also be used to evaluate ensembles. The same averaging process as described above for the RDC is assumed. Hence correct structural superimposition is essential and one alignment tensor will be optimised for the entire ensemble.

One powerful feature of relax is that the paramagnetic centre can either be fixed or be allowed to move during optimisation. This allows an unknown paramagnetic centre position to be found. Or a known position to be refined to higher accuracy than that possible with most other techniques.

10.2.3 NOEs in the N-state model

Another data type which can be used to evaluate dynamics ensembles is the NOE. This is not used in optimisation but rather is used to calculate NOE constraint violations. These violations are then compared to evaluate the ensemble. In the stereochemistry auto-analysis, these violations will also be converted to Q factors to allow direct comparison with RDC Q factors.

10.3 Determining stereochemistry in dynamic molecules

A published application of the N-state model in relax is:

- Sun, H., d'Auvergne, E. J., Reinscheid, U. M., Dias, L. C., Andrade, C. K. Z., Rocha, R. O., and Griesinger, C. (2011). Bijvoet in solution reveals unexpected stereoselectivity in a michael addition. *Chem. Eur. J.*, **17**(6), 1811–1817. ([10.1002/chem.201002520](https://doi.org/10.1002/chem.201002520))

This analysis of the stereochemistry of a small molecule consists of two steps. The first part is to determine the relative configuration. The idea is to use NMR data (consisting of RDCs and NOEs) to find the relative configuration. Ensembles of 10 members are created from molecular dynamics simulations (MD) or simulated annealing (SA). These are then ranked by the RDC Q factor and NOE violation. By converting the NOE violation into a Q factor:

$$Q_{\text{NOE}}^2 = \frac{U}{\sum_i \text{NOE}^2}, \quad (10.1)$$

where U is the quadratic flat bottom well potential, i.e. the NOE violation in \AA^2 , and the denominator is the sum of all squared NOEs. A combined Q factor is calculated as:

$$Q_{\text{total}}^2 = Q_{\text{NOE}}^2 + Q_{\text{RDC}}^2. \quad (10.2)$$

The second step is to distinguish enantiomers. As NMR data is symmetric, it cannot distinguish enantiomers. Therefore an optical technique such as [optical rotatory dispersion](#) can be used. For molecules experiencing large amounts of motion, sampling all possible conformations, calculating the expected dispersion properties, and calculating an averaged dispersion curve is not feasible. The idea is therefore to combine NMR and ORD by taking the best NMR ensembles from step one to use for ORD spectral prediction.

10.3.1 Stereochemistry – the auto-analysis

Step one of the N-state model is implemented as an auto-analysis. This is located in the module `auto_analysis.stereochem_analysis` (see http://www.nmr-relax.com/api/3.1/auto_analyses.stereochem_analysis-module.html). The auto-analysis is accessed via the `Stereochem_analysis` class, the details of which can be seen at http://www.nmr-relax.com/api/3.1/auto_analyses.stereochem_analysis.Stereochem_analysis-class.html.

10.3.2 Stereochemistry – the sample script

The following script was used for the analysis in Sun et al. (2011). It is used to complete the first step of the analysis, the determination of relative configuration, and for the generation of ensembles for the second step of the analysis. The file is located at `sample_scripts/n_state_model/stereochem_analysis.py`. The contents of the script are:

```

1     """Script for the determination of relative stereochemistry.
2
3     The analysis is preformed by using multiple ensembles of structures, randomly sampled from
4         a given set of structures. The discrimination is performed by comparing the sets of
5         ensembles using NOE violations and RDC Q factors.
6
7     This script is split into multiple stages:
8
9     1. The random sampling of the snapshots to generate the N ensembles (NUM_ENS, usually
10        10,000 to 100,000) of M members (NUM_MODELS, usually ~10). The original snapshot
11        files are expected to be named the SNAPSHOT_DIR + CONFIG + a number from SNAPSHOT_MIN
12        to SNAPSHOT_MAX + ".pdb", e.g. "snapshots/R647.pdb". The ensembles will be placed
13        into the "ensembles" directory.
14
15     2. The NOE violation analysis.
16
17     3. The superimposition of ensembles. For each ensemble, Molmol is used for
18        superimposition using the fit to first algorithm. The superimposed ensembles will be
19        placed into the "ensembles_superimposed" directory. This stage is not necessary for
20        the NOE analysis.
21
22     4. The RDC Q factor analysis.
23
24     5. Generation of Grace graphs.
25
26     6. Final ordering of ensembles using the combined RDC and NOE Q factors, whereby the
27        NOE Q factor is defined as::
28
29     Q^2 = U / sum(NOE_i^2),
30
31     where U is the quadratic flat bottom well potential - the NOE violation in Angstrom^2.
32     The denominator is the sum of all squared NOEs - this must be given as the value of
33     NOE_NORM. The combined Q is given by::
34
35     Q_total^2 = Q_NOE^2 + Q_RDC^2.
36
37     """
38
39     # relax module imports.
40     from auto_analyses.stereochem_analysis import Stereochem_analysis
41
42
43     # Stage of analysis (see the docstring above for the options).
44     STAGE = 1
45
46     # Number of ensembles.
47     NUM_ENS = 100000
48
49     # Ensemble size.
50     NUM_MODELS = 10
51
52     # Configurations.
53     CONFIGS = ["R", "S"]

```

```
42 # Snapshot directories (corresponding to CONFIGS).
43 SNAPSHOT_DIR = ["snapshots", "snapshots"]
44
45 # Min and max number of the snapshots (corresponding to CONFIGS).
46 SNAPSHOT_MIN = [0, 0]
47 SNAPSHOT_MAX = [76, 71]
48
49 # Pseudo-atoms.
50 PSEUDO = [
51     ["Q7", ["@H16", "@H17", "@H18"]],
52     ["Q9", ["@H20", "@H21", "@H22"]],
53     ["Q10", ["@H23", "@H24", "@H25"]]
54 ]
55
56 # NOE info.
57 NOE_FILE = "noes"
58 NOE_NORM = 50 * 4**2      # The NOE normalisation factor (sum of all NOEs squared).
59
60 # RDC file info.
61 RDC_NAME = "PAN"
62 RDC_FILE = "pan_rdcs"
63 RDC_SPIN_ID1_COL = 1
64 RDC_SPIN_ID2_COL = 2
65 RDC_DATA_COL = 2
66 RDC_ERROR_COL = None
67
68 # Bond length.
69 BOND_LENGTH = 1.117 * 1e-10
70
71 # Log file output (only for certain stages).
72 LOG = True
73
74 # Number of buckets for the distribution plots.
75 BUCKET_NUM = 200
76
77 # Distribution plot limits.
78 LOWER_LIM_NOE = 0.0
79 UPPER_LIM_NOE = 600.0
80 LOWER_LIM_RDC = 0.0
81 UPPER_LIM_RDC = 1.0
82
83
84 # Set up and code execution.
85 analysis = Stereochem_analysis(
86     stage=STAGE,
87     num_ens=NUM_ENS,
88     num_models=NUM_MODELS,
89     configs=CONFIGS,
90     snapshot_dir=SNAPSHOT_DIR,
91     snapshot_min=SNAPSHOT_MIN,
92     snapshot_max=SNAPSHOT_MAX,
93     pseudo=PSEUDO,
94     noe_file=NOE_FILE,
95     noe_norm=NOE_NORM,
96     rdc_name=RDC_NAME,
97     rdc_file=RDC_FILE,
98     rdc_spin_id1_col=RDC_SPIN_ID1_COL,
99     rdc_spin_id2_col=RDC_SPIN_ID2_COL,
100    rdc_data_col=RDC_DATA_COL,
101    rdc_error_col=RDC_ERROR_COL,
102    bond_length=BOND_LENGTH,
```

```
103     log=LOG,  
104     bucket_num=BUCKET_NUM,  
105     lower_lim_noe=LOWER_LIM_NOE,  
106     upper_lim_noe=UPPER_LIM_NOE,  
107     lower_lim_rdc=LOWER_LIM_RDC,  
108     upper_lim_rdc=UPPER_LIM_RDC  
109 )  
110 analysis.run()
```

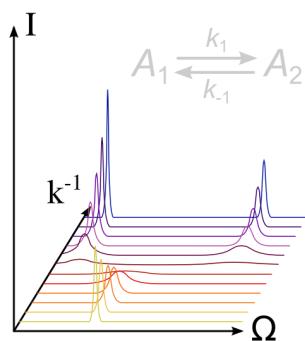
In contrast to all of the other auto-analyses, here you do not set up your own data pipe containing all of the relevant data that is then passed into the auto-analysis. This may change in the future to allow for more flexibility in how you load structures, load the RDC and NOE base data, set up pseudo-atoms and bond lengths for the RDC, etc.

Note that you need to execute this script 6 times, changing the `STAGE` variable to match. These stages are fully documented at the start of the script.

Due to the original analysis being performed prior to the addition of the `structure.superimpose` user function to relax, you will see that the auto-analysis performs superimposition of each ensemble using the external software `Molmol`. If you wish to perform this analysis without using `Molmol`, please contact the relax users mailing list “nmr-relax-users at lists.sourceforge.net” (see Section 3.3.2 on page 31). It would be rather straightforward for the relax developers to replace the complicated `Molmol` superimposition code with a single call to the `structure.superimpose` user function.

Chapter 11

The analysis of relaxation dispersion



11.1 Introduction to relaxation dispersion

Relaxation dispersion is the experimental modulation of chemical exchange relaxation. For the $R_{1\rho}$ -type experiment in which the nucleus of interest is spin-locked, either the spin-lock field strength or the offset between the spin-lock pulse and the chemical shift of the spins is used to modulate the exchange. For the CPMG-type experiment, varying the time between the pulses modules the exchange. Both experiment types are handled by relax.

The primary reference for the relaxation dispersion implemented in relax is:

- Morin, S., Linnet, T. E., Lescanne, M., Schanda, P., Thompson, G. S., Tollinger, M., Teilmann, K., Gagne, S., Marion, D., Griesinger, C., Blackledge, M., and d'Auvergne, E. J. (2014). relax: the analysis of biomolecular kinetics and thermodynamics using NMR relaxation dispersion data. *Bioinformatics*, **30**(15), 2219–2220. ([10.1093/bioinformatics/btu166](https://doi.org/10.1093/bioinformatics/btu166))

For other citations, please see the citation chapter on page [xxvii](#).

11.1.1 The modelling of dispersion data

For a system under the influence of chemical exchange, the evolution of the transverse magnetisation is given by the [Bloch \(1946\)](#) equations as modified by [McConnell \(1958\)](#) for chemical exchange – the Bloch-McConnell equations. For a two state exchange jumping between states A and B, the equation is:

$$\frac{d}{dt} \begin{bmatrix} M_A^+(t) \\ M_B^+(t) \end{bmatrix} = \begin{bmatrix} -i\Omega_A - R_{2A}^0 - p_B k_{\text{ex}} & p_A k_{\text{ex}} \\ p_B k_{\text{ex}} & -i\Omega_B - R_{2B}^0 - p_A k_{\text{ex}} \end{bmatrix} \begin{bmatrix} M_A^+(t) \\ M_B^+(t) \end{bmatrix}. \quad (11.1)$$

The analytic or closed-form frequency-domain solution for this equation however remains intractable. Solutions can nevertheless be found by either making assumptions or restrictions about the exchange process and then analytically solving [11.1](#) or by finding numeric solutions. The modelling of relaxation dispersion data can hence be categorised into these two distinct methodologies:

Analytical models: Optimisation of models based on analytical, closed-form expressions derived from the Bloch-McConnell equations subject to certain conditions (see [Section 11.3](#) on page [159](#) and [Section 11.7](#) on page [176](#)).

Numerical models: Optimisation of models based on numerically solving the Bloch-McConnell equations (see [Section 11.4](#) on page [166](#) and [Section 11.8](#) on page [180](#)).

11.1.2 Implemented models

A number of analytic and numeric models are supported within relax. These cover single quantum (SQ) CPMG-type, combined proton-heteronuclear single quantum (SQ), zero quantum (ZQ), double quantum (DQ) and multi quantum (MQ) CPMG-type experiments, and $R_{1\rho}$ -type. If the model you are interested in is not available, please see [Section 11.11](#) on page [197](#) for how you can add new models to relax.

Models which are independent of the experiment type include:

'R2eff': This is the model used to determine the $R_{2\text{eff}}$ or $R_{1\rho}$ values and errors required as the base data for all other models. See [Section 11.2.1](#) on page [156](#).

'No Rex': This is the model for no chemical exchange being present. See [Section 11.2.2](#) on page [158](#).

For the SQ CPMG-type experiments, the analytic models currently supported are:

'LM63': The original [Luz and Meiboom \(1963\)](#) 2-site fast exchange equation with parameters $\{R_2^0, \dots, \Phi_{\text{ex}}, k_{\text{ex}}\}$. See [Section 11.3.1](#) on page [159](#).

'LM63 3-site': The original [Luz and Meiboom \(1963\)](#) 3-site fast exchange equation with parameters $\{R_2^0, \dots, \Phi_{\text{ex},B}, k_B, \Phi_{\text{ex},C}, k_C\}$. The equations of [O'Connell et al. \(2009\)](#) can be used to approximately convert the parameters $\{\Phi_{\text{ex},B}, k_B, \Phi_{\text{ex},C}, k_C\}$ to more biologically relevant parameters. See [Section 11.3.2](#) on page [160](#).

‘CR72’: The reduced Carver and Richards (1972) 2-site equation for most time scales whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.3.4 on page 162.

‘CR72 full’: The full Carver and Richards (1972) 2-site equation for most time scales with parameters $\{R_{2A}^0, R_{2B}^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.3.3 on page 161.

‘IT99’: The Ishima and Torchia (1999) 2-site model for all time scales with $p_A \gg p_B$ and with parameters $\{R_2^0, \dots, p_A, \Delta\omega, \tau_{ex}\}$. See Section 11.3.5 on page 162.

‘TSMFK01’: The Tollinger et al. (2001) 2-site very-slow exchange model for time scales within range of microsecond to second time scale. Applicable in the limit of slow exchange, when $|R_{2A}^0 - R_{2B}^0| \ll k_{AB}, k_{BA} \ll 1/\tau_{CPMG}$. $2 * \tau_{CPMG}$ is the time between successive 180 degree pulses. Parameters are $\{R_{2A}^0, \dots, \Delta\omega, k_{AB}\}$. See Section 11.3.6 on page 163.

‘B14’: The reduced Baldwin (2014) 2-site exact solution equation for all time scales whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.3.8 on page 166.

‘B14 full’: The full Baldwin (2014) 2-site exact equation for all time scales with parameters $\{R_{2A}^0, R_{2B}^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.3.7 on page 164.

For the SQ CPMG-type experiments, the numeric models currently supported are:

‘NS CPMG 2-site expanded’: A model for 2-site exchange expanded using Maple by Nikolai Skrynnikov (Tollinger et al., 2001). It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.4.1 on page 166.

‘NS CPMG 2-site 3D’: The reduced model for 2-site exchange using 3D magnetisation vectors whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.4.3 on page 169.

‘NS CPMG 2-site 3D full’: The full model for 2-site exchange using 3D magnetisation vectors with parameters $\{R_{2A}^0, R_{2B}^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.4.2 on page 169.

‘NS CPMG 2-site star’: The reduced model for 2-site exchange using complex conjugate matrices whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.4.5 on page 170.

‘NS CPMG 2-site star full’: The full model for 2-site exchange using complex conjugate matrices with parameters $\{R_{2A}^0, R_{2B}^0, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.4.4 on page 169.

For the combined proton-heteronuclear SQ, ZQ, DQ and MQ CPMG-type experiments (MMQ – or multi-multiple quantum), the analytic models currently supported are:

‘MMQ CR72’: The Carver and Richards (1972) 2-site model for most time scales expanded for MMQ CPMG data by Korzhnev et al. (2004a). It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, \Delta\omega^H, k_{ex}\}$. See Section 11.5.1 on page 170.

For the combined proton-heteronuclear SQ, ZQ, DQ and MQ CPMG-type experiments (MMQ – or multi-multiple quantum), the numeric models currently supported are:

‘NS MMQ 2-site’: The model for 2-site exchange whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, \Delta\omega, \Delta\omega^H, k_{ex}\}$. See Section 11.6.1 on page 172.

‘NS MMQ 3-site linear’: The model for 3-site exchange linearised with $k_{AC} = k_{CA} = 0$ whereby the simplification $R_{2A}^0 = R_{2B}^0 = R_{2C}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, \Delta\omega_{AB}^H, \Delta\omega_{BC}^H, k_{ex}^{AB}, k_{ex}^{BC}\}$. See Section 11.6.2 on page 174.

‘NS MMQ 3-site’: The model for 3-site exchange whereby the simplification $R_{2A}^0 = R_{2B}^0 = R_{2C}^0$ is assumed. It has the parameters $\{R_2^0, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, \Delta\omega_{AB}^H, \Delta\omega_{BC}^H, k_{ex}^{AB}, k_{ex}^{BC}, k_{ex}^{AC}\}$. See Section 11.6.3 on page 175.

For the $R_{1\rho}$ -type experiments, the analytic models currently supported are:

‘M61’: The [Meiboom \(1961\)](#) 2-site fast exchange equation for on-resonance data with parameters $\{R'_{1\rho}, \dots, \Phi_{ex}, k_{ex}\}$. See Section 11.7.1 on page 177.

‘DPL94’: The [Davis et al. \(1994\)](#) extension of the ‘M61’ model for off-resonance data with parameters $\{R'_{1\rho}, \dots, \Phi_{ex}, k_{ex}\}$. See Section 11.7.3 on page 178.

‘M61 skew’: The [Meiboom \(1961\)](#) 2-site equation for all time scales with $p_A \gg p_B$ and with parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$. This model is disabled by default in the dispersion auto-analysis. See Section 11.7.2 on page 177.

‘TP02’: The [Trott and Palmer \(2002\)](#) 2-site equation for all time scales with $p_A \gg p_B$ and with parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.7.4 on page 178.

‘TAP03’: The [Trott et al. \(2003\)](#) off-resonance 2-site analytic equation for all time scales with the weak condition $p_A \gg p_B$ and with parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$.

‘MP05’: The [Miloushev and Palmer \(2005\)](#) off-resonance 2-site equation for all time scales with parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.7.6 on page 180.

For the $R_{1\rho}$ -type experiments, the numeric models currently supported are:

‘NS R1rho 2-site’: The model for 2-site exchange using 3D magnetisation vectors. It has the parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$. See Section 11.8.1 on page 181.

‘NS R_{1ρ} 3-site linear’: The model for 3-site exchange linearised with $k_{AC} = k_{CA} = 0$ whereby the simplification $R'_{1\rho A} = R'_{1\rho B} = R'_{1\rho C}$ is assumed. It has the parameters $\{R'_{1\rho}, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, k_{ex}^{AB}, k_{ex}^{BC}\}$. See Section 11.8.3 on page 183.

‘NS R_{1ρ} 3-site’: The model for 3-site exchange whereby the simplification $R'_{1\rho A} = R'_{1\rho B} = R'_{1\rho C}$ is assumed. It has the parameters $\{R'_{1\rho}, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, k_{ex}^{AB}, k_{ex}^{BC}, k_{ex}^{AC}\}$. See Section 11.8.2 on page 181.

11.1.3 Dispersion model summary

Except for ‘R2eff’ and ‘No Rex’, all models can be fit to clusterings of spins, or spin blocks. The models are described in more detail below and summarised in Table 11.1. The parameters of the models and of relaxation dispersion in general are given in Table 17.5.

R₁ parameter optimisation

For a number of models, the off-resonance R₁ value can be optimised. Normally the off-resonance models will use fixed experimental R₁ values for optimisation. However if the experimental values are not loaded, then the R₁ values will be automatically optimised. For finer control of this optimisation behaviour, see the `relax_disp.r1_fit` user function (page 583). The models which support off-resonance R₁ fitting include:

- ‘No Rex’,
- ‘DPL94’,
- ‘TP02’,
- ‘TAP03’,
- ‘MP05’,
- ‘NS R1rho 2-site’.

In the future, support for off-resonance effects in the CPMG experiments is planned (see section 11.10 on page 196).

Table 11.1: The dispersion models supported by relax.

Model name	Solution	Sites	Parameters	Restrictions	Reference
<hr/>					
Experiment independent					
R2eff	-	-	{R _{2eff} , ...}	Fixed relaxation time period	-
R2eff	-	-	{R _{2eff} , I ₀ , ...}	Variable relaxation time period	-
No Rex	Closed	1	{R ₂ ^{0, ...}}	-	-
<hr/>					
CPMG-type					
LM63	Analytic	2	{R ₂ ⁰ , ..., Φ _{ex} , k _{ex} }	Fast exchange	Luz and Meiboom (1963)
LM63 3-site	Analytic	3	{R ₂ ⁰ , ..., Φ _{ex,B} , k _B , Φ _{ex,C} , k _C }	Fast exchange, p _A > p _B and p _A > p _C	Luz and Meiboom (1963)
CR72	Analytic	2	{R _{2A} ⁰ , R _{2B} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B , not very slow exchange	Carver and Richards (1972)
CR72 full	Analytic	2	{R _{2A} ⁰ , R _{2B} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B , not very slow exchange	Carver and Richards (1972)
IT99	Analytic	2	{R ₂ ⁰ , ..., p _A , Δω, τ _{ex} }	p _A ≈ p _B	Ishma and Torchia (1999)
TSMFK01	Analytic	2	{R _{2A} ⁰ , ..., Δω, k _{AB} }	p _A ≈ p _B slow exchange	Tollinger et al. (2001)
B14	Analytic	2	{R _{2A} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B ,	Baldwin (2014)
B14 full	Analytic	2	{R _{2A} ⁰ , R _{2B} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B ,	Baldwin (2014)
NS CPMG 2-site expanded	Numeric	2	{R ₂ ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B	Tollinger et al. (2001)
NS CPMG 2-site 3D	Numeric	2	{R ₂ ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B	-
NS CPMG 2-site 3D full	Numeric	2	{R _{2A} ⁰ , R _{2B} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B	-
NS CPMG 2-site star	Numeric	2	{R ₂ ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B	-
NS CPMG 2-site star full	Numeric	2	{R _{2A} ⁰ , R _{2B} ⁰ , ..., p _A , Δω, k _{ex} }	p _A > p _B	-

Table 11.1: The dispersion models supported by relax.

Model name	Solution	Sites	Parameters	Restrictions	Reference
MMQ CPMG-type					
MMQ CR72	Analytic	2	$\{R_2^0, \dots, p_A, \Delta\omega, \Delta\omega^H, k_{ex}\}$	$p_A > p_B$	Korzhnev et al. (2004a)
NS MMQ 2-site	Numeric	2	$\{R_2^0, \dots, p_A, \Delta\omega, \Delta\omega^H, k_{ex}\}$	$p_A > p_B$	Korzhnev et al. (2005a)
NS MMQ 3-site linear	Numeric	3	$\{R_2^0, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, \Delta\omega_{AB}^H, \Delta\omega_{BC}^H, k_{ex}^H, k_{ex}^B\}$	$p_A > p_B$ and $p_B > p_C$	Korzhnev et al. (2005a)
NS MMQ 3-site	Numeric	3	$\{R_2^0, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, \Delta\omega_{AB}^H, \Delta\omega_{BC}^H, k_{ex}^H, k_{ex}^B, k_{ex}^C\}$	$p_A > p_B$ and $p_B > p_C$	Korzhnev et al. (2005a)
R_{1ρ}-type					
M61	Analytic	2	$\{R'_{1\rho}, \dots, \Phi_{ex}, k_{ex}\}$	Fast exchange, on-resonance, $R_1 = R_2$	Meiboom (1961)
DPL94	Analytic	2	$\{R'_{1\rho}, \dots, \Phi_{ex}, k_{ex}\}$	Fast exchange	Davis et al. (1994)
M61 skew	Analytic	2	$\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A \gg p_B$, on-resonance	Meiboom (1961)
TP02	Analytic	2	$\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A \gg p_B$, not fast exchange	Trott and Palmer (2002)
TAP03	Analytic	2	$\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$	Weak condition of $p_A \gg p_B$	Trott et al. (2003)
TP04 ¹	Analytic	N	$\{R'_{1\rho}, \dots, p_1, \dots, p_N, \bar{\omega}, k_{12}, \dots, k_{1N}\}$	One site dominant	Trott and Palmer (2004)
MP05	Analytic	2	$\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A > p_B$	Miloushev and Palmer (2005)
BK13	Analytic	2	$\{R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A > p_B$,	Baldwin and Kay (2013)
BK13 full	Analytic	2	$\{R_2^0, R_2^0, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A > p_B$,	Baldwin and Kay (2013)
NS R1rho 2-site	Numeric	2	$\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{ex}\}$	$p_A > p_B$	-
NS R1rho 3-site linear	Numeric	3	$\{R'_{1\rho}, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, k_{ex}^H, k_{ex}^B, k_{ex}^C\}$	$p_A > p_B$ and $p_A > p_C$	-
NS R1rho 3-site	Numeric	3	$\{R'_{1\rho}, \dots, p_A, p_B, \Delta\omega_{AB}, \Delta\omega_{BC}, k_{ex}^H, k_{ex}^B, k_{ex}^C\}$	$p_A > p_B$ and $p_A > p_C$	-

¹Not implemented yet

Table 11.2: The parameters of relaxation dispersion.

Parameter	Equation	Description	Units
ν_{CPMG}	$1/(4\tau_{\text{CPMG}})$	CPMG frequency	Hz
τ_{CPMG}	$1/(4\nu_{\text{CPMG}})$	Delay between CPMG π pulses	s
T_{relax}	-	The relaxation delay period	s
I_0	-	Reference peak intensity when T_{relax} is zero	-
I_1	-	Peak intensity for a given ν_{CPMG} or spin-lock field strength ω_1	-
R_2^0	-	R_2 relaxation rate in the absence of exchange	rad.s^{-1}
R_{2A}^0	-	R_2 relaxation rate for state A in the absence of exchange	rad.s^{-1}
R_{2B}^0	-	R_2 relaxation rate for state B in the absence of exchange	rad.s^{-1}
$R'_{1\rho}$	-	$R_{1\rho}$ relaxation rate in the absence of exchange	rad.s^{-1}
$\overline{\Omega}$	$\overline{\Omega} - \omega_{\text{rf}}$	The average resonance offset in the rotating frame	rad.s^{-1}
Ω_A	$\omega_A - \omega_{\text{rf}}$	The resonance offset in the rotating frame for state A	rad.s^{-1}
Ω_B	$\omega_B - \omega_{\text{rf}}$	The resonance offset in the rotating frame for state B	rad.s^{-1}
Ω_C	$\omega_C - \omega_{\text{rf}}$	The resonance offset in the rotating frame for state C	rad.s^{-1}
ω_A	-	The Larmor frequency of the spin in state A	rad.s^{-1}
ω_B	-	The Larmor frequency of the spin in state B	rad.s^{-1}
ω_C	-	The Larmor frequency of the spin in state C	rad.s^{-1}
ω_A^H	-	The proton Larmor frequency of the spin in state A (for MMQ data)	rad.s^{-1}
ω_B^H	-	The proton Larmor frequency of the spin in state B (for MMQ data)	rad.s^{-1}
ω_C^H	-	The proton Larmor frequency of the spin in state C (for MMQ data)	rad.s^{-1}
$\overline{\omega}$	$p_A\omega_A + p_B\omega_B$	The population averaged Larmor frequency of the spin	rad.s^{-1}
ω_1	-	Spin-lock field strength, i.e. the amplitude of the rf field	rad.s^{-1}
ω_e	$\sqrt{\overline{\Omega}^2 + \omega_1^2}$	Effective field in the rotating frame	rad.s^{-1}
ω_{rf}	-	Spin-lock offset, i.e. the frequency of the rf field	rad.s^{-1}
θ	$\arctan\left(\frac{\omega_1}{\overline{\Omega}}\right)$	Rotating frame tilt angle	rad
k_{AB}	$p_{Bk_{\text{ex}}}$	The forward exchange rate from state A to state B (2-site)	rad.s^{-1}
k_{BA}	$p_{Ak_{\text{ex}}}$	The reverse exchange rate from state B to state A (2-site)	rad.s^{-1}
k_{AB}	$p_{Bk_{AB}^{\text{ex}}}$	The forward exchange rate from state A to state B (3-site)	rad.s^{-1}
k_{BA}	$p_{Ak_{AB}^{\text{ex}}}$	The reverse exchange rate from state B to state A (3-site)	rad.s^{-1}
k_{BC}	$p_{Clk_{BC}^{\text{ex}}}$	The forward exchange rate from state B to state C (3-site)	rad.s^{-1}
k_{CB}	$p_{Bk_{BC}^{\text{ex}}}$	The reverse exchange rate from state C to state B (3-site)	rad.s^{-1}

Table 11.2: The parameters of relaxation dispersion.

Parameter	Equation	Description	Units
k_{AC}	$p_C k_{ex}^{AC}$	The forward exchange rate from state A to state C (3-site)	rad.s^{-1}
k_{CA}	$p_A k_{ex}^{AC}$	The reverse exchange rate from state C to state A (3-site)	rad.s^{-1}
k_{ex}	$1/\tau_{ex}$	Chemical exchange rate constant	rad.s^{-1}
k_{AB}^{ex}	$k_{AB} + k_{BA}$	Chemical exchange rate constant between sites A and B	rad.s^{-1}
k_{BC}^{ex}	$k_{BC} + k_{CB}$	Chemical exchange rate constant between sites B and C	rad.s^{-1}
k_{AC}^{ex}	$k_{AC} + k_{CA}$	Chemical exchange rate constant between sites A and C	rad.s^{-1}
k_B	$\approx k_{ex}^{AB}$	Approximate chemical exchange rate constant between sites A and B	rad.s^{-1}
k_C	$\approx k_{ex}^{AC}$	Approximate chemical exchange rate constant between sites A and C	rad.s^{-1}
τ_{ex}	$1/k_{ex}$	Time of exchange	s.rad^{-1}
p_A	-	Population of state A	-
p_B	$1 - p_A$	Population of state B (2-site)	-
p_B	$1 - p_A - p_C$	Population of state B (3-site)	-
p_C	$1 - p_A - p_B$	Population of state C (3-site)	-
$\Delta\omega$	$\omega_B - \omega_A$	Chemical shift difference between sites A and B (2-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{AB}$	$\omega_B - \omega_A$	Chemical shift difference between sites A and B (3-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{BC}$	$\omega_C - \omega_B$	Chemical shift difference between sites B and C (3-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{AC}$	$\Delta\omega_{AB} + \Delta\omega_{BC}$	Chemical shift difference between sites A and C (3-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_H^H$	$\omega_B^H - \omega_A^H$	Proton chemical shift difference between sites A and B (2-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{AB}^H$	$\omega_B^H - \omega_A^H$	Proton chemical shift difference between sites A and B (3-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{BC}^H$	$\omega_C^H - \omega_B^H$	Proton chemical shift difference between sites B and C (3-site)	rad.s^{-1} (stored as ppm)
$\Delta\omega_{AC}^H$	$\Delta\omega_{AB}^H + \Delta\omega_{BC}^H$	Proton chemical shift difference between sites A and C (3-site)	rad.s^{-1} (stored as ppm)
Φ_{ex}	$p_A p_B \Delta\omega^2$	Fast exchange factor	$\text{rad}^2 \cdot \text{s}^{-2}$ (stored as ppm ²)
$\Phi_{ex,B}$	See 11.20a on page 160	Fast exchange factor between sites A and B	$\text{rad}^2 \cdot \text{s}^{-2}$ (stored as ppm ²)
$\Phi_{ex,C}$	See 11.20b on page 160	Fast exchange factor between sites A and C	$\text{rad}^2 \cdot \text{s}^{-2}$ (stored as ppm ²)

11.2 The base dispersion models

11.2.1 The R_{2eff} model

This is the simplest of all models in that the dispersion component of the base data – the peak intensity values – is not modelled. It is used to determine either the R_{2eff} or R_{1ρ} values and errors as required for the base data for all other models. It can be selected by setting the model to ‘R2eff’. Depending on the experiment type, this model will be handled differently. The R_{2eff}/R_{1ρ} values determined can be later copied to the data pipes of the other dispersion models using the appropriate user functions.

Fixed relaxation period experiments

For the fixed relaxation time period CPMG-type experiments, the R_{2eff}/R_{1ρ} values are determined by direct calculation using the formula

$$R_{2\text{eff}}(\nu_{\text{CPMG}}) = -\frac{1}{T_{\text{relax}}} \cdot \ln \left(\frac{I_1(\nu_{\text{CPMG}})}{I_0} \right). \quad (11.2)$$

The values and errors are determined with a single call of the `minimise.calculate` user function. The R_{1ρ} version of the equation is essentially the same:

$$R_{1\rho}(\omega_1) = -\frac{1}{T_{\text{relax}}} \cdot \ln \left(\frac{I_1(\omega_1)}{I_0} \right). \quad (11.3)$$

Errors are calculated using the formula

$$\sigma_{R_2} = \frac{1}{T_{\text{relax}}} \sqrt{\left(\frac{\sigma_{I_1}}{I_1(\omega_1)} \right)^2 + \left(\frac{\sigma_{I_0}}{I_0} \right)^2}. \quad (11.4)$$

The derivation of this is simple enough. Rearranging 11.2,

$$R_2 \cdot T_{\text{relax}} = -\ln \left(\frac{I_1}{I_0} \right). \quad (11.5)$$

Using the rule

$$\ln \left(\frac{X}{Y} \right) = \ln(X) - \ln(Y), \quad (11.6)$$

where X and Y are normally distributed variables, then

$$R_2 \cdot T_{\text{relax}} = \ln(I_0) - \ln(I_1), \quad (11.7)$$

and

$$R_2 = -\frac{1}{T_{\text{relax}}} \cdot (\ln(I_0) - \ln(I_1)), \quad (11.8)$$

Using the estimate from https://en.wikipedia.org/wiki/Propagation_of_uncertainty that for

$$f = a \ln(A), \quad (11.9)$$

the variance of f is

$$\sigma_f^2 = \left(a * \frac{\sigma_A}{A} \right)^2, \quad (11.10)$$

then the R_2 variance is

$$\sigma_{R_2}^2 = \left(\frac{1}{T_{\text{relax}}} \cdot \frac{\sigma_{I_0}}{I_0} \right)^2 + \left(\frac{1}{T_{\text{relax}}} \cdot \frac{\sigma_{I_1}}{I_1} \right)^2. \quad (11.11)$$

Rearranging gives 11.4.

In a number of publications, the error formula from [Ishima and Torchia \(2005\)](#) has been used. This is the collapse of Equation 11.4 by setting σ_{I_0} to zero:

$$\sigma_{R_2} = \frac{\sigma_{I_1}}{T_{\text{relax}} I_1(\omega_1)}. \quad (11.12)$$

This is not implemented in relax as it can be shown by simple simulation that the formula is incorrect (see Figure 11.1). This formula significantly underestimates the real errors. The use of the same I_0 value for all dispersion points does not cause a decrease in the $R_{2\text{eff}}$ error but rather a correlation in the errors.

Variable relaxation period experiments

For the variable relaxation time period type experiments, the $R_{2\text{eff}}/R_{1\rho}$ values are determined by fitting to the simple two parameter exponential as in a R_1 or R_2 analysis. Both $R_{2\text{eff}}/R_{1\rho}$ and the initial peak intensity I_0 are optimised using the minimise user function for each exponential curve separately. Monte Carlo simulations are used to obtain the parameter errors.

Links

More information about the $R_{2\text{eff}}$ model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/R2eff>,
- the API documentation (for exponential curves) at http://www.nmr-relax.com/api/3.2/specification_specific_analyses.relax_disp.optimisation-module.html#back_calc_r2eff,
- the API documentation (for the 2-point fit) at http://www.nmr-relax.com/api/3.2/lib.dispersion.two_point-module.html#calc_two_point_r2eff,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#R2eff.

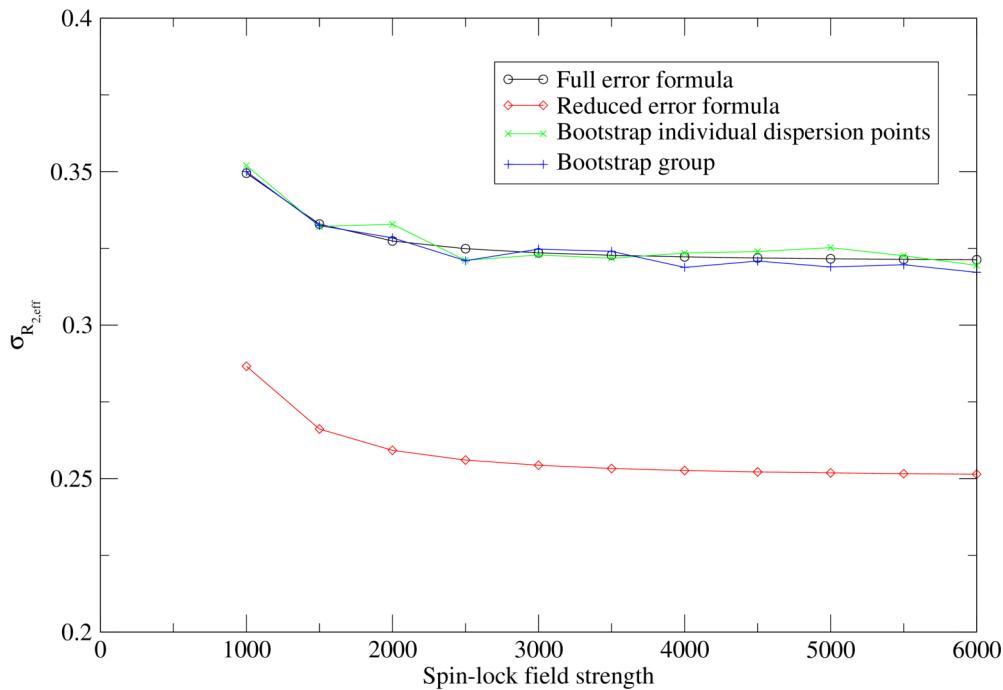


Figure 11.1: A demonstration of the inaccuracy of the error formula of Equation 11.12 from Ishima and Torchia (2005). This plot was generated using the script `test_suite/shared_data/dispersion/error_testing/simulation.py`. The bootstrapping simulation involves randomising noise-free I_0 and I_1 values for each dispersion data point assuming Gaussian errors. The full error formula is from Equation 11.4, the reduced error formula is from Equation 11.12, the bootstrapping using individual dispersion points estimates the errors assuming different I_0 randomisations for each dispersion point and each simulation, and the bootstrapping group graph uses the same randomised I_0 value for all dispersion points for each simulation.

11.2.2 The model for no chemical exchange relaxation

This model is provided for model selection purposes. In combination with frequentist methods, such as AIC, or Bayesian methods it can show if the presence of chemical exchange is statistically significant. Optimisation is still required as one R_2^0 value per magnetic field strength will be fit to the measured data for each spin system. If off-resonance CPMG or $R_{1\rho}$ data is being used, then one of the following equations will be fit:

$$R_{2\text{eff}} = R_1 \cos^2 \theta + R_2^0 \sin^2 \theta, \quad (11.13a)$$

$$R_{1\rho} = R_1 \cos^2 \theta + R'_{1\rho} \sin^2 \theta. \quad (11.13b)$$

The R_1 value can either be fixed to loaded experimental values or optimised. It is selected by setting the model to ‘No Rex’.

More information about the No Rex model is available from the:

- the relax wiki at http://wiki.nmr-relax.com/No_Rex,

- the API documentation at http://www.nmr-relax.com/api/3.2/target_functions.relax_disp.Dispersion-class.html#func_NOREX,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#No_Rex.

11.3 The analytic CPMG models

These are the analytic models designed for a single data type – the single quantum CPMG-type experiment.

11.3.1 The LM63 2-site fast exchange CPMG model

This is the original model for 2-site fast exchange for CPMG-type experiments. It is selected by setting the model to ‘LM63’, here named after [Luz and Meiboom \(1963\)](#). The original n -site Equation (7) from their paper can be written as

$$R_{\text{ex}} = \left[1 - 2\tau_{\text{ex}}g \cdot \tanh(2\tau_{\text{ex}}g)^{-1} \right] \cdot \tau_{\text{ex}} \cdot \sum_{i=2}^n p_i \Delta\omega_i, \quad (11.14)$$

where g is the pulse repetition rate defined as

$$g = 2\nu_{\text{CPMG}}. \quad (11.15)$$

It can be rearranged as

$$R_{\text{ex}} = \sum_{i=2}^n \frac{\Phi_{\text{ex},i}}{k_i} \cdot \left(1 - \frac{4\nu_{\text{CPMG}}}{k_i} \cdot \tanh\left(\frac{k_i}{4\nu_{\text{CPMG}}}\right) \right). \quad (11.16)$$

The equation for the 2-site exchange process can be expressed as

$$R_{2\text{eff}} = R_2^0 + \frac{\Phi_{\text{ex}}}{k_{\text{ex}}} \cdot \left(1 - \frac{4\nu_{\text{CPMG}}}{k_{\text{ex}}} \cdot \tanh\left(\frac{k_{\text{ex}}}{4\nu_{\text{CPMG}}}\right) \right). \quad (11.17)$$

The reference for this equation is:

- Luz, Z. and Meiboom, S. (1963). Nuclear magnetic resonance study of protolysis of trimethylammonium ion in aqueous solution - order of reaction with respect to solvent. *J. Chem. Phys.*, **39**(2), 366–370. ([10.1063/1.1734254](https://doi.org/10.1063/1.1734254))

More information about the LM63 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/LM63>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.lm63-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#LM63.

11.3.2 The LM63 3-site fast exchange CPMG model

This is the original [Luz and Meiboom \(1963\)](#) model for 3-site fast exchange for CPMG-type experiments. It is selected by setting the model to ‘LM63 3-site’. Taking the original Equation 11.16, the equation for 3-site exchange is simply:

$$R_{\text{ex}} = \sum_{i=2}^3 \frac{\Phi_{\text{ex},i}}{k_i} \cdot \left(1 - \frac{4\nu_{\text{CPMG}}}{k_i} \cdot \tanh \left(\frac{k_i}{4\nu_{\text{CPMG}}} \right) \right), \quad (11.18)$$

The reference for this equation is:

- Luz, Z. and Meiboom, S. (1963). Nuclear magnetic resonance study of protolysis of trimethylammonium ion in aqueous solution - order of reaction with respect to solvent. *J. Chem. Phys.*, **39**(2), 366–370. ([10.1063/1.1734254](https://doi.org/10.1063/1.1734254))

This model is only provided as a demonstration and should not be used for a normal analysis. Without data at multiple temperatures, a feature not currently supported within relax, that there are infinite lines of solutions and that the $\Phi_{\text{ex},B}$, $\Phi_{\text{ex},C}$, k_B and k_C parameters are all convoluted together.

This equation was made more practically relevant in the paper of [O’Connell et al. \(2009\)](#). This relies on the assumption that site 1 (or A) has a much larger equilibrium population than the other sites ($p_A \gg p_B$ and $p_A \gg p_C$). As stated, “if the different values of j_i are well-separated (by a factor of 3-10), then Eq. 3 reduces approximately to the sum of $n - 1$ independent two-state processes for exchange between site 1 and the $n - 1$ other sites”. In this situation, the following relationships hold

$$k_B \approx k_{\text{ex}}^{AB} = k_{AB} + k_{BA}, \quad (11.19a)$$

$$k_C \approx k_{\text{ex}}^{AC} = k_{AC} + k_{CA}, \quad (11.19b)$$

and

$$\Phi_{\text{ex},B} = \overline{\Phi_{\text{ex}}} \frac{(k_{\text{ex}}^{AB})^2 (\overline{k_{\text{ex}}} - k_{\text{ex}}^{AC})}{k_{\text{ex}}^2 (\overline{k_{\text{ex}}} - k_{\text{ex}}^{AC})}, \quad (11.20a)$$

$$\Phi_{\text{ex},C} = \overline{\Phi_{\text{ex}}} \frac{(k_{\text{ex}}^{AC})^2 (\overline{k_{\text{ex}}} - k_{\text{ex}}^{AB})}{k_{\text{ex}}^2 (\overline{k_{\text{ex}}} - k_{\text{ex}}^{AB})}, \quad (11.20b)$$

with

$$\overline{\Phi_{\text{ex},B}} \approx (p_A + p_C)p_B \Delta\omega_{AB}^2, \quad (11.21a)$$

$$\overline{\Phi_{\text{ex},C}} \approx (p_A + p_B)p_C \Delta\omega_{AC}^2. \quad (11.21b)$$

The parameter deconvolutions for this model can be performed after a relax analysis, if desired.

More information about the LM63 3-site model is available from:

- the relax wiki at http://wiki.nmr-relax.com/LM63_3-site,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.lm63_3site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#LM63_3-site.

11.3.3 The full CR72 2-site CPMG model

This is the model for 2-site exchange on most times scales (with the constraint that $p_A > p_B$), named after Carver and Richards (1972). It is selected by setting the model to ‘CR72 full’. The equation is

$$R_{2\text{eff}} = \frac{1}{2} \left(R_{2A}^0 + R_{2B}^0 + k_{\text{ex}} - 2\nu_{\text{CPMG}} \cosh^{-1} (D_+ \cosh(\eta_+) - D_- \cos(\eta_-)) \right), \quad (11.22)$$

where

$$D_{\pm} = \frac{1}{2} \left(\pm 1 + \frac{\Psi + 2\Delta\omega^2}{\sqrt{\Psi^2 + \zeta^2}} \right), \quad (11.23)$$

$$\eta_{\pm} = 2^{\frac{-3}{2}} \frac{1}{\nu_{\text{CPMG}}} \left(\pm \Psi + \sqrt{\Psi^2 + \zeta^2} \right)^{\frac{1}{2}}, \quad (11.24)$$

$$\Psi = (R_{2A}^0 - R_{2B}^0 - p_A k_{\text{ex}} + p_B k_{\text{ex}})^2 - \Delta\omega^2 + 4p_A p_B k_{\text{ex}}^2, \quad (11.25)$$

$$\zeta = 2\Delta\omega (R_{2A}^0 - R_{2B}^0 - p_A k_{\text{ex}} + p_B k_{\text{ex}}). \quad (11.26)$$

Note that these equations use the numerically simplified form derived in the appendix of Davis et al. (1994).

This model is not accurate when the motional process is very slow. In that case, the ‘TSMFK01’ model in Section 11.3.6 on page 163 should be used instead.

The reference for this equation is:

- Carver, J. P. and Richards, R. E. (1972). General 2-site solution for chemical exchange produced dependence of T2 upon Carr-Purcell pulse separation. *J. Magn. Reson.*, **6**(1), 89–105. ([10.1016/0022-2364\(72\)90090-X](https://doi.org/10.1016/0022-2364(72)90090-X))

More information about the CR72 full model is available from:

- the relax wiki at http://wiki.nmr-relax.com/CR72_full,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.cr72-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#CR72_full.

11.3.4 The reduced CR72 2-site CPMG model

This is the model for 2-site exchange on most times scales (with the constraint that $p_A > p_B$), named after [Carver and Richards \(1972\)](#). It is selected by setting the model to ‘CR72’. It is the same as the full CR72 model described above, but with the simplification that $R_{2A}^0 = R_{2B}^0$. This simplifies the equations to

$$R_{2\text{eff}} = R_2^0 + \frac{k_{\text{ex}}}{2} - \nu_{\text{CPMG}} \cosh^{-1} (D_+ \cosh(\eta_+) - D_- \cos(\eta_-)), \quad (11.27)$$

where D_{\pm} and η_{\pm} are unchanged and

$$\Psi = k_{\text{ex}}^2 - \Delta\omega^2, \quad (11.28)$$

$$\zeta = -2\Delta\omega(p_A k_{\text{ex}} - p_B k_{\text{ex}}). \quad (11.29)$$

As mentioned in the ‘CR72 full’ model section, this model is not accurate when the motional process is very slow. In that case please use the ‘TSMFK01’ model in Section [11.3.6](#) on page [163](#) instead.

More information about the CR72 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/CR72>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.cr72-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#CR72.

11.3.5 The IT99 2-site CPMG model

This is the model for 2-site exchange on all times scales (with the constraint that $p_A \gg p_B$), named after [Ishima and Torchia \(1999\)](#). It is selected by setting the model to ‘IT99’. The equation is:

$$R_{\text{ex}} \simeq \frac{\Phi_{\text{ex}} \tau_{\text{ex}}}{1 + \omega_a^2 \tau_{\text{ex}}^2}, \quad (11.30)$$

$$\omega_a^2 = \sqrt{\omega_{\text{1eff}}^4 + p_A^2 \Delta\omega^4}, \quad (11.31)$$

$$R_{2\text{eff}} = R_2^0 + R_{\text{ex}}. \quad (11.32)$$

The effective rotating frame field for a CPMG-type experiment is given by

$$\omega_{\text{1eff}} = 4\sqrt{3}\nu_{\text{CPMG}}, \quad (11.33)$$

and hence

$$\omega_{\text{1eff}}^4 = 2304\nu_{\text{CPMG}}^4. \quad (11.34)$$

The reference for this equation is:

- Ishima, R. and Torchia, D. A. (1999). Estimating the time scale of chemical exchange of proteins from measurements of transverse relaxation rates in solution. *J. Biomol. NMR*, **14**(4), 369–372. ([10.1023/A:1008324025406](https://doi.org/10.1023/A:1008324025406))

More information about the IT99 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/IT99>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.it99-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#IT99.

11.3.6 The TSMFK01 2-site CPMG model

This is the model for 2-site very-slow exchange model for time scales within range of microsecond to second time scale, where $p_A \gg p_B$, and named after Tollinger et al. (2001). It is selected by setting the model to ‘TSMFK01’. A particularly interesting feature of the dispersion curves is the damped oscillations, which occur at low CPMG field strengths, and is solely a function of the chemical shift difference between the two sites (i.e. independent of the rate of exchange).

The equation is:

$$R_{2\text{eff}} = R_{2A}^0 + k_{AB} - k_{AB} \frac{\sin(\Delta\omega \cdot \tau_{\text{CPMG}})}{\Delta\omega \cdot \tau_{\text{CPMG}}} \quad (11.35)$$

The reference for this equation is:

- Tollinger, M., Skrynnikov, N. R., Mulder, F. A. A., Forman-Kay, J. D., and Kay, L. E. (2001). Slow dynamics in folded and unfolded states of an sh3 domain. *J. Am. Chem. Soc.*, **123**(46), 11341–11352. ([10.1021/ja011300z](https://doi.org/10.1021/ja011300z))

More information about the TSMFK01 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/TSMFK01>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.tsmfk01-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#TSMFK01.

11.3.7 The full B14 2-site CPMG model

This is the model for 2-site exchange exact analytical derivation on all time scales (with the constraint that $p_A > p_B$), named after [Baldwin \(2014\)](#). It is selected by setting the model to ‘B14 full’. The equation is

$$\begin{aligned} R_{2\text{eff}}^0 &= \frac{R_{2A}^0 + R_{2B}^0 + k_{\text{ex}}}{2} - \frac{N_{\text{cyc}}}{T_{\text{relax}}} \cosh^{-1}(\nu_{1c}) \\ &\quad - \frac{1}{T_{\text{relax}}} \ln \left(\frac{1+y}{2} + \frac{1-y}{2\sqrt{\nu_{1c}^2 - 1}} (\nu_2 + 2k_{AB}p_D) \right), \end{aligned} \quad (11.36a)$$

$$= R_{2\text{eff}}^{\text{CR72}} - \frac{1}{T_{\text{relax}}} \ln \left(\frac{1+y}{2} + \frac{1-y}{2\sqrt{\nu_{1c}^2 - 1}} (\nu_2 + 2k_{AB}p_D) \right), \quad (11.36b)$$

where Appendix 1 in [Baldwin \(2014\)](#) lists the recipe for the exact calculation of $R_{2\text{eff}}$. Note that the following definitions are different to those in the original publication, but match both the reference implementation and the relax implementation. The definitions are functionally equivalent. First establish the complex free precession eigenfrequency with

$$\Delta R_2^0 = R_{2A}^0 - R_{2B}^0, \quad (11.37a)$$

$$\alpha_- = \Delta R_2^0 + k_{AB} - k_{BA}, \quad (11.37b)$$

$$\zeta = 2\Delta\omega\alpha_-, \quad (11.37c)$$

$$\Psi = \alpha_-^2 + 4k_{AB}k_{BA} - \Delta\omega^2, \quad (11.37d)$$

$$h_3 = \frac{1}{\sqrt{2}} \sqrt{\Psi + \sqrt{\zeta^2 + \Psi^2}}, \quad (11.37e)$$

$$h_4 = \frac{1}{\sqrt{2}} \sqrt{-\Psi + \sqrt{\zeta^2 + \Psi^2}}. \quad (11.37f)$$

The ground state ensemble evolution frequency f_{00} expressed in separated real and imaginary components in terms of definitions ζ , Ψ , and h_4 is

$$f_{00} = \frac{1}{2} (R_{2A}^0 + R_{2B}^0 + k_{\text{ex}}) + \frac{i}{2} (\Delta\omega - h_4). \quad (11.38)$$

Define substitutions for ‘stay’ and ‘swap’ factors as

$$N = h_3 + ih_4, \quad (11.39a)$$

$$NN^* = h_3^2 + h_4^2, \quad (11.39b)$$

$$F_0 = (\Delta\omega^2 + h_3^2) / NN^*, \quad (11.39c)$$

$$F_2 = (\Delta\omega^2 - h_4^2) / NN^*, \quad (11.39d)$$

$$F_1^b = (\Delta\omega + h_4) (\Delta\omega - ih_3) / NN^*, \quad (11.39e)$$

$$F_1^{a+b} = (2\Delta\omega^2 + i\zeta) / NN^*. \quad (11.39f)$$

The weighting factors for frequencies E_{0-2} emerging from a single CPMG block, F_{0-2} , are

$$E_0 = 2\tau_{\text{CPMG}} \cdot h_3, \quad (11.40a)$$

$$E_2 = 2\tau_{\text{CPMG}} \cdot h_4, \quad (11.40b)$$

$$E_1 = (h_3 - ih_4) \cdot \tau_{\text{CPMG}}. \quad (11.40c)$$

Here $\tau_{\text{CPMG}} = 1/4\nu_{\text{CPMG}}$. The final result, with identities to assist efficient matrix exponentiation optimised for numerical calculation, is

$$\nu_{1c} = F_0 \cosh(E_0) - F_2 \cos(E_2), \quad (11.41\text{a})$$

$$\nu_{1s} = F_0 \sinh(E_0) - \imath F_2 \sin(E_2), \quad (11.41\text{b})$$

$$\nu_3 = \sqrt{\nu_{1c}^2 - 1}, \quad (11.41\text{c})$$

$$\nu_4 = F_1^b(-\alpha_- - h_3) + \imath F_1^b(\Delta\omega - h_4), \quad (11.41\text{d})$$

$$\nu_5 = (-\Delta R_2^0 + k_{\text{ex}} + \imath\Delta\omega) \nu_{1s} + 2(\nu_4 + k_{AB}F_1^{a+b}) \sinh(E_1), \quad (11.41\text{e})$$

$$y = \left(\frac{\nu_{1c} - \nu_3}{\nu_{1c} + \nu_3} \right)^{N_{\text{cyc}}}, \quad (11.41\text{f})$$

$$T = \frac{1+y}{2} + \frac{1-y}{2} \frac{\nu_5}{\nu_3 N}, \quad (11.41\text{g})$$

$$R_{2\text{eff}}^{\text{CR72}} = \frac{R_{2A}^0 + R_{2B}^0 + k_{\text{ex}}}{2} - \frac{N_{\text{cyc}}}{T_{\text{relax}}} \text{arccosh}(\Re(\nu_{1c})), \quad (11.41\text{h})$$

$$R_{2\text{eff}} = R_{2\text{eff}}^{\text{CR72}} - \frac{1}{T_{\text{relax}}} \log(\Re(T)). \quad (11.41\text{i})$$

The advantage of these equations is that you will always obtain the correct answer provided you have 2-site exchange, in-phase magnetisation and on-resonance pulses.

The term p_D is based on product of the off diagonal elements in the CPMG propagator, see supplementary Section 3 ([Baldwin, 2014](#)).

It is interesting to consider the region of validity of the Carver and Richards result. The two results are equal when the correction is zero, which is true when

$$\sqrt{\nu_{1c}^2 - 1} \approx \nu_2 + 2k_{AB}p_D. \quad (11.42)$$

This occurs when $k_{AB}p_D$ tends to zero, and so $\nu_2 = \nu_3$. Setting $k_{AB}p_D$ to zero amounts to neglecting magnetisation that starts on the ground state ensemble and end on the excited state ensemble and vice versa. This will be a good approximation when $p_A \gg p_B$. In practise, significant deviations from the Carver and Richards equation can be incurred if $p_B > 1\%$. Incorporation of the correction term results in an improved description of the CPMG experiment over [Carver and Richards \(1972\)](#).

The reference for this equation is:

- Baldwin, A. J. (2014). An exact solution for $R_{2,\text{eff}}$ in CPMG experiments in the case of two site chemical exchange. *J. Magn. Reson.*, **244**, 114–124. ([10.1016/j.jmr.2014.02.023](https://doi.org/10.1016/j.jmr.2014.02.023))

More information about the B14 full model is available from:

- the relax wiki at http://wiki.nmr-relax.com/B14_full,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.b14-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#B14_full.

11.3.8 The reduced B14 2-site CPMG model

This is the model for 2-site exchange exact analytical derivation on all time scales (with the constraint that $p_A > p_B$), named after [Baldwin \(2014\)](#). It is selected by setting the model to ‘B14’. It is the same as the full B14 model described above, but with the simplification that $R_{2A}^0 = R_{2B}^0$.

More information about the B14 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/B14>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.b14-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#B14.

11.4 The numeric CPMG models

These are the numeric models designed for a single data type – the single quantum CPMG-type experiment.

11.4.1 The NS 2-site expanded CPMG model

This is the numerical model for 2-site exchange expanded using Maple by Nikolai Skrynnikov. It is selected by setting the model to ‘NS CPMG 2-site expanded’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

This model will give the same results as the other numerical solutions whereby $R_{2A}^0 = R_{2B}^0$. The following is the set of equations of the expansion used in relax. It has been modified from the original for speed. See the `lib.dispersion.ns_cpmg_2site_expanded` module for more details including the original code. Further simplifications can be found in the code.

$$t_3 = \iota, \tag{11.43.1}$$

$$t_4 = t_3\Delta\omega, \tag{11.43.2}$$

$$t_5 = k_{BA}^2, \tag{11.43.3}$$

$$t_8 = 2t_4k_{BA}, \tag{11.43.4}$$

$$t_{10} = 2k_{BA}k_{AB}, \tag{11.43.5}$$

$$t_{11} = \Delta\omega^2, \tag{11.43.6}$$

$$t_{14} = 2t_4k_{AB}, \tag{11.43.7}$$

$$t_{15} = k_{AB}^2, \tag{11.43.8}$$

$$t_{17} = \sqrt{t_5 - t_8 + t_{10} - t_{11} + t_{14} + t_{15}}, \tag{11.43.9}$$

$$t_{21} = \exp\left(\frac{(-k_{BA} + t_4 - k_{AB} + t_{17})\tau_{CPMG}}{2}\right), \quad (11.43.10)$$

$$t_{22} = \frac{1}{t_{17}}, \quad (11.43.11)$$

$$t_{28} = \exp\left(\frac{(-k_{BA} + t_4 - k_{AB} - t_{17})\tau_{CPMG}}{2}\right), \quad (11.43.12)$$

$$t_{31} = t_{22}k_{AB}(t_{21} - t_{28}), \quad (11.43.13)$$

$$t_{33} = \sqrt{t_5 + t_8 + t_{10} - t_{11} - t_{14} + t_{15}}, \quad (11.43.14)$$

$$t_{34} = k_{BA} + t_4 - k_{AB} + t_{33}, \quad (11.43.15)$$

$$t_{37} = \exp((-k_{BA} - t_4 - k_{AB} + t_{33})\tau_{CPMG}), \quad (11.43.16)$$

$$t_{39} = \frac{1}{t_{33}}, \quad (11.43.17)$$

$$t_{41} = k_{BA} + t_4 - k_{AB} - t_{33}, \quad (11.43.18)$$

$$t_{44} = \exp((-k_{BA} - t_4 - k_{AB} - t_{33})\tau_{CPMG}), \quad (11.43.19)$$

$$t_{47} = \frac{t_{39}}{2}(t_{34}t_{37} - t_{41}t_{44}), \quad (11.43.20)$$

$$t_{49} = k_{BA} - t_4 - k_{AB} - t_{17}, \quad (11.43.21)$$

$$t_{51} = t_{21}t_{49}t_{22}, \quad (11.43.22)$$

$$t_{52} = k_{BA} - t_4 - k_{AB} + t_{17}, \quad (11.43.23)$$

$$t_{54} = t_{28}t_{52}t_{22}, \quad (11.43.24)$$

$$t_{55} = t_{54} - t_{51}, \quad (11.43.25)$$

$$t_{60} = \frac{1}{2}t_{39}k_{AB}(t_{37} - t_{44}), \quad (11.43.26)$$

$$t_{62} = t_{31}t_{47} + t_{55}t_{60}, \quad (11.43.27)$$

$$t_{63} = \frac{1}{k_{AB}}, \quad (11.43.28)$$

$$t_{68} = \frac{t_{63}}{2}(t_{49}t_{54} - t_{52}t_{51}), \quad (11.43.29)$$

$$t_{69} = \frac{t_{62}t_{68}}{2}, \quad (11.43.30)$$

$$t_{72} = t_{37}t_{41}t_{39}, \quad (11.43.31)$$

$$t_{76} = t_{44}t_{34}t_{39}, \quad (11.43.32)$$

$$t_{78} = \frac{t_{63}}{2}(t_{41}t_{76} - t_{34}t_{72}), \quad (11.43.33)$$

$$t_{80} = \frac{1}{2}(t_{76} - t_{72}), \quad (11.43.34)$$

$$t_{82} = \frac{1}{2}(t_{31}t_{78} + t_{55}t_{80}), \quad (11.43.35)$$

$$t_{83} = \frac{t_{82}t_{55}}{2}, \quad (11.43.36)$$

$$t_{88} = \frac{t_{22}}{2}(t_{52}t_{21} - t_{49}t_{28}), \quad (11.43.37)$$

$$t_{91} = t_{88}t_{47} + t_{68}t_{60}, \quad (11.43.38)$$

$$t_{92} = t_{91}t_{88}, \quad (11.43.39)$$

$$t_{95} = \frac{1}{2}(t_{88}t_{78} + t_{68}t_{80}), \quad (11.43.40)$$

$$t_{96} = t_{95}t_{31}, \quad (11.43.41)$$

$$t_{97} = t_{69} + t_{83}, \quad (11.43.42)$$

$$t_{98} = t_{97}^2, \quad (11.43.43)$$

$$t_{99} = t_{92} + t_{96}, \quad (11.43.44)$$

$$t_{102} = t_{99}^2, \quad (11.43.45)$$

$$t_{108} = t_{62}t_{88} + t_{82}t_{31}, \quad (11.43.46)$$

$$t_{112} = \sqrt{t_{98} - 2t_{99}t_{97} + t_{102} + 2(t_{91}t_{68} + t_{95}t_{55})t_{108}}, \quad (11.43.47)$$

$$t_{113} = t_{97} - t_{99} - t_{112}, \quad (11.43.48)$$

$$t_{115} = n_{\text{CPMG}}, \quad (11.43.49)$$

$$t_{116} = \left(\frac{t_{97} + t_{99} + t_{112}}{2} \right)^{t_{115}}, \quad (11.43.50)$$

$$t_{118} = \frac{1}{t_{112}}, \quad (11.43.51)$$

$$t_{120} = t_{97} - t_{99} + t_{112}, \quad (11.43.52)$$

$$t_{122} = \left(\frac{t_{97} + t_{99} - t_{112}}{2} \right)^{t_{115}}, \quad (11.43.53)$$

$$t_{127} = \frac{1}{2t_{108}}, \quad (11.43.54)$$

$$t_{139} = \frac{1}{2(k_{AB} + k_{BA})} \left[(t_{120}t_{122} - t_{113}t_{116})t_{118}k_{BA} + (t_{120}t_{122} - t_{116}t_{120})t_{113}t_{118}t_{127}k_{AB} \right]. \quad (11.43.55)$$

The relative peak intensities, magnetisation, and effective R₂ relaxation rate are calculated as

$$I_0 = p_A, \quad (11.44a)$$

$$I_1 = \Re(t_{139}) \exp(-T_{\text{relax}}R_2^0), \quad (11.44b)$$

$$M_x = I_1/I_0, \quad (11.44c)$$

$$R_{2\text{eff}} = -\frac{1}{T_{\text{relax}}} \cdot \ln(M_x). \quad (11.44d)$$

In these equations τ_{CPMG} and n_{CPMG} are numpy arrays and hence t_{139} is also a numpy array. This avoids a Python loop over the dispersion points until the very end of the calculation, required to populate the $R_{2\text{eff}}$ data structure, resulting in very fast calculations.

The reference for this model is:

- Tollinger, M., Skrynnikov, N. R., Mulder, F. A. A., Forman-Kay, J. D., and Kay, L. E. (2001). Slow dynamics in folded and unfolded states of an sh3 domain. *J. Am. Chem. Soc.*, **123**(46), 11341–11352. ([10.1021/ja011300z](https://doi.org/10.1021/ja011300z))

More information about the NS CPMG 2-site expanded model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_CPMG_2-site_expanded,

- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_cpmg_2site_expanded-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_CPMG_2-site_expanded.

11.4.2 The full NS 2-site 3D CPMG model

This is the numerical model for 2-site exchange using 3D magnetisation vectors. It is selected by setting the model to ‘NS CPMG 2-site 3D full’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

More information about the NS CPMG 2-site 3D full model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_CPMG_2-site_3D_full,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_cpmg_2site_3d-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_CPMG_2-site_3D_full.

11.4.3 The reduced NS 2-site 3D CPMG model

This is the numerical model for 2-site exchange using 3D magnetisation vectors, whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It is selected by setting the model to ‘NS CPMG 2-site 3D’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

More information about the NS CPMG 2-site 3D model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_CPMG_2-site_3D,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_cpmg_2site_3d-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_CPMG_2-site_3D.

11.4.4 The full NS 2-site star CPMG model

This is the numerical model for 2-site exchange using complex conjugate matrices. It is selected by setting the model to ‘NS CPMG 2-site star full’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

More information about the NS CPMG 2-site star full model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_CPMG_2-site_star_full,

- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_cpmg_2site_star-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_CPMG_2-site_star_full.

11.4.5 The reduced NS 2-site star CPMG model

This is the numerical model for 2-site exchange using complex conjugate matrices, whereby the simplification $R_{2A}^0 = R_{2B}^0$ is assumed. It is selected by setting the model to ‘NS CPMG 2-site star’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

More information about the NS CPMG 2-site star model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_CPMG_2-site_star,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_cpmg_2site_star-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_CPMG_2-site_star.

11.5 The analytic MMQ CPMG models

These are the analytic models designed for handling combined proton-heteronuclear SQ, ZQ, DQ, and MQ CPMG-type experiments. This data combination is labelled as multiple multiple quantum data or MMQ.

11.5.1 The MMQ CR72 model

This is the analytic CR72 model for 2-site exchange on most times scales (Section 11.3.4 on page 162) extended for multiple types of multiple quantum data (MMQ) by Korzhnev et al. (2004a). It is selected by setting the model to ‘MMQ CR72’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces. The equation for the exchange process is

$$R_{\text{eff}} = \Re(\lambda_1) - \frac{1}{T_{\text{relax}}} \ln(Q), \quad (11.45)$$

where

$$\lambda_1 = R_{2,\text{MQ}}^0 + \frac{k_{\text{ex}}}{2} - \nu_{\text{CPMG}} \cosh^{-1} (D_+ \cosh(\eta_+) - D_- \cos(\eta_-)), \quad (11.46)$$

$$D_{\pm} = \frac{1}{2} \left(\pm 1 + \frac{\Psi + 2\Delta\omega^2}{\sqrt{\Psi^2 + \zeta^2}} \right), \quad (11.47)$$

$$\eta_{\pm} = 2^{\frac{2}{3}} \frac{1}{\nu_{\text{CPMG}}} \left(\pm \Psi + \sqrt{\Psi^2 + \zeta^2} \right)^{\frac{1}{2}}, \quad (11.48)$$

$$\Psi = (\imath\Delta\omega^H + p_A k_{\text{ex}} - p_B k_{\text{ex}})^2 - \Delta\omega^2 + 4p_A p_B k_{\text{ex}}^2, \quad (11.49)$$

$$\zeta = -2\Delta\omega (\imath\Delta\omega^H + p_A k_{\text{ex}} - p_B k_{\text{ex}}), \quad (11.50)$$

and where

$$Q = \Re \left(1 - m_{D+}^2 + m_{D+}m_{Z+} - m_{Z+}^2 + \frac{m_{D+} + m_{Z+}}{2} \sqrt{\frac{p_B}{p_A}} \right), \quad (11.51)$$

and

$$m_{D\pm} = \pm \frac{\imath k_{\text{ex}} \sqrt{p_A p_B}}{d_{\pm} z_{\pm}} \left(z_{\pm} + 2\Delta\omega \frac{\sin(z_{\pm}\delta)}{\sin((d_{\pm} + z_{\pm})\delta)} \right), \quad (11.52)$$

$$m_{Z\mp} = \pm \frac{\imath k_{\text{ex}} \sqrt{p_A p_B}}{d_{\pm} z_{\pm}} \left(d_{\pm} - 2\Delta\omega \frac{\sin(d_{\pm}\delta)}{\sin((d_{\pm} + z_{\pm})\delta)} \right), \quad (11.53)$$

and

$$d_{\pm} = (\Delta\omega^H + \Delta\omega) \pm \imath k_{\text{ex}}, \quad (11.54)$$

$$z_{\pm} = (\Delta\omega^H - \Delta\omega) \pm \imath k_{\text{ex}}. \quad (11.55)$$

The symbol δ is half of τ_{CPMG} or

$$\delta = \frac{1}{4\nu_{\text{CPMG}}}. \quad (11.56)$$

The references for this model are:

- Korzhnev, D. M., Kloiber, K., Kanelis, V., Tugarinov, V., and Kay, L. E. (2004a). Probing slow dynamics in high molecular weight proteins by methyl-TROSY NMR spectroscopy: application to a 723-residue enzyme. *J. Am. Chem. Soc.*, **126**(12), 3964–3973. ([10.1021/ja039587i](https://doi.org/10.1021/ja039587i))
- Korzhnev, D. M., Kloiber, K., and Kay, L. E. (2004b). Multiple-quantum relaxation dispersion NMR spectroscopy probing millisecond time-scale dynamics in proteins: theory and application. *J. Am. Chem. Soc.*, **126**(23), 7320–7329. ([10.1021/ja049968b](https://doi.org/10.1021/ja049968b))
- Korzhnev, D. M., Neudecker, P., Mittermaier, A., Orekhov, V. Y., and Kay, L. E. (2005a). Multiple-site exchange in proteins studied with a suite of six NMR relaxation dispersion experiments: an application to the folding of a Fyn SH3 domain mutant. *J. Am. Chem. Soc.*, **127**(44), 15602–15611. ([10.1021/ja054550e](https://doi.org/10.1021/ja054550e))

More information about the MMQ CR72 model is available from:

- the relax wiki at http://wiki.nmr-relax.com/MMQ_CR72,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.mmq_cr72-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#MMQ_CR72.

11.6 The numeric MMQ CPMG models

These are the numeric models designed for handling combined proton-heteronuclear SQ, ZQ, DQ, and MQ CPMG-type experiments. This data combination is labelled as multiple multiple quantum data or MMQ.

11.6.1 The NS MMQ 2-site model

This is the numerical model for 2-site exchange for proton-heteronuclear SQ, ZQ, DQ and MQ CPMG data, as derived in (Korzhnev et al., 2004a,b, 2005a). It is selected by setting the model to ‘NS MMQ 2-site’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces. Different sets of equations are used for the different data types.

The SQ, ZD and DQ equations

The basic evolution matrices for single, zero and double quantum CPMG-type data for this model are

$$R_{2\text{eff}} = -\frac{1}{T_{\text{relax}}} \log \frac{\mathbf{M}_A(T_{\text{relax}})}{\mathbf{M}_A(0)}, \quad (11.57)$$

where $\mathbf{M}_A(0)$ is proportional to the vector $[p_A, p_B]^T$ and

$$\mathbf{M}_A(T_{\text{relax}}) = (\mathbf{A}_{\pm} \mathbf{A}_{\mp} \mathbf{A}_{\mp} \mathbf{A}_{\pm})^n \mathbf{M}_A(0) \quad (11.58)$$

The evolution matrix \mathbf{A} is defined as

$$\mathbf{A}_{\pm} = e^{\mathbf{a}_{\pm} \cdot \tau_{\text{CPMG}}}, \quad (11.59)$$

where

$$\mathbf{a}_{\pm} = \begin{pmatrix} -k_{AB} - R_{2A}^0 & k_{BA} \\ k_{AB} & -k_{BA} \pm i\Delta\omega - R_{2B}^0 \end{pmatrix}. \quad (11.60)$$

For different data types $\Delta\omega$ is defined as: $\Delta\omega$ (^{15}N SQ-type data); $\Delta\omega^H$ (^1H SQ-type data); $\Delta\omega^H - \Delta\omega$ (ZQ-type data); and $\Delta\omega^H + \Delta\omega$ (DQ-type data).

The MQ equations

The equation for the exchange process for multiple quantum CPMG-type data is

$$R_{\text{eff}} = -\frac{1}{T} \log \left\{ \operatorname{Re} \left[\frac{0.5}{p_A} (1 \ 0) \cdot (\mathbf{AB} + \mathbf{CD}) \cdot \begin{pmatrix} p_A \\ p_B \end{pmatrix} \right] \right\}, \quad (11.61)$$

where T is the constant time interval, and the matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} are dually defined. When n is even, they are defined as

$$\mathbf{A} = (\mathbf{M}_1 \mathbf{M}_2 \mathbf{M}_2 \mathbf{M}_1)^{\frac{n}{2}}, \quad (11.62a)$$

$$\mathbf{B} = (\mathbf{M}_2^* \mathbf{M}_1^* \mathbf{M}_1^* \mathbf{M}_2^*)^{\frac{n}{2}}, \quad (11.62b)$$

$$\mathbf{C} = (\mathbf{M}_2 \mathbf{M}_1 \mathbf{M}_1 \mathbf{M}_2)^{\frac{n}{2}}, \quad (11.62c)$$

$$\mathbf{D} = (\mathbf{M}_1^* \mathbf{M}_2^* \mathbf{M}_2^* \mathbf{M}_1^*)^{\frac{n}{2}}. \quad (11.62d)$$

When n is odd, they are defined as

$$\mathbf{A} = (\mathbf{M}_1 \mathbf{M}_2 \mathbf{M}_2 \mathbf{M}_1)^{\frac{n-1}{2}} \mathbf{M}_1 \mathbf{M}_2, \quad (11.63a)$$

$$\mathbf{B} = (\mathbf{M}_1^* \mathbf{M}_2^* \mathbf{M}_2^* \mathbf{M}_1^*)^{\frac{n-1}{2}} \mathbf{M}_1^* \mathbf{M}_2^*, \quad (11.63b)$$

$$\mathbf{C} = (\mathbf{M}_2 \mathbf{M}_1 \mathbf{M}_1 \mathbf{M}_2)^{\frac{n-1}{2}} \mathbf{M}_2 \mathbf{M}_1, \quad (11.63c)$$

$$\mathbf{D} = (\mathbf{M}_2^* \mathbf{M}_1^* \mathbf{M}_1^* \mathbf{M}_2^*)^{\frac{n-1}{2}} \mathbf{M}_2^* \mathbf{M}_1^*. \quad (11.63d)$$

When n is zero, to avoid matrix powers of zero they are defined as

$$\mathbf{A} = \mathbf{M}_1 \mathbf{M}_2, \quad (11.64a)$$

$$\mathbf{B} = \mathbf{M}_1^* \mathbf{M}_2^*, \quad (11.64b)$$

$$\mathbf{C} = \mathbf{M}_2 \mathbf{M}_1, \quad (11.64c)$$

$$\mathbf{D} = \mathbf{M}_2^* \mathbf{M}_1^*. \quad (11.64d)$$

The \mathbf{M} matrices are defined as:

$$\mathbf{M}_j = \exp(\mathbf{m}_j \delta), \quad (11.65)$$

where 2δ is the spacing between successive 180° pulses and where The references for this model are:

$$\mathbf{m}_1 = \begin{pmatrix} -p_B k_{\text{ex}} - R_{2,\text{DQ}}^A & p_A k_{\text{ex}} \\ p_B k_{\text{ex}} & -p_A k_{\text{ex}} - i(\Delta\omega^H + \Delta\omega) - R_{2,\text{DQ}}^B \end{pmatrix}, \quad (11.66a)$$

$$\mathbf{m}_2 = \begin{pmatrix} -p_B k_{\text{ex}} - R_{2,\text{ZQ}}^A & p_A k_{\text{ex}} \\ p_B k_{\text{ex}} & -p_A k_{\text{ex}} - i(\Delta\omega^H - \Delta\omega) - R_{2,\text{ZQ}}^B \end{pmatrix}. \quad (11.66b)$$

For the model it is assumed that $R_{2,\text{DQ}}^A = R_{2,\text{ZQ}}^A = R_{2A}^0$ and $R_{2,\text{DQ}}^B = R_{2,\text{ZQ}}^B = R_{2B}^0$. The references for this model are:

- Korzhnev, D. M., Kloiber, K., Kanelis, V., Tugarinov, V., and Kay, L. E. (2004a). Probing slow dynamics in high molecular weight proteins by methyl-TROSY NMR spectroscopy: application to a 723-residue enzyme. *J. Am. Chem. Soc.*, **126**(12), 3964–3973. ([10.1021/ja039587i](https://doi.org/10.1021/ja039587i))
- Korzhnev, D. M., Kloiber, K., and Kay, L. E. (2004b). Multiple-quantum relaxation dispersion NMR spectroscopy probing millisecond time-scale dynamics in proteins: theory and application. *J. Am. Chem. Soc.*, **126**(23), 7320–7329. ([10.1021/ja049968b](https://doi.org/10.1021/ja049968b))
- Korzhnev, D. M., Neudecker, P., Mittermaier, A., Orekhov, V. Y., and Kay, L. E. (2005a). Multiple-site exchange in proteins studied with a suite of six NMR relaxation dispersion experiments: an application to the folding of a Fyn SH3 domain mutant. *J. Am. Chem. Soc.*, **127**(44), 15602–15611. ([10.1021/ja054550e](https://doi.org/10.1021/ja054550e))

More information about the NS MMQ 2-site model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_MMQ_2-site,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_mmq_2site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_MMQ_2-site.

11.6.2 The NS MMQ 3-site linear model

This is the numerical model for 3-site exchange for proton-heteronuclear SQ, ZQ, DQ and MQ CPMG data, as derived in (Korzhnev et al., 2004a,b, 2005a). As this model is linear, the assumption that $k_{AC} = k_{CA} = 0$ has been made. To simplify the optimisation space for the model, the assumption $R_{2A}^0 = R_{2B}^0 = R_{2C}^0 = R_2^0$ has also been made.

The SQ, ZD and DQ equations

The basic evolution matrices for single, zero and double quantum CPMG-type data for this model are

$$\mathbf{A}_\pm = e^{\mathbf{a}_\pm \cdot \tau_{\text{CPMG}}}, \quad (11.67)$$

where

$$\begin{aligned} \mathbf{a}_\pm = & \begin{pmatrix} -k_{AB} & k_{BA} & 0 \\ k_{AB} & -k_{BA} - k_{BC} \pm i\Delta\omega_{AB} & k_{CB} \\ 0 & k_{BC} & -k_{CB} \pm i\Delta\omega_{AC} \end{pmatrix} \\ & - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}. \end{aligned} \quad (11.68)$$

The MQ equations

The formulae for multiple quantum CPMG-type data are the same as for the ‘NS MMQ 2-site’ model except for the R_{eff} calculation and the \mathbf{m}_j matrices. The rate is calculated as

$$R_{\text{eff}} = -\frac{1}{T} \log \left\{ \text{Re} \left[\frac{0.5}{p_A} (1 \ 0 \ 0) \cdot (\mathbf{AB} + \mathbf{CD}) \cdot \begin{pmatrix} p_A \\ p_B \\ p_C \end{pmatrix} \right] \right\}. \quad (11.69)$$

The \mathbf{m}_j matrices are

$$\mathbf{m}_1 = \begin{pmatrix} -k_{AB} & k_{BA} & 0 \\ k_{AB} & -k_{BA} - k_{BC} - i(\Delta\omega_{AB}^H + \Delta\omega_{AB}) & k_{CB} \\ 0 & k_{BC} & -k_{CB} - i(\Delta\omega_{AC}^H + \Delta\omega_{AC}) \end{pmatrix} - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}, \quad (11.70a)$$

$$\mathbf{m}_2 = \begin{pmatrix} -k_{AB} & k_{BA} & 0 \\ k_{AB} & -k_{BA} - k_{BC} - i(\Delta\omega_{AB}^H - \Delta\omega_{AB}) & k_{CB} \\ 0 & k_{BC} & -k_{CB} - i(\Delta\omega_{AC}^H - \Delta\omega_{AC}) \end{pmatrix} - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}. \quad (11.70b)$$

For the model, the assumption $R_{2A}^0 = R_{2B}^0 = R_{2C}^0 = R_2^0$ is made.

More information about the NS MMQ 3-site linear model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_MMQ_3-site_linear,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_mmq_3site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_MMQ_3-site_linear.

11.6.3 The NS MMQ 3-site model

This is the numerical model for 3-site exchange for proton-heteronuclear SQ, ZQ, DQ and MQ CPMG data, as derived in (Korzhnev et al., 2004a,b, 2005a). However it has been extended to allow the $A \leftrightarrow C$ transition. To simplify the optimisation space for the model as in the ‘NS MMQ 3-site linear’ model, the assumption $R_{2A}^0 = R_{2B}^0 = R_{2C}^0 = R_2^0$ has been made.

The SQ, ZD and DQ equations

The basic evolution matrices for single, zero and double quantum CPMG-type data for this model are

$$\mathbf{A}_\pm = e^{\mathbf{a}_\pm \cdot \tau_{\text{CPMG}}}, \quad (11.71)$$

where

$$\mathbf{a}_\pm = \begin{pmatrix} -k_{AB} - k_{AC} & k_{BA} & k_{CA} \\ k_{AB} & -k_{BA} - k_{BC} \pm i\Delta\omega_{AB} & k_{CB} \\ k_{AC} & k_{BC} & -k_{CB} - k_{CA} \pm i\Delta\omega_{AC} \end{pmatrix} - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}. \quad (11.72)$$

The MQ equations

The \mathbf{m}_j matrices for this model are

$$\mathbf{m}_1 = \begin{pmatrix} -k_{AB} - k_{AC} & k_{BA} & k_{CA} \\ k_{AB} & -k_{BA} - k_{BC} - i(\Delta\omega_{AB}^H + \Delta\omega_{AB}) & k_{CB} \\ k_{AC} & k_{BC} & -k_{CB} - k_{CA} - i(\Delta\omega_{AC}^H + \Delta\omega_{AC}) \end{pmatrix} - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}, \quad (11.73a)$$

$$\mathbf{m}_2 = \begin{pmatrix} -k_{AB} - k_{AC} & k_{BA} & k_{CA} \\ k_{AB} & -k_{BA} - k_{BC} - i(\Delta\omega_{AB}^H - \Delta\omega_{AB}) & k_{CB} \\ k_{AC} & k_{BC} & -k_{CB} - k_{CA} - i(\Delta\omega_{AC}^H - \Delta\omega_{AC}) \end{pmatrix} - \begin{pmatrix} R_{2A}^0 & 0 & 0 \\ 0 & R_{2B}^0 & 0 \\ 0 & 0 & R_{2C}^0 \end{pmatrix}. \quad (11.73b)$$

More information about the NS MMQ 3-site model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_MMQ_3-site,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_mmq_3site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_MMQ_3-site.

11.7 The analytic $R_{1\rho}$ models

These are the analytic models designed for $R_{1\rho}$ -type experiments.

11.7.1 The M61 2-site fast exchange $R_{1\rho}$ model

This is the model for 2-site fast exchange for on-resonance $R_{1\rho}$ -type data. It is selected by setting the model to ‘M61’, here named after Meiboom (1961). The equation for the exchange process is

$$R_{1\rho} = R'_{1\rho} + \frac{\Phi_{\text{ex}} k_{\text{ex}}}{k_{\text{ex}}^2 + \omega_e^2}. \quad (11.74)$$

The reference for this equation is:

- Meiboom, S. (1961). Nuclear magnetic resonance study of proton transfer in water. *J. Chem. Phys.*, **34**(2), 375–388. ([10.1063/1.1700960](https://doi.org/10.1063/1.1700960))

More information about the M61 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/M61>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.m61-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#M61.

11.7.2 The M61 skew 2-site fast exchange $R_{1\rho}$ model

This is the second model for 2-site fast exchange for on-resonance $R_{1\rho}$ -type data from Meiboom (1961). It is selected by setting the model to ‘M61 skew’. The equation for the exchange process is

$$R_{1\rho} = R'_{1\rho} + \frac{p_A^2 p_B \Delta\omega^2 k_{\text{ex}}}{k_{\text{ex}}^2 + p_A^2 \Delta\omega^2 + \omega_1^2}. \quad (11.75)$$

Care must be taken as this model appears to have infinite lines of solutions – p_A and $\Delta\omega$ are convoluted. Hence this model is disabled in the dispersion auto-analysis.

More information about the M61 skew model is available from:

- the relax wiki at http://wiki.nmr-relax.com/M61_skew,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.m61b-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#M61_skew.

11.7.3 The DPL94 2-site fast exchange $R_{1\rho}$ model

This is the model for 2-site fast exchange for $R_{1\rho}$ -type data. It is selected by setting the model to ‘DPL94’, here named after [Davis et al. \(1994\)](#). It extends the [Meiboom \(1961\)](#) model to off-resonance data. The model collapses to the M61 model for on-resonance data. The equation for the exchange process is

$$R_{1\rho} = R_1 \cos^2 \theta + \left(R'_{1\rho} + \frac{\Phi_{\text{ex}} k_{\text{ex}}}{k_{\text{ex}}^2 + \omega_e^2} \right) \sin^2 \theta, \quad (11.76)$$

where θ is the rotating frame tilt angle. The reference for this equation is:

- Davis, D. G., Perlman, M. E., and London, R. E. (1994). Direct measurements of the dissociation-rate constant for inhibitor-enzyme complexes via the T1rho and T2 (CPMG) methods. *J. Magn. Reson.*, **104**(3), 266–275. ([10.1006/jmrb.1994.1084](https://doi.org/10.1006/jmrb.1994.1084))

More information about the DPL94 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/DPL94>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.dpl94-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#DPL94.

11.7.4 The TP02 2-site exchange $R_{1\rho}$ model

This is the model for 2-site exchange for off-resonance $R_{1\rho}$ -type data from [Trott and Palmer \(2002\)](#). It is selected by setting the model to ‘TP02’. The equation for the exchange process is

$$R_{1\rho} = R_1 \cos^2 \theta + R'_{1\rho} \sin^2 \theta + \frac{\sin^2 \theta p_A p_B \Delta \omega^2 k_{\text{ex}}}{\omega_{\text{Aeff}}^2 \omega_{\text{Beff}}^2 / \omega_{\text{eff}}^2 + k_{\text{ex}}^2}, \quad (11.77)$$

in which

$$\delta_A = \omega_A - \omega_{\text{rf}}, \quad (11.78a)$$

$$\delta_B = \omega_B - \omega_{\text{rf}}, \quad (11.78b)$$

$$\overline{\omega} = p_A \omega_A + p_B \omega_B, \quad (11.78c)$$

$$\overline{\Omega} = \overline{\omega} - \omega_{\text{rf}}, \quad (11.78d)$$

$$\omega_{\text{Aeff}}^2 = \omega_1^2 + \delta_A^2, \quad (11.78e)$$

$$\omega_{\text{Beff}}^2 = \omega_1^2 + \delta_B^2, \quad (11.78f)$$

$$\omega_{\text{eff}}^2 = \omega_1^2 + \overline{\Omega}^2, \quad (11.78g)$$

$$\theta = \arctan \left(\frac{\omega_1}{\overline{\Omega}} \right). \quad (11.78h)$$

The equation is accurate only when populations are highly skewed with $p_A \gg p_B$. And it is valid only for exchange processes which are not fast. Note that this model has been superseded by the ‘TAP03’ and ‘MP05’ models. The reference for this equation is:

- Trott, O. and Palmer, 3rd, A. G. (2002). R1rho relaxation outside of the fast-exchange limit. *J. Magn. Reson.*, **154**(1), 157–160. ([10.1006/jmre.2001.2466](https://doi.org/10.1006/jmre.2001.2466))

More information about the TP02 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/TP02>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.tp02-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#TP02.

11.7.5 The TAP03 2-site exchange R_{1ρ} model

This is the model for 2-site exchange for off-resonance R_{1ρ}-type data from [Trott et al. \(2003\)](#). It is selected by setting the model to ‘TAP03’. The equation for the exchange process is

$$R_{1\rho} = R_1 \cos^2 \theta + R'_{1\rho} \sin^2 \theta + \left(\frac{1}{\gamma} \right) \frac{\sin^2 \hat{\theta} p_A p_B \Delta \omega^2 k_{\text{ex}}}{\hat{\omega}_{\text{Aeff}}^2 \hat{\omega}_{\text{Beff}}^2 / \hat{\omega}_{\text{eff}}^2 + k_{\text{ex}}^2 - 2 \sin^2 \hat{\theta} p_A p_B \Delta \omega^2 + (1 - \gamma) \omega_1^2}, \quad (11.79)$$

in which, in addition to those parameters defined above for the ‘TP02’ model,

$$\sigma = p_B \delta_A + p_A \delta_B, \quad (11.80a)$$

$$\gamma = 1 - p_A p_B \Delta \omega^2 \frac{\sigma^2 - k_{\text{ex}}^2 + \omega_1^2}{(\sigma^2 + k_{\text{ex}}^2 + \omega_1^2)^2}, \quad (11.80b)$$

$$\hat{\omega}_{\text{Aeff}}^2 = \gamma \omega_1^2 + \delta_A^2, \quad (11.80c)$$

$$\hat{\omega}_{\text{Beff}}^2 = \gamma \omega_1^2 + \delta_B^2, \quad (11.80d)$$

$$\hat{\omega}_{\text{eff}}^2 = \gamma \omega_1^2 + \bar{\Omega}^2, \quad (11.80e)$$

$$\hat{\theta} = \arctan \left(\frac{\sqrt{\gamma} \omega_1}{\bar{\Omega}} \right). \quad (11.80f)$$

The equation is accurate when populations are less skewed than the ‘TP02’ model ($p_A \gg p_B$). Note that this model, as with the ‘TP02’ model, has been superseded by the ‘MP05’ model in the next section. The reference for this equation is:

- Trott, O., Abergel, D., and Palmer, A. (2003). An average-magnetization analysis of R-1 rho relaxation outside of the fast exchange. *Mol. Phys.*, **101**(6), 753–763. ([10.1080/0026897021000054826](https://doi.org/10.1080/0026897021000054826))

More information about the TAP03 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/TAP03>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.tap03-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#TAP03.

11.7.6 The MP05 2-site exchange $R_{1\rho}$ model

This is the model for 2-site exchange for off-resonance $R_{1\rho}$ -type data for all time scales from [Miloushev and Palmer \(2005\)](#). It is selected by setting the model to ‘MP05’. The equation for the exchange process is

$$R_{1\rho} = R_1 \cos^2 \theta + R'_{1\rho} \sin^2 \theta + \frac{\sin^2 \theta p_A p_B \Delta \omega^2 k_{\text{ex}}}{\omega_{A_{\text{eff}}}^2 \omega_{B_{\text{eff}}}^2 / \omega_{\text{eff}}^2 + k_{\text{ex}}^2 - \sin^2 \theta p_A p_B \Delta \omega^2 \left(1 + \frac{2k_{\text{ex}}^2 (p_A \omega_{A_{\text{eff}}}^2 + p_B \omega_{B_{\text{eff}}}^2)}{\omega_{A_{\text{eff}}}^2 \omega_{B_{\text{eff}}}^2 + \omega_{\text{eff}}^2 k_{\text{ex}}^2} \right)}, \quad (11.81)$$

in which the parameters are defined as in the ‘TP02’ model above. This model supersedes both the ‘TP02’ and ‘TAP03’ models. The reference for this equation is:

- Miloushev, V. Z. and Palmer, 3rd, A. G. (2005). $R(1\rho)$ relaxation for two-site chemical exchange: general approximations and some exact solutions. *J. Magn. Reson.*, **177**(2), 221–227. ([10.1016/j.jmr.2005.07.023](https://doi.org/10.1016/j.jmr.2005.07.023))

More information about the MP05 model is available from:

- the relax wiki at <http://wiki.nmr-relax.com/MP05>,
- the API documentation at <http://www.nmr-relax.com/api/3.2/lib.dispersion.mp05-module.html>,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#MP05.

11.8 The numeric $R_{1\rho}$ models

These are the numeric models designed for $R_{1\rho}$ -type experiments.

11.8.1 The NS 2-site R_{1ρ} model

This is the numerical model for 2-site exchange using 3D magnetisation vectors. It is selected by setting the model to ‘NS R1rho 2-site’. The simple constraint $p_A > p_B$ is used to halve the optimisation space, as both sides of the limit are mirror image spaces.

For this model, the equations from Korzhnev et al. (2005b) have been used. The R_{1ρ} value for state A magnetisation is defined as

$$R_{1\rho} = -\frac{1}{T_{\text{relax}}} \cdot \ln(M_0^T \cdot e^{R \cdot T_{\text{relax}}} \cdot M_0), \quad (11.82)$$

where

$$M_0 = \begin{pmatrix} \sin \theta \\ 0 \\ \cos \theta \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (11.83)$$

$$\theta = \arctan \left(\frac{\omega_1}{\Omega_A} \right). \quad (11.84)$$

The relaxation evolution matrix is defined as

$$R = \begin{pmatrix} -R'_{1\rho} - k_{AB} & -\delta_A & 0 & k_{BA} & 0 & 0 \\ \delta_A & -R'_{1\rho} - k_{AB} & -\omega_1 & 0 & k_{BA} & 0 \\ 0 & \omega_1 & -R_1 - k_{AB} & 0 & 0 & k_{BA} \\ k_{AB} & 0 & 0 & -R'_{1\rho} - k_{BA} & -\delta_B & 0 \\ 0 & k_{AB} & 0 & \delta_B & -R'_{1\rho} - k_{BA} & -\omega_1 \\ 0 & 0 & k_{AB} & 0 & \omega_1 & -R_1 - k_{BA} \end{pmatrix}, \quad (11.85)$$

where $\delta_{A,B}$ is defined in Equations 11.78a and 11.78b.

More information about the NS R1rho 2-site model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_R1rho_2-site,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_r1rho_2site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_R1rho_2-site.

11.8.2 The NS 3-site R_{1ρ} model

This is the numerical model for 3-site exchange using 3D magnetisation vectors. It is selected by setting the model to ‘NS R1rho 3-site’. The constraints $p_A > p_B$ and $p_A > p_C$

is used to decrease the size of the optimisation space, as both sides of the limit are mirror image spaces.

For this model, as for the 2-site model above, the equations from Korzhnev et al. (2005b) have been used. These have been however rearranged to match the notation in Palmer and Massi (2006). The $R_{1\rho}$ value for state A magnetisation is defined as

$$R_{1\rho} = -\frac{1}{T_{\text{relax}}} \cdot \ln(M_0^T \cdot e^{R \cdot T_{\text{relax}}} \cdot M_0), \quad (11.86)$$

where

$$M_0 = \begin{pmatrix} \sin \theta \\ 0 \\ \cos \theta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (11.87)$$

$$\theta = \arctan\left(\frac{\omega_1}{\Omega_A}\right). \quad (11.88)$$

This assumes that the starting magnetisation has an X and Z component only for the A state. The relaxation evolution matrix is defined as

$$R = \begin{pmatrix} -R'_{1\rho A} - k_{AB} - k_{AC} & -\delta_A & 0 & \cdots \\ \delta_A & -R'_{1\rho A} - k_{AB} - k_{AC} & -\omega_1 & \cdots \\ 0 & \omega_1 & -R_{1A} - k_{AB} - k_{AC} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\ + \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \ddots \\ \cdots & -R'_{1\rho B} - k_{BA} - k_{BC} & -\delta_B & 0 & \cdots \\ \cdots & \delta_B & -R'_{1\rho B} - k_{BA} - k_{BC} & -\omega_1 & \cdots \\ \cdots & 0 & \omega_1 & -R_{1B} - k_{BA} - k_{BC} & \cdots \\ \ddots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\ + \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \ddots \\ \cdots & -R'_{1\rho C} - k_{CA} - k_{CB} & -\delta_C & 0 & \cdots \\ \cdots & \delta_C & -R'_{1\rho C} - k_{CA} - k_{CB} & -\omega_1 & \cdots \\ \cdots & 0 & \omega_1 & -R_{1C} - k_{CA} - k_{CB} & \cdots \end{pmatrix} \\ + \begin{pmatrix} & k_{BA} & 0 & 0 & \cdots \\ & \ddots & 0 & k_{BA} & 0 & \cdots \\ & & 0 & 0 & k_{BA} & \cdots \end{pmatrix} \\ + \begin{pmatrix} k_{AB} & 0 & 0 & & \\ 0 & k_{AB} & 0 & \ddots & \cdots \\ 0 & 0 & k_{AB} & & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\begin{aligned}
 & + \begin{pmatrix} & \cdots & k_{CA} & 0 & 0 \\ \ddots & \cdots & 0 & k_{CA} & 0 \\ & \cdots & 0 & 0 & k_{CA} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{AC} & 0 & 0 & \cdots & \\ 0 & k_{AC} & 0 & \cdots & \ddots \\ 0 & 0 & k_{AC} & \cdots & \end{pmatrix} \\
 & + \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots \\ & & k_{CB} & 0 & 0 \\ \dots & \ddots & 0 & k_{CB} & 0 \\ & & 0 & 0 & k_{CB} \\ \dots & k_{BC} & 0 & 0 & \\ \dots & 0 & k_{BC} & 0 & \ddots \\ \dots & 0 & 0 & k_{BC} & \end{pmatrix}, \tag{11.89}
 \end{aligned}$$

where $\delta_{A,B,C}$ are defined as in Equations 11.78a and 11.78b. For the model, the assumptions $R'_{1\rho A} = R'_{1\rho B} = R'_{1\rho C} = R'_{1\rho}$ and $R_{1A} = R_{1B} = R_{1C} = R_1$ have been made.

More information about the NS R1rho 3-site model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_R1rho_3-site,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_r1rho_3site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_R1rho_3-site.

11.8.3 The NS 3-site linear R_{1ρ} model

This is the numerical model for 3-site linear exchange using 3D magnetisation vectors. The assumption that $k_{AC} = k_{CA} = 0$ has been made to linearise this model. It is selected by setting the model to ‘NS R1rho 3-site linear’. The constraints $p_A > p_B$ and $p_A > p_C$ is used to decrease the size of the optimisation space, as both sides of the limit are mirror image spaces. To simplify the optimisation space for the model as in the ‘NS R_{1ρ} 3-site’ model, the assumptions $R^0_{2A} = R^0_{2B} = R^0_{2C} = R^0_2$ and $R_{1A} = R_{1B} = R_{1C} = R_1$ have been made.

The equations are the same as for the ‘NS R1rho 3-site’ model except for the relaxation

evolution matrix which simplifies to

$$\begin{aligned}
 R = & \begin{pmatrix} -R'_{1\rho A} - k_{AB} & -\delta_A & 0 & \dots \\ \delta_A & -R'_{1\rho A} - k_{AB} & -\omega_1 & \dots \\ 0 & \omega_1 & -R_{1A} - k_{AB} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\
 + & \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \ddots \\ \dots & -R'_{1\rho B} - k_{BA} - k_{BC} & -\delta_B & 0 & \dots \\ \dots & \delta_B & -R'_{1\rho B} - k_{BA} - k_{BC} & -\omega_1 & \dots \\ \dots & 0 & \omega_1 & -R_{1B} - k_{BA} - k_{BC} & \dots \\ \ddots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\
 + & \begin{pmatrix} \ddots & \vdots & \vdots & \vdots \\ \dots & -R'_{1\rho C} - k_{CB} & -\delta_C & 0 \\ \dots & \delta_C & -R'_{1\rho C} - k_{CB} & -\omega_1 \\ \dots & 0 & \omega_1 & -R_{1C} - k_{CB} \end{pmatrix} \\
 + & \begin{pmatrix} k_{BA} & 0 & 0 & \dots \\ \ddots & 0 & k_{BA} & 0 & \dots \\ & 0 & 0 & k_{BA} & \dots \\ k_{AB} & 0 & 0 & \ddots & \dots \\ 0 & k_{AB} & 0 & \ddots & \dots \\ 0 & 0 & k_{AB} & \ddots & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \\
 + & \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots \\ & k_{CB} & 0 & 0 & 0 \\ \dots & \ddots & 0 & k_{CB} & 0 \\ & 0 & 0 & 0 & k_{CB} \\ k_{BC} & 0 & 0 & \ddots & \dots \\ \dots & 0 & k_{BC} & 0 & \ddots \\ \dots & 0 & 0 & k_{BC} & \ddots \end{pmatrix}, \tag{11.90}
 \end{aligned}$$

where $\delta_{A,B,C}$ are defined as in Equations 11.78a and 11.78b. For the model, the assumptions $R'_{1\rho A} = R'_{1\rho B} = R'_{1\rho C} = R'_{1\rho}$ and $R_{1A} = R_{1B} = R_{1C} = R_1$ have been made.

More information about the NS R1rho 3-site linear model is available from:

- the relax wiki at http://wiki.nmr-relax.com/NS_R1rho_3-site_linear,
- the API documentation at http://www.nmr-relax.com/api/3.2/lib.dispersion.ns_r1rho_3site-module.html,
- the relaxation dispersion page of the relax website at http://www.nmr-relax.com/analyses/relaxation_dispersion.html#NS_R1rho_3-site_linear.

11.9 Relaxation dispersion optimisation theory

The implementation of optimisation in relax is discussed in detail in Chapter 14. To understand the concepts in this subsection, it is best to look at that chapter first.

11.9.1 The relaxation dispersion auto-analysis

In relax, optimisation can either be performed manually or one of the auto-analyses can be employed. Note that if you are using the relax GUI, you will be using the dispersion auto-analysis. The auto-analysis is a fully self-contained protocol designed to make the analysis as simple as possible. All details can be seen in the `auto_analyses/relax_disp.py` file which, in reality, is simply a large relax script.

The relaxation dispersion auto-analysis implements many of the concepts described in detail in the next sections. It can be summarised as:

Peak intensity error analysis: An error analysis is performed to determine the peak intensity errors, if not already calculated (see Section 17.2.213 on page 608).

'R_{2eff}' model optimisation: Firstly the 'R_{2eff}' model is either optimised (using the `minimise.execute` user function) or simply calculated (using the `minimise.calculate` user function) to find the R_{2eff} or R_{1ρ} values used as the base data for all other dispersion models (see Section 11.2.1 on page 156).

Dispersion curve insignificance: Spins with insignificant dispersion profiles will be deselected with the `relax_disp.insignificance` user function, as described below, excluding the 'No Rex' model.

Model optimisation: Sequential optimisation of each of the specified dispersion models. This consists of a grid search followed by Nelder-Mead simplex optimisation constrained using the log-barrier constraint algorithm. Each model will be stored in a different data pipe. See Section 17.2.70 on page 500 for the grid search and Section 17.2.69 on page 497 for minimisation.

Grid search avoidance: A number of tricks are used to speed up optimisation by skipping or decreasing the size of the initial grid search:

Pre-run directory: If a pre-run directory is supplied – a separate directory containing the dispersion auto-analysis results from a previous run – the optimised parameters from these previous results will be used as the starting point for optimisation rather than performing a grid search. This is used in a clustered analysis whereby the pre-run directory contains results from a non-clustered analysis. This is essential for when large spin clusters are specified, as a grid search becomes prohibitively expensive with clusters of three or more spins. At some point a RelaxError will occur because the grid search is impossibly large. For the cluster specific parameters, i.e. the populations of the states and the exchange parameters, an average value will be used as the starting point. For all other parameters, the R₂⁰ values for each spin and magnetic field, as well as the parameters related to the chemical shift difference Δω, the optimised values of the previous run will be directly copied.

Model nesting: If two models are nested, then the parameters of the simpler will be used as the starting point for optimisation of the more complex. The currently supported nested model sets are presented in Table 11.3 on page 188. The models are optimised in the order presented in that table. In some cases, the R_{2A}^0 and R_{2B}^0 parameter values are set to the simpler model R_2^0 value and the grid search is bypassed.

Model equivalence: When two models are equivalent, the optimised parameters of one model can be used as the starting point of the other rather than performing a grid search. This is used in the auto-analysis for avoiding the grid search in the numeric models. The optimised ‘CR72’ model is used for the ‘NS CPMG 2-site expanded’, ‘NS CPMG 2-site 3D’, and ‘NS CPMG 2-site star’ models. The optimised ‘MMQ CR72’ model is used for the ‘NS MMQ 2-site’ model. And the ‘MP05’ model is used for the ‘NS R1rho 2-site’ model.

Interruption: The optimisation procedure of the auto-analysis can read saved results files if a previous calculation was interrupted.

Model elimination: As it is quite common that some of the dispersion models fail to optimise to reasonable values, or will even optimise to non-physically possible values where the global minimum is located, model elimination is performed to remove these models. The relax implementation is described in [d’Auvergne and Gooley \(2006\)](#). This needs to be performed prior to model selection as a failed model will often provide a statistically better fit than a non-failed model.

Per-model error analysis: If desired, Monte Carlo simulations for error propagation can be performed for each model. This does however require far greater computation time.

Model selection: If more than one model is analysed, AIC model selection will be performed to judge statistical significance of the models ([Akaike, 1973](#)). This is used to determine if statistically significant R_{ex} contributions can be extracted from the data, as well as determine if one model is better than the other. Different statistical techniques such as AICc and BIC can be used when using the script UI ([Hurvich and Tsai, 1989; Schwarz, 1978](#)). The AIC, AICc and BIC equations for NMR relaxation data were derived in [d’Auvergne and Gooley \(2003\)](#). In most cases, the list models to choose from should be severely limited. The results will be stored in a new ‘final’ data pipe and output files placed in the `final` directory.

Error analysis: Monte Carlo simulations for error propagation is performed on the final data pipe (see Section 17.2.96 on page 535 as well as the descriptions for all of the other `monte_carlo` user functions). Model elimination is performed again to remove the Monte Carlo simulations which have failed.

Output file creation: For each of the models and the final model selection results, the `relax_disp.plot_disp_curves` (Section 17.2.170 on page 581), `relax_disp.plot_exp_curves` (Section 17.2.171 on page 582), `relax_disp.write_disp_curves` (Section 17.2.182 on page 590), `grace.write` (Section 17.2.56 on page 485) and `value.write` (Section 17.2.269 on page 660) user functions will be called to generate all the output files you would need. These generate both Grace 2D plots of the data as well as plain text files. Additional output files can be created after the analysis by using the user functions manually.

All these steps will be shown in full detail in the relax logs. You should check very carefully for any relax warnings as these can be an indication that something has not been set up correctly.

If you are a power user, you are free to use all of the relax user functions, the relax library, and the relax data store to implement your own protocol. If you wish, the protocol can be converted into a new auto-analysis and distributed as part of relax. The relax test suite will ensure the protocol remains functional for the lifetime of relax.

Table 11.3: Model nesting for the relaxation dispersion auto-analysis.

Model	Nested models ¹
Base models	
$R_{\text{eff}}/R'_{1\rho}$	-
No R_{ex}	-
Single quantum (SQ) CPMG-type	
LM63	LM63
LM63 3-site	NS CPMG 2-site expanded, NS CPMG 2-site 3D, NS CPMG 2-site star, B14
CR72	NS CPMG 2-site 3D full, NS CPMG 2-site star full, B14 full, NS CPMG 2-site expanded,
CR72 full	NS CPMG 2-site 3D, NS CPMG 2-site star, B14, CR72
IT99	-
TSMFK01	NS CPMG 2-site expanded, NS CPMG 2-site 3D, NS CPMG 2-site star, CR72
B14	NS CPMG 2-site 3D full, NS CPMG 2-site star full, CR72 full, NS CPMG 2-site expanded,
B14 full	NS CPMG 2-site 3D, NS CPMG 2-site star, B14, CR72
NS CPMG 2-site expanded	NS CPMG 2-site 3D, NS CPMG 2-site star, B14, CR72
NS CPMG 2-site 3D	NS CPMG 2-site expanded, NS CPMG 2-site star, B14, CR72
NS CPMG 2-site 3D full	NS CPMG 2-site star full, B14 full, CR72 full, NS CPMG 2-site expanded, NS CPMG 2-site 3D,
NS CPMG 2-site star	NS CPMG 2-site star, B14, CR72
NS CPMG 2-site 3D full, B14 full, CR72 full, NS CPMG 2-site expanded, NS CPMG 2-site 3D,	NS CPMG 2-site 3D full, B14 full, CR72 full, NS CPMG 2-site expanded, NS CPMG 2-site 3D,
NS CPMG 2-site star full	NS CPMG 2-site star, B14, CR72
MMQ (SQ, ZQ, DQ, & MQ) CPMG-type	
MMQ CR72	NS MMQ 2-site
NS MMQ 2-site	MMQ CR72
NS MMQ 3-site linear	NS MMQ 3-site, NS MMQ 2-site, MMQ CR72
NS MMQ 3-site	NS MMQ 3-site linear, NS MMQ 2-site, MMQ CR72

Table 11.3: Model nesting for the relaxation dispersion auto-analysis.

Model	Nested models ¹
$R_{1\rho}$ -type	
M61	-
M61 skew	-
DPL94	-
DPL94 R_1 fit	-
TP02	MP05, TAP03
TP02 R_1 fit	MP05 R_1 fit, TAP03 R_1 fit
TAP03	MP05, TP02
TAP03 R_1 fit	MP05 R_1 fit, TP02 R_1 fit
MP05	TAP03, TP02
MP05 R_1 fit	TAP03 R_1 fit, TP02 R_1 fit
NS $R_{1\rho}$ 2-site	MP05, TAP03, TP02
NS $R_{1\rho}$ 2-site R_1 fit	MP05 R_1 fit, TAP03 R_1 fit, TP02 R_1 fit
NS $R_{1\rho}$ 2-site, MP05, TAP03, TP02	NS $R_{1\rho}$ 2-site, MP05, TAP03, TP02
NS $R_{1\rho}$ 3-site linear	NS $R_{1\rho}$ 3-site linear, NS $R_{1\rho}$ 2-site, MP05, TAP03, TP02
NS $R_{1\rho}$ 3-site	NS $R_{1\rho}$ 3-site linear, NS $R_{1\rho}$ 2-site, NS $R_{1\rho}$ 3-site

¹The nested models are ordered by preference. The earliest model in the list which has been optimised in the auto-analysis will be used as the nested model. For example for the 'B14 full' model, the 'CR72 full' model is the first preference, followed by 'B14', then the final fall back is 'CR72 full' or 'B14' have been optimised. If none of the nested models have been optimised, the grid search will be performed. In this example, 'CR72 full' is preferred as it has perfect parameter nesting – all parameters of 'B14 full' are found in 'CR72 full'. The B14 and CR72 are fallbacks, and for these R20A and R20B are copied from R20 so they start optimisation as R20A == R20B. Hence 'CR72 full' whereby R20A != R20B is a much better starting point as R20A and R20B have been optimised to different values. But because of the large model instability in the 'CR72 full' model, you may wish to instead start with 'B14'.

11.9.2 Dispersion curve insignificance

To avoid severe model failure due to the fitting of statistically insignificant dispersion curves, the `relax_disp.insignificance` user function can be used. This is activated by default in the auto-analysis. The user function takes a single insignificance value and if the difference between the smallest and largest $R_{2\text{eff}}$ or $R_{1\rho}$ value for an individual spin for all dispersion curves is less than this, then that spin will be deselected. See Section 17.2.167 on page 580 for more details.

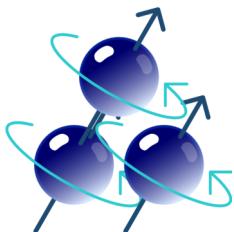
11.9.3 The relaxation dispersion space

In a dispersion analysis the target function $f(\theta)$ is the chi-squared equation

$$\chi^2(\theta) = \sum_{i=1}^n \frac{(R_{2\text{eff}} - R_{2\text{eff}}(\theta))^2}{\sigma_i^2}, \quad (11.91)$$

where i is the summation index, $R_{2\text{eff}}$ is the experimental relaxation data which belongs to the data set R and includes the $R_{2\text{eff}}$ and $R_{1\rho}$ values for all experiments at all magnetic field strengths, $R_{2\text{eff}}(\theta)$ is the back calculated relaxation data belonging to the set $R(\theta)$, and σ_i is the experimental $R_{2\text{eff}}/R_{1\rho}$ error. For standard optimisation, the summation index ranges over the relaxation data of an individual spin. However this can be changed with clustering whereby the relaxation data from a group of spin systems are optimised using a shared set of parameters.

11.9.4 The clustered relaxation dispersion analysis



Often in a relaxation dispersion analysis you will wish to fit a number of spin systems together using global parameters such as k_{ex} , p_A , etc. This can be achieved through the concept of clustering. A cluster is defined by an ID string and can contain any number of spins. Multiple clusters in one analysis can be defined. Any spins not included in a cluster will be treated as a free spin whereby all parameters of the dispersion model are local to that spin. Spin clusters can be defined using the `relax_disp.cluster` user function (see Section 17.2.162 on page 577) or via the spin cluster GUI element.

For the spin clustering, the special `relax_disp.parameter_copy` user function has been designed to help avoid the impossibly large grid search. This user function will copy the parameters from one non-clustered data pipe to another pipe, taking the median of the parameters from the first data pipe. Rather than taking the median, the $R_2^0/R_{1\rho}'$ and $\Delta\omega$ related parameters which are independent of the clustering are simply copied.

11.9.5 Dispersion parameter grid search

One of the most statistically unbiased methods for determining an initial parameter estimate prior to optimisation is to perform a grid search. This is performed via the `minimise.grid_search` user function (see Section 17.2.70 on page 500).

For some dispersion models the grid search can be too computationally expensive. In this case, some tricks can be used to bypass the parts of the grid search or the whole grid search:

Model nesting: Using the optimised parameters of a simpler nested model as the starting point for optimisation.

Model equivalence: Using the optimised solution of an equivalent analytic model as the starting point for a numeric model.

These tricks are implemented in the relaxation dispersion auto-analysis protocol as described above. If you do not use the auto-analysis or the GUI, then you are free to implement your own solutions.

The grid search lower and upper bounds default to:

$$5 \leq R_2^0 \leq 20, \quad (11.92a)$$

$$5 \leq R_{2A}^0 \leq 20, \quad (11.92b)$$

$$5 \leq R_{2B}^0 \leq 20, \quad (11.92c)$$

$$0 \leq \Phi_{ex} \leq 10, \quad (11.92d)$$

$$0 \leq \Phi_{ex,B} \leq 10, \quad (11.92e)$$

$$0 \leq \Phi_{ex,C} \leq 10, \quad (11.92f)$$

$$0 \leq p_A \Delta \omega^2 \leq 10, \quad (11.92g)$$

$$0 \leq \Delta \omega \leq 10, \quad (11.92h)$$

$$0 \leq \Delta \omega_{AB} \leq 10, \quad (11.92i)$$

$$0 \leq \Delta \omega_{BC} \leq 10, \quad (11.92j)$$

$$0 \leq \Delta \omega^H \leq 3, \quad (11.92k)$$

$$0 \leq \Delta \omega_{AB}^H \leq 3, \quad (11.92l)$$

$$0 \leq \Delta \omega_{BC}^H \leq 3, \quad (11.92m)$$

$$0.5 \leq p_A \leq 1, \quad (11.92n)$$

$$0.0 \leq p_B \leq 0.5, \quad (11.92o)$$

$$1 \leq k_{ex} \leq 1e^4, \quad (11.92p)$$

$$1 \leq k_{ex}^{AB} \leq 1e^4, \quad (11.92q)$$

$$1 \leq k_{ex}^{BC} \leq 1e^4, \quad (11.92r)$$

$$1 \leq k_A \leq 1e^4, \quad (11.92s)$$

$$1 \leq k_B \leq 1e^4, \quad (11.92t)$$

$$0.1 \leq k_{AB} \leq 20, \quad (11.92u)$$

$$1e^{-4} \leq \tau_{ex} \leq 1. \quad (11.92v)$$

For the MMQ models, the grid bounds are slightly different with

$$-10 \leq \Delta\omega \leq 10, \quad (11.93a)$$

$$-10 \leq \Delta\omega_{AB} \leq 10, \quad (11.93b)$$

$$-10 \leq \Delta\omega_{BC} \leq 10, \quad (11.93c)$$

$$-3 \leq \Delta\omega^H \leq 3, \quad (11.93d)$$

$$-3 \leq \Delta\omega_{AB}^H \leq 3, \quad (11.93e)$$

$$-3 \leq \Delta\omega_{BC}^H \leq 3. \quad (11.93f)$$

These values can be changed when not using the auto-analysis. Linear constraints can decrease the number of grid points searched through.

11.9.6 Dispersion parameter optimisation

For a description of gradients and Hessians, see Section 14.3.2 on page 304 and Section 14.3.3 on page 304 respectively. The relaxation dispersion model gradients or Hessians have either not been calculated for the analytic models or cannot be calculated as the solution is not analytic. Numeric gradients and Hessians could be calculated but this is too computationally expensive, especially for the numeric models where this adds a second layer of numeric approximation.

Optimisation in relax is via the minfx package <https://sourceforge.net/projects/minfx/>. This allows the Nelder-Mead simplex optimisation technique (see Section 14.4.5 on page 309) and the log-barrier constraint algorithm (see Section 14.5.2 on page 312) to be used. The advantage of these two techniques is that it enables extremely reliable and high precision optimisation without the use of gradients or Hessians, hence can significantly increase optimisation speeds. They however do not avoid the multiple local minimum problem present in the MMQ models – for that a highly accurate grid search is a reasonable solution.

11.9.7 Relaxation dispersion parameter constraints

To understand this section, please see Section 14.5 on page 310. For a dispersion analysis, linear constraints are the most useful type of constraint.

For most models, the linear constraints in the notation of (14.18) for the relaxation rates are

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} R_2^0 \\ R_{2A}^0 \\ R_{2B}^0 \end{pmatrix} \geq \begin{pmatrix} 0 \\ -200 \\ 0 \\ -200 \\ 0 \\ -200 \end{pmatrix}, \quad (11.94)$$

for the Φ_{ex} and $\Delta\omega$ parameters as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \Phi_{\text{ex}} \\ \Phi_{\text{ex},B} \\ \Phi_{\text{ex},C} \\ p_A \Delta\omega^2 \\ \Delta\omega \\ \Delta\omega_{AB} \\ \Delta\omega_{BC} \\ \Delta\omega^H \\ \Delta\omega_{AB}^H \\ \Delta\omega_{BC}^H \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (11.95)$$

for the population parameters as

$$\begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 1 & 0 \\ -1 & -1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} p_A \\ p_B \end{pmatrix} \geq \begin{pmatrix} -1 \\ 0.5 \\ 0.85 \\ -1 \\ 1 \end{pmatrix}, \quad (11.96)$$

and for the exchange rate and time parameters as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} k_{\text{ex}} \\ k_{\text{ex}}^{AB} \\ k_{\text{ex}}^{BC} \\ k_B \\ k_C \\ k_{AB} \\ \tau_{\text{ex}} \end{pmatrix} \geq \begin{pmatrix} 0 \\ -2e^6 \\ 0 \\ -2e^6 \\ 0 \\ -2e^6 \\ 0 \\ -2e^6 \\ 0 \\ -100 \\ 0 \end{pmatrix}. \quad (11.97)$$

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

$$0 \leq R_2^0 \leq 200, \quad (11.98a)$$

$$0 \leq R_{2A}^0 \leq 200, \quad (11.98b)$$

$$0 \leq R_{2B}^0 \leq 200, \quad (11.98c)$$

$$\Phi_{\text{ex}} \geq 0, \quad (11.98d)$$

$$\Phi_{\text{ex},B} \geq 0, \quad (11.98e)$$

$$\Phi_{\text{ex},C} \geq 0, \quad (11.98f)$$

$$\Delta\omega \geq 0, \quad (11.98g)$$

$$\Delta\omega_{AB} \geq 0, \quad (11.98h)$$

$$\begin{aligned}
\Delta\omega_{BC} &\geq 0, & (11.98i) \\
\Delta\omega^H &\geq 0, & (11.98j) \\
\Delta\omega_{AB}^H &\geq 0, & (11.98k) \\
\Delta\omega_{BC}^H &\geq 0, & (11.98l) \\
p_A \Delta\omega^2 &\geq 0, & (11.98m) \\
0 \leq p_A \leq 1, & & (11.98n) \\
0 \leq p_B \leq p_A, & & (11.98o) \\
0 \leq p_C \leq p_A, & & (11.98p) \\
p_A \geq 0.85 \quad (\text{the skewed condition, } p_A \gg p_B), & & (11.98q) \\
0 \leq k_{ex} \leq 2e^6, & & (11.98r) \\
0 \leq k_{ex}^{AB} \leq 2e^6, & & (11.98s) \\
0 \leq k_{ex}^{BC} \leq 2e^6, & & (11.98t) \\
0 \leq k_A \leq 2e^6, & & (11.98u) \\
0 \leq k_B \leq 2e^6, & & (11.98v) \\
0 \leq k_{AB} \leq 100, & & (11.98w) \\
\tau_{ex} \geq 0. & & (11.98x)
\end{aligned}$$

Note that the $\Delta\omega$ and $\Delta\omega^H$ constraints are not used for any of the MMQ-type models as sign differentiation is possible. These constraints are also turned off for the ‘NS $R_{1\rho}$ 3-site linear’ and ‘NS $R_{1\rho}$ 3-site’ models. And that the $p_A \geq 0.85$ constraint is used instead of the $p_A \geq 0.5$ constraint for all models which require $p_A \gg p_B$. When not using the auto-analysis, constraints can be modified or turned off.

11.9.8 Relaxation dispersion diagonal scaling

The concept of diagonal scaling is explained in Section 14.6 on page 312.

For the dispersion analysis the scaling factor of 10 is used for the relaxation rates, $1e^4$ for the exchange rates, $1e^{-4}$ for exchange times, and 1 for all other parameters. The scaling matrix for the parameters $\{R_2^0, R_{2A}^0, R_{2B}^0, \Phi_{ex}, \Phi_{ex,B}, \Phi_{ex,C}, p_A \Delta\omega^2, \Delta\omega, \Delta\omega^H, p_A, p_B,$

$k_{\text{ex}}, k_B, k_C, k_{AB}, \tau_{\text{ex}}$ } is

$$\begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-4} \end{pmatrix}. \quad (11.99)$$

11.9.9 Relaxation dispersion model elimination

Relaxation dispersion models will often fail. This may be due to data quality and quantity issues, inherent instability in certain models, or the use of analytic models outside of the range of their defined viability. Model elimination is therefore required to remove these failed models prior to model selection, as failed models will often fit the experimental data statistically better than non-failed models. The user function `eliminate` (see Section 17.2.41 on page 475) is used to remove the failed models. Model elimination was implemented in relax as described in:

- 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. ([10.1007/s10858-006-9007-z](https://doi.org/10.1007/s10858-006-9007-z))

The following hard coded rules are used to eliminate models:

$$p_A \leqslant 0.501, \quad (11.100a)$$

$$p_A \geqslant 0.999, \quad (11.100b)$$

$$\tau_{\text{ex}} \geqslant 1.0. \quad (11.100c)$$

If a parameter falls outside of these limits, the entire spin cluster will be deselected. When not using the auto-analysis, custom model elimination rules can be defined and used with the `eliminate` user function.

11.9.10 Monte Carlo simulation elimination

Just as models can fail, often Monte Carlo simulations will also experience optimisation failures (see Figure 4 of d'Auvergne and Gooley (2006) for such a failure in the model-free

optimisation space). The minimum can be warped so much by the data randomisation that a new minimum appears at an unreasonable position in the optimisation space. Even when the original model optimisation is successful, this can affect a small portion of the simulations. These must be removed prior to calculating the parameter errors otherwise the errors will be significantly over estimated. The simulation model failures are outliers which skew the error estimate, introducing a bias. This can result in parameter error estimates which are too large. The solution is the use of the `eliminate` user function when Monte Carlo simulations are turned on – this will automatically deselect simulations rather than spins using the rules from the previous section. Note that relax is the only software which provides this feature.

11.9.11 Relaxation dispersion on a computer cluster using OpenMPI

If the optimisation is too slow on a single computer, the dispersion analysis has been parallelised on the level of both the spin cluster and the Monte Carlo simulation. The scaling efficiency is very close to perfect so if you have access to a computer cluster and the OpenMPI protocol, then the calculations can be run much faster. The implementation uses Gary Thompson's multi-processor package. See Section 1.3 on page 17 for details on how to use this.

11.10 To do – dispersion features yet to be implemented

The capabilities of the relaxation dispersion analysis in relax is expansive but it cannot be called complete. There are a number of features and models yet to be implemented. Missing features include:

- The handling of off-resonance effects in the models of the numeric solution for CPMG-type data. This is specifically the ‘NS CPMG 2-site 3D’ and ‘NS CPMG 2-site star’ models and their ‘* full’ equivalents. The necessary infrastructure is in place, but not activated yet (mainly due to a lack of synthetic data to test against). Currently only the software CATIA can handle this effect.
- Multi-state data – the handling of data from two sets of peaks from the same spin system is not properly supported. It is currently handled by assuming each state is a separate spin system but this means that $\Delta\omega$, $\Delta\omega^H$ and related parameters are not shared as they should be.
- The van't Hoff analysis of multi-temperature dispersion data (see https://en.wikipedia.org/wiki/Van_%27t_Hoff_equation).
- The Korzhnev et al. (2005b) correction for constant-time $R_{1\rho}$ experiments for the analytic models ($R_{1\rho} = -\lambda_1 - 1/T_{\text{relax}} \log a_1$, where $a_1 = 1 - p_B \cos^2(\theta_A - \theta_B)$, and $\theta_A = \arctan(\omega_1/\omega_A)$ and $\theta_B = \arctan(\omega_1/\omega_B)$).
- Calculation of the dispersion α value from Millet et al. (2000).
- Support for CEST-type data.

- Offset support for the ground state chemical shift rather than the observed peak position. This is really only needed for $R_{1\rho}$ -type data.

If you would like one of these features, please contact the “nmr-relax-devel at lists.sourceforge.net” mailing list. The most useful would be if you have synthetic data whereby you know what the true solution should be. This can then be incorporated into a relax system test and the feature implemented to allow the test to pass. Note that such data should never be emailed to a public mailing list! Synthetic data, or experimental data, can be obtained from the literature.

Some of the missing models include:

‘TP04’: The $R_{1\rho}$ -type data [Trott and Palmer \(2004\)](#) N-site analytic equation for all time scales with parameters $\{R'_{1\rho}, \dots, p_1, \dots, p_N, \bar{\omega}, k_{12}, \dots, k_{1N}\}$.

‘* $R_{1\rho}$ ’: All of the 3-site and N-site models as summarised in Table 1 of [Palmer and Massi \(2006\)](#).

‘BK13’: The $R_{1\rho}$ -type data [Baldwin and Kay \(2013\)](#) off-resonance 2-site equation for all time scales with parameters $\{R'_{1\rho}, \dots, p_A, \Delta\omega, k_{\text{ex}}\}$. For details and code, see the [archived Gna! support request #3155](#).

‘BK13 full’: The $R_{1\rho}$ -type data full [Baldwin and Kay \(2013\)](#) off-resonance 2-site equation for all time scales with parameters $\{R'_{1\rho A}, R'_{1\rho B}, \dots, p_A, \Delta\omega, k_{\text{ex}}\}$. For details and code, see the [archived Gna! support request #3155](#).

Information for how these can be added is given in the next section.

11.11 Tutorial for adding relaxation dispersion models

As the field of NMR relaxation dispersion has a very long history, it is not possible to include all analytic and numeric relaxation dispersion models for both CPMG-type or $R_{1\rho}$ -type experiments in relax. However it is not too difficult to add new models for your own needs if you have some Python, Matlab, Mathematica, or similar scripting skills. The steps required are detailed on the relax wiki page http://wiki.nmr-relax.com/Tutorial_for_adding_relaxation_dispersion_models_to_relax.

11.12 Comparison of dispersion analysis software

Diverse software exists for analysing relaxation dispersion data. The following is a list of the officially released software which you can use instead of relax:

CPMGFit Art Palmer’s original dispersion analysis software at <http://www.palmer.hs.columbia.edu/software/cpmgfit.html>.

cpmg_fit Dmitry Korzhnev’s dispersion software available upon request.

CATIA Flemming Hansen’s dispersion software at <http://www.biochem.ucl.ac.uk/hansen/catia/>. The reference is Hansen et al. (2008).

NESSY Michael Bieri’s dispersion software at <https://sourceforge.net/projects/nmr-nessy/>. The reference is Bieri and Gooley (2011).

GUARDD Ian Kleckner’s dispersion software at <http://code.google.com/p/guardd/>. The reference is Kleckner and Foster (2012).

ShereKhan See the web server at <http://sherekhan.bionmr.org/>. The reference is Mazur et al. (2013).

GLOVE Peter Wright’s dispersion software at <http://www.scripps.edu/wright/>. The reference is Sugase et al. (2013).

chemex Guillaume Bouvignies’ dispersion software which can be found at <http://code.google.com/p/chemex/>.

There is currently support in relax for generating the input files for CPMGFit, CATIA, NESSY, and ShereKhan and for running CPMGFit and CATIA from within relax.

The features of the different software are compared in Table 11.4 on page 199. Note that this table is likely to be incomplete so please see the websites of the respective software for an up to date list of features. The aim of this table is to provide a fair comparison between all of the available dispersion software. Therefore if you do find deficiencies or errors in this table please report these either to the relax users mailing list at “nmr-relax-users at lists.sourceforge.net” or submit a bug report (see section 3.4 on page 31) so that the details can be corrected.

Table 11.4: Comparison of the features for the different dispersion software.

	CPMGFit	cpmg.fit	CATIA	NESSY	GUARDD	ShereKhan	GLOVE	chemex	relax
Dispersion models									
Base models									
$R_{2\text{eff}}/R'_{1\rho}$	-	-	-	✓	✓	✓	✓	-	✓
No Rex	-	✓	-	✓	✓	-	✓	-	✓
Single quantum (SQ) CPMG-type									
LM63	✓	-	-	✓	-	✓	✓	-	✓
LM63 3-site	✓	-	-	✓	-	-	-	-	✓
CR72	✓	-	-	✓	-	✓	✓	-	✓
IT99	✓	-	-	-	-	-	✓	-	✓
TSMFK01	-	-	-	-	-	-	-	-	✓
B14	-	-	-	-	-	-	-	-	✓
NS 2-site	-	✓	✓	-	-	✓	?	-	✓
MMQ (SQ, ZQ, DQ, & MQ) CPMG-type									
MMQ CR72	-	✓	-	-	✓	-	?	-	✓
NS MMQ 2-site	-	✓	-	-	-	-	?	-	✓
NS MMQ 3-site linear	-	✓	-	-	-	-	?	-	✓
NS MMQ 3-site	-	✓	-	-	-	-	?	-	✓
$R_{1\rho}$ -type									
M61	-	-	-	-	-	-	-	-	✓
DPL94	-	-	-	-	-	-	-	-	✓
TP02	-	✓	-	-	-	-	-	-	✓
TAP03	-	-	-	-	-	-	-	-	✓
TP04	-	-	-	-	-	-	-	-	-
MP05	-	-	-	-	-	-	-	-	✓
BK13	-	-	-	-	-	-	-	-	-
NS $R_{1\rho}$ 2-site	-	✓	-	-	-	-	?	-	✓
NS $R_{1\rho}$ 3-site linear	-	✓	-	-	-	-	-	-	✓
NS $R_{1\rho}$ 3-site	-	✓	-	-	-	-	-	-	✓
General features									
Parallelisation via MPI for running on clusters	-	-	-	-	-	-	-	-	✓
Off-resonance effects (CPMG-type data)	-	-	✓	-	-	-	-	-	-
Off-resonance effects ($R_{1\rho}$ -type data)	-	✓	-	-	-	-	-	-	✓
Support for TROSY-type data	-	-	✓	-	-	-	-	-	-
Support for CEST-type data	-	-	-	-	-	-	-	✓	-
Support for scalar coupling effects	-	-	✓	-	-	-	-	-	-
Arrhenius / Van't Hoff analysis	-	✓	-	✓	✓	-	-	-	-
Dispersion data back calculation (BC)	-	✓	-	✓	✓	-	-	-	✓

Table 11.4: Comparison of the features for the different dispersion software.

	CPMGFit	cpmg_fit	CATIA	NESSY	GUARDD	ShereKhan	GLOVE	chemex	relax
User interface									
Graphical user interface (GUI)	-	-	-	✓	✓	-	-	-	✓
Web user interface (Web UI)	-	-	-	-	-	✓	-	-	-
Scripting user interface (Script UI)	-	✓	✓	-	-	✓	✓	-	✓
Shell interface	✓	-	-	-	-	-	-	✓	-
Temperature and field-dependent simulator	-	-	-	-	-	-	-	-	-
UI for experimental planning (data BC)	-	-	-	-	✓	-	-	-	-
Optimisation									
Grid search algorithm	✓	-	-	✓	✓	✓	✓	-	✓
Grid search (via scripting)	-	-	✓	-	-	-	-	-	-
Nelder-Mead simplex algorithm	-	-	-	-	-	-	-	-	✓
Levenberg-Maquardt algorithm	✓	✓	✓	✓	-	✓	✓	✓	-
Numeric gradient approximation	✓	✓	✓	✓	-	✓	✓	✓	-
Logarithmic-barrier constraint algorithm	-	-	-	-	-	-	✓	-	✓
MATLAB interior-point black magic	-	-	-	-	✓	-	-	-	-
Visualisation of the chi-squared space	-	-	-	-	✓	-	-	-	✓
Error propagation									
Covariance matrix (lowest quality)	✓	✓	✓	-	✓	✓	✓	-	-
Jackknife simulations (for missing errors)	✓	-	-	-	-	-	✓	-	-
Bootstrapping simulations (false errors)	-	-	-	-	-	-	-	-	-
Monte Carlo simulations (gold standard)	✓	-	-	✓	✓	-	✓	✓	✓
Inbuilt statistical comparisons									
Akaike's Information Criterion (AIC)	-	-	-	✓	-	-	-	-	✓
Small sample size AIC (AICc)	-	-	-	✓	-	-	-	-	✓
Bayesian Information Criterion (BIC)	-	-	-	-	-	-	-	-	✓
F-testing (ANOVA statistics)	-	-	-	✓	✓	-	-	-	-
Data and model testing									
Insignificant dispersion curve tests	-	-	-	✓	-	-	-	-	✓
Model elimination tests	-	-	-	-	-	-	-	-	✓
Programming languages									
Python	-	-	-	✓	-	✓	-	✓	✓
C	-	✓	✓	-	-	-	✓	-	✓
Perl	-	-	-	-	-	-	✓	-	-
MATLAB	-	-	-	-	✓	-	-	-	-
FORTRAN	✓	-	-	-	-	-	-	-	-
Ruby on Rails	-	-	-	-	-	✓	-	-	-
Free software licencing									
GNU General Public Licence (version 2+)	✓	-	-	-	-	-	-	-	-
GNU General Public Licence (version 3+)	-	-	-	✓	✓	-	-	✓	✓
Free software infrastructure	-	-	-	✓	✓	-	-	✓	✓
Proprietary	-	✓	✓	-	-	✓	✓	-	-

11.13 Analysing dispersion in the prompt/script UI mode

Before reading this section, please read Chapter 4 covering the relax data model first. It will explain many of the concepts used within the following example script. For detailed information on how to run a relax script, please see section 1.2.8 on page 12. The dispersion analysis is parallelised on the level of the spin cluster and Monte Carlo simulations so, if you have access to an MPI cluster or multi-core system with OpenMPI installed, please see section 1.3 on page 17 for how to run the calculations much quicker.

11.13.1 Dispersion script mode – the sample script

The following is a verbatim copy of the contents of the `sample_scripts/relax_disp/cpmg_analysis.py` file. You will need to first copy this script to a dedicated analysis directory containing peak lists, a sequence or PDB file and a file listing unresolved spin systems, and then modify its contents to suit your specific analysis. The script contents are:

```
1 """Script for performing a full relaxation dispersion analysis using CPMG-type data."""
2
3
4 # Python module imports.
5 from os import sep
6
7 # relax module imports.
8 from auto_analyses.relax_disp import Relax_disp
9
10
11 # Analysis variables.
12 #####
13
14 # The dispersion models.
15 MODELS = ['R2eff', 'No Rex', 'CR72', 'N2 CPMG 2-site expanded']
16
17 # The grid search size (the number of increments per dimension).
18 GRID_INC = 11
19
20 # The number of Monte Carlo simulations to be used for error analysis at the end of the
21 # analysis.
22 MC_NUM = 500
23
24 # The results directory.
25 RESULTS_DIR = '.'
26
27 # The model selection technique to use.
28 MODSEL = 'AIC'
29
30 # The flag for only using numeric models in the final model selection.
31 NUMERIC_ONLY = False
32
33 # The R2eff value in rad/s by which to judge insignificance. If the maximum difference
34 # between two points on all dispersion curves for a spin is less than this value, that
35 # spin will be deselected.
```

```

33  INSIGNIFICANCE = 1.0
34
35
36
37 # Set up the data pipe.
38 #####
39
40 # Create the data pipe.
41 pipe_name = 'base pipe'
42 pipe_bundle = 'relax_disp'
43 pipe.create(pipe_name=pipe_name, bundle=pipe_bundle, pipe_type='relax_disp')
44
45 # Load the sequence.
46 sequence.read('fake_sequence.in', res_num_col=1, res_name_col=2)
47
48 # Name the spins so they can be matched to the assignments, and the isotope for field
49 # strength scaling.
50 spin.name(name='N')
51 spin.isotope(isotope='15N')
52
53 # The spectral data - spectrum ID, peak list file name, CPMG frequency (Hz), spectrometer
54 # frequency in Hertz.
55 data = [
56     ['500_reference.in',      '500_MHz'+sep+'reference.in_sparky',      None,   500e6],
57     ['500_66.667.in',        '500_MHz'+sep+'66.667.in_sparky',        66.6666, 500e6],
58     ['500_133.33.in',        '500_MHz'+sep+'133.33.in_sparky',        133.3333, 500e6],
59     ['500_133.33.in.bis',    '500_MHz'+sep+'133.33.in.bis_sparky',    133.3333, 500e6],
60     ['500_200.in',           '500_MHz'+sep+'200.in_sparky',           200.0000, 500e6],
61     ['500_266.67.in',        '500_MHz'+sep+'266.67.in_sparky',        266.6666, 500e6],
62     ['500_333.33.in',        '500_MHz'+sep+'333.33.in_sparky',        333.3333, 500e6],
63     ['500_400.in',           '500_MHz'+sep+'400.in_sparky',           400.0000, 500e6],
64     ['500_466.67.in',        '500_MHz'+sep+'466.67.in_sparky',        466.6666, 500e6],
65     ['500_533.33.in',        '500_MHz'+sep+'533.33.in_sparky',        533.3333, 500e6],
66     ['500_533.33.in.bis',   '500_MHz'+sep+'533.33.in.bis_sparky',   533.3333, 500e6],
67     ['500_600.in',           '500_MHz'+sep+'600.in_sparky',           600.0000, 500e6],
68     ['500_666.67.in',        '500_MHz'+sep+'666.67.in_sparky',        666.6666, 500e6],
69     ['500_733.33.in',        '500_MHz'+sep+'733.33.in_sparky',        733.3333, 500e6],
70     ['500_800.in',           '500_MHz'+sep+'800.in_sparky',           800.0000, 500e6],
71     ['500_866.67.in',        '500_MHz'+sep+'866.67.in_sparky',        866.6666, 500e6],
72     ['500_933.33.in',        '500_MHz'+sep+'933.33.in_sparky',        933.3333, 500e6],
73     ['500_933.33.in.bis',   '500_MHz'+sep+'933.33.in.bis_sparky',   933.3333, 500e6],
74     ['500_1000.in',          '500_MHz'+sep+'1000.in_sparky',          1000.0000, 500e6],
75     ['800_reference.in',      '800_MHz'+sep+'reference.in_sparky',      None,   800e6],
76     ['800_66.667.in',        '800_MHz'+sep+'66.667.in_sparky',        66.6666, 800e6],
77     ['800_133.33.in',        '800_MHz'+sep+'133.33.in_sparky',        133.3333, 800e6],
78     ['800_133.33.in.bis',   '800_MHz'+sep+'133.33.in.bis_sparky',   133.3333, 800e6],
79     ['800_200.in',           '800_MHz'+sep+'200.in_sparky',           200.0000, 800e6],
80     ['800_266.67.in',        '800_MHz'+sep+'266.67.in_sparky',        266.6666, 800e6],
81     ['800_333.33.in',        '800_MHz'+sep+'333.33.in_sparky',        333.3333, 800e6],
82     ['800_400.in',           '800_MHz'+sep+'400.in_sparky',           400.0000, 800e6],
83     ['800_466.67.in',        '800_MHz'+sep+'466.67.in_sparky',        466.6666, 800e6],
84     ['800_533.33.in',        '800_MHz'+sep+'533.33.in_sparky',        533.3333, 800e6],
85     ['800_533.33.in.bis',   '800_MHz'+sep+'533.33.in.bis_sparky',   533.3333, 800e6],
86     ['800_600.in',           '800_MHz'+sep+'600.in_sparky',           600.0000, 800e6],
87     ['800_666.67.in',        '800_MHz'+sep+'666.67.in_sparky',        666.6666, 800e6],
88     ['800_733.33.in',        '800_MHz'+sep+'733.33.in_sparky',        733.3333, 800e6],
89     ['800_800.in',           '800_MHz'+sep+'800.in_sparky',           800.0000, 800e6],
90     ['800_866.67.in',        '800_MHz'+sep+'866.67.in_sparky',        866.6666, 800e6],
91     ['800_933.33.in',        '800_MHz'+sep+'933.33.in_sparky',        933.3333, 800e6],
92     ['800_933.33.in.bis',   '800_MHz'+sep+'933.33.in.bis_sparky',   933.3333, 800e6],
93     ['800_1000.in',          '800_MHz'+sep+'1000.in_sparky',          1000.0000, 800e6]

```

```

92     ]
93
94     # Loop over the spectra.
95     for id, file, cpmg_frq, H_frq in data:
96         # Load the peak intensities.
97         spectrum.read_intensities(file=file, spectrum_id=id, int_method='height')
98
99         # Set the relaxation dispersion experiment type.
100        relax_disp.exp_type(spectrum_id=id, exp_type='SQ CPMG')
101
102        # Set the relaxation dispersion CPMG frequencies.
103        relax_disp.cpmg_setup(spectrum_id=id, cpmg_frq=cpmg_frq)
104
105        # Set the NMR field strength of the spectrum.
106        spectrometer.frequency(id=id, freq=H_frq)
107
108        # Relaxation dispersion CPMG constant time delay T (in s).
109        relax_disp.relax_time(spectrum_id=id, time=0.030)
110
111    # Specify the duplicated spectra.
112    spectrum.replicated(spectrum_ids=['500_133.33.in', '500_133.33.in.bis'])
113    spectrum.replicated(spectrum_ids=['500_533.33.in', '500_533.33.in.bis'])
114    spectrum.replicated(spectrum_ids=['500_933.33.in', '500_933.33.in.bis'])
115    spectrum.replicated(spectrum_ids=['800_133.33.in', '800_133.33.in.bis'])
116    spectrum.replicated(spectrum_ids=['800_533.33.in', '800_533.33.in.bis'])
117    spectrum.replicated(spectrum_ids=['800_933.33.in', '800_933.33.in.bis'])
118
119    # Peak intensity error analysis.
120    spectrum.error_analysis(subset=['500_reference.in', '500_66.667.in', '500_133.33.in', '500
121        _133.33.in.bis', '500_200.in', '500_266.67.in', '500_333.33.in', '500_400.in', '500
122        _466.67.in', '500_533.33.in', '500_533.33.in.bis', '500_600.in', '500_666.67.in', '500
123        _733.33.in', '500_800.in', '500_866.67.in', '500_933.33.in', '500_933.33.in.bis', '500
124        _1000.in'])
125    spectrum.error_analysis(subset=['800_reference.in', '800_66.667.in', '800_133.33.in', '800
126        _133.33.in.bis', '800_200.in', '800_266.67.in', '800_333.33.in', '800_400.in', '800
127        _466.67.in', '800_533.33.in', '800_533.33.in.bis', '800_600.in', '800_666.67.in', '800
128        _733.33.in', '800_800.in', '800_866.67.in', '800_933.33.in', '800_933.33.in.bis', '800
129        _1000.in'])
130
131    # Deselect unresolved spins.
132    deselect.read(file='unresolved', dir='500_MHz', res_num_col=1)
133    deselect.read(file='unresolved', dir='800_MHz', res_num_col=1)
134
135
136    # Auto-analysis execution.
137    #####
138
139    # Do not change!
140    Relax_disp(pipe_name=pipe_name, pipe_bundle=pipe_bundle, results_dir=RESULTS_DIR, models=
141        MODELS, grid_inc=GRID_INC, mc_sim_num=MC_NUM, modsel=MODSEL, insignificance=
142        INSIGNIFICANCE, numeric_only=NUMERIC_ONLY)

```

11.13.2 Dispersion script mode – imports

At the very start of the script are two import statements. This is simply the standard Python import system for modules. The first will import the `sep` variable which is the operating system independent directory separator:

```
4 # Python module imports.
5 from os import sep
```

This `sep` variable will be used later on in the script. The second import is that of the automated relaxation dispersion class `Relax_disp` which will be used at the very end of the script to perform the full analysis:

```
7 # relax module imports.
8 from auto_analyses.relax_disp import Relax_disp
```

11.13.3 Dispersion script mode – analysis variables

The next part of the script is the definition of a number of analysis variables. As the example in this section is for CPMG-type experiments, the relaxation dispersion models which will be used in the auto-analysis are:

```
14 # The dispersion models.
15 MODELS = ['R2eff', 'No Rex', 'CR72', 'N2 CPMG 2-site expanded']
```

This list can be expanded to most of the 2-site exchange models, for example as:

```
MODELS = ['R2eff', 'No Rex', 'LM63', 'CR72', 'IT99', 'TSMFK01', 'NS CPMG 2-site expanded']
```

But note that the selection of which models to use is incredibly important. Do not use models which are not suitable for the data as that will cause the final results to contain rubbish. If you have $R_{1\rho}$ -type off-resonance data, the models could be changed to:

```
MODELS = ['R2eff', 'No Rex', 'DPL94', 'NS R1rho 2-site']
```

The next variable affects the optimisation precision:

```
17 # The grid search size (the number of increments per dimension).
18 GRID_INC = 21
```

The number of grid search increments may be decreased, but only after careful checking with a higher number of increments. Setting this value too low may place the initial optimisation too far away from the minimum. Although as-of-yet undetected and unpublished, if multiple local minima do exist then optimisation may not reach the global minimum. Too little grid search increments can also cause the total optimisation time to increase as the Nelder-Mead simplex optimisation together with the Logarithmic-barrier penalty function as used in the auto-analysis may require more time to reach the minimum.

The Monte Carlo simulation number `MC_NUM` variable affects the error estimate precision:

```
20 # The number of Monte Carlo simulations to be used for error analysis at the end of the
   analysis.
21 MC_NUM = 500
```

For accurate parameter errors this number should not be decreased. Ideally it should be increased however this will significantly increase the total analysis time. The next variable allows you to change the directory in which all results files from the auto-analysis will be saved.

```
23 # The results directory.
24 RESULTS_DIR = '.'
```

The MODSEL variable defines how the best dispersion model for the measured data is chosen:

```
26 # The model selection technique to use.
27 MODSEL = 'AIC'
```

For the automated analysis, currently only AIC, AICc, and BIC are supported. For details about these frequentist model selection techniques and their application to NMR data, see [d'Auvergne and Gooley \(2003\)](#). Post-analysis comparisons can also be preformed if desired. The NUMERIC_ONLY variable can be used to choose if only numeric or all models will be used in the model selection for the final results:

```
29 # The flag for only using numeric models in the final model selection.
30 NUMERIC_ONLY = False
```

To only use numeric models in the model selection while allowing models such as 'CR72' to be optimised and used as the starting point for the numeric models, change this variable to:

```
NUMERIC_ONLY = True
```

The last variable allows spins with insignificant dispersion profiles to be deselected:

```
32 # The R2eff value in rad/s by which to judge insignificance. If the maximum difference
   between two points on all dispersion curves for a spin is less than this value, that
   spin will be deselected.
33 INSIGNIFICANCE = 1.0
```

This is often needed due to the errors in the dispersion curves being underestimated, hence the 'No Rex' model is not selected when clearly it should be. To use all data in the analysis, this variable should be set to 0.0.

11.13.4 Dispersion script mode – initialisation of the data pipe

The data pipe is created using the lines:

```
40 # Create the data pipe.
41 pipe_name = 'base_pipe'
42 pipe_bundle = 'relax_disp'
43 pipe.create(pipe_name=pipe_name, bundle=pipe_bundle, pipe_type='relax_disp')
```

The first two lines define variables for the data pipe name and the pipe bundle name. The pipe bundle is used to group together all of the data pipes created by the automated protocol. See section [4.2.1](#) on page [36](#) for more details.

The `pipe.create` user function will then create a relaxation dispersion specific data pipe labelled with the pipe and bundle names. The third argument sets the pipe type to that of relaxation dispersion. The rest of the script is used to fill this base data pipe with all of the data required for a dispersion analysis. The auto-analysis will then copy the data from this pipe as it sees fit.

11.13.5 Dispersion script mode – setting up the spin systems

The first thing which needs to be completed prior to any spin specific command is to generate the molecule, residue and spin data structures for storing the spin specific data. In the sample script above, this is generated from a plain text file with the sequence information, however a PDB file can be used instead (see the `structure.read_pdb` user function on page 638 for more details). In the case of the sample script, the command:

```
45 # Load the sequence.
46 sequence.read('fake_sequence.in', res_num_col=1, res_name_col=2)
```

will load residue names and numbers from the `fake_sequence.in` file into relax, creating one spin per residue. Then:

```
48 # Name the spins so they can be matched to the assignments, and the isotope for field
  strength scaling.
49 spin.name(name='N')
50 spin.isotope(isotope='15N')
```

will set up the spin information required for loading the peak intensity data from Sparky peak lists and for the analysis of the dispersion data.

11.13.6 Dispersion script mode – loading the data

To load the peak intensities into relax, a large data structure is first defined:

```
52 # The spectral data - spectrum ID, peak list file name, CPMG frequency (Hz), spectrometer
  frequency in Hertz.
53 data = [
54     ['500_reference.in',      '500_MHz'+sep+'reference.in_sparky',           None,   500e6],
55     ['500_66.667.in',        '500_MHz'+sep+'66.667.in_sparky',             66.6666, 500e6],
56     ['500_133.33.in',        '500_MHz'+sep+'133.33.in_sparky',            133.3333, 500e6],
57     ['500_133.33.in.bis',    '500_MHz'+sep+'133.33.in.bis_sparky',       133.3333, 500e6],
58     ['500_200.in',           '500_MHz'+sep+'200.in_sparky',              200.0000, 500e6],
59     ['500_266.67.in',        '500_MHz'+sep+'266.67.in_sparky',            266.6666, 500e6],
60     ['500_333.33.in',        '500_MHz'+sep+'333.33.in_sparky',            333.3333, 500e6],
61     ['500_400.in',           '500_MHz'+sep+'400.in_sparky',              400.0000, 500e6],
62     ['500_466.67.in',        '500_MHz'+sep+'466.67.in_sparky',            466.6666, 500e6],
63     ['500_533.33.in',        '500_MHz'+sep+'533.33.in_sparky',            533.3333, 500e6],
64     ['500_533.33.in.bis',    '500_MHz'+sep+'533.33.in.bis_sparky',       533.3333, 500e6],
65     ['500_600.in',           '500_MHz'+sep+'600.in_sparky',              600.0000, 500e6],
66     ['500_666.67.in',        '500_MHz'+sep+'666.67.in_sparky',            666.6666, 500e6],
67     ['500_733.33.in',        '500_MHz'+sep+'733.33.in_sparky',            733.3333, 500e6],
68     ['500_800.in',           '500_MHz'+sep+'800.in_sparky',              800.0000, 500e6],
69     ['500_866.67.in',        '500_MHz'+sep+'866.67.in_sparky',            866.6666, 500e6],
70     ['500_933.33.in',        '500_MHz'+sep+'933.33.in_sparky',            933.3333, 500e6],
71     ['500_933.33.in.bis',    '500_MHz'+sep+'933.33.in.bis_sparky',       933.3333, 500e6],
72     ['500_1000.in',          '500_MHz'+sep+'1000.in_sparky',             1000.0000, 500e6],
73     ['800_reference.in',      '800_MHz'+sep+'reference.in_sparky',           None,   800e6],
74     ['800_66.667.in',        '800_MHz'+sep+'66.667.in_sparky',             66.6666, 800e6],
75     ['800_133.33.in',        '800_MHz'+sep+'133.33.in_sparky',            133.3333, 800e6],
76     ['800_133.33.in.bis',    '800_MHz'+sep+'133.33.in.bis_sparky',       133.3333, 800e6],
77     ['800_200.in',           '800_MHz'+sep+'200.in_sparky',              200.0000, 800e6],
78     ['800_266.67.in',        '800_MHz'+sep+'266.67.in_sparky',            266.6666, 800e6],
79     ['800_333.33.in',        '800_MHz'+sep+'333.33.in_sparky',            333.3333, 800e6],
80     ['800_400.in',           '800_MHz'+sep+'400.in_sparky',              400.0000, 800e6],
81     ['800_466.67.in',        '800_MHz'+sep+'466.67.in_sparky',            466.6666, 800e6],
```

```

92      ['800_533.33.in',      '800_MHz'+sep+'533.33.in_sparky',      533.3333,  800e6],
93      ['800_533.33.in.bis', '800_MHz'+sep+'533.33.in.bis_sparky', 533.3333,  800e6],
94      ['800_600.in',        '800_MHz'+sep+'600.in_sparky',        600.0000, 800e6],
95      ['800_666.67.in',    '800_MHz'+sep+'666.67.in_sparky',    666.6666, 800e6],
96      ['800_733.33.in',    '800_MHz'+sep+'733.33.in_sparky',    733.3333, 800e6],
97      ['800_800.in',       '800_MHz'+sep+'800.in_sparky',       800.0000, 800e6],
98      ['800_866.67.in',   '800_MHz'+sep+'866.67.in_sparky',   866.6666, 800e6],
99      ['800_933.33.in',   '800_MHz'+sep+'933.33.in_sparky',   933.3333, 800e6],
100     ['800_933.33.in.bis', '800_MHz'+sep+'933.33.in.bis_sparky', 933.3333, 800e6],
101     ['800_1000.in',     '800_MHz'+sep+'1000.in_sparky',     1000.0000, 800e6]
102   ]

```

In Python terminology, this is a list of lists data structure. It is essentially a matrix of information which is used in the subsequent `for` loop. The comment explains what each element is. For $R_{1\rho}$ -type experiments, the CPMG frequency column can be replaced with the spin-lock field strength. This data structure will need to be tailored to your data. It can be seen that the `sep` variable is now being used to specify that the Sparky files are either located in the `500_MHz` or `800_MHz` directories. It is used here to make this script independent of the operating system.

The Python `for` loop starts with the lines:

```

94 # Loop over the spectra.
95 for id, file, cpmg_frq, H_frq in data:

```

and includes all subsequently indented lines. This line of code takes the elements of the `data` data structure and splits it into 4 variables. Therefore for the first line, `id` will be set to '`500_reference.in`', `file` will be set to '`500_MHz/reference.in_sparky`' on a GNU/Linux machine, `cpmg_frq` will be `None`, and `H_frq` will be 500 MHz. For $R_{1\rho}$ -type data, you could change the `cpmg_frq` variable to `field` for example.

The first user function in the block loads the peak intensity data from the peak lists:

```

96 # Load the peak intensities.
97 spectrum.read_intensities(file=file, spectrum_id=id, int_method='height')

```

This assumes that peak heights were measured. All data will be tagged with the given ID string. For examples of peak list formats supported by relax, see Section 5.4.4 on page 59. The next step is to specify the dispersion experiment type for each spectrum:

```

99 # Set the relaxation dispersion experiment type.
100 relax_disp.exp_type(spectrum_id=id, exp_type='SQ CPMG')

```

This can be '`SQ CPMG`', '`DQ CPMG`', '`ZQ CPMG`', '`MQ CPMG`', '`1H SQ CPMG`', '`1H MQ CPMG`' or '`R1rho`'. The next user function sets the CPMG frequencies for each spectrum:

```

102 # Set the relaxation dispersion CPMG frequencies.
103 relax_disp.cpmg_setup(spectrum_id=id, cpmg_frq=cpmg_frq)

```

For an $R_{1\rho}$ -type experiment, these lines could be changed to:

```

# Set the relaxation dispersion R1rho spin lock field strength.
relax_disp.spin_lock_field(spectrum_id=id, field=field)

```

Then the NMR spectrometer field strength is set:

```
105     # Set the NMR field strength of the spectrum.
106     spectrometer.frequency(id=id, freq=H_frq)
```

And finally the relaxation time period is set with:

```
108     # Relaxation dispersion CPMG constant time delay T (in s).
109     relax_disp.relax_time(spectrum_id=id, time=0.030)
```

If exponential data has been collected rather than fixed time period data, then the data data structure can have an additional column added for the relaxation times, and then this same user function can be used. The `for` loop will need one extra variable for the times, and this should be passed into this `relax_disp.relax_time` user function for the time argument.

Finally, once the `for` loop has completed, replicated spectra are defined with the commands:

```
111     # Specify the duplicated spectra.
112     spectrum.replicated(spectrum_ids=['500_133.33.in', '500_133.33.in.bis'])
113     spectrum.replicated(spectrum_ids=['500_533.33.in', '500_533.33.in.bis'])
114     spectrum.replicated(spectrum_ids=['500_933.33.in', '500_933.33.in.bis'])
115     spectrum.replicated(spectrum_ids=['800_133.33.in', '800_133.33.in.bis'])
116     spectrum.replicated(spectrum_ids=['800_533.33.in', '800_533.33.in.bis'])
117     spectrum.replicated(spectrum_ids=['800_933.33.in', '800_933.33.in.bis'])
```

11.13.7 Dispersion script mode – 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 spins 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 functions:

```
119     # Peak intensity error analysis.
120     spectrum.error_analysis(subset=['500_reference.in', '500_66.667.in', '500_133.33.in', '500
121         _133.33.in.bis', '500_200.in', '500_266.67.in', '500_333.33.in', '500_400.in', '500
122         _466.67.in', '500_533.33.in', '500_533.33.in.bis', '500_600.in', '500_666.67.in', '500
123         _733.33.in', '500_800.in', '500_866.67.in', '500_933.33.in', '500_933.33.in.bis', '500
124         _1000.in'])
125     spectrum.error_analysis(subset=['800_reference.in', '800_66.667.in', '800_133.33.in', '800
126         _133.33.in.bis', '800_200.in', '800_266.67.in', '800_333.33.in', '800_400.in', '800
127         _466.67.in', '800_533.33.in', '800_533.33.in.bis', '800_600.in', '800_666.67.in', '800
128         _733.33.in', '800_800.in', '800_866.67.in', '800_933.33.in', '800_933.33.in.bis', '800
129         _1000.in'])
```

Here the 500 MHz and 800 MHz peak intensity errors have been calculated separately as they should not be the same.

Any spins which cannot be resolved due to peak overlap were included in a file called `unresolved`. This file can consist of optional columns of the molecule name, the residue name and number, and the spin name and number. The matching spins are excluded from the analysis by the user functions:

```
123     # Deselect unresolved spins.
124     deselect.read(file='unresolved', dir='500_MHz', res_num_col=1)
125     deselect.read(file='unresolved', dir='800_MHz', res_num_col=1)
```

11.13.8 Dispersion script mode – execution

Once the data has set up and you have modified your script to match your analysis needs, then the data pipe, pipe bundle and analysis variables are passed into the `Relax_disp` class. This is the final lines of the script:

```
129 # Auto-analysis execution.  
130 #####  
131  
132 # Do not change!  
133 Relax_disp(pipe_name=pipe_name, pipe_bundle=pipe_bundle, results_dir=RESULTS_DIR, models=  
    MODELS, grid_inc=GRID_INC, mc_sim_num=MC_NUM, modsel=MODSEL, insignificance=  
    INSIGNIFICANCE, numeric_only=NUMERIC_ONLY)
```

This will start the auto-analysis. If you are adventurous, you can replace this line with your own `minimise.grid_search`, `minimise.execute`, and `monte_carlo.*` user function calls and design your own protocol. For ideas in designing your own advanced analysis, see the `auto_analysis/relax_disp.py` file.

11.14 The relaxation dispersion auto-analysis in the GUI

The following demonstration of the relaxation dispersion analysis in the graphical user interface (GUI) uses the experimental single quantum (SQ) CPMG-type data from Flemming Hansen located in the relax directory `test_suite/shared_data/dispersion/Hansen`. This is the data from the paper:

- Hansen, D. F., Vallurupalli, P., Lundstrom, P., Neudecker, P., and Kay, L. E. (2008). Probing chemical shifts of invisible states of proteins with relaxation dispersion NMR spectroscopy: how well can we do? *J. Am. Chem. Soc.*, **130**(8), 2667–2675. ([10.1021/ja078337p](https://doi.org/10.1021/ja078337p))

More details can be seen in the `README` file in that directory. By using the data in this directory and following the instructions below, a complete dispersion analysis can be performed.

Note that the dispersion analysis in the GUI uses the automated protocol as implemented in the `auto_analyses/relax_disp.py` relax script. If you wish to perform a custom analysis or implement your own protocol, please use the prompt/scripting user interface instead to have access to the full flexibility and power of relax.

When running the analysis, it is best to keep a permanent log of all of the printouts, warnings, errors and messages produced by relax. Such information will be an invaluable reference when comparing the non-clustered and clustered results, as well as understanding what happened to each spin system under study. This can be achieved by running relax with the command line options:

```
$ relax --log log --gui
```

All output from relax will then be visible both in the relax controller window (see Figure 1.9 on page 19) and in the `log` file. Other relax options can be seen by running:

```
$ relax --help
```

11.14.1 Dispersion GUI mode – two analyses

To process this test data, two separate analyses will be performed:

- The first analysis will consist of treating all spins independently from each other. This will use model selection to determine if any statistically significant relaxation dispersion is present by comparing to the ‘No Rex’ dispersion model.
- The second analysis will consist of clustering spins with similar kinetics and thermodynamics parameters (exchange rates and populations respectively) and optimising these clusters using a common set of exchange parameters.

In this example, Gary Thompson’s multiprocessor framework (see section 11.9.11 on page 196) will be used with OpenMPI on a single four-core, hyper-threaded computer with the command:

```
$ mpirun -np 8 /data/relax/relax-trunk/relax --multi='mpi4py' --log
~/tmp/dispersion/log_non_clustered --gui
```

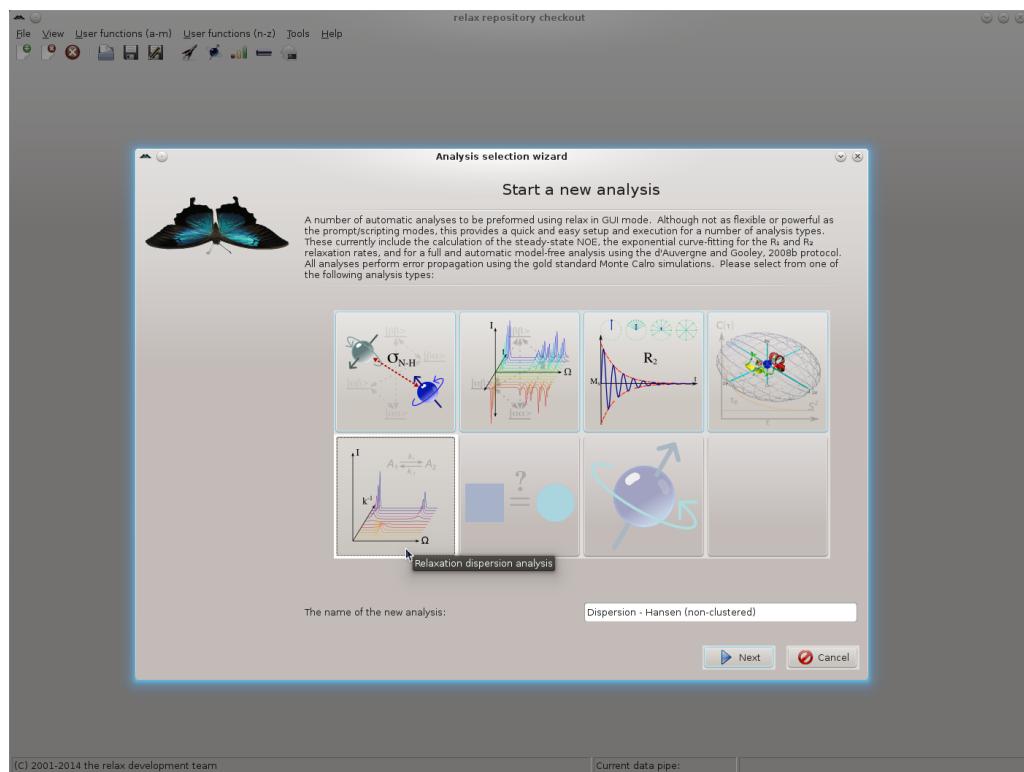
Note that the `~/tmp/dispersion` directory should be created before starting relax. If running on a system where your home directory is not defined as `~/`, MS Windows for example, the `~` part will need to be replaced with some other directory. A different command will be used for the clustered analysis to store the log messages in a separate file. The two logs will be used to compare the two analysis types at the end.

11.14.2 Dispersion GUI mode – computation time

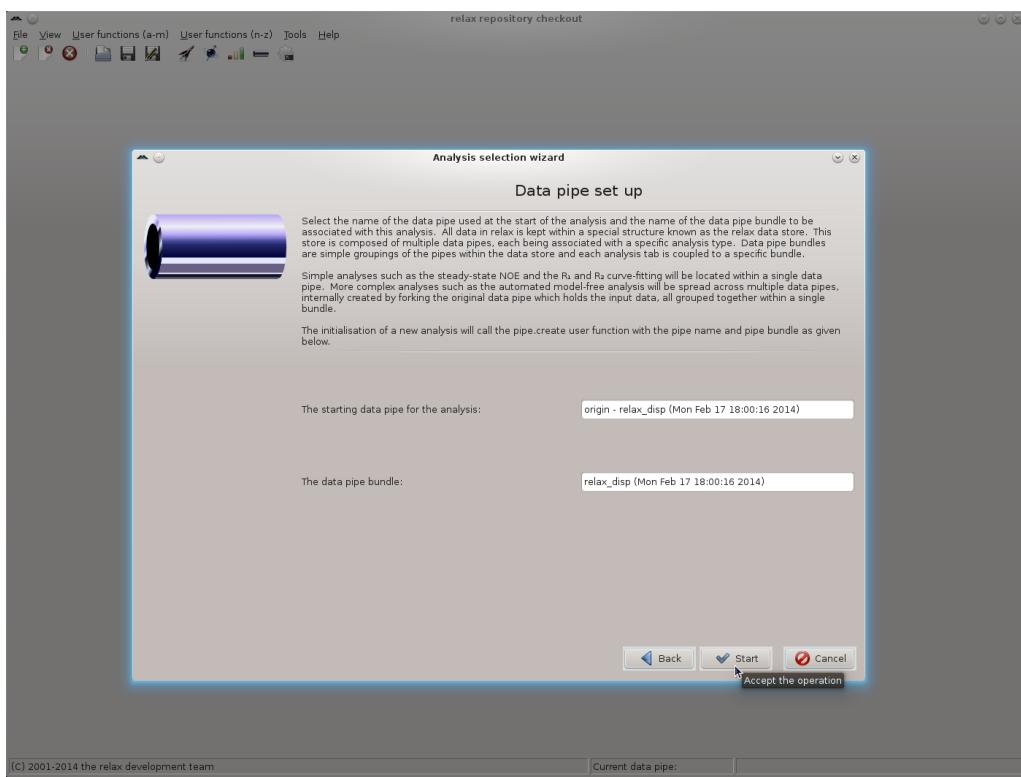
The time required to complete these two analyses is highly dependent on the computer being used as well as how many nodes can be used for running the calculations parallelised with OpenMPI. On a large cluster with many nodes both analyses should be completed in under an hour. But when running the analyses without OpenMPI on old single core computer, the analyses could take days and even up to a week.

11.14.3 Dispersion GUI mode – initialisation of the data pipe

After starting relax in the GUI mode, the dispersion analysis should be initialised by launching the analysis selection wizard (see Figure 1.4 on page 12). The relaxation dispersion analysis should be selected and the name changed if multiple dispersion analyses are to be performed within one relax session. In this case the name “Dispersion - Hansen (non-clustered)” will be used:



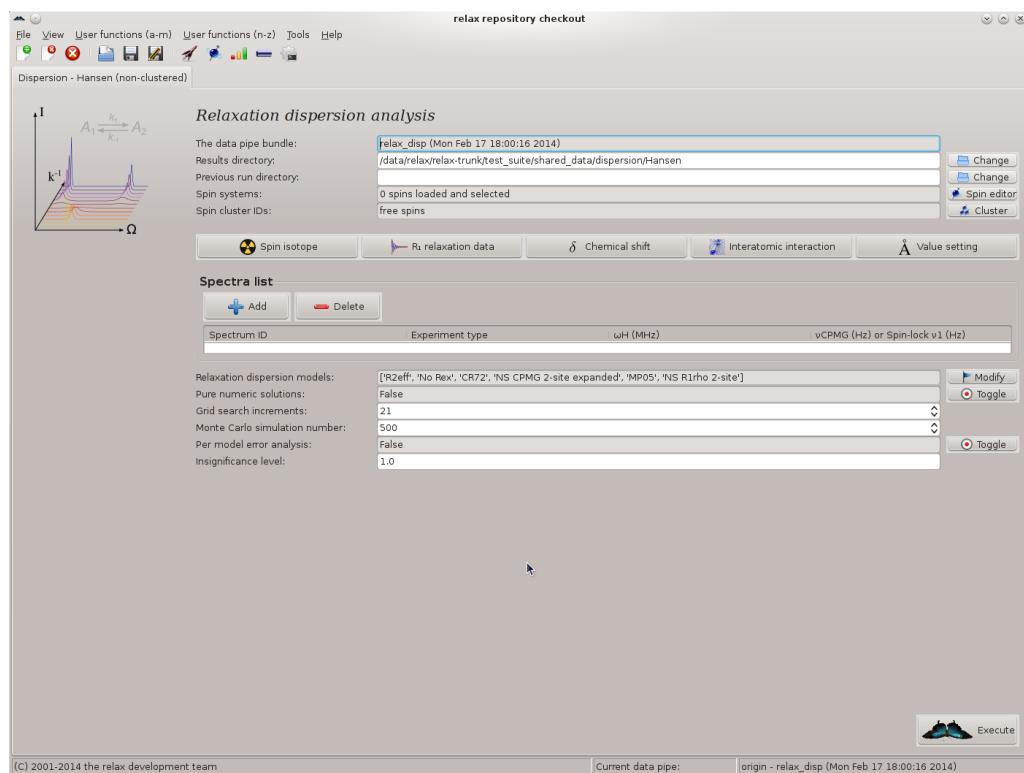
Click on the “Next” button to move to the second wizard page:



Here the values need not be changed. The data pipe bundle will be used to hold all the separate data pipes for each dispersion model type together.

11.14.4 Dispersion GUI mode – general setup

A blank analysis tab should now be visible:



The first step will be to change the “Results directory” where all of the automatically created results file, 2D Grace plots, and relax state files will be saved. The directory `~/tmp/dispersion/non_clustered` will be used for this initial non-clustered analysis.

11.14.5 Dispersion GUI mode – setting up the spin systems

As the relaxation dispersion data is specific to individual nuclear spins, the molecule, residue and spin data structures need to be set up. For this, the special “Spin systems” GUI element can be used. The initial state will be “0 spins loaded and selected”. Click on the “Spin editor” button to launch the spin viewer window. The steps for setting up the spin containers using PDB files are described in section 4.5.2 on page 42 or for sequence files in section 4.5.3 on page 45.

In this tutorial, the sequence file `fake_sequence.in` in the `test_suite/shared_data/dispersion/Hansen` directory will be loaded. In the spin loading wizard, which can be launched by clicking on the “Load spins” button, select the “From a file containing sequence data” option and click on “Next”. In the `sequence.read` user function wizard page, select the `fake_sequence.in` file. As this file only contains residue numbers and names (click on the “preview” button to see the file contents), edit the “Free format file settings” to set the residue number and name columns to 1 and 2 respectively and all other columns to blank values. Click on “Save” to store the free format settings and close the window. Back in the `sequence.read` user function wizard page, click on “Next” to load the sequence. Finally click on “Finish” to close the wizard. Do not close the spin viewer window yet. Back in the main analysis tab, the “Spin systems” GUI element will now say “73 spins loaded and selected”.

11.14.6 Dispersion GUI mode – unresolved spins

As in the prompt/script UI section 11.13.7, the spins can be deselected at this point using the same `unresolved` files. This is described in detail in section 4.5.5 on page 46.

Within the open spin viewer window, click on the “User functions→deselect→read” menu item. In the `deselect.read` user function window, select the file `test_suite/shared_data/dispersion/Hansen/500_MHz/unresolved`. As this file only contains residue number, edit the “Free format file settings” to set the residue number column to 1 and all other columns to empty values. Click the “Apply” button rather than “OK” to allow a second file to be read. The relax controller window may appear and can be closed. Select the new file `test_suite/shared_data/dispersion/Hansen/800_MHz/unresolved` and click on “OK”. Now all spins from these two files will be deselected and skipped in the analysis.

11.14.7 Dispersion GUI mode – dispersion setup

The next step is to set up the data defining the relaxation dispersion process. This is performed by clicking on the buttons between the “Spin cluster ID” and “Spectra list” GUI elements. The buttons are as follows:

Spin isotope: This is needed to specify the spin isotope information for each spin in the system, for example if the data is from ^{15}N , ^{13}C , ^1H , etc. The button launches the `spin.isotope` user function (see section 17.2.226 on page 619).

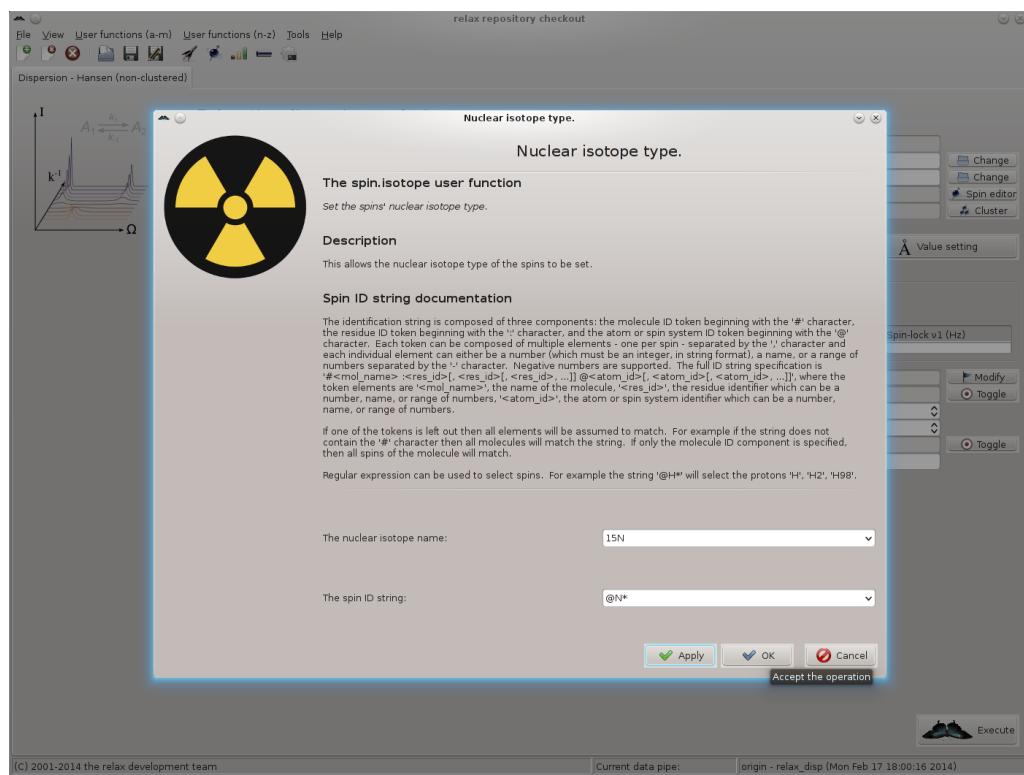
R_1 relaxation data: This is used to load a text file containing R_1 relaxation data for each spin of interest (see Chapter 5 for calculating R_1 values). The button launches the `relax_data.read` user function (see section 17.2.155 on page 572). It is currently only used for handling off-resonance effects in the $R_{1\rho}$ -type dispersion data but may be extended, in the future, to handle off-resonance effects in the CPMG-type experiments.

Chemical shift: As with the R_1 button, this is for properly handling off-resonance effects in the $R_{1\rho}$ -type dispersion data. The button launches the `chemical_shift.read` user function (see section 17.2.23 on page 459).

Interatomic interaction: This button launches the `interatom.define` user function (see section 17.2.58 on page 489). This is needed to link, for example, ^1H to ^{15}N spins into a spin system so that MMQ-type data, which relies on a multiple-spin system, can be handled.

Value setting: When one of the parameters of the dispersion model is already known, its value can be set by clicking on this button which launches the `value.set` user function (see section 17.2.268 on page 656). Setting a parameter value will fix the parameter in the initial grid search. However it will be free to vary in the non-linear least squares optimisation used to refine its value.

For SQ CPMG-type dispersion data, only the spin isotope information is required:



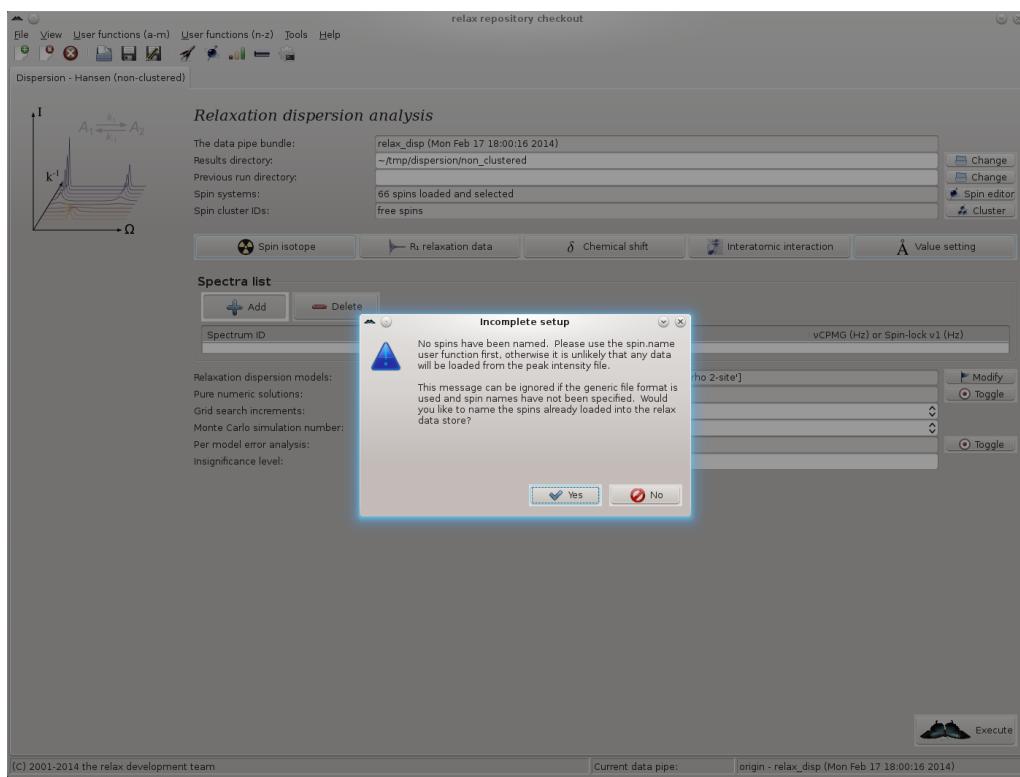
In this case, all spins are ^{15}N . However as no spins are named yet, the text “@N*” cannot be used. So simply change this to “@*” and click on “OK”. The other buttons can be ignored.

11.14.8 Dispersion GUI mode – loading the data

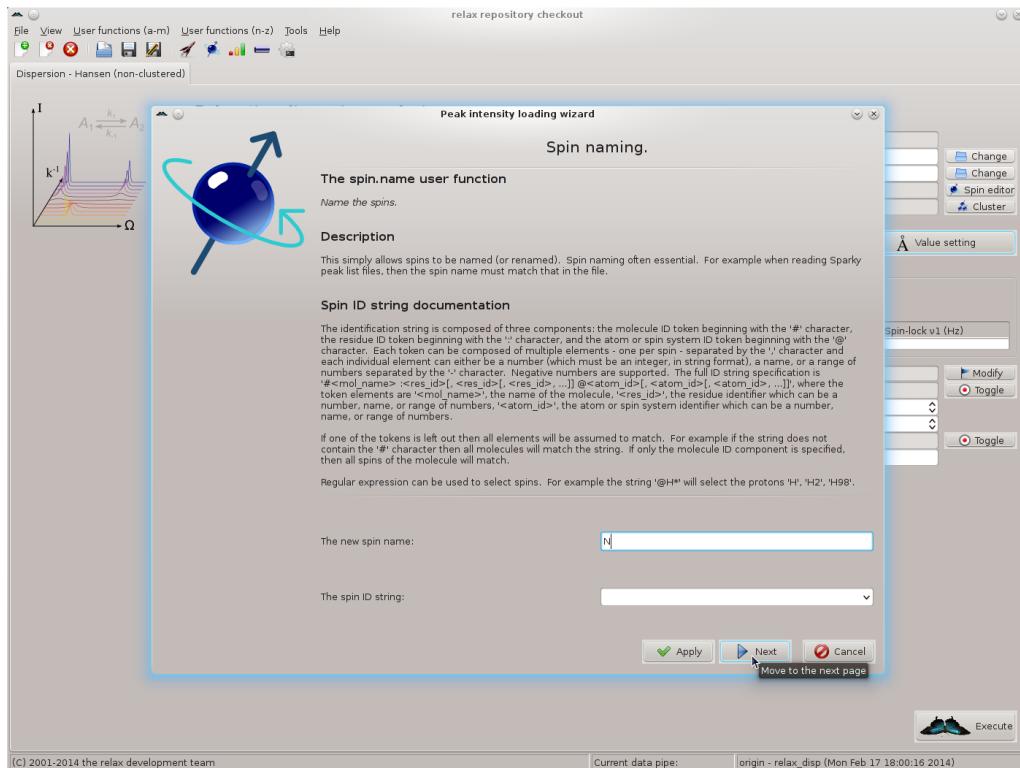
The standard way for handling experimental NMR data for starting a relaxation dispersion analysis in the relax GUI is to load the peak intensity values (either height or volume) from a peak list. For a list of all the currently supported peak list formats, see the `spectrum.read_intensities` user function documentation in section 17.2.216 on page 611.

Note that relax also accepts pre-fitted or pre-calculated $R_{2\text{eff}}$ or $R_{1\rho}$ values via the `relax_disp.r2eff_read` and `relax_disp.r2eff_read_spin` user functions (see section 17.2.175 on page 584 and section 17.2.176 on page 585 respectively), however this is not the standard way of using the GUI. As this is not tested, if you decide to work with pre-calculated relaxation rates please report any bugs encountered as described in section 3.4 on page 31. To access the user functions, click on “User functions→relax_disp→r2eff_read” or “User functions→relax_disp→r2eff_read_spin”.

In this tutorial, the Sparky formatted peak lists in the `test_suite/shared_data/disperssion/Hansen/500_MHz` and `test_suite/shared_data/disperssion/Hansen/800_MHz` directories will be loaded. First click on the “Add” button in the “Spectra list” GUI element. A warning message will appear as the spins have not yet been named, spin names were not present in the `fake_sequence.in` file, hence they cannot be matched to the data in the peak lists:



Simply click on “Yes” to allow the spins to be named in the next step. The `spin.name` user function wizard page should now appear. As all data is from ^{15}N spins and these spins have been named as “N” in the Sparky peak lists, set the new spin name to “N” and click on “Next”:



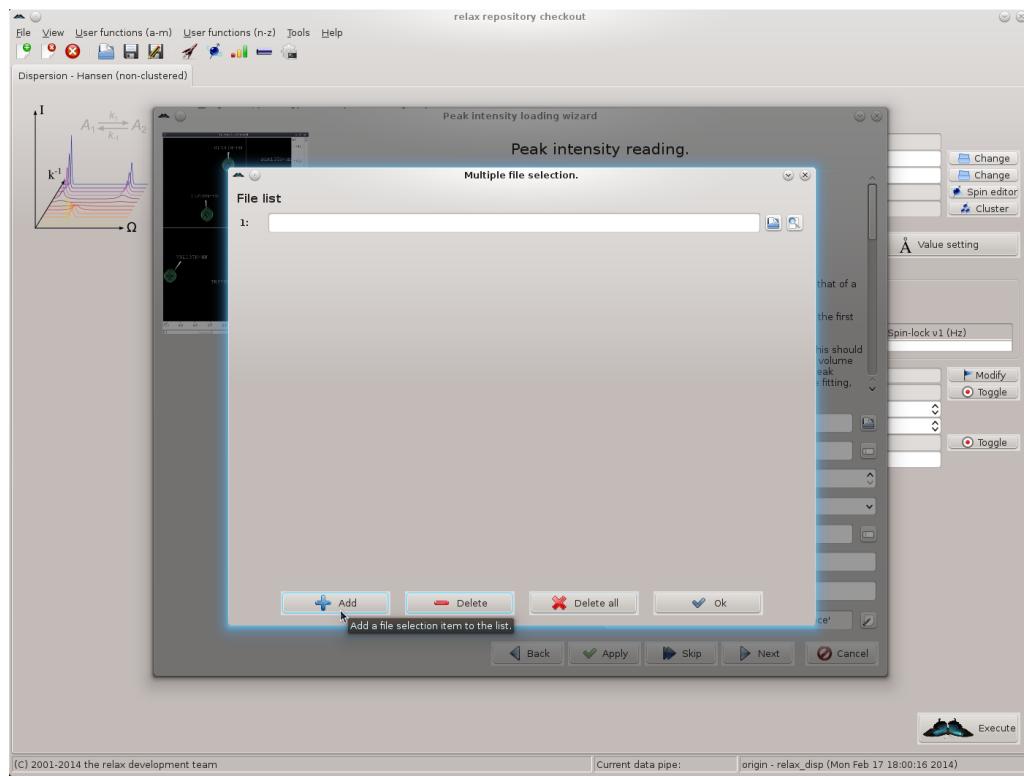
The `spectrum.read_intensities` user function wizard page should appear. To help un-

derstand the next steps of the peak intensity loading wizard, note the following table which summarises the peak list metadata for this analysis:

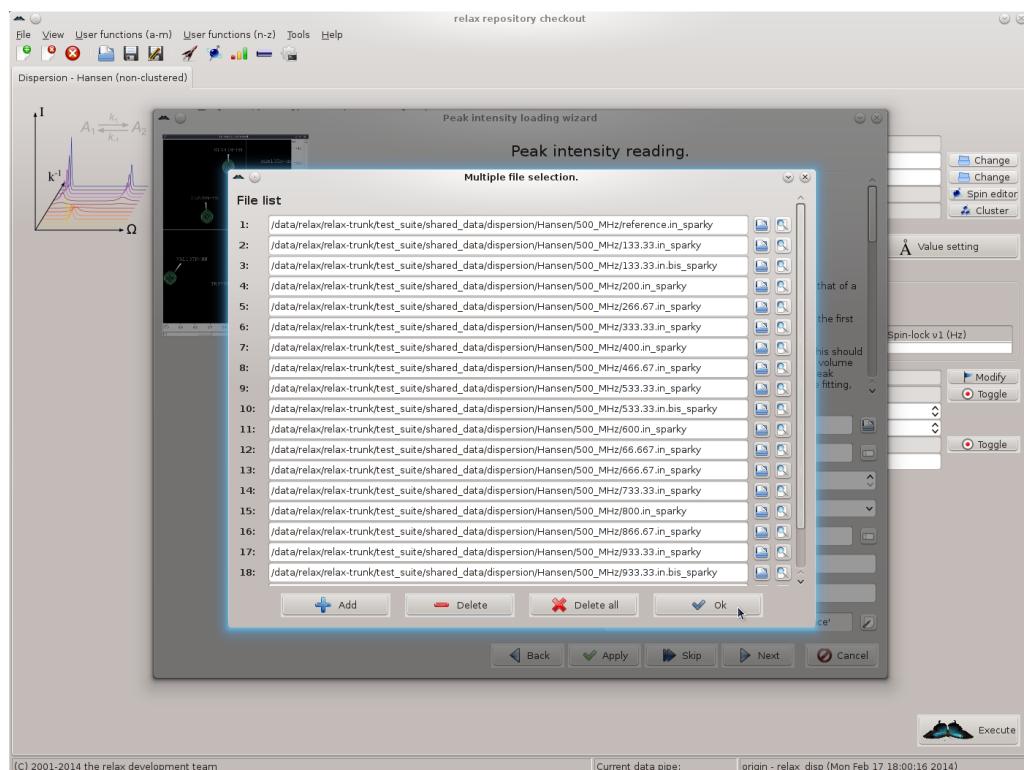
File name	Spectrum ID	Spectrometer frq. (MHz)	ν_{CPMG} (Hz)	Replicate IDs
500_MHz/reference.in_sparky	500_ref	500.0	None	
500_MHz/66.667.in_sparky	500_66.667	500.0	66.6666	
500_MHz/133.33.in_sparky	500_133.33	500.0	133.3333	500_133.33b
500_MHz/133.33.in.bis_sparky	500_133.33b	500.0	133.3333	500_133.33
500_MHz/200.in_sparky	500_200.00	500.0	200.0000	
500_MHz/266.67.in_sparky	500_266.67	500.0	266.6666	
500_MHz/333.33.in_sparky	500_333.33	500.0	333.3333	
500_MHz/400.in_sparky	500_400.00	500.0	400.0000	
500_MHz/466.67.in_sparky	500_466.67	500.0	466.6666	
500_MHz/533.33.in_sparky	500_533.33	500.0	533.3333	500_533.33b
500_MHz/533.33.in.bis_sparky	500_533.33b	500.0	533.3333	500_533.33
500_MHz/600.in_sparky	500_600.00	500.0	600.0000	
500_MHz/666.67.in_sparky	500_666.67	500.0	666.6666	
500_MHz/733.33.in_sparky	500_733.33	500.0	733.3333	
500_MHz/800.in_sparky	500_800.00	500.0	800.0000	
500_MHz/866.67.in_sparky	500_866.67	500.0	866.6666	
500_MHz/933.33.in_sparky	500_933.33	500.0	933.3333	500_933.33b
500_MHz/933.33.in.bis_sparky	500_933.33b	500.0	933.3333	500_933.33
500_MHz/1000.in_sparky	500_1000.0	500.0	1000.0000	
800_MHz/reference.in_sparky	800_ref	800.0	None	
800_MHz/66.667.in_sparky	800_66.667	800.0	66.6666	
800_MHz/133.33.in_sparky	800_133.33	800.0	133.3333	800_133.33b
800_MHz/133.33.in.bis_sparky	800_133.33b	800.0	133.3333	800_133.33
800_MHz/200.in_sparky	800_200.00	800.0	200.0000	
800_MHz/266.67.in_sparky	800_266.67	800.0	266.6666	
800_MHz/333.33.in_sparky	800_333.33	800.0	333.3333	
800_MHz/400.in_sparky	800_400.00	800.0	400.0000	
800_MHz/466.67.in_sparky	800_466.67	800.0	466.6666	
800_MHz/533.33.in_sparky	800_533.33	800.0	533.3333	800_533.33b
800_MHz/533.33.in.bis_sparky	800_533.33b	800.0	533.3333	800_533.33
800_MHz/600.in_sparky	800_600.00	800.0	600.0000	
800_MHz/666.67.in_sparky	800_666.67	800.0	666.6666	
800_MHz/733.33.in_sparky	800_733.33	800.0	733.3333	
800_MHz/800.in_sparky	800_800.00	800.0	800.0000	
800_MHz/866.67.in_sparky	800_866.67	800.0	866.6666	
800_MHz/933.33.in_sparky	800_933.33	800.0	933.3333	800_933.33b
800_MHz/933.33.in.bis_sparky	800_933.33b	800.0	933.3333	800_933.33
800_MHz/1000.in_sparky	800_1000.0	800.0	1000.0000	

For all of these files, the experiment type is SQ CPMG and the relaxation delay time is 30 ms. All files are located in the `test_suite/shared_data/dispersion/Hansen` base directory.

To simplify the loading of the data, all of the 500 MHz data will be read simultaneously (followed by all of the 800 MHz data). Firstly, to select all of the 500 MHz files, click on the button at the end of the “file name(s)” GUI element. This will launch the multiple file selection window:

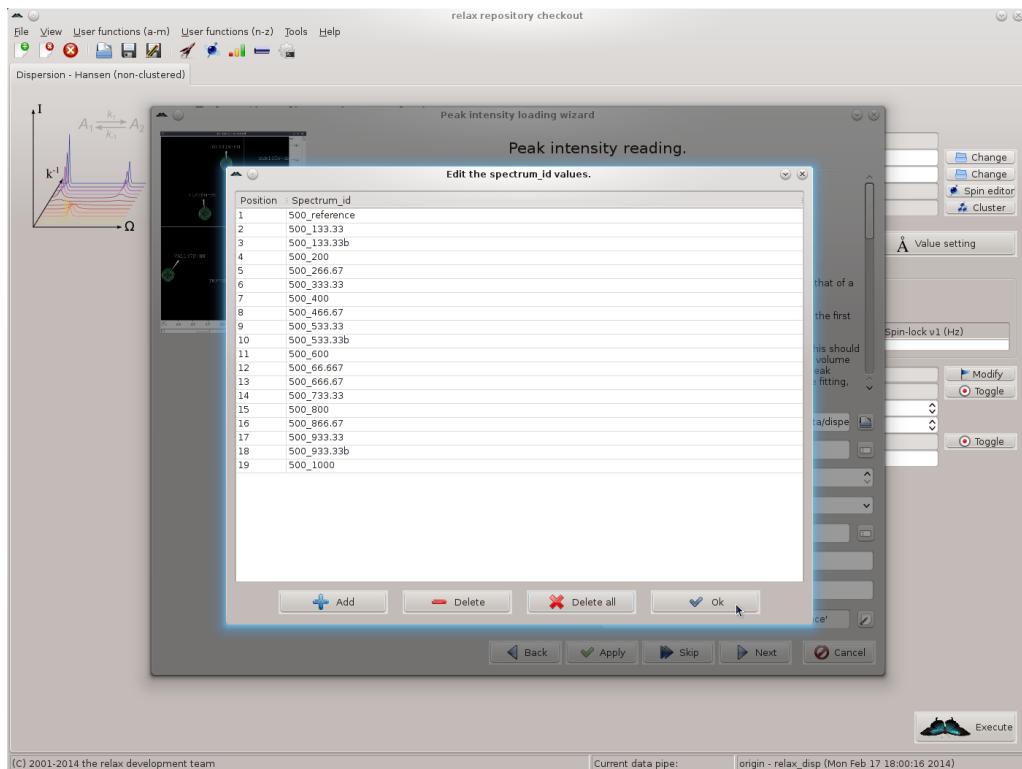


Add the first file by typing the file name or clicking on the file selection button to the right of the file input field and choosing the file. A preview button is included to allow the file to be checked. Then click on the “Add” button to insert a new file selection item, add the next file, and continue until completion. You should now see 19 file names:



Click on “OK” to return to the `spectrum.read_intensities` user function wizard page.

Next, the spectrum IDs should be set. For this, click on the button to the far right of the “spectrum ID string” GUI element. Click on “Add” and fill in the spectrum IDs corresponding to the files selected. As 19 files exist for the 500 MHz data, 19 spectrum IDs should be added. When complete you should see:



Click on “OK” to return to the `spectrum.read_intensities` user function wizard page.

There is a shortcut to these previous steps in that the file names and spectrum IDs can be manually typed into the input elements of the `spectrum.read_intensities` user function wizard page. For the file names, assuming relax is installed in the base directory `/data/relax/relax-trunk` and a Unix-like system is being used, the following text can be added:

```
'/data/relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/
reference.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/dispersion/
Hansen/500_MHz/133.33.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/
dispersion/Hansen/500_MHz/133.33.in.bis_sparky', '/data/relax/relax-trunk/test_suite/
shared_data/dispersion/Hansen/500_MHz/200.in_sparky', '/data/relax/relax-trunk/
test_suite/shared_data/dispersion/Hansen/500_MHz/266.67.in_sparky', '/data/relax/relax-
trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/333.33.in_sparky', '/data/
relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/400.in_sparky', '/
data/relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/466
.67.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500
_MHz/533.33.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/dispersion/
Hansen/500_MHz/533.33.in.bis_sparky', '/data/relax/relax-trunk/test_suite/shared_data/
dispersion/Hansen/500_MHz/600.in_sparky', '/data/relax/relax-trunk/test_suite/
shared_data/dispersion/Hansen/500_MHz/66.667.in_sparky', '/data/relax/relax-trunk/
test_suite/shared_data/dispersion/Hansen/500_MHz/666.67.in_sparky', '/data/relax/relax-
trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/733.33.in_sparky', '/data/
relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/800.in_sparky', '/
data/relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500_MHz/866
.67.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/dispersion/Hansen/500
_MHz/933.33.in_sparky', '/data/relax/relax-trunk/test_suite/shared_data/dispersion/
Hansen/500_MHz/933.33.in.bis_sparky', '/data/relax/relax-trunk/test_suite/shared_data/
```

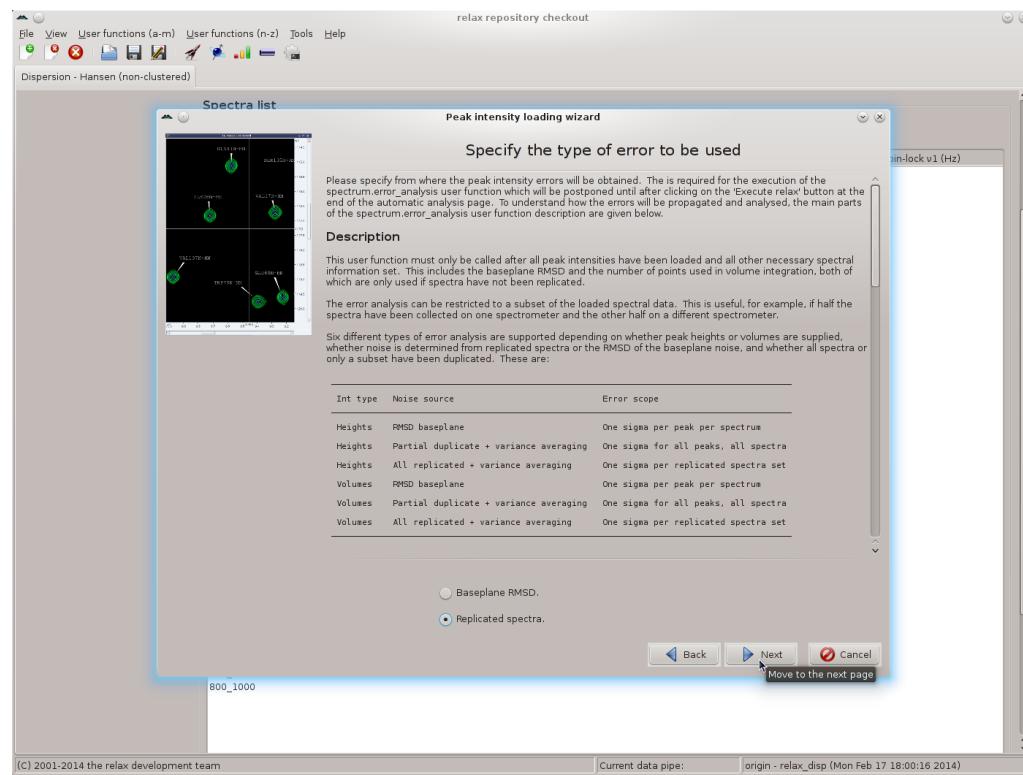
```
dispersion/Hansen/500_MHz/1000.in_sparky']
```

The spectrum IDs can be added by copying and pasting the following text, making sure all text is on one line:

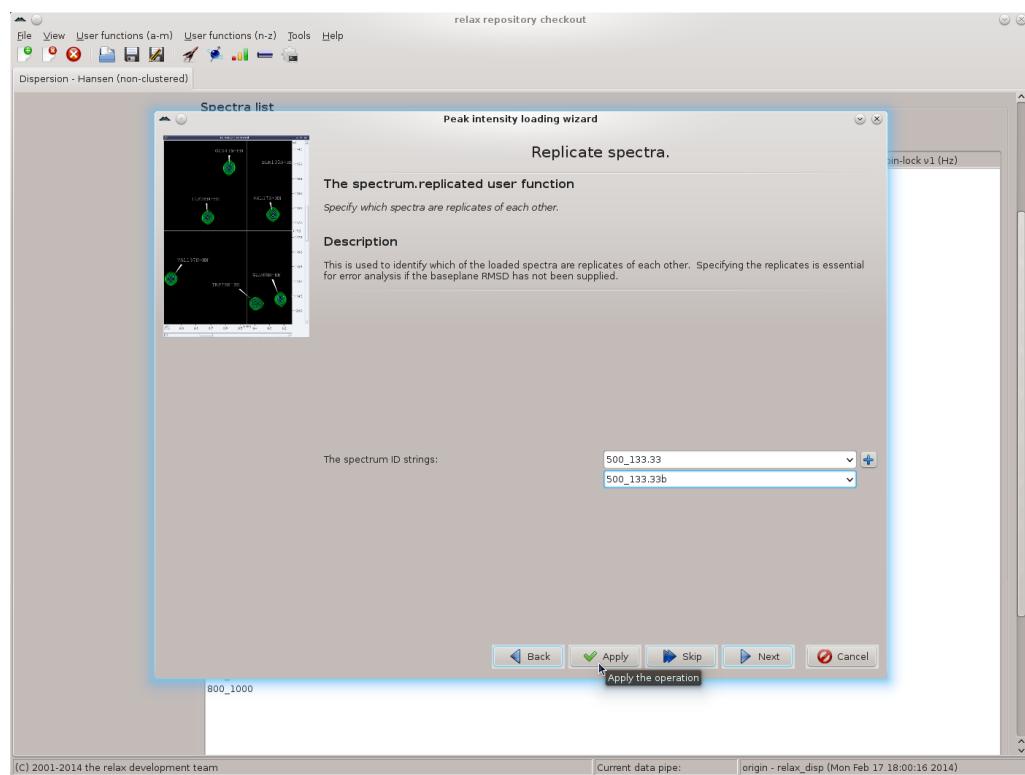
```
['500_reference', '500_133.33', '500_133.33b', '500_200', '500_266.67', '500_333.33', '500_400', '500_466.67', '500_533.33', '500_533.33b', '500_600', '500_66.667', '500_666.67', '500_733.33', '500_800', '500_866.67', '500_933.33', '500_933.33b', '500_1000']
```

To see if the data has been correctly entered, click on the buttons to the right of the GUI elements. You should see the files and spectrum IDs correctly listed in the multiple file selection window and the sequence input window respectively.

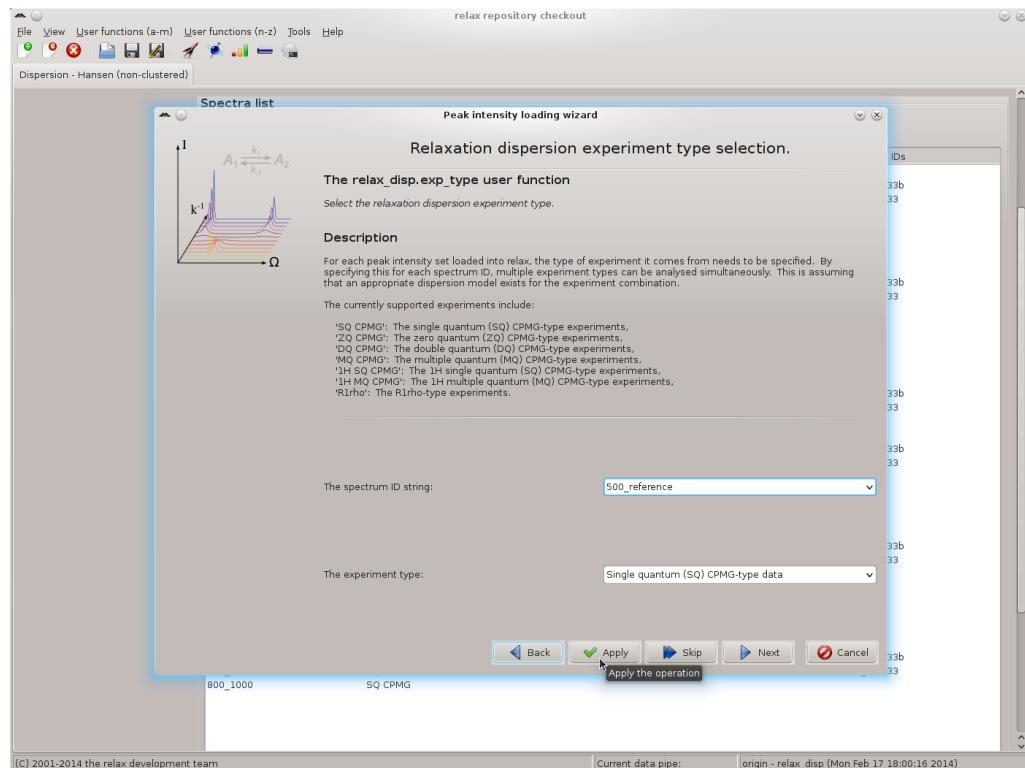
For the SQ CPMG data of this tutorial, none of the other settings in the `spectrum.read_intensities` user function wizard page need to be changed. By clicking on the “Apply” button, the 500 MHz data will be loaded but the wizard will stay on the same page. Note that the contents of the main relax window will have changed. Clicking “Apply” rather than “Next” will allow the 800 MHz data to be loaded next. Change all of the file names and spectrum IDs for the 800 MHz and click on the “Next” button to read the data and to move to the next wizard page. This will be the page for specifying the types of errors to use:



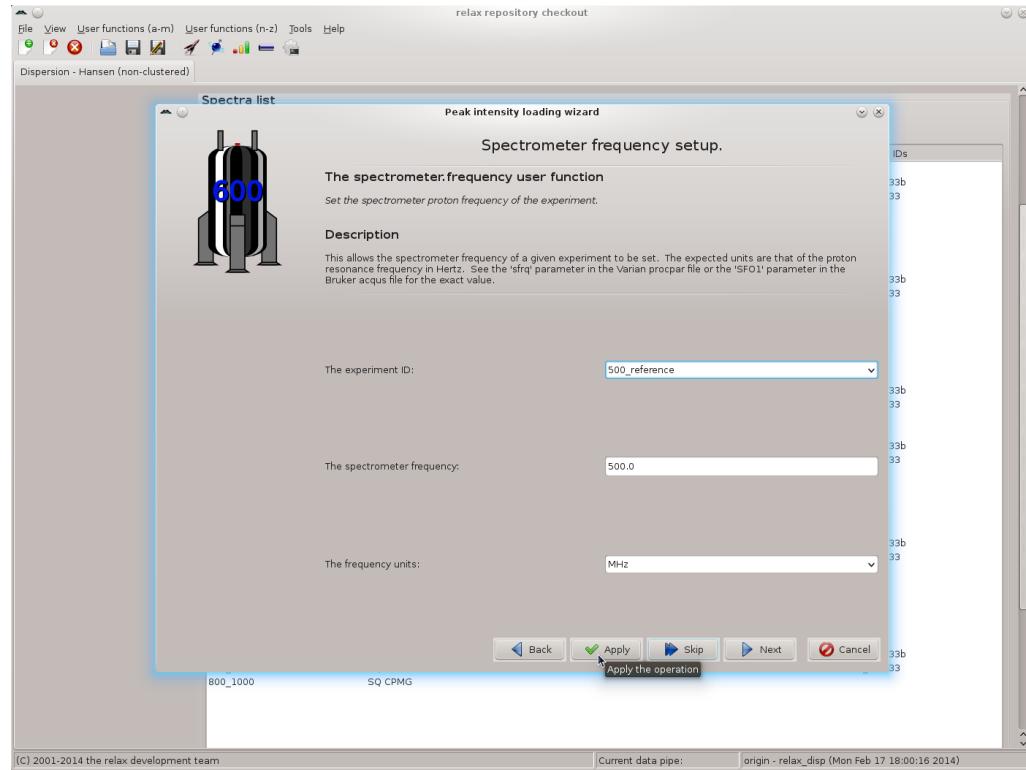
As replicated spectra have been collected, set the value to “Replicated spectra” and then click the “Next” button. Select the pairs of spectra which have been replicated and click the “Apply” button to allow all of the replicates to be specified:



Repeat for each pair of replicates, then click the “Next” button to move to the next wizard page – the `relax_disp.exp_type` user function. Here the tedious operation of labelling all spectrum IDs as being “Single quantum (SQ) CPMG-type data” must be performed. Select the spectrum ID and the SQ CPMG data type and click on the “Apply” button, repeating for each ID:



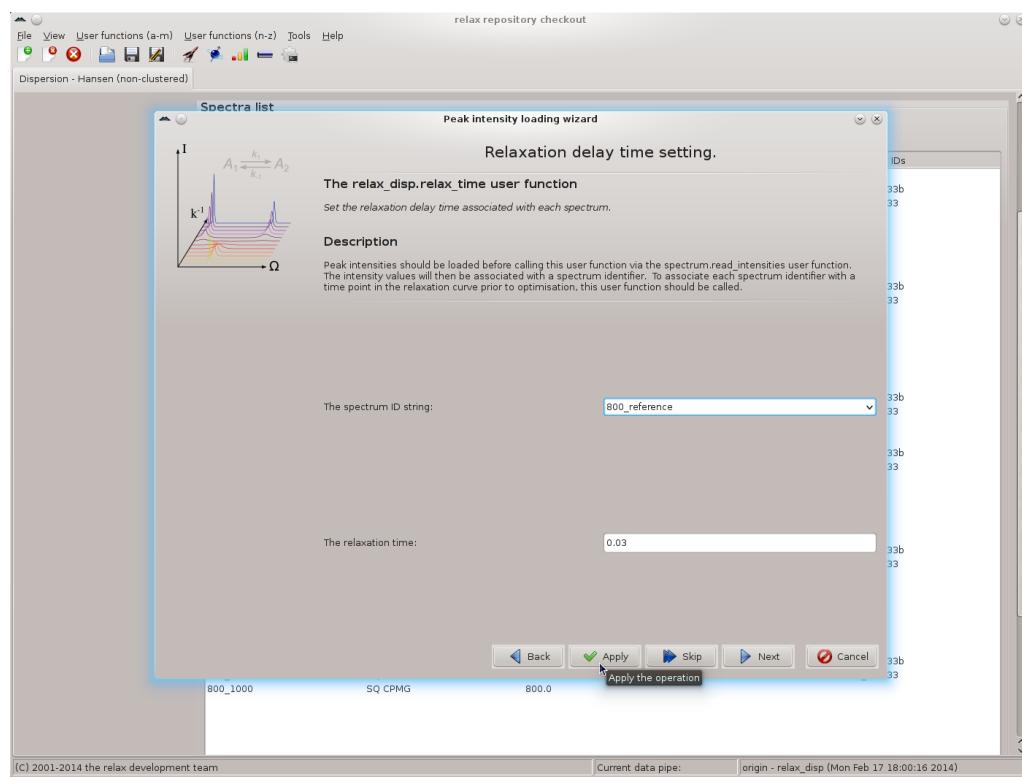
Click the “Next” button when finished. The next wizard page will be that of the `spectrometer.frequency` user function. Here the exact spectrometer frequency values should be specified. These values should be those of the “`sfrq`” parameter in the Varian `procpar` file or the “`SF01`” parameter in the Bruker `acqus` file. As the exact values are not known for the data of this tutorial, the values of 500.0 and 800.0 MHz will be used:



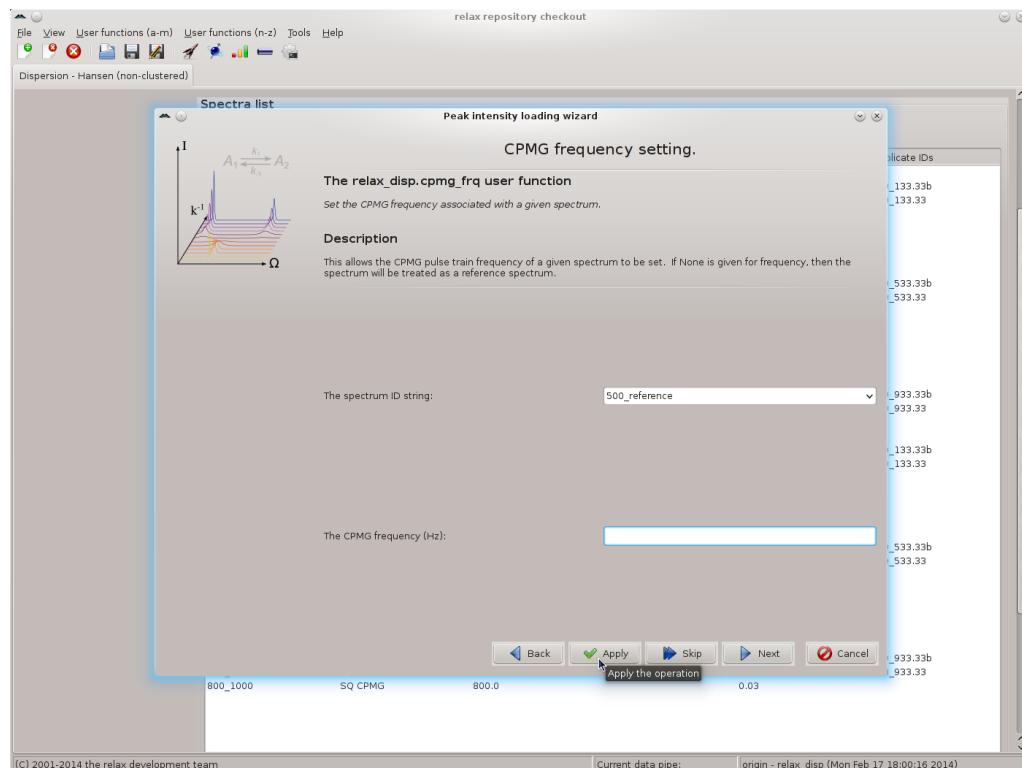
Again use the “Apply” button to set the value, repeating for all matching spectrum IDs. Change the frequency to 800.0 MHz and continue for the next set of matching spectrum IDs. Finally click on “Next” to move to the next wizard page. If you have not used the exact values from the files, the relax controller window will appear with the warning:

```
relax> spectrometer.frequency(id='500_reference', frq=500.0, units='MHz')
RelaxWarning: The precise spectrometer frequency should be supplied, a value such as
500000000 or 5e8 for a 500 MHz machine is not acceptable. Please see the 'sfrq'
parameter in the Varian procpar file or the 'SF01' parameter in the Bruker acqus file.
```

The controller window can be safely closed. Though, to avoid frustration when setting the frequency for all spectrum IDs, it may be better to shift the window to the side of the screen. The next wizard page is that of the `relax_disp.relax_time` user function. Set the time to 0.03 and click on the “Apply” button for all spectrum IDs, one after the other:

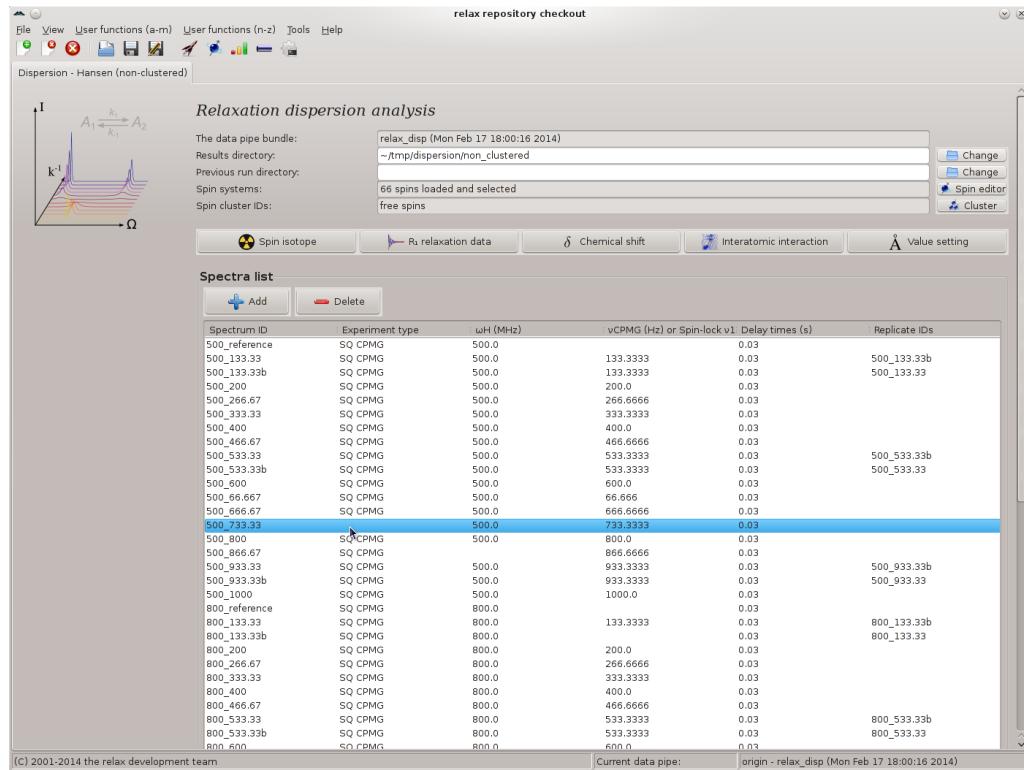


Click on “Next” to move to the `relax_disp.cpmg_frq` user function wizard page. If $R_{1\rho}$ data has been collected, the `relax_disp.spin_lock_field` user function wizard page would appear instead. For the reference spectrum IDs, leave the CPMG frequency value blank and click on “Apply”:



This will label these spectra as the reference. For all other spectrum IDs, use the CPMG

frequencies as given in the table above, using “Apply” to execute the `relax_disp.cpmg-freq` user function while staying on the same wizard page. Click on “Next” to finish. You should now see the main relax window:



In this screenshot it can be seen that some of the metadata is missing. This often happens due to the large amounts of metadata specified in the peak intensity loading wizard and human error. The missing metadata can now easily be filled in by right clicking on the corresponding row. For example here the “500_733.33” spectrum ID does not have the experiment type set. Simply right click and select the “Set the experiment type” menu entry from the popup menu. For the missing spectrometer frequency for the “500_866.67” spectrum ID, the “Set the spectrometer frequency” menu entry can be used. For the missing ν_{CPMG} value for the “800_133.33” spectrum ID, the “Set the CPMG pulse frequency ν_{CPMG} ” menu entry can be used. All of the metadata should be double and triple checked and fixed where required using the popup menu. Any errors in this metadata would be catastrophic for the subsequent dispersion analysis.

11.14.9 Dispersion GUI mode – choosing the models to optimise

The next step is to specify which relaxation dispersion models will be used in the analysis. The number of models used should be limited based on your knowledge of the problem. Depending on experimental information, a number of models can be ruled out. For example for the data used in this tutorial, the exchange is known to be slow. Hence all of the fast exchange models can be excluded. For the conditions under which the various dispersion models can be used, please refer to the original references. In relax many dispersion models are provided, however you should never use them all.

For this tutorial, we will only use the following dispersion models:

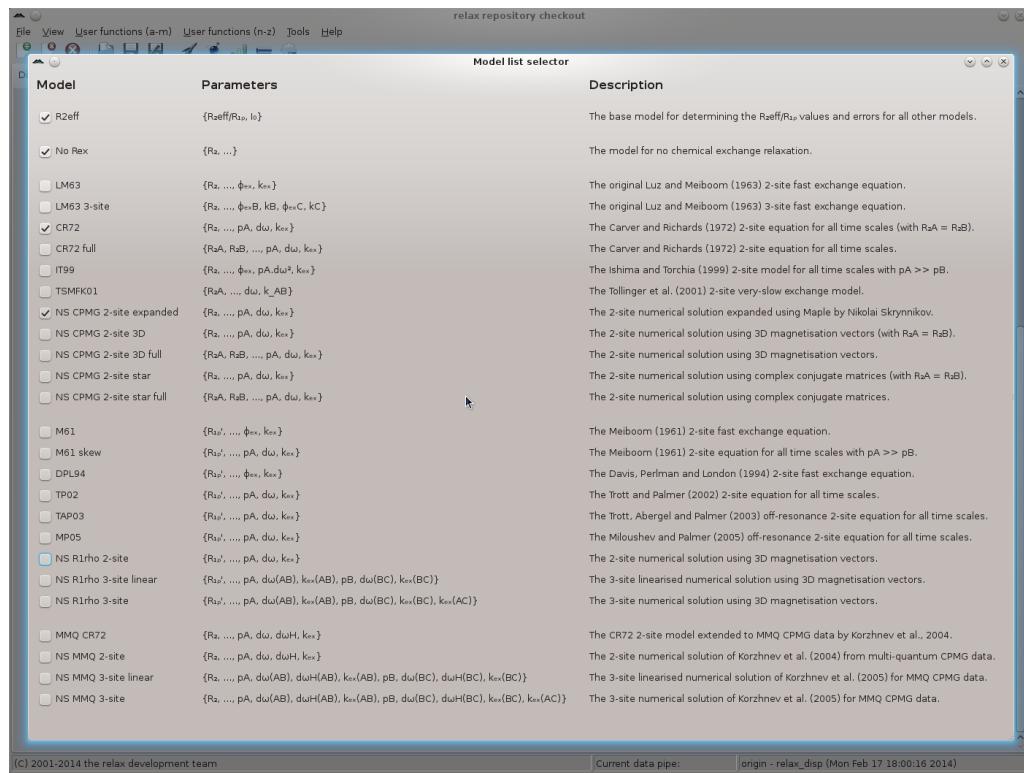
'R_{2eff}': This is essential for calculating the R_{2eff} values used as the input data for all other models (see section 11.2.1 on page 156).

'No Rex': This model will be important in the model selection stage to determine if any statistically significant relaxation dispersion is present in the R_{2eff} data (see section 11.2.2 on page 158).

'CR72': The Carver and Richards model for most time scales (see Section 11.3.4 on page 162). This is the standard model for slow timescale CPMG-type data (for fast timescales use the 'LM63' model instead and for very slow exchange the 'TSMFK01' model).

'NS CPMG 2-site expanded': The best and fastest of the numerical models for 2-site exchange (see Section 11.4.1 on page 166).

To chose which models will be used in the analysis, click on the "Modify" button of the "Relaxation dispersion models" GUI element in the main relax window and change the model list to:



The "Pure numerical solution" flag will be left on "False". But if you prefer to only have results from the numerical models, this can be changed to "True" by clicking on the "Toggle" button (note that the 'CR72' model should nevertheless be used to speed up optimisation – see section 11.9.5 on page 191 for details).

11.14.10 Dispersion GUI mode – optimisation settings

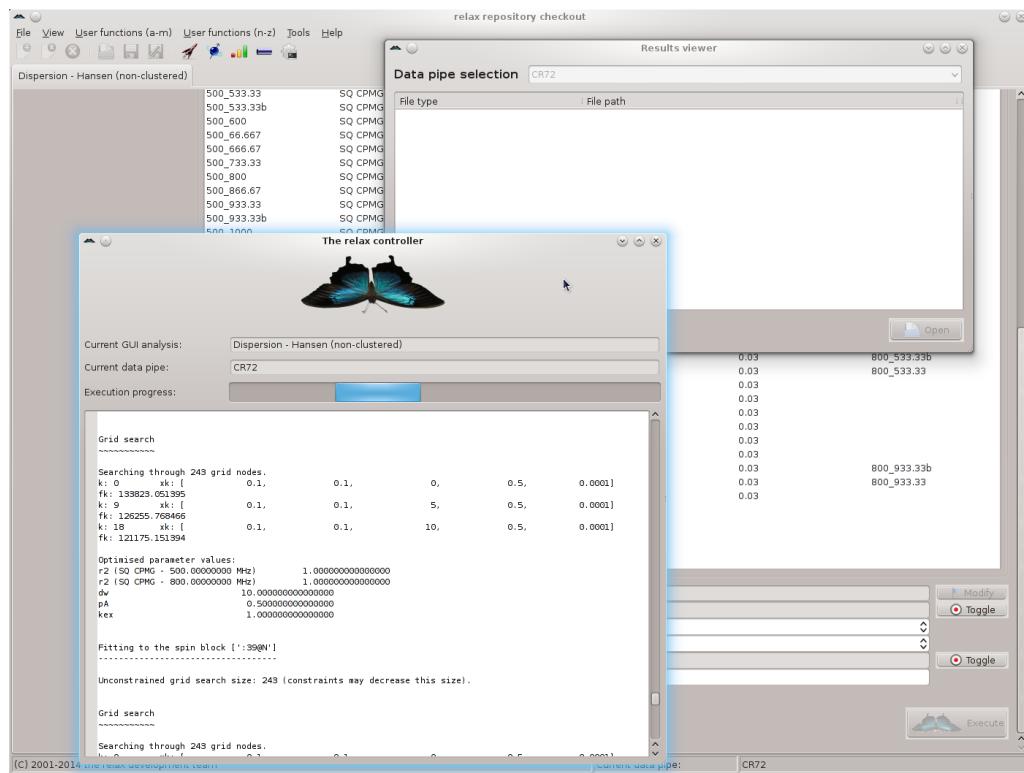
The grid search increment number of 21 might be excessive, especially if the multiprocessor framework is not used to speed up calculations (see section 11.9.11 on page 196). This

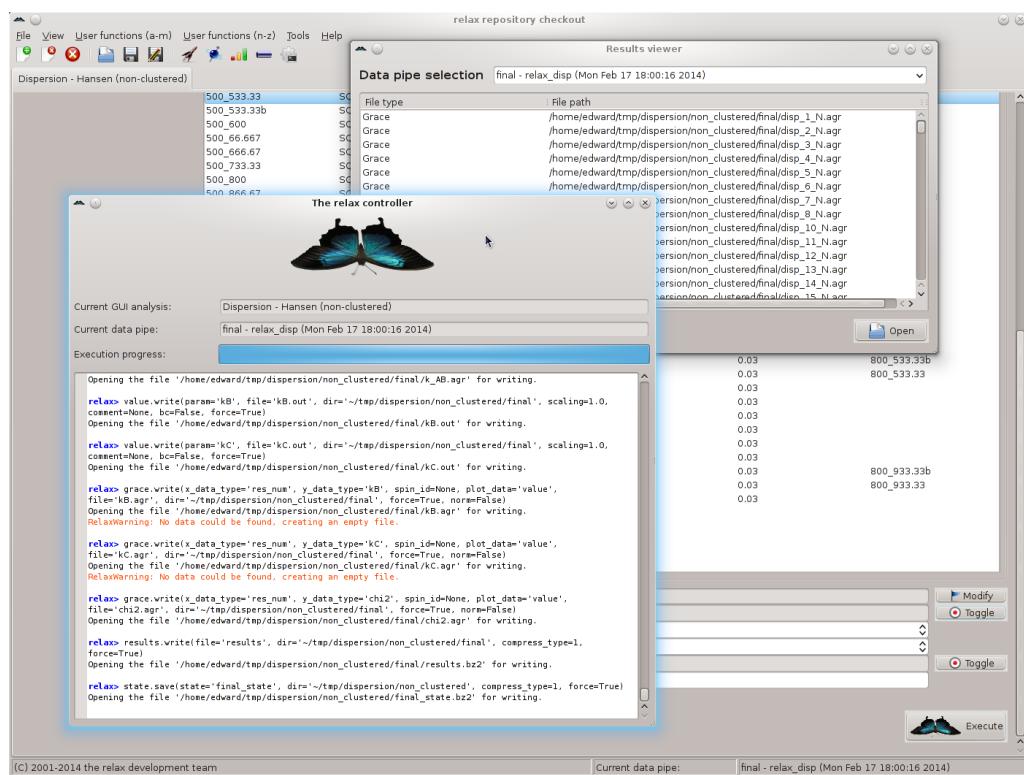
value can be changed to 11, as this number is probably sufficient for most dispersion analyses.

Also for speed, the number of Monte Carlo simulations will be set to 200. However for real analyses, the minimum number of 500 should be used for accurate parameter error estimates. The “Per model error analysis” setting will be left to “False”. And the “Insignificance level” will be left at 1.0 (see section 11.9.2 on page 190).

11.14.11 Dispersion GUI mode – execution of the non-clustered analysis

To start the analysis, simply click the “Execute” button to start the analysis:





The relax controller window can be closed. At all stages of the analysis, the logs should be checked for all warnings. For example searching for the text ‘RelaxWarning’ in the log file `~/dispersion/log_non_clustered`, the only warnings can be seen at the end of the analysis. These can be seen to be harmless as they are indicating that certain Grace 2D plots contain no data – specifically those for dispersion parameters not present in the set of models optimised. Any other warnings should however be very carefully noted and checked as these could be pointing to a serious problem.

11.14.12 Dispersion GUI mode – inspection of the results

To view the results of the analysis, the relax results window will have been automatically opened. If it was closed, click on the “View→Results viewer” menu entry or the “Results viewer window” button in the toolbar to open the window. This window will list all of the `*.out` text files and `*.agr` 2D Grace plots created by the auto-analysis. By double clicking on the file, these can be opened. The operating specific text editor will be launched for the text files. For the Grace files, the installed Grace version (Xmgrace, QtGrace, or GraceGTK) will be launched.

Firstly, to see which models have been chosen for the spin systems and which have no statistically significant dispersion, find the `model.out` file and double click on it. You should see the following:

# Parameter description: The dispersion model.						
#	# mol_name	res_num	res_name	spin_num	spin_name	value
None	1	GLY	None	N		'NS CPMG 2-site expanded'
None	2	GLY	None	N		'No Rex'
None	3	GLY	None	N		'No Rex'
None	4	GLY	None	N		'No Rex'
None	5	GLY	None	N		'CR72'

None	6	GLY	None	N	'No Rex'	None
None	7	GLY	None	N	'No Rex'	None
None	8	GLY	None	N	'NS CPMG 2-site expanded'	None
None	9	GLY	None	N	None	None
None	10	GLY	None	N	'No Rex'	None
None	11	GLY	None	N	'No Rex'	None
None	12	GLY	None	N	'NS CPMG 2-site expanded'	None
None	13	GLY	None	N	'No Rex'	None
None	14	GLY	None	N	'CR72'	None
None	15	GLY	None	N	'No Rex'	None
None	16	GLY	None	N	'No Rex'	None
None	17	GLY	None	N	'No Rex'	None
None	18	GLY	None	N	'No Rex'	None
None	19	GLY	None	N	'No Rex'	None
None	20	GLY	None	N	'No Rex'	None
None	21	GLY	None	N	'CR72'	None
None	22	GLY	None	N	'No Rex'	None
None	23	GLY	None	N	'CR72'	None
None	24	GLY	None	N	'No Rex'	None
None	25	GLY	None	N	'No Rex'	None
None	26	GLY	None	N	'CR72'	None
None	27	GLY	None	N	None	None
None	28	GLY	None	N	None	None
None	29	GLY	None	N	'No Rex'	None
None	30	GLY	None	N	None	None
None	31	GLY	None	N	'NS CPMG 2-site expanded'	None
None	32	GLY	None	N	'NS CPMG 2-site expanded'	None
None	33	GLY	None	N	'No Rex'	None
None	34	GLY	None	N	'NS CPMG 2-site expanded'	None
None	35	GLY	None	N	'No Rex'	None
None	36	GLY	None	N	'No Rex'	None
None	37	GLY	None	N	'No Rex'	None
None	38	GLY	None	N	'NS CPMG 2-site expanded'	None
None	39	GLY	None	N	'NS CPMG 2-site expanded'	None
None	40	GLY	None	N	'No Rex'	None
None	41	GLY	None	N	'No Rex'	None
None	42	GLY	None	N	'NS CPMG 2-site expanded'	None
None	43	GLY	None	N	'No Rex'	None
None	44	GLY	None	N	'No Rex'	None
None	45	GLY	None	N	'NS CPMG 2-site expanded'	None
None	46	GLY	None	N	'NS CPMG 2-site expanded'	None
None	47	GLY	None	N	'NS CPMG 2-site expanded'	None
None	48	GLY	None	N	'No Rex'	None
None	49	GLY	None	N	'No Rex'	None
None	50	GLY	None	N	'No Rex'	None
None	51	GLY	None	N	'CR72'	None
None	52	GLY	None	N	'NS CPMG 2-site expanded'	None
None	53	GLY	None	N	'NS CPMG 2-site expanded'	None
None	54	GLY	None	N	None	None
None	55	GLY	None	N	'NS CPMG 2-site expanded'	None
None	56	GLY	None	N	'No Rex'	None
None	57	GLY	None	N	'No Rex'	None
None	58	GLY	None	N	'No Rex'	None
None	59	GLY	None	N	'CR72'	None
None	60	GLY	None	N	'No Rex'	None
None	61	GLY	None	N	'NS CPMG 2-site expanded'	None
None	62	GLY	None	N	'NS CPMG 2-site expanded'	None
None	63	GLY	None	N	'CR72'	None
None	64	GLY	None	N	'CR72'	None
None	65	GLY	None	N	'CR72'	None
None	66	GLY	None	N	'NS CPMG 2-site expanded'	None
None	67	GLY	None	N	'NS CPMG 2-site expanded'	None
None	68	GLY	None	N	None	None
None	69	GLY	None	N	None	None
None	70	GLY	None	N	'NS CPMG 2-site expanded'	None
None	71	GLY	None	N	'NS CPMG 2-site expanded'	None
None	72	GLY	None	N	'NS CPMG 2-site expanded'	None
None	73	GLY	None	N	'CR72'	None

Excluding the deselected spin systems with the model ‘None’ and those with no statistically significant dispersion, ‘No Rex’, it can be seen that in some cases the ‘CR72’ model is selected whereas in others the ‘NS CPMG 2-site expanded’ model is selected. The differences between the ‘CR72’ analytic model and the ‘NS CPMG 2-site expanded’ numeric model are insignificant. To see this, open the CR72/chi2.out and NS CPMG 2-site expanded/chi2.out text files in the `~/dispersion/log_non_clustered` directory and compare the optimised chi-squared values (this can be performed with the mouse by using the pipe editor window, changing the current data pipe, and double clicking on the files in the results viewer window for each data pipe – see below for more details).

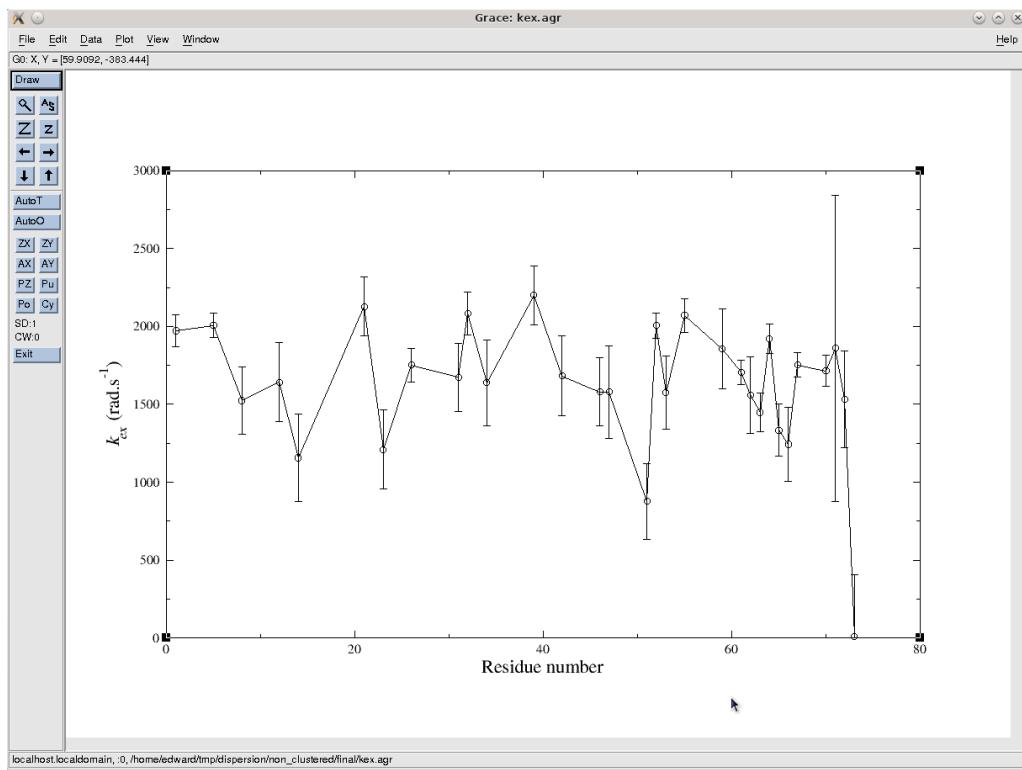
To see which spins have been assigned the model ‘No Rex’ due to the $R_{2\text{eff}}/R_{1\rho}$ significance level of 1.0 rad/s, search the log messages for ‘insignificance’. You should see:

```
relax> relax_disp.insignificance(level=1.0)
Deselecting spin ':2@N', the maximum dispersion curve difference for all curves is 0.772528040762 rad/s.
Deselecting spin ':3@N', the maximum dispersion curve difference for all curves is 0.572686080104 rad/s.
Deselecting spin ':4@N', the maximum dispersion curve difference for all curves is 0.20753288407 rad/s.
Deselecting spin ':7@N', the maximum dispersion curve difference for all curves is 0.184120905625 rad/s.
Deselecting spin ':10@N', the maximum dispersion curve difference for all curves is 0.746360942576 rad/s.
Deselecting spin ':11@N', the maximum dispersion curve difference for all curves is 0.372702361421 rad/s.
Deselecting spin ':13@N', the maximum dispersion curve difference for all curves is 0.261522940719 rad/s.
Deselecting spin ':15@N', the maximum dispersion curve difference for all curves is 0.743965404051 rad/s.
Deselecting spin ':16@N', the maximum dispersion curve difference for all curves is 0.198783344901 rad/s.
Deselecting spin ':17@N', the maximum dispersion curve difference for all curves is 0.469568638477 rad/s.
Deselecting spin ':18@N', the maximum dispersion curve difference for all curves is 0.720840385542 rad/s.
Deselecting spin ':19@N', the maximum dispersion curve difference for all curves is 0.290773963568 rad/s.
Deselecting spin ':20@N', the maximum dispersion curve difference for all curves is 0.983669594767 rad/s.
Deselecting spin ':22@N', the maximum dispersion curve difference for all curves is 0.507488886603 rad/s.
Deselecting spin ':24@N', the maximum dispersion curve difference for all curves is 0.984086643389 rad/s.
Deselecting spin ':25@N', the maximum dispersion curve difference for all curves is 0.638104572082 rad/s.
Deselecting spin ':29@N', the maximum dispersion curve difference for all curves is 0.525261970487 rad/s.
Deselecting spin ':33@N', the maximum dispersion curve difference for all curves is 0.822112754666 rad/s.
Deselecting spin ':35@N', the maximum dispersion curve difference for all curves is 0.713976877685 rad/s.
Deselecting spin ':36@N', the maximum dispersion curve difference for all curves is 0.413602640091 rad/s.
Deselecting spin ':37@N', the maximum dispersion curve difference for all curves is 0.302953864843 rad/s.
Deselecting spin ':40@N', the maximum dispersion curve difference for all curves is 0.401535026433 rad/s.
Deselecting spin ':41@N', the maximum dispersion curve difference for all curves is 0.805657060225 rad/s.
Deselecting spin ':43@N', the maximum dispersion curve difference for all curves is 0.582523964429 rad/s.
Deselecting spin ':44@N', the maximum dispersion curve difference for all curves is 0.325638582443 rad/s.
Deselecting spin ':45@N', the maximum dispersion curve difference for all curves is 0.947956877682 rad/s.
Deselecting spin ':49@N', the maximum dispersion curve difference for all curves is 0.872396631779 rad/s.
Deselecting spin ':50@N', the maximum dispersion curve difference for all curves is 0.403543891199 rad/s.
Deselecting spin ':56@N', the maximum dispersion curve difference for all curves is 0.468272490195 rad/s.
Deselecting spin ':57@N', the maximum dispersion curve difference for all curves is 0.634215047495 rad/s.
Deselecting spin ':58@N', the maximum dispersion curve difference for all curves is 0.953109267554 rad/s.
```

Search for ‘eliminate’ to see which models have been removed due to the model elimination step (see section 11.9.9 on page 195 for details). For example for the CR72 model:

```
relax> eliminate(function=None, args=None)
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50003 is less than 0.50100, eliminating the spin cluster [':@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50001 is less than 0.50100, eliminating the spin cluster [':@8@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50001 is less than 0.50100, eliminating the spin cluster [':@42@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50001 is less than 0.50100, eliminating the spin cluster [':@45@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50000 is less than 0.50100, eliminating the spin cluster [':@46@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50002 is less than 0.50100, eliminating the spin cluster [':@47@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50000 is less than 0.50100, eliminating the spin cluster [':@48@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50002 is less than 0.50100, eliminating the spin cluster [':@60@N'].
Data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)': The pA parameter of 0.50000 is less than 0.50100, eliminating the spin cluster [':@62@N'].
```

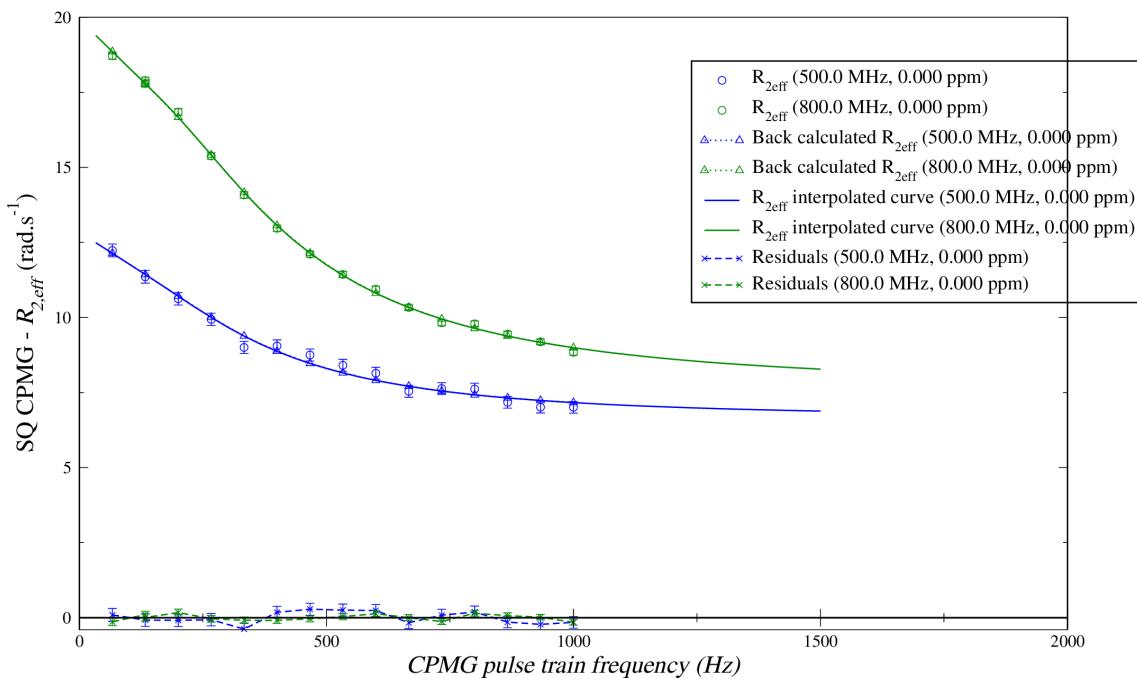
To see the optimised parameter values, double click on the `kex.agr` file to see the exchange rates. This will open the `grace.view` user function window by which the file can be opened. The default settings produces the following graph:



The exchange rate for most spins experiencing exchange is between 1000 and 2000 s⁻¹. Opening the pA.agr file, it can be seen that the population of state A is approximately 0.98. A number of spins have much lower values than this, but their errors are huge meaning that all but 3 are statistically the same as 0.98 (note that a proper ANOVA statistics analysis would be required to make such statements).

To see one of the dispersion curves, open the disp_64_N.agr file in Grace. The default settings will produce the graph:

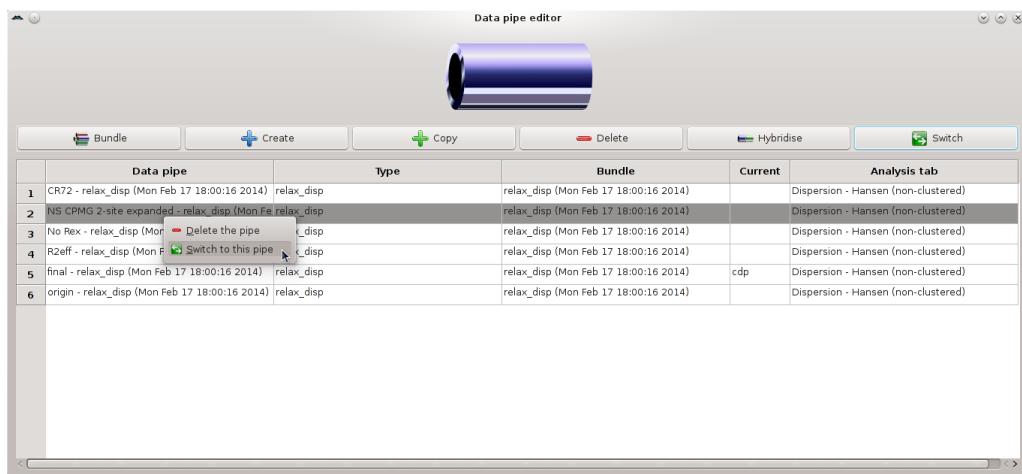
Relaxation dispersion plot



To improve the appearance of this plot, please refer to the Grace software documentation.

11.14.13 Dispersion GUI mode – comparing models

To compare the results of the different optimised models, the pipe editor window can be used to switch between the data pipes containing the results of the individual models. Select the “View→Data pipe editor” menu entry. Right click on the model of interest and select “Switch to this pipe”:



The results viewer window can then be used to open the text files and Grace plots for that model. Switch to the data pipe of another model and open the same file to compare the results.

11.14.14 Dispersion GUI mode – the clustered analysis

Before the second analysis with spin clustering will be performed, the relax state will be saved in the file `state.bz2` and the program closed. To store the details of the second analysis in a separate log file, relax will be restarted with the command:

```
$ mpirun -np 8 /data/relax/relax-trunk/relax --multi='mpi4py' --log
~/tmp/dispersion/log_clustered --gui
```

When the GUI has started, load the `state.bz2` file. If relax is not restarted, all messages will be in a single log file.

For the clustered analysis, we will focus on one group of spins – those from residues :59 to :67. These can be seen to have very similar dynamics:

```
# Parameter description: The population for state A.
#
# mol_name  res_num  res_name  spin_num  spin_name  value          error
[snip]
None      59       GLY        None       N           0.991169677577733  0.0254974551085798
None      60       GLY        None       N           None                   None
None      61       GLY        None       N           0.989169345780449  0.000173707304433962
None      62       GLY        None       N           0.892612114003636  0.156741312688688
None      63       GLY        None       N           0.991579380015928  0.00101555844987099
None      64       GLY        None       N           0.983519617639107  0.00141882200997569
None      65       GLY        None       N           0.993831316342131  0.00840481515172743
None      66       GLY        None       N           0.996816227018878  0.00495523034494496
None      67       GLY        None       N           0.987206586948786  0.000217099775069814
[snip]

# Parameter description: The exchange rate.
#
# mol_name  res_num  res_name  spin_num  spin_name  value          error
[snip]
None      59       GLY        None       N           1856.39029180567  258.796681611922
None      60       GLY        None       N           None                   None
None      61       GLY        None       N           1706.42820099893  79.5779060629935
None      62       GLY        None       N           1560.77926730839  246.876174669559
None      63       GLY        None       N           1448.69535431372  121.940593279104
None      64       GLY        None       N           1922.2405164604   96.0557804598977
None      65       GLY        None       N           1333.8423680145   168.000554346898
None      66       GLY        None       N           1243.45993122534  238.215664971556
None      67       GLY        None       N           1753.32557147779  78.2028166128168
[snip]
```

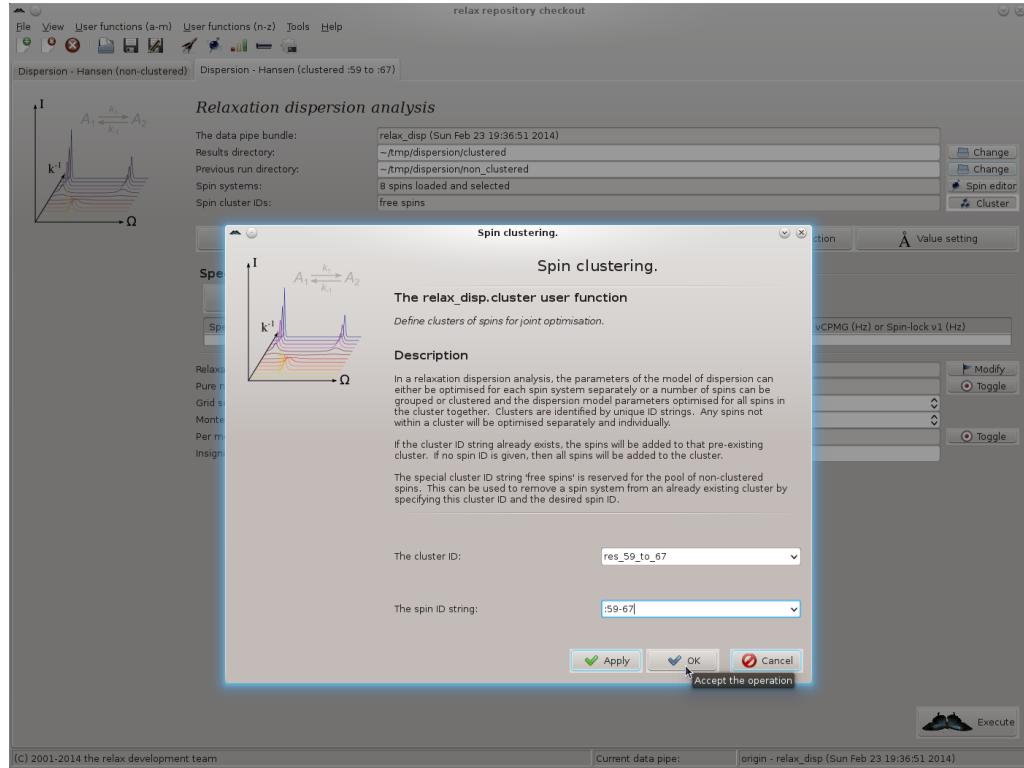
All other spins will be deselected.

From the non-clustered results, it could be argued that all spins in the entire system experience the same dynamic process, i.e. they have the same p_A and k_{ex} values. Such an analysis could be performed at a later stage if desired. The dispersion curves for the residue :60 could also be inspected to see that dispersion is likely to be present and another clustered analysis including this spin performed. The number of clustered analyses performed is up to the user – imagination is the only limit.

To start the analysis, open the analysis selection wizard as was performed previously. Name the analysis as ‘Dispersion - Hansen (clustered :59 to :67)’. Once the analysis is initialised, change the results directory to `~/tmp/dispersion/clustered`. To use the results of the previous analysis to speed up this analysis, as clustering will cause the grid search to be impossibly long, change the “Previous run directory” value to `~/tmp/dispersion/non-clustered`.

Set up the spin systems as for the non-clustered analysis. This time however deselect all spins except for those of residues :59 to :67 (excluding :60). This can be performed by right clicking on the spins in the spin viewer window.

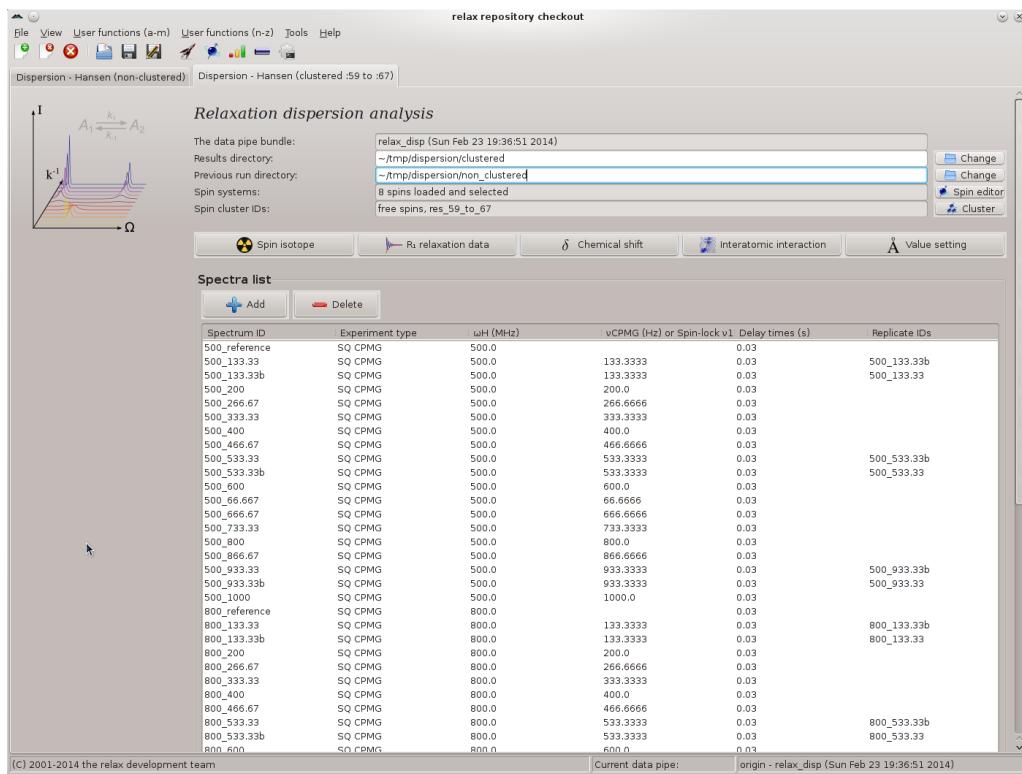
The next step is to cluster these eight spins. In the “Spin cluster IDs” GUI element, click on the “Cluster” button. This launches the `relax_disp.cluster` user function. Set the cluster ID to “`res_59_to_67`”, for example, and the spin ID string to “`:59-67`” (this says all residues from :59 to :67, see section 4.2.2 on page 38 for details):



As spin :60 is deselected, that residue will be skipped in the analysis.

Then set the spin isotope and load all of the Sparky peak lists as before. Chose the models ‘R2eff’, ‘No Rex’, ‘CR72’, and ‘NS CPMG 2-site expanded’. As the ‘CR72’ and ‘NS CPMG 2-site expanded’ were seen as being statistically equivalent in the non-clustered analysis, click on the “Toggle” button of the “Pure numerical solutions” GUI element to avoid the ‘CR72’ model in the model selection step. Actually, as the initial parameters for the ‘NS CPMG 2-site expanded’ model in the dispersion auto-analysis will be taken as the average from the non-clustered analysis, the ‘CR72’ model could be completely skipped.

Ignore the grid search increment setting as this will have no effect. No grid searches will be performed because the results from the non-clustered analysis will be used as the optimisation starting point. Set the Monte Carlo simulations to 200. The main window should now look like:



Start the analysis by clicking on the “Execute” button. You should notice that the spin cluster printout in the log messages in the relax controller window now show the text:

```
The spin cluster [:59', ':60', ':61', ':62', ':63', ':64', ':65', ':66', ':67'].
```

As residue 60 is deselected, it will not be used in the optimisation or any part of the analysis. The full analysis should take a few hours to complete.

11.14.15 Dispersion GUI mode – comparison of the analyses

To statistically compare the non-clustered and clustered analyses, the advanced Akaike’s Information Criterion (AIC) as derived in [d’Auvergne and Gooley \(2003\)](#) can be used. This information is stored within the recorded log files. Open the `~/tmp/dispersion/log_non_clustered` file and search for the model selection section. The text for residues 59 to 67 should be:

```
The spin cluster [:59@N'].
# Data pipe
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)          Num_params_(k)    Num_data_sets_(n)    Chi2      Criterion
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)          2                  30                 1577.42286   1581.42286
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014)  5                  30                 31.48415     41.48415
The model from the data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.
```

```
The spin cluster [:60@N'].
# Data pipe
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)          Num_params_(k)    Num_data_sets_(n)    Chi2      Criterion
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)          2                  30                 2647.97449   2651.97449
The model from the data pipe 'No Rex - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.
```

```
The spin cluster [:61@N'].
# Data pipe
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)          Num_params_(k)    Num_data_sets_(n)    Chi2      Criterion
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)          5                  30                 77.50622     87.50622
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014)  5                  30                 74.73334     84.73334
The model from the data pipe 'NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.
```

```
The spin cluster [:62@N'].
```

```

# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                722.91592  726.91592
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                30.11618   40.11618
The model from the data pipe 'NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

The spin cluster [':63@N'].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                5455.72135 5459.72135
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)       5              30                58.56731   68.56731
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                59.90738   69.90738
The model from the data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

The spin cluster [':64@N'].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                13736.91051 13740.91051
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)       5              30                28.66223   38.66223
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                29.54008   39.54008
The model from the data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

The spin cluster [':65@N'].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                2498.29408 2502.29408
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)       5              30                35.13518   45.13518
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                36.17043   46.17043
The model from the data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

The spin cluster [':66@N'].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                962.74016  966.74016
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)       5              30                15.02929   25.02929
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                15.08439   25.08439
The model from the data pipe 'CR72 - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

The spin cluster [':67@N'].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Mon Feb 17 18:00:16 2014)    2              30                16773.20431 16777.20431
CR72 - relax_disp (Mon Feb 17 18:00:16 2014)       5              30                118.17857  128.17857
NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014) 5              30                111.56710  121.56710
The model from the data pipe 'NS CPMG 2-site expanded - relax_disp (Mon Feb 17 18:00:16 2014)' has been selected.

```

For the log file from the clustered analysis (the `~/tmp/dispersion/log_clustered` file), the text should be as follows:

```
The spin cluster [:59, :60, :61, :62, :63, :64, :65, :66, :67].
# Data pipe                                         Num_params_(k)  Num_data_sets_(n)   Chi2      Criterion
No Rex - relax_disp (Sun Feb 23 19:36:51 2014)    16           240          56914.82514 56946.82514
NS CPMG 2-site expanded - relax_disp (Sun Feb 23 19:36:51 2014) 26           240          510.96553 562.96553
The model from the data pipe 'NS CPMG 2-site expanded - relax_disp (Sun Feb 23 19:36:51 2014)' has been selected.
```

The numbers for the ‘NS CPMG 2-site expanded’ model can be directly compared. This is because the parameter number, data set number, chi-squared value and AIC value (labelled as ‘Criterion’ in the logs) can be summed for the non-clustered analysis and then compared to the clustered values.

Analysis	Parameter number (k)	Data set number (n)	Chi-squared value	AIC value
Non-clustered	40	240	388.966	468.966
Clustered	26	240	510.966	562.966

The Akaike Information Criterion value is much less for the non-clustered analysis. Therefore this result is the most parsimonious – the result closest to Occam’s razor as defined by frequentist statistics. Therefore the non-clustered analysis is a statistically better description of the experimental data for this set of residues. For a different cluster of spins, the result may be different. If the assumptions of the same dynamics for all spins (both populations p_A and exchange rates k_{ex}) is correct, the results of the clustered analysis are nevertheless useful as it can decrease parameter uncertainty. If the assumption is not

correct, then the decrease in parameter uncertainty will be coupled with a parameter bias – a shift of the parameter away from reality. This should be avoided at all costs.

To perform a relaxation dispersion analysis on your own system, care in the setup, model choice and design of the clustering should be taken:

- Inspect the dispersion curves of all spin systems one by one and decide if any spins should be deselected for the entire analysis (due to the data being of insufficient quality).
- Depending on the dynamics of the system, the type of data collected (SQ CPMG vs. MMQ CPMG vs. $R_{1\rho}$), and personal preferences, chose which limited set of models will be used in the analysis. For this, the published literature should be consulted. Only use models for which you are sure are suited to the system being studied.
- Decide on a number of spin clustering schemes to compare to the non-clustered analysis.
- For deciding which analysis is best for representing the dynamics of the system, a balance between the statistical significance (based on modern frequentist statistics such as AIC), the decrease in parameter uncertainty, and the increase in parameter bias needs to be made.
- All results for all spins should be carefully inspected and compared.

Chapter 12

The ordering of frames

12.1 Introduction of frame ordering

12.1.1 Tensors of frame ordering

The frame order theory is defined as a bridging physics theory for rigid body motions based on the statistical mechanical ordering of reference frames. It is implemented as a new analysis type in relax designed for the study of rigid body motions in molecules. The theory aims to unify all rotational molecular physics techniques via a single statistical mechanical molecular dynamics (MD) model, by defining a series of rank- $2n$ frame order tensors. These tensors encapsulate the maximum information content of a rotational molecular physics experiment, as well as what type of information is contained in the data. To use the frame order theory, two steps are required:

1. The physics of the experiment should be decomposed into statistically mechanically averaged product of rotation matrix elements.
2. The MD model should be expressed in terms of a rotation matrix which modulates the motion of the rigid body, within the reference frame matching that of the experimental data.

12.1.2 Ln^{3+} aligned RDC and PCS data

For the current implementation in relax, the frame order theory has been derived for molecules internally aligned using paramagnetic lanthanide ions. The data required for the analysis includes both residual dipolar couplings (RDCs) and pseudocontact shifts (PCSs). Both data sets are required as they are complementary, each carrying different dynamics information:

RDCs: This data is dominated by the amplitudes of the MD motions, and the orientation of the statical mechanical average structure.

PCSs: These mainly provide information about the directions of the MD motions, and the orientation and position of the statical mechanical average structure.

For a successful analysis, the data must be of the highest quality. In addition, multiple alignments are required, either using different lanthanide ions and/or a different attachment point for the lanthanide ion. Note that the current implementation only handles alignment of one rigid body or domain.

12.2 Frame order theory

12.2.1 Frame order introduction

The ordering of a vector

Let $\mu(t)$ be a time dependent vector defined within an arbitrary fixed frame F as

$$\mu(t) = [\delta_x, \delta_y, \delta_z]^T \quad (12.1)$$

where δ_i is the time dependent direction cosine between the unit vector and the axis i of frame F . Key for understanding the statistical mechanics of a second rank rotational process is the time dependence of the outer product

$$P(t) = \mu(t) \otimes \mu(t), \quad (12.2)$$

$$= \begin{bmatrix} \delta_x^2 & \delta_x\delta_y & \delta_x\delta_z \\ \delta_y\delta_x & \delta_y^2 & \delta_y\delta_z \\ \delta_z\delta_x & \delta_z\delta_y & \delta_z^2 \end{bmatrix}. \quad (12.3)$$

Assuming statistical mechanical ensemble averaging, the observable expected value of the matrix $P(t)$ is a matrix which defines the ordering of the vector $\mu(t)$ within the frame F . This order matrix is

$$S(t) \equiv \overline{P(t)}, \quad (12.4a)$$

$$= \overline{\mu(t) \otimes \mu(t)}, \quad (12.4b)$$

$$= \overline{\begin{bmatrix} \delta_x^2 & \delta_x\delta_y & \delta_x\delta_z \\ \delta_y\delta_x & \delta_y^2 & \delta_y\delta_z \\ \delta_z\delta_x & \delta_z\delta_y & \delta_z^2 \end{bmatrix}}, \quad (12.4c)$$

$$= \begin{bmatrix} S_{xx}(t) & S_{xy}(t) & S_{xz}(t) \\ S_{yx}(t) & S_{yy}(t) & S_{yz}(t) \\ S_{zx}(t) & S_{zy}(t) & S_{zz}(t) \end{bmatrix}, \quad (12.4d)$$

where

$$S_{ij}(t) = \overline{\delta_i\delta_j}. \quad (12.5)$$

Because of the symmetry $S_{ij}(t) = S_{ji}(t)$, the order matrix has 6 unique elements.

Assuming that the time dependent process modulating $\mu(t)$ is much faster than the evolution period t_{\max} of the observed physical interaction, for example the weak molecular alignment process which induces residual dipolar couplings (RDCs) and pseudo-contact

shifts (PCSS) in NMR, the order matrix which gives rise to the non-isotropic effect is equation 12.4c at $t_{\max} = \infty$. Hence the non-zero order matrix is

$$S(\infty) = \begin{bmatrix} S_{xx}(\infty) & S_{xy}(\infty) & S_{xz}(\infty) \\ S_{yx}(\infty) & S_{yy}(\infty) & S_{yz}(\infty) \\ S_{zx}(\infty) & S_{zy}(\infty) & S_{zz}(\infty) \end{bmatrix}. \quad (12.6)$$

The ordering of a frame

Let the frame $C(t)$ be time dependent within an arbitrary fixed frame F . After a time period t the shift from $C(0)$ to $C(t)$ is given by the rotation

$$R(t) = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} \equiv \begin{bmatrix} \delta_{xx} & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_{yy} & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_{zz} \end{bmatrix}, \quad (12.7)$$

where rotation matrix element c_{ij} is equivalent to the direction cosine δ_{ij} between axis i of $C(t)$ and axis j of $C(0)$. For second rank physical processes modulated by rotational motions, analogously to the outer product expected value of 12.4b, the time dependence of the process is governed by the outer product

$$\mathbb{M}^{(2)}(t) = \overline{R(t) \otimes R(t)} \quad (12.8)$$

This is a rank-4, three dimensional rotational tensor defining the ordering of the frame $C(t)$ after a period t within the original frame $C(0)$. This is the definition of the second degree frame order tensor.

The matrix form of the second degree frame order tensor in rank-2, 9D Kronecker product notation is

$$\mathbb{M}^{(2)}(t) = \begin{bmatrix} \delta_{xx}^2 & \delta_{xx}\delta_{xy} & \delta_{xx}\delta_{xz} & \delta_{xy}\delta_{xx} & \delta_{xy}^2 & \delta_{xy}\delta_{xz} & \delta_{xz}\delta_{xx} & \delta_{xz}\delta_{xy} & \delta_{xz}^2 \\ \delta_{xx}\delta_{yx} & \delta_{xx}\delta_{yy} & \delta_{xx}\delta_{yz} & \delta_{xy}\delta_{yx} & \delta_{xy}\delta_{yy} & \delta_{xy}\delta_{yz} & \delta_{xz}\delta_{yx} & \delta_{xz}\delta_{yy} & \delta_{xz}\delta_{yz} \\ \delta_{xx}\delta_{zx} & \delta_{xx}\delta_{zy} & \delta_{xx}\delta_{zz} & \delta_{xy}\delta_{zx} & \delta_{xy}\delta_{zy} & \delta_{xy}\delta_{zz} & \delta_{xz}\delta_{zx} & \delta_{xz}\delta_{zy} & \delta_{xz}\delta_{zz} \\ \delta_{yx}\delta_{xx} & \delta_{yx}\delta_{xy} & \delta_{yx}\delta_{xz} & \delta_{yy}\delta_{xx} & \delta_{yy}\delta_{xy} & \delta_{yy}\delta_{xz} & \delta_{yz}\delta_{xx} & \delta_{yz}\delta_{xy} & \delta_{yz}\delta_{xz} \\ \delta_{yx}^2 & \delta_{yx}\delta_{yy} & \delta_{yx}\delta_{yz} & \delta_{yy}\delta_{yx} & \delta_{yy}^2 & \delta_{yy}\delta_{yz} & \delta_{yz}\delta_{yx} & \delta_{yz}\delta_{yy} & \delta_{yz}^2 \\ \delta_{yx}\delta_{zx} & \delta_{yx}\delta_{zy} & \delta_{yx}\delta_{zz} & \delta_{yy}\delta_{zx} & \delta_{yy}\delta_{zy} & \delta_{yy}\delta_{zz} & \delta_{yz}\delta_{zx} & \delta_{yz}\delta_{zy} & \delta_{yz}\delta_{zz} \\ \delta_{zx}\delta_{xx} & \delta_{zx}\delta_{xy} & \delta_{zx}\delta_{xz} & \delta_{zy}\delta_{xx} & \delta_{zy}\delta_{xy} & \delta_{zy}\delta_{xz} & \delta_{zz}\delta_{xx} & \delta_{zz}\delta_{xy} & \delta_{zz}\delta_{xz} \\ \delta_{zx}\delta_{yx} & \delta_{zx}\delta_{yy} & \delta_{zx}\delta_{yz} & \delta_{zy}\delta_{yx} & \delta_{zy}\delta_{yy} & \delta_{zy}\delta_{yz} & \delta_{zz}\delta_{yx} & \delta_{zz}\delta_{yy} & \delta_{zz}\delta_{yz} \\ \delta_{zx}^2 & \delta_{zx}\delta_{zy} & \delta_{zx}\delta_{zz} & \delta_{zy}\delta_{zx} & \delta_{zy}^2 & \delta_{zy}\delta_{zz} & \delta_{zz}\delta_{zx} & \delta_{zz}\delta_{zy} & \delta_{zz}^2 \end{bmatrix}, \quad (12.9)$$

where

$$\overline{\delta_{ij}\delta_{kl}} \equiv \overline{c_{ij}c_{kl}}. \quad (12.10)$$

This is a rank-2, 3D order matrix of rank-2, 3D order matrices. To see this, the T_{14} rank-4

matrix transpose of $\mathbb{M}^{(2)}$ in Kronecker product notation is

$$\mathbb{M}^{T_{14}}(t) = \begin{bmatrix} \delta_{xx}^2 & \delta_{yx}\delta_{xx} & \delta_{zx}\delta_{xx} & \delta_{xy}\delta_{xx} & \delta_{yy}\delta_{xx} & \delta_{zy}\delta_{xx} & \delta_{xz}\delta_{xx} & \delta_{yz}\delta_{xx} & \delta_{zz}\delta_{xx} \\ \delta_{xx}\delta_{yx} & \delta_{yx}^2 & \delta_{zx}\delta_{yx} & \delta_{xy}\delta_{yx} & \delta_{yy}\delta_{yx} & \delta_{zy}\delta_{yx} & \delta_{xz}\delta_{yx} & \delta_{yz}\delta_{yx} & \delta_{zz}\delta_{yx} \\ \delta_{xx}\delta_{zx} & \delta_{yx}\delta_{zx} & \delta_{zx}^2 & \delta_{xy}\delta_{zx} & \delta_{yy}\delta_{zx} & \delta_{zy}\delta_{zx} & \delta_{xz}\delta_{zx} & \delta_{yz}\delta_{zx} & \delta_{zz}\delta_{zx} \\ \delta_{xx}\delta_{xy} & \delta_{yx}\delta_{xy} & \delta_{zx}\delta_{xy} & \delta_{xy}^2 & \delta_{yy}\delta_{xy} & \delta_{zy}\delta_{xy} & \delta_{xz}\delta_{xy} & \delta_{yz}\delta_{xy} & \delta_{zz}\delta_{xy} \\ \delta_{xx}\delta_{yy} & \delta_{yx}\delta_{yy} & \delta_{zx}\delta_{yy} & \delta_{xy}\delta_{yy} & \delta_{yy}^2 & \delta_{zy}\delta_{yy} & \delta_{xz}\delta_{yy} & \delta_{yz}\delta_{yy} & \delta_{zz}\delta_{yy} \\ \delta_{xx}\delta_{zy} & \delta_{yx}\delta_{zy} & \delta_{zx}\delta_{zy} & \delta_{xy}\delta_{zy} & \delta_{yy}\delta_{zy} & \delta_{zy}^2 & \delta_{xz}\delta_{zy} & \delta_{yz}\delta_{zy} & \delta_{zz}\delta_{zy} \\ \delta_{xx}\delta_{xz} & \delta_{yx}\delta_{xz} & \delta_{zx}\delta_{xz} & \delta_{xy}\delta_{xz} & \delta_{yy}\delta_{xz} & \delta_{zy}\delta_{xz} & \delta_{xz}^2 & \delta_{yz}\delta_{xz} & \delta_{zz}\delta_{xz} \\ \delta_{xx}\delta_{yz} & \delta_{yx}\delta_{yz} & \delta_{zx}\delta_{yz} & \delta_{xy}\delta_{yz} & \delta_{yy}\delta_{yz} & \delta_{zy}\delta_{yz} & \delta_{xz}\delta_{yz} & \delta_{yz}^2 & \delta_{zz}\delta_{yz} \\ \delta_{xx}\delta_{zz} & \delta_{yx}\delta_{zz} & \delta_{zx}\delta_{zz} & \delta_{xy}\delta_{zz} & \delta_{yy}\delta_{zz} & \delta_{zy}\delta_{zz} & \delta_{xz}\delta_{zz} & \delta_{yz}\delta_{zz} & \delta_{zz}^2 \end{bmatrix}. \quad (12.11)$$

The 3D matrix in the top left corner is the ordering of the x-axis with itself, the central matrix is the ordering of the y-axis with itself, and the bottom right is the ordering of the z-axis with itself. The off-diagonal 3D matrices are the cross-correlations between the three axes. Using the notation e_x , e_y and e_z for the orthogonal axis system of the time dependent frame $C(t)$, the second degree frame order matrix can be written as

$$\mathbb{M}^{T_{14}}(t) = \begin{bmatrix} \overline{e_x \otimes e_x} & \overline{e_x \otimes e_y} & \overline{e_x \otimes e_z} \\ \overline{e_y \otimes e_x} & \overline{e_y \otimes e_y} & \overline{e_y \otimes e_z} \\ \overline{e_z \otimes e_x} & \overline{e_z \otimes e_y} & \overline{e_z \otimes e_z} \end{bmatrix}. \quad (12.12)$$

If the rank-2, 3D order matrix between the axes A and B is denoted as

$$S_{AB}(t) = \overline{e_A \otimes e_B}, \quad (12.13)$$

then the frame order matrix is

$$\mathbb{M}^{T_{14}}(t) = \begin{bmatrix} S_{XX}(t) & S_{XY}(t) & S_{XZ}(t) \\ S_{YX}(t) & S_{YY}(t) & S_{YZ}(t) \\ S_{ZX}(t) & S_{ZY}(t) & S_{ZZ}(t) \end{bmatrix}. \quad (12.14)$$

The frame order matrix is diagonally symmetric, as can be seen in the T_{14} transpose of the matrix in rank-2, 9D Kronecker product form (equation 12.11, hence for the second degree frame order matrix there are 45 unique elements. For the 9D Kronecker product notation of equation 12.9, this transformed diagonal symmetry can be schematically represented as

$$\mathbb{M}^{(2)}(t) = \begin{bmatrix} \cdot & \text{---} & \text{---} & \text{---} & \cdot \\ \text{---} & \cdot & \text{---} & \text{---} & \cdot \\ \cdot & \text{---} & \cdot & \text{---} & \cdot \\ \text{---} & \text{---} & \text{---} & \cdot & \cdot \\ \cdot & \text{---} & \text{---} & \text{---} & \cdot \\ \text{---} & \text{---} & \text{---} & \text{---} & \cdot \\ \cdot & \text{---} & \text{---} & \text{---} & \cdot \\ \text{---} & \text{---} & \text{---} & \text{---} & \cdot \\ \cdot & \text{---} & \text{---} & \text{---} & \cdot \end{bmatrix}.$$

When rotational symmetries are present in the time modulation of the frame $C(t)$ then, according to Perrin (1936), the averages of the double products $\overline{c_{ij}c_{kl}}$ where an index appears only once is zero. In this case, the active frame order matrix elements are

$$\mathbb{M}^{(2)}(t) = \begin{bmatrix} \overline{\delta_{xx}^2} & \cdot & \cdot & \cdot & \overline{\delta_{xy}^2} & \cdot & \cdot & \cdot & \overline{\delta_{xz}^2} \\ \cdot & \overline{\delta_{xx}\delta_{yy}} & \cdot & \cdot & \overline{\delta_{xy}\delta_{yx}} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \overline{\delta_{xx}\delta_{zz}} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \overline{\delta_{yy}\delta_{xx}} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \overline{\delta_{yx}^2} & \cdot & \cdot & \cdot & \overline{\delta_{yy}^2} & \cdot & \cdot & \cdot & \overline{\delta_{yz}^2} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \overline{\delta_{yy}\delta_{zz}} & \cdot & \cdot & \overline{\delta_{yz}\delta_{zy}} \\ \cdot & \cdot & \cdot & \overline{\delta_{zx}\delta_{xz}} & \cdot & \cdot & \overline{\delta_{zz}\delta_{xx}} & \cdot & \cdot \\ \overline{\delta_{zx}^2} & \cdot & \cdot & \cdot & \overline{\delta_{zy}^2} & \cdot & \cdot & \overline{\delta_{zz}\delta_{yy}} & \cdot \\ \end{bmatrix} . \quad (12.15)$$

This matrix consists of 15 unique elements. It is the weighted sum of the three rank-4 identity matrices I_1 , I_2 and I_3 .

The fourth rank identity matrices

According to Spencer (1980), the rank-4 identity matrices are defined as

$$I_1 = \delta_{ij}\delta_{kl}e_i \otimes e_j \otimes e_k \otimes e_l, \quad (12.16a)$$

$$I_2 = \delta_{ik}\delta_{jl}e_i \otimes e_j \otimes e_k \otimes e_l, \quad (12.16b)$$

$$I_3 = \delta_{il}\delta_{kj}e_i \otimes e_j \otimes e_k \otimes e_l, \quad (12.16c)$$

where δ_{ij} is the Kronecker delta and e_i are the axes. In general, the identity matrix is

$$I = (\lambda \delta_{ij} \delta_{kl} + \mu \delta_{ik} \delta_{jl} + \nu \delta_{il} \delta_{kj}) e_i \otimes e_j \otimes e_k \otimes e_l. \quad (12.17)$$

Expanding 12.16a to 12.16c to 9D Kronecker product matrix form,

$$I_1 = \begin{bmatrix} 1 & . & . & . & . & 1 & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ 1 & . & . & . & . & 1 & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ 1 & . & . & . & . & 1 & . & . & . & . & 1 \end{bmatrix}, \quad (12.18a)$$

$$I_2 = \begin{bmatrix} 1 & . & . & . & . & . & . & . & . & . \\ . & 1 & . & . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . & . & . \\ . & . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & . & 1 & . \end{bmatrix}, \quad (12.18b)$$

$$I_3 = \begin{bmatrix} 1 & . & . & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . & . \\ . & . & . & . & . & . & 1 & . & . \\ . & 1 & . & . & . & . & . & . & . \\ . & . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & . & 1 & . \\ . & . & 1 & . & . & . & . & . & . \\ . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & . & 1 \end{bmatrix}. \quad (12.18c)$$

The identity matrices are related to each other via the rank-4 matrix transposes

$$I_2 = I_1^{T_{14}} = I_1^{T_{23}}, \quad (12.19a)$$

$$I_3 = I_1^{T_{24}} = I_1^{T_{13}}. \quad (12.19b)$$

In the case of unrestricted motions, the time limits of the frame order matrix are

$$\mathbb{M}^{(2)}(t=0) = I_1, \quad (12.20)$$

and

$$\mathbb{M}^{(2)}(t=\infty) = \frac{1}{3}I_2. \quad (12.21)$$

Tensor power of the frame order

The rank-4, 3D frame order tensor of equation 12.8 on page 239 was derived for second order rotational physical processes. However this can be generalised for physical processes of all orders. The tensor power of the time dependent rotation matrix $R(t)$ is defined as

$$R^{\otimes n}(t) \stackrel{\text{def}}{=} R(t) \otimes \cdots \otimes R(t), \quad (12.22)$$

where the outer product is repeated n times. Therefore let the frame order tensor be defined as

$$\mathbb{M}^{(n)}(t) = \overline{R^{\otimes n}(t)}, \quad (12.23)$$

where n is the order of the physical process. The rank of the 3D tensors is $2n$. The first few frame order tensors of rank-2, rank-4, rank-6, and rank-8 are

$$\mathbb{M}^{(1)}(t) = \overline{R(t)}, \quad (12.24a)$$

$$\mathbb{M}^{(2)}(t) = \overline{R(t) \otimes R(t)}, \quad (12.24b)$$

$$\mathbb{M}^{(3)}(t) = \overline{R(t) \otimes R(t) \otimes R(t)}, \quad (12.24c)$$

$$\mathbb{M}^{(4)}(t) = \overline{R(t) \otimes R(t) \otimes R(t) \otimes R(t)}. \quad (12.24d)$$

In index and direction cosine notation,

$$\mathbb{M}_{ij}^{(1)}(t) = \overline{\delta_{ij}}, \quad (12.25a)$$

$$\mathbb{M}_{ijkl}^{(2)}(t) = \overline{\delta_{ij}\delta_{kl}}, \quad (12.25b)$$

$$\mathbb{M}_{ijklmn}^{(3)}(t) = \overline{\delta_{ij}\delta_{kl}\delta_{mn}}, \quad (12.25c)$$

$$\mathbb{M}_{ijklmno}^{(4)}(t) = \overline{\delta_{ij}\delta_{kl}\delta_{mn}\delta_{op}}. \quad (12.25d)$$

The frame order motional eigenframe

The rotation matrices of the general frame order tensor of equation 12.23 can be decomposed into a time dependent and time independent component. The original frame F can be defined as the motional eigenframe of the system and a new arbitrary frame F' introduced. The forward rotation from the reference frame F' to the motional eigenframe F will be denoted as R_{eigen} . The rotation matrix decomposition is

$$R'(t) = R_{\text{eigen}} \cdot R(t) \cdot R_{\text{eigen}}^T. \quad (12.26)$$

Hence the second degree frame order tensor is

$$\mathbb{M}^{(2)} = \overline{R'(t) \otimes R'(t)}, \quad (12.27)$$

Using the mixed product property

$$AC \otimes BD = (A \otimes B)(C \otimes D), \quad (12.28)$$

the arbitrary frame, second degree frame order matrix is

$$\mathbb{M}^{(2)} = \overline{\left(R_{\text{eigen}} \cdot R(t) \cdot R_{\text{eigen}}^T \right) \otimes \left(R_{\text{eigen}} \cdot R(t) \cdot R_{\text{eigen}}^T \right)}, \quad (12.29a)$$

$$= \overline{(R_{\text{eigen}} \otimes R_{\text{eigen}}) \cdot \left(R(t) \cdot R_{\text{eigen}}^T \right) \otimes \left(R(t) \cdot R_{\text{eigen}}^T \right)}, \quad (12.29b)$$

$$= \overline{(R_{\text{eigen}} \otimes R_{\text{eigen}}) \cdot (R(t) \otimes R(t)) \cdot \left(R_{\text{eigen}}^T \otimes R_{\text{eigen}}^T \right)}, \quad (12.29c)$$

$$= \overline{(R_{\text{eigen}} \otimes R_{\text{eigen}}) \cdot (R(t) \otimes R(t))} \cdot \overline{(R_{\text{eigen}}^T \otimes R_{\text{eigen}}^T)}. \quad (12.29d)$$

Generalising from the 2nd to the nth-order, the generalised frame order tensor rotation is

$$\mathbb{M}^{(n)}(t) = R_{\text{eigen}}^{\otimes n} \cdot \overline{R^{\otimes n}(t)} \cdot R_{\text{eigen}}^{T \otimes n}. \quad (12.30)$$

Rotation to the average position frame of the rigid body

For the modelling aspect of the frame order theory, one more rotation is required. In equation 12.30, it is assumed that the starting position for the moving rigid body is that of its motional average. However in the initial 3D structure, this is not the case and an additional rotation to the average position R_{ave} is required. Taking this into account, the generalised frame order tensor is defined as

$$\mathbb{M}^{(n)}(t) = R_{\text{eigen}}^{\otimes n} \cdot \overline{R^{\otimes n}(t)} \cdot R_{\text{eigen}}^{T \otimes n} \cdot R_{\text{ave}}^{T \otimes n}, \quad (12.31)$$

where R_{eigen} is the eigenframe rotation matrix, $R(t)$ is the time dependent rotation matrix, R_{ave} is the rotation from the average domain position to the motional eigenframe, and $\otimes n$ is the nth tensor power. In applications to physical processes which require numerical integration, pre-rotating the rigid body by R_{ave} to the average position is equivalent but more numerically efficient. Therefore the R_{ave} can be dropped and equation 12.30 used instead.

12.2.2 Frame order and the alignment tensor

The RDC and PCS

For the residual dipolar coupling (RDC) and pseudo-contact shift (PCS) NMR phenomena, both effects are governed by the partial molecular alignment tensor A . For a two domain molecular system, when one domain is internally aligned with for example a paramagnetic lanthanide ion within a magnetic field, the other domain experiences a reduced alignment \overline{A} due to the interdomain motions.

The RDC The residual dipolar coupling is given by

$$D = d \hat{r}^T \cdot A \cdot \hat{r}, \quad (12.32)$$

where \hat{r} is the internuclear unit vector, d is the dipolar constant defined as

$$d = -\frac{3}{2\pi} \frac{\mu_0}{4\pi} \frac{\gamma_I \gamma_S \hbar}{\langle r \rangle^3}, \quad (12.33)$$

μ_0 is the permeability of free space, γ_i is the gyromagnetic ratio of the nucleus i , \hbar is Planck's constant divided by 2π , $\langle r \rangle$ is the time averaged internuclear distance, the factor of $\frac{1}{2\pi}$ is to convert the constant from radians per second to Hertz, and the factor of three is associated with the alignment tensor. In the presence of an alignment tensor reduction, and assuming that the fast vibrational and librational internal motions of the vector are statistically self-decoupled from the rigid body motions, the RDC is simply

$$D = d \hat{r}^T \cdot \overline{A} \cdot \hat{r}, \quad (12.34)$$

as the vector \hat{r} is considered time independent in the molecular reference frame.

The PCS The pseudo-contact shift equation is simply

$$\delta = \frac{\mu_0}{4\pi} \frac{15kT}{B_0^2} \frac{1}{|r|^5} \hat{r}^T \cdot A \cdot \hat{r}, \quad (12.35a)$$

$$= \frac{c}{|r|^5} \hat{r}^T \cdot A \cdot \hat{r}, \quad (12.35b)$$

$$= \frac{1}{4\pi} \frac{1}{|r|^5} \hat{r}^T \cdot \chi \cdot \hat{r}, \quad (12.35c)$$

where A is the alignment tensor, χ is the magnetic susceptibility tensor, \hat{r} is the lanthanide nuclear unit vector, and c is the PCS constant defined as

$$c = \frac{\mu_0}{4\pi} \frac{15kT}{B_0^2}. \quad (12.36)$$

The alignment tensor reduction process is complicated by the inverse $|r|^5$ normalisation factor, as r is not time independent in the molecular reference frame.

Alignment tensor reduction

The statistical mechanics behind the alignment tensor reduction can be expressed as

$$\bar{A} = \left\langle \int_0^{t_{\max}} R^{-1}(\Omega_t) \cdot A \cdot R(\Omega_t) dt \right\rangle, \quad (12.37)$$

where the angular brackets denote the ensemble averaging, the time integration is for a single molecule over the evolution period of the physical interaction, Ω_t are the SO(3) rotational angles describing the change in position of the moving rigid body, and A is the full alignment tensor. Here the alignment tensor has been created by an averaging of the partially restricted Brownian diffusion process of the non-moving component, again both over the ensemble and time, as

$$A = \left\langle \int_0^{t_{\max}} R^{-1}(\Omega_t) \cdot F \cdot R(\Omega_t) dt \right\rangle, \quad (12.38)$$

where F is the molecular frame. It is assumed that the alignment process of the non-moving domain and the motions of the moving domain are decoupled.

Using the ergodic hypothesis, the averaging process which generates the reduced alignment tensor can be simplified as

$$\bar{A} = \overline{R^{-1} \cdot A \cdot R}, \quad (12.39a)$$

$$= \overline{R^T \cdot A \cdot R}. \quad (12.39b)$$

The index notation for a tensor rotation is

$$T'_{ij} = \sum_{kl} R_{ki} R_{lj} T_{kl}. \quad (12.40)$$

Therefore the reduced alignment tensor in index notation is

$$\overline{A}_{ij} = \sum_{kl} \overline{c_{ki} c_{lj}} A_{kl}, \quad (12.41)$$

$$= \sum_{kl} \mathbb{M}_{kilj}^{(2)} A_{kl}, \quad (12.42)$$

where $\mathbb{M}^{(2)}$ is a rank-4, 3D orientational tensor which will be called the frame order tensor.

Expanding the sum,

$$\begin{aligned} \overline{A}_{ij} &= \mathbb{M}_{xixj} A_{xx} + \mathbb{M}_{xiyj} A_{xy} + \mathbb{M}_{xizj} A_{xz} \\ &\quad + \mathbb{M}_{yixj} A_{yx} + \mathbb{M}_{yiyy} A_{yy} + \mathbb{M}_{yizj} A_{yz} \\ &\quad + \mathbb{M}_{zixj} A_{zx} + \mathbb{M}_{ziyj} A_{zy} + \mathbb{M}_{zizj} A_{zz}. \end{aligned} \quad (12.43)$$

As

$$A_{ij} = A_{ji}, \quad (12.44a)$$

$$A_{zz} = -A_{xx} - A_{yy}, \quad (12.44b)$$

equation 12.43 becomes

$$\begin{aligned}\overline{A}_{ij} &= (\mathbb{M}_{xixj} - \mathbb{M}_{zizj}) A_{xx} + (\mathbb{M}_{yiyj} - \mathbb{M}_{zizj}) A_{yy} \\ &\quad + (\mathbb{M}_{xiyj} + \mathbb{M}_{yixj}) A_{xy} + (\mathbb{M}_{xizj} + \mathbb{M}_{zixj}) A_{xz} + (\mathbb{M}_{yizj} + \mathbb{M}_{ziyj}) A_{yz}. \end{aligned}\quad (12.45)$$

A single element of the reduced tensor is simply a linear combination of all elements of the original tensor multiplied by constants consisting of different combinations of frame order matrix components.

Converting from the rank-2, 3D, symmetric and traceless space of alignment tensors to the rank-1, 5D space, a non-linear frame order superoperator can be written as

$$\overline{A} = \mathbb{M}_{5D}^{(2)} \cdot A. \quad (12.46)$$

In matrix notation, this is

$$\begin{bmatrix} \overline{A}_{xx} \\ \overline{A}_{yy} \\ \overline{A}_{xy} \\ \overline{A}_{xz} \\ \overline{A}_{yz} \end{bmatrix} = \begin{bmatrix} \mathbb{M}_{xxxx} - \mathbb{M}_{zzzz} & \mathbb{M}_{yxyx} - \mathbb{M}_{zxzx} & \mathbb{M}_{xxyx} + \mathbb{M}_{yxxx} & \mathbb{M}_{xxzx} + \mathbb{M}_{zzxx} & \mathbb{M}_{yxzx} + \mathbb{M}_{zzyx} \\ \mathbb{M}_{xyxy} - \mathbb{M}_{zyzy} & \mathbb{M}_{yyyy} - \mathbb{M}_{zyzy} & \mathbb{M}_{xyyy} + \mathbb{M}_{yyxy} & \mathbb{M}_{xyzy} + \mathbb{M}_{zyxy} & \mathbb{M}_{yyzy} + \mathbb{M}_{zyyy} \\ \mathbb{M}_{xxyy} - \mathbb{M}_{zzyy} & \mathbb{M}_{yxyy} - \mathbb{M}_{zxzy} & \mathbb{M}_{xxyy} + \mathbb{M}_{yxyy} & \mathbb{M}_{xxzy} + \mathbb{M}_{zzxy} & \mathbb{M}_{yxzy} + \mathbb{M}_{zzyy} \\ \mathbb{M}_{xxzz} - \mathbb{M}_{zzzz} & \mathbb{M}_{yxyz} - \mathbb{M}_{zxzz} & \mathbb{M}_{xxyz} + \mathbb{M}_{gaxz} & \mathbb{M}_{xxzz} + \mathbb{M}_{zzzz} & \mathbb{M}_{yxzz} + \mathbb{M}_{zxyz} \\ \mathbb{M}_{xyxz} - \mathbb{M}_{zyzz} & \mathbb{M}_{yyyz} - \mathbb{M}_{zyzz} & \mathbb{M}_{xyyz} + \mathbb{M}_{yyxz} & \mathbb{M}_{xyzz} + \mathbb{M}_{zyxz} & \mathbb{M}_{yyzz} + \mathbb{M}_{zyyz} \end{bmatrix} \cdot \begin{bmatrix} A_{xx} \\ A_{yy} \\ A_{xy} \\ A_{xz} \\ A_{yz} \end{bmatrix}. \quad (12.47)$$

Let

$$\{A_0, A_1, A_2, A_3, A_4\} = \{A_{xx}, A_{yy}, A_{xy}, A_{xz}, A_{yz}\}, \quad (12.48)$$

and assuming the rank-2, 9D Kronecker product form of $\mathbb{M}_{ij}^{(2)}$ using numerical indices where $\{i, j\} = 0, 1, \dots, 8$, then

$$\begin{bmatrix} \overline{A}_0 \\ \overline{A}_1 \\ \overline{A}_2 \\ \overline{A}_3 \\ \overline{A}_4 \end{bmatrix} = \begin{bmatrix} \mathbb{M}_{00} - \mathbb{M}_{80} & \mathbb{M}_{40} - \mathbb{M}_{80} & \mathbb{M}_{10} + \mathbb{M}_{30} & \mathbb{M}_{20} + \mathbb{M}_{60} & \mathbb{M}_{50} + \mathbb{M}_{70} \\ \mathbb{M}_{04} - \mathbb{M}_{84} & \mathbb{M}_{44} - \mathbb{M}_{84} & \mathbb{M}_{14} + \mathbb{M}_{34} & \mathbb{M}_{24} + \mathbb{M}_{64} & \mathbb{M}_{54} + \mathbb{M}_{74} \\ \mathbb{M}_{01} - \mathbb{M}_{81} & \mathbb{M}_{41} - \mathbb{M}_{81} & \mathbb{M}_{11} + \mathbb{M}_{31} & \mathbb{M}_{21} + \mathbb{M}_{61} & \mathbb{M}_{51} + \mathbb{M}_{71} \\ \mathbb{M}_{02} - \mathbb{M}_{82} & \mathbb{M}_{42} - \mathbb{M}_{82} & \mathbb{M}_{12} + \mathbb{M}_{32} & \mathbb{M}_{22} + \mathbb{M}_{62} & \mathbb{M}_{52} + \mathbb{M}_{72} \\ \mathbb{M}_{05} - \mathbb{M}_{85} & \mathbb{M}_{45} - \mathbb{M}_{85} & \mathbb{M}_{15} + \mathbb{M}_{35} & \mathbb{M}_{25} + \mathbb{M}_{65} & \mathbb{M}_{55} + \mathbb{M}_{75} \end{bmatrix} \cdot \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix}. \quad (12.49)$$

For the alignment tensor, the 81 elements of the frame order matrix have recombined into 25 unique scaling factors.

The alignment tensor is the anisotropic part of a frame order matrix

The alignment tensor is related to the orientational probability tensor by

$$A = P - \frac{1}{3}I. \quad (12.50)$$

The P probability tensor is the average orientation position of the molecule, hence is the average molecular frame \overline{F} . As this frame is simply the rotation matrix relative to the

laboratory frame, then

$$P = \bar{F}, \quad (12.51a)$$

$$= \bar{R}, \quad (12.51b)$$

$$= \mathbb{M}^{(1)}. \quad (12.51c)$$

Therefore the alignment tensor can then be written as the anisotropic part of the first degree frame order matrix

$$A = \mathbb{M}^{(1)} - \frac{1}{3}I. \quad (12.52)$$

12.2.3 Single pivoted motions

Atomic level mechanics of the single pivot

For the PCS, the lanthanide ion to nuclear vector is

$$r = p_N - p_{Ln^{3+}}, \quad (12.53)$$

where p_N is the Cartesian coordinates of the nucleus of interest and $p_{Ln^{3+}}$ is the position of the aligning lanthanide ion. r is defined in the alignment frame, and $p_{Ln^{3+}}$ is constant in this frame. After a forward rotation to the discrete state i , the new atomic position in the reference frame is

$$p'_N = R_i \cdot (p_N - p_P) + p_P. \quad (12.54)$$

where p_P is the pivot point of the rotation. Hence the transformed vector is

$$r_i = p_N^i - p_{Ln^{3+}}, \quad (12.55a)$$

$$= R_i \cdot (p_N - p_P) + p_P - p_{Ln^{3+}}. \quad (12.55b)$$

The set of three vectors are defining this pivoted system are

$$r_{LN} = p_N - p_{Ln^{3+}}, \quad (12.56a)$$

$$r_{PN} = p_N - p_P, \quad (12.56b)$$

$$r_{LP} = p_P - p_{Ln^{3+}}. \quad (12.56c)$$

Let the pre-rotation vectors be

$$r_{LN}^{(0)} = r_{LP}^{(0)} + r_{PN}^{(0)}, \quad (12.57a)$$

$$r_{PN}^{(0)}, \quad (12.57b)$$

$$r_{LP}^{(0)}. \quad (12.57c)$$

The post-rotation vectors are

$$r_{LN}^{(1)} = r_{LP}^{(0)} + R_i^{(1)} \cdot r_{PN}^{(0)}, \quad (12.58a)$$

$$r_{PN}^{(1)} = R_i^{(1)} \cdot r_{PN}^{(0)}, \quad (12.58b)$$

$$r_{LP}^{(1)} = r_{LP}^{(0)}. \quad (12.58c)$$

The vector r_{PN} is independent of alignment so can be calculated once per atom, and r_{LP} is independent of alignment and atom position so can be calculate once.

PCS and single pivoted motions

For a single state i , the PCS value when substituting 12.55b into 12.35 is

$$\delta = \frac{c}{|r_i|^{15}} r_i^T \cdot A \cdot r_i. \quad (12.59)$$

Expanding for the single motion of the lanthanide-atom vector $r_{\text{LN}}^{(1)}$, this becomes

$$\delta = \frac{c}{|r_{\text{LN}}^{(1)}|^5} r_{\text{LN}}^{(1)T} \cdot A \cdot r_{\text{LN}}^{(1)}, \quad (12.60\text{a})$$

$$\delta = \frac{c}{|r_{\text{LN}}^{(1)}|^5} \left(r_{\text{LP}}^{(0)} + R_i^{(1)} \cdot r_{\text{PN}}^{(0)} \right)^T \cdot A \cdot \left(r_{\text{LP}}^{(0)} + R_i^{(1)} \cdot r_{\text{PN}}^{(0)} \right), \quad (12.60\text{b})$$

$$\delta = \frac{c}{|r_{\text{LN}}^{(1)}|^5} \left(r_{\text{LP}}^{(0)T} + r_{\text{PN}}^{(0)T} \cdot R_i^{(1)T} \right) \cdot A \cdot \left(r_{\text{LP}}^{(0)} + R_i^{(1)} \cdot r_{\text{PN}}^{(0)} \right), \quad (12.60\text{c})$$

$$\begin{aligned} \delta = \frac{c}{|r_{\text{LN}}^{(1)}|^5} & \left[r_{\text{PN}}^{(0)T} \cdot R_i^{(1)T} \cdot A \cdot R_i^{(1)} \cdot r_{\text{PN}}^{(0)} \right. \\ & + r_{\text{LP}}^{(0)T} \cdot A \cdot r_{\text{PN}}^{(0)} + r_{\text{PN}}^{(0)T} \cdot A \cdot r_{\text{LP}}^{(0)} \\ & \left. + r_{\text{LP}}^{(0)T} \cdot A \cdot r_{\text{LP}}^{(0)} \right]. \end{aligned} \quad (12.60\text{d})$$

Due to the distance normalisation factor in these equations, the symbolic integration for the modelling of specific motional modes is currently intractable.

12.2.4 Double pivoted motions

When the motion of a multiple rigid body system can be described as two rotations about two different pivots, the modulation of the PCS becomes more complicated. Figure 12.1 shows this motion for a single lanthanide to atom vector.

Atomic level mechanics of the double pivot

The six vectors at the original position are

$$r_{\text{LN}}^{(0)} = r_{\text{LP}_2}^{(0)} + r_{\text{P}_2\text{P}_1}^{(0)} + r_{\text{P}_1\text{N}}^{(0)}, \quad (12.61\text{a})$$

$$r_{\text{P}_2\text{N}}^{(0)} = r_{\text{P}_2\text{P}_1}^{(0)} + r_{\text{P}_1\text{N}}^{(0)}, \quad (12.61\text{b})$$

$$r_{\text{P}_1\text{N}}^{(0)}, \quad (12.61\text{c})$$

$$r_{\text{LP}_1}^{(0)} = r_{\text{LP}_2}^{(0)} + r_{\text{P}_2\text{P}_1}^{(0)}, \quad (12.61\text{d})$$

$$r_{\text{P}_2\text{P}_1}^{(0)}, \quad (12.61\text{e})$$

$$r_{\text{LP}_2}^{(0)}. \quad (12.61\text{f})$$

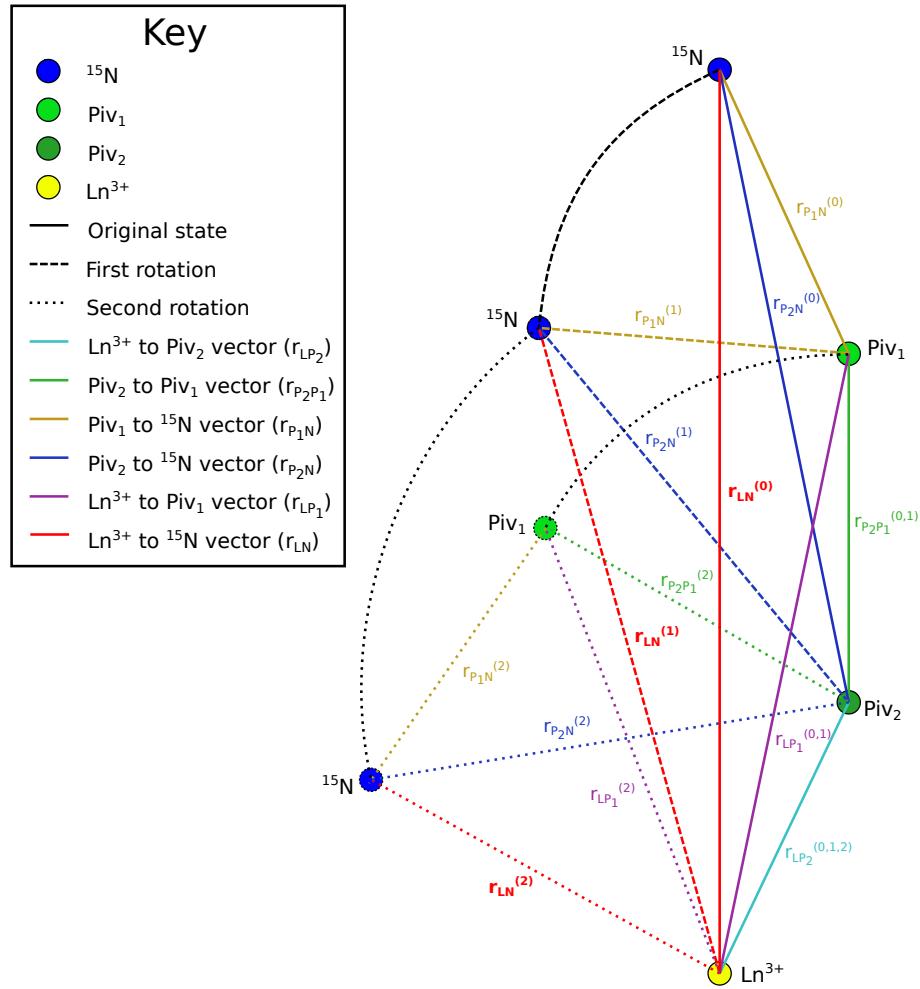


Figure 12.1: Frame order in the double pivot system. The lanthanide position is denoted by Ln^{3+} or simply L, the position of the first pivot by Piv_1 , the position of the second pivot by Piv_2 , and the position of the nucleus of interest by ${}^{15}\text{N}$. In the vector notation these are L, P₁, P₂ and N. The original position is denoted by (0), the position after the first rotation by (1), and the position after the second rotation by (2).

The six vectors after the first rotation for state i , $R_i^{(1)}$, are

$$r_{LN}^{(1)} = r_{LP_2}^{(0)} + r_{P_2P_1}^{(0)} + R_i^{(1)} \cdot r_{P_1N}^{(0)}, \quad (12.62a)$$

$$r_{P_2N}^{(1)} = r_{P_2P_1}^{(0)} + R_i^{(1)} \cdot r_{P_1N}^{(0)}, \quad (12.62b)$$

$$r_{P_1N}^{(1)} = R_i^{(1)} \cdot r_{P_1N}^{(0)}, \quad (12.62c)$$

$$r_{LP_1}^{(1)} = r_{LP_2}^{(0)} + r_{P_2P_1}^{(0)}, \quad (12.62d)$$

$$r_{P_2P_1}^{(1)} = r_{P_2P_1}^{(0)}, \quad (12.62e)$$

$$r_{LP_2}^{(1)} = r_{LP_2}^{(0)}. \quad (12.62f)$$

The six vectors after the second rotation for state i , $R_i^{(2)}$, are

$$r_{LN}^{(2)} = r_{LP_2}^{(1)} + R_i^{(2)} \cdot r_{P_2P_1}^{(1)} + R_i^{(2)} \cdot r_{P_1N}^{(1)}, \quad (12.63a)$$

$$r_{P_2N}^{(2)} = R_i^{(2)} \cdot r_{P_2P_1}^{(1)} + R_i^{(2)} \cdot r_{P_1N}^{(1)}, \quad (12.63b)$$

$$r_{P_1N}^{(2)} = R_i^{(2)} \cdot r_{P_1N}^{(1)}, \quad (12.63c)$$

$$r_{LP_1}^{(2)} = r_{LP_2}^{(1)} + R_i^{(2)} \cdot r_{P_2P_1}^{(1)}, \quad (12.63d)$$

$$r_{P_2P_1}^{(2)} = R_i^{(2)} \cdot r_{P_2P_1}^{(1)}, \quad (12.63e)$$

$$r_{LP_2}^{(2)} = r_{LP_2}^{(1)}. \quad (12.63f)$$

PCS and double pivoted motions

As defined in equation 12.59 on page 248, the PCS for state i is

$$\delta = \frac{c}{|r_i|} r_i^T \cdot A \cdot r_i. \quad (12.64)$$

For the double motion of the lanthanide-atom vector $r_{LN}^{(2)}$, this becomes

$$\delta = \frac{c}{|r_{LN}^{(2)}|^5} r_{LN}^{(2)T} \cdot A \cdot r_{LN}^{(2)}, \quad (12.65a)$$

$$= \frac{c}{|r_{LN}^{(2)}|^5} \left(r_{LP_2}^{(1)} + R_i^{(2)} \cdot r_{P_2P_1}^{(1)} + R_i^{(2)} \cdot r_{P_1N}^{(1)} \right)^T \cdot A \cdot \left(r_{LP_2}^{(1)} + R_i^{(2)} \cdot r_{P_2P_1}^{(1)} + R_i^{(2)} \cdot r_{P_1N}^{(1)} \right), \quad (12.65b)$$

$$= \frac{c}{|r_{LN}^{(2)}|^5} \left(r_{LP_2}^{(0)} + R_i^{(2)} \cdot r_{P_2P_1}^{(0)} + R_i^{(2)} \cdot R_i^{(1)} \cdot r_{P_1N}^{(0)} \right)^T \cdot A \cdot \left(r_{LP_2}^{(0)} + R_i^{(2)} \cdot r_{P_2P_1}^{(0)} + R_i^{(2)} \cdot R_i^{(1)} \cdot r_{P_1N}^{(0)} \right). \quad (12.65c)$$

12.2.5 Frame order in rotational Brownian diffusion and NMR relaxation

Free ellipsoidal Brownian diffusion

In Perrin's equations for free ellipsoidal Brownian diffusion ([Perrin, 1934, 1936](#)), the second degree frame order matrix elements $\overline{c_{ij}c_{kl}}$ are an essential step of the derivation. From [Perrin \(1936\)](#), the solution for the ellipsoidal diffusion equation is

$$\overline{c_{jj}c_{kk}} + \overline{c_{jk}c_{kj}} = e^{-(4\mathfrak{D}_i + \mathfrak{D}_j + \mathfrak{D}_k)t}, \quad (12.66)$$

$$\overline{c_{ii}^2} = \frac{1}{3} + \frac{1}{6}(2 + \mu_i) e^{-6(\mathfrak{D}_{iso} - \sqrt{\mathfrak{D}_{iso}^2 - \mathfrak{L}^2})t} + \frac{1}{6}(2 - \mu_i) e^{-6(\mathfrak{D}_{iso} + \sqrt{\mathfrak{D}_{iso}^2 - \mathfrak{L}^2})t}, \quad (12.67)$$

$$\overline{c_{jk}^2} = \frac{1}{3} - \frac{1}{6}(1 - \mu_i) e^{-6(\mathfrak{D}_{iso} - \sqrt{\mathfrak{D}_{iso}^2 - \mathfrak{L}^2})t} - \frac{1}{6}(1 + \mu_i) e^{-6(\mathfrak{D}_{iso} + \sqrt{\mathfrak{D}_{iso}^2 - \mathfrak{L}^2})t}, \quad (12.68)$$

where

$$\mu_i = \frac{\mathfrak{D}_i - \mathfrak{D}_{iso}}{\sqrt{\mathfrak{D}_{iso}^2 - \mathfrak{L}^2}}, \quad (12.69)$$

$$\mathfrak{D}_{iso} = \frac{1}{3} \sum_i \mathfrak{D}_i, \quad (12.70)$$

$$\mathfrak{L}^2 = \frac{1}{3} \sum_{i < j} \mathfrak{D}_i \mathfrak{D}_j, \quad (12.71)$$

\mathfrak{D}_i are the three diffusion rates, and c_{ij} are the direction cosines in the diffusion frame. According to [Perrin \(1936\)](#), because of the symmetry of the rotation the averages of the double-products $\overline{c_{ij}c_{kl}}$ where an index appears only once are zero and the second degree frame order matrix is represented by equation 12.15 on page 241. At time $t = 0$, the frame order matrix simplifies to

$$\mathbb{M}^{(n)}(0) = I_1, \quad (12.72)$$

and at time $t = \infty$ the matrix decays to

$$\mathbb{M}^{(n)}(\infty) = \frac{1}{3}I_2. \quad (12.73)$$

NMR relaxation

The free ellipsoid Brownian diffusion equations form the base theory for interpreting NMR relaxation data - the spheroidal and spherical diffusion equations are simply parametric restrictions of the full ellipsoid equations. As they are the definition of the frame order matrix, the frame order tensor can be seen as the modulator of all NMR relaxation processes.

12.3 Frame order modelling

12.3.1 Rigid body motions for a two domain system

Ball and socket joint

For a molecule consisting of two rigid bodies with pivoted inter-domain or inter-segment motions, the most natural mechanical description of the motion would be that of the spherical joint. This is also known as the ball and socket joint. The mechanical system consists of a single pivot point and three rotational degrees of freedom.

Tilt and torsion angles from robotics

To describe the motional mechanics of a ball and socket joint, the Euler angle system is a poor representation as the angles do not correspond to the mechanical modes of motion. In the field of robotics, many different orientation parameter sets have been developed for describing three degree-of-freedom joint systems. For the spherical joint description of intra-molecular rigid body motions, an angle system for describing symmetrical spherical parallel mechanisms (SPMs), a parallel manipulator, was found to be ideal. This is the ‘tilt-and-torsion’ angle system ([Huang et al., 1999](#); [Bonev and Gosselin, 2006](#)). These angles were derived independently a number of times to model human joint mechanics, originally as the ‘halfplane-deviation-twist’ angles ([Korein, 1985](#)), and then as the ‘tilt/twist’ angles ([Crawford et al., 1999](#)).

In the tilt-and-torsion angle system, the rigid body is first tilted by the angle θ about the horizontal axis a . The axis a , which lies in the xy -plane, is defined by the azimuthal angle ϕ (the angle is between the rotated z' axis projection onto the xy -plane and the x -axis). The tilt component is hence defined by both θ and ϕ . Finally the domain is rotated about the z' axis by the torsion angle σ . The resultant rotation matrix is

$$R(\theta, \phi, \sigma) = R_z(\phi)R_y(\theta)R_z(\sigma - \phi), \quad (12.74a)$$

$$= R_{zyz}(\sigma - \phi, \theta, \phi), \quad (12.74b)$$

$$= \begin{pmatrix} c_\phi c_\theta c_{\sigma-\phi} - s_\phi s_{\sigma-\phi} & -c_\phi c_\theta s_{\sigma-\phi} - s_\phi c_{\sigma-\phi} & c_\phi s_\theta \\ s_\phi c_\theta c_{\sigma-\phi} + c_\phi s_{\sigma-\phi} & -s_\phi c_\theta s_{\sigma-\phi} + c_\phi c_{\sigma-\phi} & s_\phi s_\theta \\ -s_\theta c_{\sigma-\phi} & s_\theta s_{\sigma-\phi} & c_\theta \end{pmatrix}, \quad (12.74c)$$

where $c_\eta = \cos(\eta)$ and $s_\eta = \sin(\eta)$ and R_{zyz} is the Euler rotation in zyz axis rotation notation where

$$\alpha = \sigma - \phi, \quad (12.75a)$$

$$\beta = \theta, \quad (12.75b)$$

$$\gamma = \phi. \quad (12.75c)$$

As $\sigma = \alpha + \gamma$, it can be seen that both these Euler angles influence the torsion angle, demonstrating the problem with this parameterisation.

Modelling torsion An advantage of this angle system is that the tilt and torsion components can be treated separately in the modelling of domain motions. The simplest model for the torsion angle would be the restriction

$$-\sigma_{\max} \leq \sigma \leq \sigma_{\max}. \quad (12.76)$$

The angle can be completely restricted as $\sigma_{\max} = 0$ to create torsionless models. In this case, the tilt and torsion rotation matrix simplifies to

$$R(\theta, \phi, 0) = R_z(\phi)R_y(\theta)R_z(-\phi), \quad (12.77a)$$

$$= R_{zyz}(-\phi, \theta, \phi), \quad (12.77b)$$

$$= \begin{pmatrix} c_\phi^2 c_\theta + s_\phi^2 & c_\phi s_\phi c_\theta - c_\phi s_\phi & c_\phi s_\theta \\ c_\phi s_\phi c_\theta - c_\phi s_\phi & s_\phi^2 c_\theta + c_\phi^2 & s_\phi s_\theta \\ -c_\phi s_\theta & -s_\phi s_\theta & c_\theta \end{pmatrix}. \quad (12.77c)$$

Modelling tilt The tilt angles θ and ϕ are related to spherical angles, hence the modelling of this component relates to a distribution on the surface of a sphere. At the simplest level, this can be modelled as both isotropic and anisotropic cones of uniform distribution.

Model list

For the modelling of the ordering of the motional frame, the tilt and torsion angle system will be used together with uniform distributions of rigid body position. For the torsion angle σ_{\max} , this can be modelled as being rigid ($\sigma_{\max} = 0$), being a free rotor ($\sigma_{\max} = \pi$), or as having a torsional restriction ($0 < \sigma_{\max} < \pi$). For the θ and ϕ angles of the tilt component, the rigid body motion can be modelled as being rigid ($\theta = 0$), as moving in an isotropic cone, or moving anisotropically in a pseudo-elliptic cone. Both single and double modes of motion have been modelled. The total list of models so far implemented are:

1. Rigid
2. Rotor
3. Free rotor
4. Isotropic cone
5. Isotropic cone, torsionless
6. Isotropic cone, free rotor
7. Pseudo-ellipse
8. Pseudo-ellipse, torsionless
9. Pseudo-ellipse, free rotor
10. Double rotor

The equations for these models are derived in Chapter 16 on page 375.

12.3.2 Frame order axis permutations

Multiple local minima exist in the optimisation space for the isotropic and pseudo-elliptic cone frame order models. In the case of the pseudo-ellipse, the eigenframes at each minimum are identical, however the θ_x , θ_y , and σ_{\max} half-angles are permuted. Because of the constraint $\theta_x \leq \theta_y$ in the pseudo-ellipse model, there are exactly three local minima (out of 6 possible permutations). In the isotropic cone, the $\theta_x \equiv \theta_y$ condition collapses this to two. The multiple minima correspond to permutations of the motional system - the eigenframe x, y and z-axes as well as the cone opening angles θ_x , θ_y , and σ_{\max} associated with these axes. But as the mechanics of the cone angles is not identical to that of the torsion angle, only one of the three local minima is the global minimum.

As the [minfx library](#) used in the frame order analysis currently only implements local optimisation algorithms, and because a global optimiser cannot be guaranteed to converge to the correct minima, a different approach is required:

- Optimise to one solution.
- Duplicate the data pipe for the model as ‘permutation A’.
- Permute the axes and amplitude parameters to jump from one local minimum to the other.
- Optimise the new permuted model, as the permuted parameters will not be exactly at the minimum.
- Repeat for the remaining ‘permutation B’ solution (only for the pseudo-ellipse models).

These steps have been incorporated into the automated analysis protocol.

The permutation step has been implemented as the `frame_order.permute_axes` user function. It is complicated by the fact that θ_x is defined as a rotation about the y-axis and θ_y is about the x-axis. See table 12.1 on page 257 for the pseudo-ellipse model permutations. These are also illustrated in figure 12.2 on page 255.

For the isotropic cone model, the same permutations exist but with some differences:

- The x and y axes are not defined in the x-y plane, therefore there are only two permutations (the first solution and ‘permutation A’).
- Any axis in the x-y plane can be used for the permutation, however different axes will result in different χ^2 values.
- As $\theta_x \equiv \theta_y$, the condition $\theta_x \leq \sigma_{\max} \leq \theta_y$ can only exist if the torsion and cone angles are identical.
- Permutations A and B create identical cones as the x and y axes are equivalent.

The new isotropic cone angle is defined as

$$\theta' = \frac{\theta'_x + \theta'_y}{2}. \quad (12.78)$$

The isotropic cone axis permutations are shown in figure 12.3 on page 256.

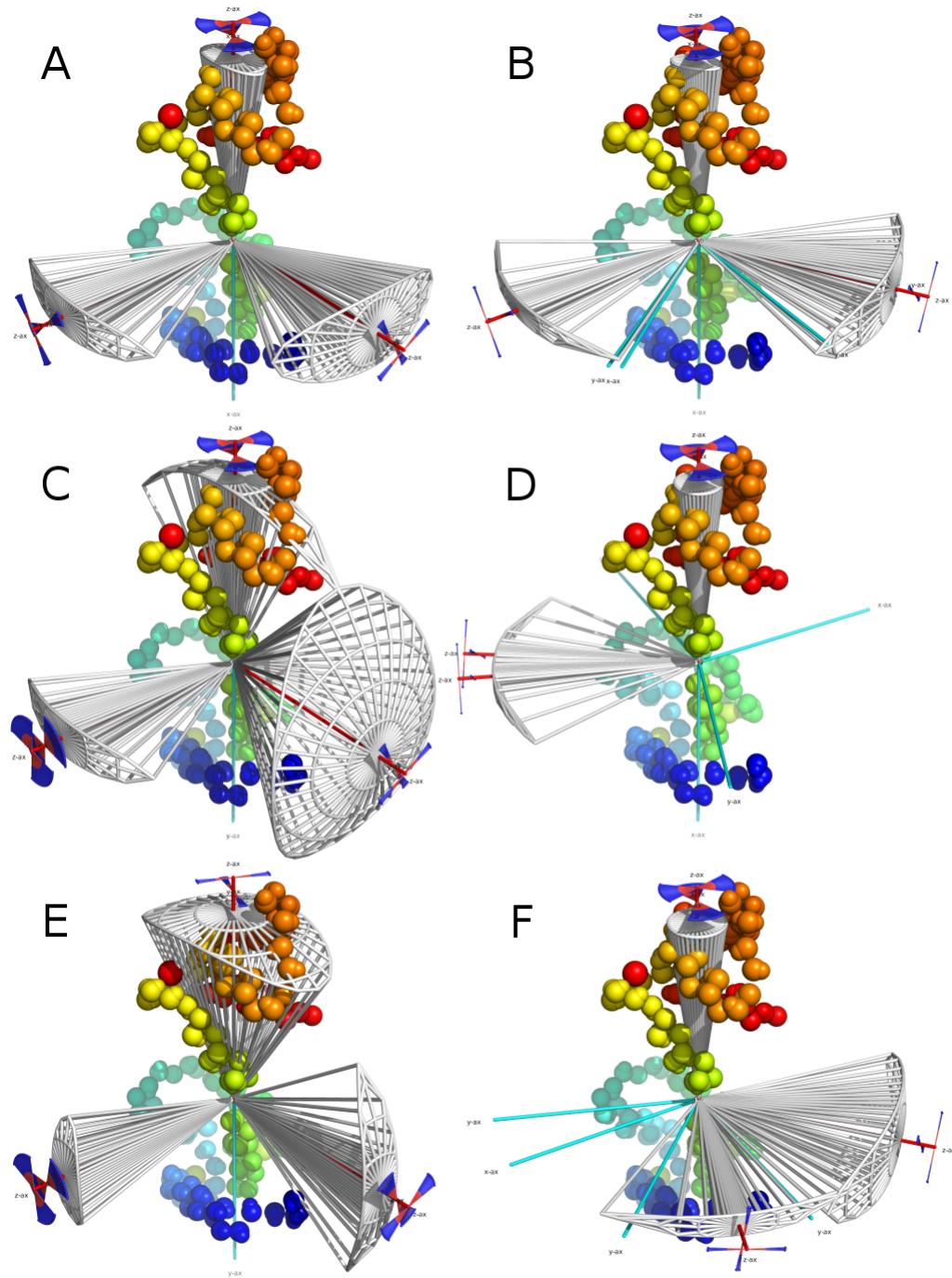


Figure 12.2: Pseudo-ellipse axis permutations. This uses synthetic data for a rotor model applied to CaM, with the rotor axis defined as being between the centre of the two helices between the domains (the centre of all cones in the figure) and the centre of mass of the C-terminal domain, and the rotor half-angle set to 30° . The condition $\theta_x \leq \theta_y \leq \sigma_{\max}$ is shown in A and B. The condition $\theta_x \leq \sigma_{\max} \leq \theta_y$ is shown in C and D. The condition $\sigma_{\max} \leq \theta_x \leq \theta_y$ is shown in E and F. A, C, and E are the axis permutations for a set of starting half-angles and B, D, and F are the results after low quality optimisation demonstrating the presence of the multiple local minima.

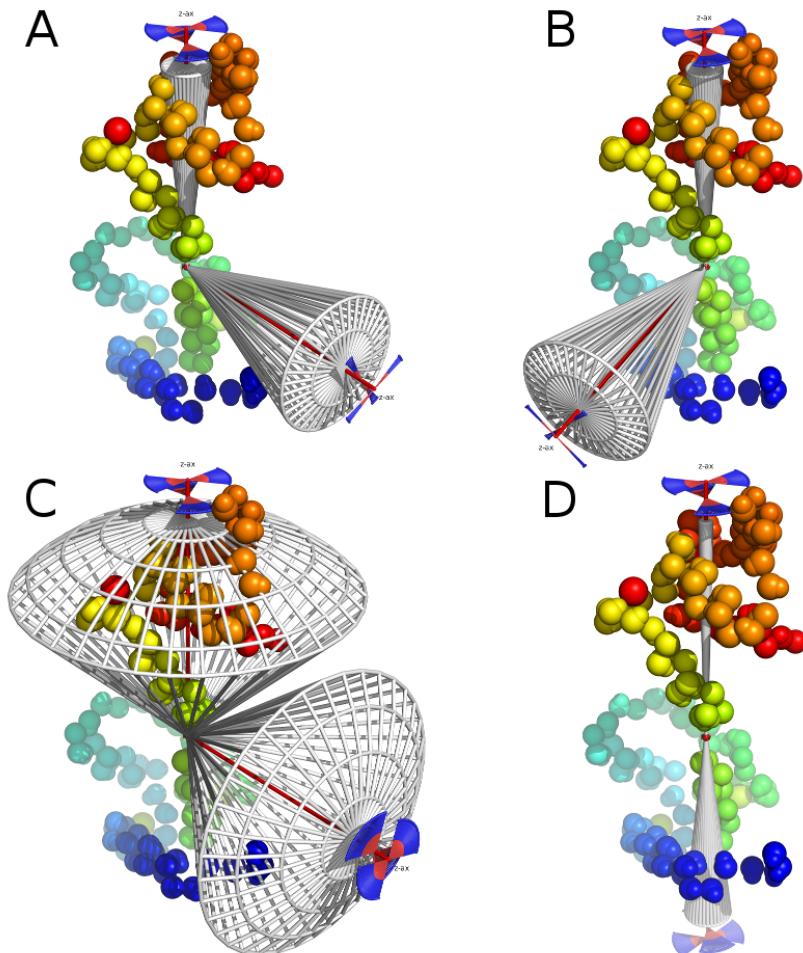


Figure 12.3: Isotropic cone axis permutations. This uses synthetic data for a rotor model applied to CaM, with the rotor axis defined as being between the centre of the two helices between the domains (the centre of all cones in the figure) and the centre of mass of the C-terminal domain, and the rotor half-angle set to 30° . The condition $\theta \leq \sigma_{\max}$ is shown in A and B. The condition $\sigma_{\max} \leq \theta$ is shown in C and D. A and C are the axis permutations for a set of starting half-angles and B and D are the results after low quality optimisation demonstrating the presence of the multiple local minima.

Table 12.1: The pseudo-ellipse motional eigenframe and half-angle permutations implemented in the `frame_order.permute_axes` user function.

Condition	Permutation name	Cone half-angles [$\theta'_x, \theta'_y, \sigma'_{\max}$]	Axes [x', y', z']
$\theta_x \leq \theta_y \leq \sigma_{\max}$	Self ¹	[$\theta_x, \theta_y, \sigma_{\max}$]	[x, y, z]
	A	[$\theta_x, \sigma_{\max}, \theta_y$]	[$-z, y, x$]
	B	[$\theta_y, \sigma_{\max}, \theta_x$]	[z, x, y]
$\theta_x \leq \sigma_{\max} \leq \theta_y$	Self ¹	[$\theta_x, \theta_y, \sigma_{\max}$]	[x, y, z]
	A	[$\theta_x, \sigma_{\max}, \theta_y$]	[$-z, y, x$]
	B	[$\sigma_{\max}, \theta_y, \theta_x$]	[$x, -z, y$]
$\sigma_{\max} \leq \theta_x \leq \theta_y$	Self ¹	[$\theta_x, \theta_y, \sigma_{\max}$]	[x, y, z]
	A	[$\sigma_{\max}, \theta_x, \theta_y$]	[y, z, x]
	B	[$\sigma_{\max}, \theta_y, \theta_x$]	[$x, -z, y$]

¹ The first optimised solution.

12.3.3 Linear constraints for the frame order models

Linear constraints are implemented for the frame order models using the log-barrier constraint algorithm in `minfx`, as this does not require the derivation of gradients.

The pivot point and average domain position parameter constraints in Ångstrom are:

$$-500 \leq P_x \leq 500, \quad (12.79a)$$

$$-500 \leq P_y \leq 500, \quad (12.79b)$$

$$-500 \leq P_z \leq 500, \quad (12.79c)$$

$$-999 \leq p_x \leq 999, \quad (12.79d)$$

$$-999 \leq p_y \leq 999, \quad (12.79e)$$

$$-999 \leq p_z \leq 999. \quad (12.79f)$$

These translation parameter restrictions are simply to stop the optimisation in the case of model failures. Converting these to the $A \cdot x \geq b$ matrix notation required for the optimisation constraint algorithm, the constraints become

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} P_x \\ P_y \\ P_z \\ p_x \\ p_y \\ p_z \end{pmatrix} \geq \begin{pmatrix} -500 \\ -500 \\ -500 \\ -500 \\ -500 \\ -500 \\ -500 \\ -999 \\ -999 \\ -999 \\ -999 \end{pmatrix} \quad (12.80)$$

For the order or motional amplitude parameters of the set \mathfrak{S} , the constraints used are

$$0 \leq \theta \leq \pi, \quad (12.81a)$$

$$0 \leq \theta_x \leq \theta_y \leq \pi, \quad (12.81b)$$

$$0 \leq \sigma_{\max} \leq \pi, \quad (12.81c)$$

$$0 \leq \sigma_{\max,2} \leq \pi. \quad (12.81d)$$

These reflect the range of validity of these parameters. Converting to the $A \cdot x \geq b$ notation, the constraints are

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \theta \\ \theta_x \\ \theta_y \\ \sigma_{\max} \\ \sigma_{\max,2} \end{pmatrix} \geq \begin{pmatrix} 0 \\ -\pi \\ 0 \\ -\pi \\ 0 \\ 0 \\ -\pi \\ 0 \\ -\pi \\ 0 \\ -\pi \end{pmatrix} \quad (12.82)$$

The pseudo-elliptic cone model constraint $\theta_x \geq \theta_y$ is used to simplify the optimisation space by eliminating symmetry.

12.4 Computation time and the numerical integration of the PCS

The numerical integration of the PCS using standard quadratic integration for the frame order models is impractical and cannot be used. A rough estimate for the computation time required for a full analysis of all the frame order models is in the order of 10^7 years. Therefore a number of tricks have been implemented to speed up the calculations.

12.4.1 Numerical integration techniques

The numerical integration is approximated as

$$\int f dV \approx V \langle f \rangle \pm V \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}}, \quad (12.83)$$

where the angular brackets are the means. As the average PCS value in the frame order models is defined as

$$\bar{\delta} = \int_S \delta dS / \int_S dS, \quad (12.84)$$

then

$$\bar{\delta} \approx \langle f \rangle \pm \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}}. \quad (12.85)$$

This simply means that the average PCS value of a set of N domain positions which satisfy the constraints of the current model parameter values can be used as the back-calculated PCS value for the model.

The simplest method to calculate the PCS value would be to generate a uniform distribution of domain positions. The PCS value is calculated for each state in the distribution of N structures and then averaged. However this technique has non-ideal convergence properties, hence the number N needs to be high to obtain a reasonable estimate of the PCS value. Two techniques with better convergence properties are the Monte Carlo integration algorithm and the quasi-random integration algorithms.

Monte Carlo numerical integration

As the convergence properties are better than that of a uniform distribution, the Monte Carlo integration algorithm is a viable option for using the PCS in the frame order analysis. Less states N are required for a reasonable estimate of the back-calculated PCS value. By randomly selecting N orientations of the domain which lie within the cone half-angle limits, back-calculating the PCS value for each state, and then averaging over all N states, the PCS value for the model can be numerically integrated. The original implementation used this technique.

Quasi-random numerical integration – Sobol' point sequence

Although the Monte Carlo numerical integration algorithm is a huge improvement on both the quadratic integration and the uniform distribution numerical integration techniques, it was nevertheless still too slow for optimising the frame order models. Therefore the quasi-random integration techniques were investigated, specifically using the Sobol' point sequence (Sobol', 1967).

To implement the numerical integration of the PCS using the quasi-random Sobol' sequence, the LGPL licenced Sobol library of John Burkardt and Corrado Chisari from http://people.sc.fsu.edu/~jburkardt/py_src/sobol/sobol.html was integrated into the relax library.

For each point coordinate s_i , the following functions were used to translate from linear space sampling to rotational space

$$\theta = \arccos(2s_i - 1), \quad (12.86a)$$

$$\phi = 2\pi s_i, \quad (12.86b)$$

$$\sigma = 2\pi(s_i - \frac{1}{2}), \quad (12.86c)$$

where θ is the frame order tilt angle (the angle of rotation about the x-y plane rotation axis), ϕ is the angle defining the x-y plane rotation axis, and σ is the torsion angle (the angle of rotation about the z' axis). Each frame order model uses a different set of θ , ϕ , and σ angles, therefore 1D, 2D, and 3D Sobol' sequences are generated.

Oversampling the Sobol' sequence points

As generating Sobol' points is computationally expensive, for speed this operation occurs during target function initialisation prior to optimisation. Hence the Sobol' points are not dynamically generated and a special algorithm is required to ensure adequate 1D, 2D and 3D sampling in the torsion-tilt angle space. The problem is that as the number of dimensions M increases, the density of a fixed N number of points in the M dimensions decreases. For example if a fixed value of N is chosen so that the pseudo-ellipse model is properly sampled, then the rotor model will be severely oversampled and take far too long to optimise. The protocol implemented to avoid this problem is:

- Generate $N.Ov.10^M$ points, where N is the maximum number of Sobol' points to be used, Ov is the oversampling factor defaulting to 1, and M is the dimensions of the torsion-tilt angle space.
- Convert all Sobol' points into torsion-tilt angles.
- Convert all angles to rotation matrices.

During optimisation, the following two checks are implemented:

- Skip points outside of the limits of the current parameter values.
- Terminate the loop over the Sobol' points once N is reached.

For most cases, N should be reached. However if the cone or torsion half-angles are extremely small, then the points used may be less than N . This is therefore monitored and printed out after each optimisation step. For these cases, Ov can be increased for better sampling. This is implemented in the `frame_order.sobol_setup` user function.

12.4.2 Parallelization and running on a cluster

Four different attempts at parallelizing the calculations using the MPI 2 protocol resulted in no speeds up. In each case, the calculations were up to 2 times slower. It appears as though the data transfer of the PCS, atomics positions, vectors, etc. between nodes is slower than the calculations. As parallelization for speeding up the calculations can only achieve around two orders of magnitude faster calculations, the technique was abandoned.

12.4.3 Frame order model nesting

The concept of model nesting is used to hugely speed up the optimisation in the automated protocol. The most complex models have 15 independent parameters, and performing a grid search over 15 dimensions of the pseudo-ellipse frame order model is not feasible when using PCS numerical integration. The idea is to use the optimised parameters of a simpler model as the starting point for a more complex model, avoiding the need for a grid search for those copied parameters. This appears to work as the PCS value is dominated by the average domain position, hence the average domain parameters are very similar in all models.

Model categories

The modelling of the σ torsion angle gives a number of categories of related models, those with no torsion, those with restricted torsion, and the free rotors.

No torsion When $\sigma = 0$, the following models are defined:

- Rigid,
- Isotropic cone, torsionless,
- Pseudo-ellipse, torsionless.

Restricted torsion When $0 < \sigma < \pi$, the following models are defined:

- Rotor,
- Isotropic cone,
- Pseudo-ellipse.

Free rotors When $\sigma = \pi$, i.e. there is no torsional restriction, the following models are defined:

- Free rotor,
- Isotropic cone, free rotor,
- Pseudo-ellipse, free rotor.

Multiple torsion angles This covers a single model – the double rotor.

Parameter categories

There are three major parameter categories – the average domain position, the eigenframe of the motion, and the amplitude of the motion.

Average domain position Let the translational parameters be

$$\mathfrak{T} = \{P_x, P_y, P_z\}, \quad (12.87)$$

and the rotational or orientational parameters be

$$\mathfrak{O} = \{P_\alpha, P_\beta, P_\gamma\}. \quad (12.88)$$

Two full average position parameter sets used in the frame order models are

$$\mathfrak{P} = \mathfrak{T} + \mathfrak{O} = \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\}, \quad (12.89a)$$

$$\mathfrak{P}' = \{P_x, P_y, P_z, P_\beta, P_\gamma\}. \quad (12.89b)$$

Table 12.2: The nesting of frame order model parameters and the resultant grid search dimensionality. The boxes highlight parameter sets which are optimised in the initial grid search. The start of each train of arrows are the optimised parameters which will be copied for the more complex model and excluded from the grid search. The non-nested grid search dimensionality is given in brackets.

Model	Parameter sets			Order parameters				Grid search dimensionality
Rigid	\mathfrak{P}	-	-	-	-	-	-	6 (6)
Rotor	\mathfrak{P}	$\mathfrak{E}_{\text{ax}}^{\alpha}$	\mathfrak{p}_1	-	-	-	σ_{\max}	-
Isotropic cone	\mathfrak{P}	\mathfrak{E}_{ax}	\mathfrak{p}_1	-	θ	-	σ_{\max}	-
Pseudo-ellipse	\mathfrak{P}	$\mathfrak{E}_{\alpha\beta\gamma}$	\mathfrak{p}_1	-	θ_x	θ_y	σ_{\max}	-
Isotropic cone, torsionless	\mathfrak{P}	\mathfrak{E}_{ax}	\mathfrak{p}_1	-	θ	-	-	0 (12)
Pseudo-ellipse, torsionless	\mathfrak{P}	$\mathfrak{E}_{\alpha\beta\gamma}$	\mathfrak{p}_1	-	θ_x	θ_y	-	0 (14)
Free rotor	\mathfrak{P}'	$\mathfrak{E}_{\text{ax}}^{\alpha}$	\mathfrak{p}_1	-	-	-	-	5 (9)
Isotropic cone, free rotor	\mathfrak{P}'	\mathfrak{E}_{ax}	\mathfrak{p}_1	-	θ	-	-	0 (11)
Pseudo-ellipse, free rotor	\mathfrak{P}'	$\mathfrak{E}_{\alpha\beta\gamma}$	\mathfrak{p}_1	-	θ_x	θ_y	-	0 (13)
Double rotor	\mathfrak{P}	$\mathfrak{E}_{\alpha\beta\gamma}$	\mathfrak{p}_1	\mathfrak{p}_2	-	-	σ_{\max}	$\sigma_{\max,2}$
								3 (15)

The motional eigenframe This consists of either the full eigenframe or a single axis, combined with the pivot point(s) defining the origin of the frame(s) within the PDB space. The eigenframe parameters themselves are

$$\mathfrak{E}_{\alpha\beta\gamma} = \{E_{\alpha}, E_{\beta}, E_{\gamma}\}, \quad (12.90a)$$

$$\mathfrak{E}_{\text{ax}} = \{E_{\theta}, E_{\phi}\}, \quad (12.90b)$$

$$\mathfrak{E}_{\text{ax}}^{\alpha} = \{E_{\alpha}^{\text{ax}}\}. \quad (12.90c)$$

The pivot parameter sets are

$$\mathfrak{p}_1 = \{p_x, p_y, p_z\}, \quad (12.91a)$$

$$\mathfrak{p}_2 = \{p_d\}, \quad (12.91b)$$

The rigid body ordering The parameters of order are

$$\mathfrak{S} = \{\theta, \theta_x, \theta_y, \sigma_{\max}, \sigma_{\max,2}\}. \quad (12.92)$$

Frame order parameter nesting in the automated protocol

The parameter nesting used in the automated protocol is shown in table 12.2. This massively collapses the dimensionality of the initial grid search.

12.4.4 PCS subset

Another trick that can be used to speed up the optimisation is to use a subset of all PCS values. The PCS data for a rigid domain can consist of hundreds of data points. Rather than using all these values, a very small subset of data points well chosen throughout the 3D structure – far apart, in rigid locations, and all nearby atoms having a similar value – can be used for the initial grid search and optimisation. As the PCS is most sensitive to the average domain position and less to the amplitude of motions, if the subset of atoms is well chosen the optimisation minimum for the subset should be very close to the optimisation minimum for the full data set. The subset minimum optimisation should be about two orders of magnitude faster as the number of PCS data points are linearly correlated with computation time. At the end of the analysis, a final stage of slower optimisation using all PCS data can be performed to find the full data set minimum.

In the case of the CaM analyses, a subset of five points was used. In the peptide bound calmodulin analyses, the H data of residues {Tyr 99, Val 108, Glu 114, Glu 119, Arg 126} was used. As data was not available for the same residues in the free calmodulin analysis, instead the proton data for residues {Tyr 99, Met 109, Lys 115, Glu 119, Glu 127} was used.

12.4.5 Optimisation of the frame order models

Due to the numerical integration of the PCS, optimisation is extremely slow. A number of optimisation techniques can help speed up this part including using a low precision initial grid search, a zooming grid search, an alternating grid search, and zooming precision optimisation.

Low precision grid search

Using 10,000 Sobol' points appears to be the minimum while still delivering about 2-3 decimal places of accuracy for the frame order models (when values are not close to zero). But rather than using this number of points, 100 can be used in the initial grid search. This is not accurate enough from the parameter perspective, but is close enough to the real minimum (local or global, see section [12.3.2](#) on page [254](#)). This speeds up the calculation by two orders of magnitude.

The zooming grid search

This is implemented in relax rather than in the [minfx optimisation library](#). As the grid search is parallelised to run on a cluster using OpenMPI, it can sometimes be advantageous to use a fine grid search to find a better optimisation starting position for the Nelder-Mead simplex algorithm. Rather than simply increasing the number of increments in the grid search, iteratively performing the grid search while zooming into the optimised parameter values is a more efficient alternative. This zooming grid search can be seen in the frame order optimisation script on page [268](#).

The alternating grid search

For finding the average domain position in the rigid frame order model, an alternating grid search has been implemented in the automated analysis protocol. The idea is to speed up the slow 6D grid search by first searching over the 3D rotational space, then the 3D translational space. The grid is then zoomed by one level and alternating grid search is repeated. As this technique does not guarantee convergence, it is not turned on by default. The results from the alternation should be carefully checked and the technique avoided if the average domain position is not reasonable.

Zooming precision optimisation

One trick is to perform an iterative optimisation whereby each iteration increases in precision. For this the function tolerance cutoff for optimisation is used. But the use of the amount of sampling through the Sobol' sequence can result in much greater speed ups. The reason is because each point in the Sobol' sequence requires a fixed amount of time to calculate the PCSs for all spins, so time is linearly correlated with the number of points.

Using synthetic CaM data found in `test_suite/shared_data/frame_order/cam/`, the ideal number of Sobol' points was found to be $\{100, 1000, 10000\}$. This is combined with a function tolerance cutoff of $\{1e^{-2}, 1e^{-3}, 1e^{-4}\}$.

12.4.6 Error analysis

Low precision Monte Carlo simulations

The propagation of errors via Monte Carlo simulation, which is extremely computationally expensive, can also be sped up. By only using a small number of simulations (50 to 100), 100 Sobol' points, and a function tolerance of $1e-3$, the computational time required for Monte Carlo simulations can be dramatically reduced. However, despite using the lower quality settings, with the current implementation of the frame order theory this error analysis is currently not computationally feasible.

12.5 The frame order data analysis

12.5.1 Introduction to frame order data analysis

The analysis consists of two parts. The first is the single structure or ensemble analysis of the combined RDC and PCS alignment tensor for the non-moving paramagnetically aligned domain or rigid body. The resultant alignment tensor and optimised paramagnetic Ln^{3+} position is then used as input into the frame order analysis.

12.5.2 The N-state model analysis scripts

For determining the alignment tensor and paramagnetic Ln^{3+} position from the RDC and PCS data, the N-state model or ensemble analysis should be used. The script is:

```

1 # Python imports.
2 from numpy import array
3
4 # relax imports.
5 from lib.physical_constants import NH_BOND_LENGTH_RDC
6
7
8 # Create a data pipe for all the data.
9 pipe.create('CaM N-dom', 'N-state')
10
11 # Load the CaM structure.
12 structure.read_pdb('2BE6_core_II_III.pdb', dir='../../../../structures/2BE6_superimpose/
    Ndom_II_III', set_mol_name=['CaM_A', 'IQ_A', 'Metals_A', 'CaM_B', 'IQ_B', 'Metals_B',
    'CaM_C', 'IQ_C', 'Metals_C'])
13
14 # Load the spins.
15 structure.load_spins('@N', from_mols=['CaM_A', 'CaM_B', 'CaM_C'], mol_name_target='CaM',
    ave_pos=False)
16 structure.load_spins('@H', from_mols=['CaM_A', 'CaM_B', 'CaM_C'], mol_name_target='CaM',
    ave_pos=False)
17
18 # Select only the superimposed spins (skipping mobile residues :2-4,42,56-57,76-80,
    identified from model-free order parameters).
19 select.spin(':17-41,43-55,58-67', change_all=True)
20 select.display()
21
22 # Define the magnetic dipole-dipole relaxation interaction.
23 interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
24 interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=NH_BOND_LENGTH_RDC)
25 interatom.unit_vectors(ave=False)
26
27 # Set the nuclear isotope and element.
28 spin.isotope(isotope='15N', spin_id='@N')
29 spin.isotope(isotope='1H', spin_id='@H')
30
31 # The alignment data.
32 align_data = [
33     ['dy', 'RDC_DY_111011_spinID.txt', 'PCS_DY_200911.txt', 900.00423401],
34     ['tb', 'RDC_TB_111011_spinID.txt', 'PCS_TB_200911.txt', 900.00423381],
35     ['tm', 'RDC_TM_111011_spinID.txt', 'PCS_TM_200911.txt', 900.00423431],
36     ['er', 'RDC_ER_111011_spinID.txt', 'PCS_ER_200911.txt', 899.90423151],
37     ['yb', 'RDC_YB_110112_spinID.txt', 'PCS_YB_211111.txt', 899.90423111],
38     ['ho', 'RDC_HO_300512_spinID.txt', 'PCS_HO_300412.txt', 899.80423481]
39 ]
40
41 # Loop over the alignments.
42 for i in range(len(align_data)):
43     # Alias the data.
44     TAG = align_data[i][0]
45     RDC_FILE = align_data[i][1]
46     PCS_FILE = align_data[i][2]
47     FRQ = align_data[i][3]
48
49     # RDCs.

```

```

50     rdc.read(align_id=TAG, file=RDC_FILE, dir='../../', data_type='D', spin_id1_col=1,
51     spin_id2_col=2, data_col=3, error_col=4)
52
53     # PCSs.
54     pcs.read(align_id=TAG, file=PCS_FILE, dir='../../', res_num_col=1, data_col=2,
55     error_col=4, spin_id='@N')
56     pcs.read(align_id=TAG, file=PCS_FILE, dir='../../', res_num_col=1, data_col=3,
57     error_col=4, spin_id='@H')
58
59     # The temperature.
60     spectrometer.temperature(id=TAG, temp=303.0)
61
62     # The frequency.
63     spectrometer.frequency(id=TAG, frq=FRQ, units='MHz')
64
65     # The paramagnetic centre (average Ca2+ position).
66     ave = array([7.608, 5.402, 16.725]) + array([7.295, 4.684, 16.168]) + array([7.338, 5.275,
67     16.086])
68     ave = ave / 3
69     paramag.centre(pos=ave)
70
71     # Set up the model.
72     n_state_model.select_model('fixed')
73
74     # Tensor optimisation.
75     print("\n\n# Tensor optimisation.\n\n")
76     minimise.grid_search(inc=5)
77     minimise.execute('newton', constraints=False)
78     state.save('tensor_only_fit', force=True)
79
80     # Optimisation of everything.
81     paramag.centre(fix=False)
82     minimise.execute('bfgs', constraints=False)
83
84     # PCS structural noise.
85     print("\n\n# Tensor optimisation with PCS structural noise.\n\n")
86     pcs.structural_noise(rmsd=0.3, sim_num=10000, file='structural_noise.agr', force=True)
87
88     # Optimisation of everything.
89     paramag.centre(fix=False)
90     minimise.execute('bfgs', constraints=False)
91
92     # Monte Carlo simulations.
93     monte_carlo.setup(number=500)
94     monte_carlo.create_data()
95     monte_carlo.initial_values()
96     minimise.execute('bfgs', constraints=False)
97     monte_carlo.error_analysis()
98
99     # Show the tensors.
100    align_tensor.display()
101
102    # Q-factors.
103    rdc.calc_q_factors()
104    pcs.calc_q_factors()
105
106    # Correlation plots.
107    rdc.corr_plot(file="rdc_corr.agr", force=True)
108    pcs.corr_plot(file="pcs_corr.agr", force=True)
109
110    # Save the program state.

```

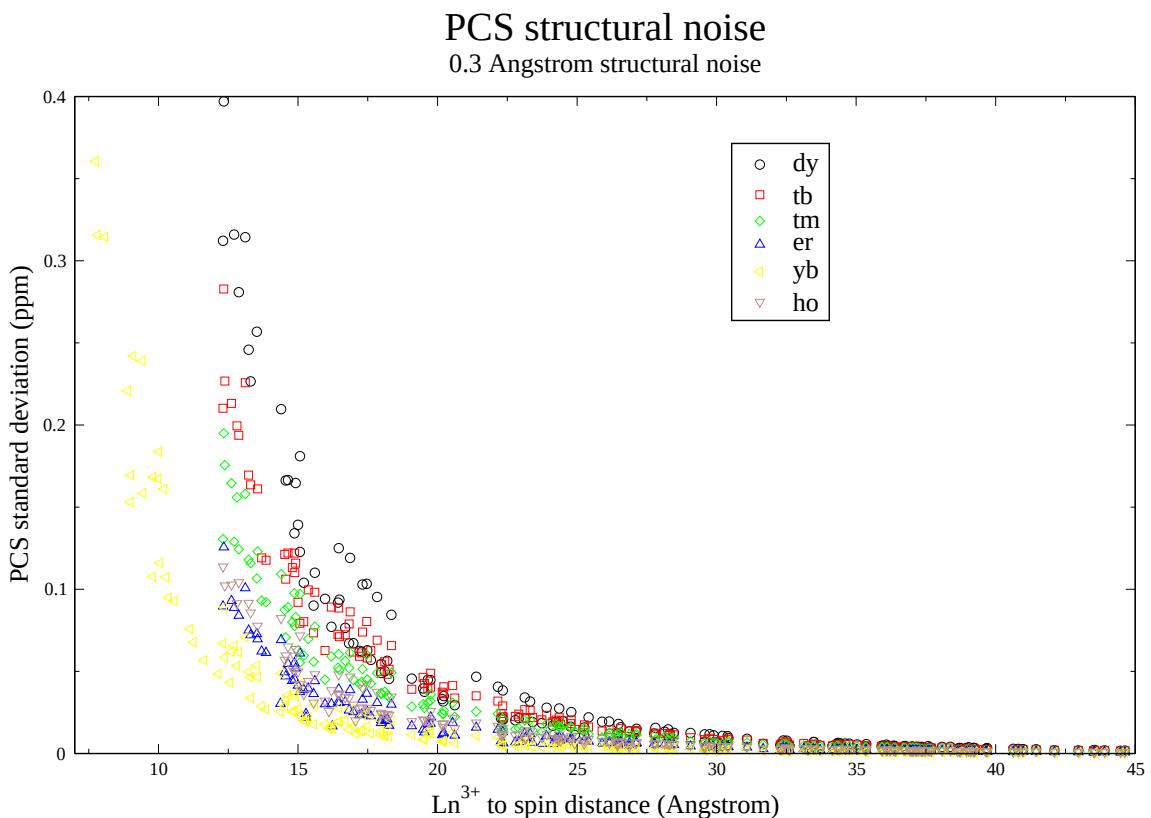


Figure 12.4: Structural noise simulation using the 2BE6 CaM-IQ X-ray ABC ensemble as an example. The simulated structural error, with the atom position uncertainty set to 0.3 Å, is an extra contribution to the measured PCS error. The PCS structural error is compared to the distance from the paramagnetic centre to illustrate the relationship between the two. The atomic positions were randomised 10,000 times with the lanthanide position assumed fixed, the PCS value back-calculated using a pre-fit alignment tensor, and the PCS standard deviations calculated.

```
107 state.save('tensor_fit', force=True)
```

The calculation of the additional PCS error due to structural noise is very important for the subsequent frame order analysis. A plot of such a calculation is shown in Figure 12.4

Statistics for the tensors, including the matrix or inter-tensor angles, the singular value decomposition (SVD) values and condition number, and a printout of the tensors for copying directly in the frame order analysis script, are obtained via the script:

```
1 # Script for determining the tensor angles and SVD values of the CaM-IQ tensors.
2
3 # Load the state.
4 state.load('tensor_fit')
5
6 # Loop over the alignment tensors, producing a string of relax user function commands.
7 print("\nTensor strings for relax input:")
8 string = ""
9 for A in cdp.align_tensors:
10     string += "align_tensor.init(tensor='%s', params=(%s, %s, %s, %s, %s), param_types=2)\\
11     % (A.name, A.Axx, A.Ayy, A.Axy, A.Axz, A.Ayz)
```

```

11     string += "align_tensor.init(tensor='%s', params=(%s, %s, %s, %s, %s), param_types=2,
12     errors=True)\n" % (A.name, A.Axx_err, A.Ayy_err, A.Axy_err, A.Axz_err, A.Ayz_err)
13 print(string)
14
15 # The matrix angles.
16 align_tensor.matrix_angles(basis_set=0)
17 align_tensor.matrix_angles(basis_set=1)
18
19 # The SVD analysis.
20 align_tensor.svd(basis_set=0)
21 align_tensor.svd(basis_set=1)

```

12.5.3 The frame order analysis scripts

The following is the example frame order analysis script located at `sample_scripts/frame_order/full_analysis.py`:

```

1 """Script for black-box Frame Order analysis.
2
3 This is for the CaM-IQ data.
4
5 The free rotor pseudo-elliptic cone model is not used in this script as the cone X and Y
6      opening angles cannot be differentiated with simply RDC and PCS data, hence this model
7      is perfectly approximated by the free rotor isotropic cone.
8 """
9
10
11 # Python module imports.
12 from numpy import array
13 from time import asctime, localtime
14
15 # relax module imports.
16 from auto_analyses.frame_order import Frame_order_analysis, Optimisation_settings
17
18 # Analysis variables.
19 #####
20
21 # The frame order models to use.
22 MODELS = [
23     'rigid',
24     'rotor',
25     'iso cone',
26     'pseudo-ellipse',
27     'iso cone, torsionless',
28     'pseudo-ellipse, torsionless',
29     'double rotor'
30 ]
31
32 # The number of Monte Carlo simulations to be used for error analysis at the end of the
33 # protocol.
34 MC_NUM = 10
35
36 # Rigid model optimisation setup.
37 OPT_RIGID = Optimisation_settings()
38 OPT_RIGID.add_grid(inc=21, zoom=0)
39 OPT_RIGID.add_grid(inc=21, zoom=1)
40 OPT_RIGID.add_grid(inc=21, zoom=2)
41 OPT_RIGID.add_grid(inc=21, zoom=3)

```

```

39  OPT_RIGID.add_min(min_algor='simplex')
40
41  # PCS subset optimisation setup.
42  OPT_SUBSET = Optimisation_settings()
43  OPT_SUBSET.add_grid(inc=11, zoom=0, sobol_max_points=100)
44  OPT_SUBSET.add_grid(inc=11, zoom=1, sobol_max_points=100)
45  OPT_SUBSET.add_grid(inc=11, zoom=2, sobol_max_points=100)
46  OPT_SUBSET.add_min(min_algor='simplex', func_tol=1e-2, sobol_max_points=100)
47
48  # Full data set optimisation setup.
49  OPT_FULL = Optimisation_settings()
50  OPT_FULL.add_grid(inc=11, zoom=2, sobol_max_points=100)
51  OPT_FULL.add_grid(inc=11, zoom=3, sobol_max_points=100)
52  OPT_FULL.add_min(min_algor='simplex', func_tol=1e-2, sobol_max_points=100)
53  OPT_FULL.add_min(min_algor='simplex', func_tol=1e-3, sobol_max_points=1000)
54  OPT_FULL.add_min(min_algor='simplex', func_tol=1e-4, sobol_max_points=10000)
55
56  # Monte Carlo simulation optimisation setup.
57  OPT_MC = Optimisation_settings()
58  OPT_MC.add_min(min_algor='simplex', func_tol=1e-3, sobol_max_points=100)
59
60
61  # Set up the base data pipes.
62  #####
63
64  # The data pipe bundle to group all data pipes.
65  PIPE_BUNDLE = "Frame Order (%s)" % asctime(localtime())
66
67  # Create the base data pipe containing only a subset of the PCS data.
68  SUBSET = "Data subset - " + PIPE_BUNDLE
69  pipe.create(pipe_name=SUBSET, pipe_type='frame order', bundle=PIPE_BUNDLE)
70
71  # Read the structures.
72  structure.read_pdb('2BE6_ndom_truncN.pdb', dir='../../../../structures/2BE6_superimpose',
73                      set_mol_name='N-dom')
73  structure.read_pdb('2BE6_cdom_truncC.pdb', dir='../../../../structures/2BE6_superimpose',
74                      set_mol_name='C-dom')
75
75  # Set up the 15N and 1H spins.
76  structure.load_spins(spin_id='@N', mol_name_target='CaM', ave_pos=False)
77  structure.load_spins(spin_id='@H', mol_name_target='CaM', ave_pos=False)
78  spin.isotope(isotope='15N', spin_id='@N')
79  spin.isotope(isotope='1H', spin_id='@H')
80
81  # Define the magnetic dipole-dipole relaxation interaction.
82  interatom.define(spin_id1='@N', spin_id2='@H', direct_bond=True)
83  interatom.set_dist(spin_id1='@N', spin_id2='@H', ave_dist=1.041 * 1e-10)
84  interatom.unit_vectors()
85
86  # Deselect mobile spins and vectors (from the CaM model-free order parameters from the
87  # BMRB).
87  deselect.spin(':2-4')
88  deselect.spin(':42')
89  deselect.spin(':56-57')
90  deselect.spin(':76-82')
91  deselect.spin(':114-117')
92  deselect.spin(':129-130')
93  deselect.spin(':146-148')
94  deselect.interatom(':2-4')
95  deselect.interatom(':42')
96  deselect.interatom(':56-57')

```

```

97 deselect.interatom(':76-82')
98 deselect.interatom(':114-117')
99 deselect.interatom(':129-130')
100 deselect.interatom(':146-148')

101 # The lanthanides and data files.
102 ln = ['dy', 'tb', 'tm', 'er', 'yb', 'ho']
103 pcs_files = [
104     'PCS_DY_200911.txt',
105     'PCS_TB_200911.txt',
106     'PCS_TM_200911.txt',
107     'PCS_ER_200911.txt',
108     'PCS_YB_211111.txt',
109     'PCS_HO_300412.txt'
110 ]
111 pcs_files_subset = [
112     'PCS_DY_200911_subset.txt',
113     'PCS_TB_200911_subset.txt',
114     'PCS_TM_200911_subset.txt',
115     'PCS_ER_200911_subset.txt',
116     'PCS_YB_211111_subset.txt',
117     'PCS_HO_300412_subset.txt'
118 ]
119 rdc_files = [
120     'RDC_DY_111011_spinID.txt',
121     'RDC_TB_111011_spinID.txt',
122     'RDC_TM_111011_spinID.txt',
123     'RDC_ER_111011_spinID.txt',
124     'RDC_YB_110112_spinID.txt',
125     'RDC_HO_300512_spinID.txt'
126 ]
127 ]
128
129 # The spectrometer frequencies for Luigi's measurements (matching the above lanthanide
# ordering, taken from the acqus SF01 parameter).
130 pcs_frq = [
131     701.2533001,    # Dy3+.
132     701.2533002,    # Tb3+.
133     701.2533005,    # Tm3+.
134     701.2533003,    # Er3+.
135     701.2533004,    # Yb3+.
136     701.2533005    # Ho3+.
137 ]
138 rdc_frq = [
139     900.00423401,    # Dy3+.
140     900.00423381,    # Tb3+.
141     900.00423431,    # Tm3+.
142     899.90423151,    # Er3+.
143     899.90423111,    # Yb3+.
144     899.80423481,    # Ho3+.
145 ]
146
147 # Loop over the alignments.
148 for i in range(len(ln)):
149     # Load the RDCs.
150     rdc.read(alin_id=ln[i], file=rdc_files[i], dir='../../../../align_data/CaM_IQ/',
151             spin_id1_col=1, spin_id2_col=2, data_col=3, error_col=4)
152
153     # The 1H PCS (only a subset of ~5 spins for fast initial optimisations).
154     pcs.read(alin_id=ln[i], file=pcs_files_subset[i], dir='../../../../align_data/CaM_IQ/',
155             res_num_col=1, data_col=3, error_col=4, spin_id='@H')

```

```

155 # The temperature and field strength.
156 spectrometer.temperature(id=ln[i], temp=303.0)
157 spectrometer.frequency(id=ln[i], frq=rdc_frq[i], units="MHz")
158
159 # Set up the tensors from the CaM-IQ ABC :6-74@N,CA,C,O N-state model fit.
160 align_tensor.init(tensor='Dy N-dom', align_id='dy', params=(-0.000895122969134, 0
161 .000200206126748, 0.000350783562498, 0.000789321179176, -0.000185956794915),
162 param_types=2)
163 align_tensor.init(tensor='Dy N-dom', align_id='dy', params=(1.2293468401e-05, 1
164 .74966511177e-05, 1.07296910627e-05, 1.21471537359e-05, 9.98472771055e-06),
165 param_types=2, errors=True)
166 align_tensor.init(tensor='Tb N-dom', align_id='tb', params=(-0.000386773980249, -0
167 .000252229451755, 0.000289345115245, 0.00077454551221, -0.000411564842864),
168 param_types=2)
169 align_tensor.init(tensor='Tb N-dom', align_id='tb', params=(9.08568851197e-06, 1
170 .25046911688e-05, 9.13279347879e-06, 8.66699785438e-06, 8.17084290029e-06),
171 param_types=2, errors=True)
172 align_tensor.init(tensor='Tm N-dom', align_id='tm', params=(0.000138832563763, 0
173 .00019276873546, -0.000401761891364, -0.00053984778662, 0.000385156710458),
174 param_types=2)
175 align_tensor.init(tensor='Tm N-dom', align_id='tm', params=(7.45028293534e-06, 1
176 .15087527652e-05, 7.80160598908e-06, 7.48687231235e-06, 8.44077530542e-06),
177 param_types=2, errors=True)
178 align_tensor.init(tensor='Er N-dom', align_id='er', params=(0.00013266928235, 6
179 .08491225722e-05, -0.000249892897607, -0.000344865388853, 0.000118692962249),
180 param_types=2)
181 align_tensor.init(tensor='Er N-dom', align_id='er', params=(6.24728334522e-06, 8
182 .68937486363e-06, 7.96726504939e-06, 6.43064935791e-06, 1.00354375045e-05),
183 param_types=2, errors=True)
184 align_tensor.init(tensor='Yb N-dom', align_id='yb', params=(0.000150564392882, -7
185 .59743643441e-05, -0.00013958907081, -0.000188379895441, 0.000102722198261),
186 param_types=2)
187 align_tensor.init(tensor='Yb N-dom', align_id='yb', params=(4.42731599871e-06, 5
188 .1565091874e-06, 5.18051425981e-06, 3.9225664592e-06, 4.49007020445e-06), param_types
189 =2, errors=True)
190 align_tensor.init(tensor='Ho N-dom', align_id='ho', params=(-0.000307522207243, 2
191 .76511812842e-05, 0.000152789357344, 0.000307999279733, -0.000235201851074),
param_types=2)
192 align_tensor.init(tensor='Ho N-dom', align_id='ho', params=(6.56971189673e-06, 1
193 .0420422445e-05, 8.05282585054e-06, 7.42469124453e-06, 7.25413636142e-06), param_types
194 =2, errors=True)
195
196 # Define the domains.
197 domain(id='N', spin_id=":1-78")
198 domain(id='C', spin_id=":80-148")
199
200 # The tensor domains and reductions.
201 full = ['Dy N-dom', 'Tb N-dom', 'Tm N-dom', 'Er N-dom', 'Yb N-dom', 'Ho N-dom']
202 red = ['Dy C-dom', 'Tb C-dom', 'Tm C-dom', 'Er C-dom', 'Yb C-dom', 'Ho C-dom']
203 ids = ['dy', 'tb', 'tm', 'er', 'yb', 'ho']
204 for i in range(len(full)):
205     # Initialise the reduced tensors (fitted during optimisation).
206     align_tensor.init(tensor=red[i], align_id=ids[i], params=(0, 0, 0, 0, 0))
207
208     # Set the domain info.
209     align_tensor.set_domain(tensor=full[i], domain='N')
210     align_tensor.set_domain(tensor=red[i], domain='C')
211
212     # Specify which tensor is reduced.
213     align_tensor.reduction(full_tensor=full[i], red_tensor=red[i])

```

```

192 # Set the reference domain.
193 frame_order.ref_domain('N')
194
195 # Set the initial pivot point.
196 pivot = array([ 21.863, 5.270, 5.934])
197 frame_order.pivot(pivot, fix=False)
198
199 # Set the paramagnetic centre position.
200 paramag.centre(pos=[6.518, 8.520, 13.767])
201
202 # Duplicate the PCS data subset data pipe to create a data pipe containing all the PCS
203 # data.
204 DATA = "Data - " + PIPE_BUNDLE
205 pipe.copy(pipe_from=SUBSET, pipe_to=DATA, bundle_to=PIPE_BUNDLE)
206 pipe.switch(DATA)
207
208 # Load the complete PCS data into the already filled data pipe.
209 for i in range(len(ln)):
210     # The 15N PCS.
211     pcs.read(align_id=ln[i], file=pcs_files[i], dir='../../../../align_data/CaM_IQ/',
212             res_num_col=1, data_col=2, error_col=4, spin_id='@N')
213
214     # The 1H PCS.
215     pcs.read(align_id=ln[i], file=pcs_files[i], dir='../../../../align_data/CaM_IQ/',
216             res_num_col=1, data_col=3, error_col=4, spin_id='@H')
217
218 # Execution.
219 #####
220 Frame_order_analysis(data_pipe_full=DATA, data_pipe_subset=SUBSET, pipe_bundle=PIPE_BUNDLE,
221                     , results_dir=None, opt_rigid=OPT_RIGID, opt_subset=OPT_SUBSET, opt_full=OPT_FULL,
222                     opt_mc=OPT_MC, mc_sim_num=MC_NUM, models=MODELS)

```

Once this analysis has been completed then, a refinement step is required. This is due to the low amount of motion in the system which causes the pivot point to be less well defined and hence strongly affected by artifacts of discrete sampling of a continuous and uniform distribution. The collapse of certain cone open half-angles and torsion angles to zero also causes the number of Sobol' points N to sometimes be zero. The refinement script is:

```

1 """Script for black-box Frame Order analysis.
2
3 This is for the CaM-IQ data.
4
5 The free rotor pseudo-elliptic cone model is not used in this script as the cone X and Y
6 opening angles cannot be differentiated with simply RDC and PCS data, hence this model
7 is perfectly approximated by the free rotor isotropic cone.
8 """
9
10 # Python module imports.
11 from time import asctime, localtime
12
13 # relax module imports.
14 from auto_analyses.frame_order import Frame_order_analysis, Optimisation_settings
15
16 # Analysis variables.
17 #####

```

```

17
18 # The frame order models to use.
19 MODELS = [
20     'rigid',
21     'rotor',
22     'iso cone',
23     'pseudo-ellipse',
24     'iso cone, torsionless',
25     'pseudo-ellipse, torsionless',
26     'double rotor'
27 ]
28
29 # The number of Monte Carlo simulations to be used for error analysis at the end of the
30 # protocol.
31 MC_NUM = 10
32
33 # Full data set optimisation setup.
34 OPT_FULL = Optimisation_settings()
35 OPT_FULL.add_min(min_algor='simplex', func_tol=1e-4, quad_int=True)
36
37 # Monte Carlo simulation optimisation setup.
38 OPT_MC = Optimisation_settings()
39 OPT_MC.add_min(min_algor='simplex', func_tol=1e-3, quad_int=True)
40
41 # Set up the base data pipes.
42 #####
43
44 # The data pipe bundle to group all data pipes.
45 PIPE_BUNDLE = "Frame Order (%s)" % asctime(localtime())
46
47
48 # Execution.
49 #####
50
51 # Do not change!
52 Frame_order_analysis(pipe_bundle=PIPE_BUNDLE, results_dir='refinement', pre_run_dir='.',
53                      opt_full=OPT_FULL, opt_mc=OPT_MC, mc_sim_num=MC_NUM, models=MODELS)

```

12.5.4 Computation times

It should be noted that computation times are currently very long, in the order of months. This depends on the models chosen, the amount and quality of the input data, and the refinement step. This is due to the numeric integration for the PCS data. Calculations can range from one to six months on a very fast computer. Unfortunately parallelization was not possible, but multiple analyses can be performed simultaneously on multiple core or multiple CPU systems. A dedicated machine with a reliable UPS system is highly recommended.

Part III

Power users

Chapter 13

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 section 3.3 on page 30). Announcements are sent to “nmr-relax-announce at lists.sourceforge.net” whereas “nmr-relax-users at lists.sourceforge.net” is the list where discussions about the usage of relax should be posted. “nmr-relax-devel at lists.sourceforge.net” 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, “nmr-relax-commits at lists.sourceforge.net” 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.

13.1 The relax source code repositories

Although the downloadable distribution archives can be modified, it is best that the most current and up to date code from relax’s version control repositories be modified instead.

13.1.1 relax repositories

The source and data files for relax are stored in version control repositories. This allows every single change which has ever been made to be recorded within the repository. Originally a single Subversion repository was used to hold the source code and webpages but, since the Gna! free software infrastructure shutdown, the sources and data were migrated to git repositories (see Section 3.1 on page 29 for a detailed history). If not already installed, you can download git from <https://git-scm.com/downloads>. More information about the basics of version control and how this is implemented in git can be found in the [git reference manual](#).

The source and data files are currently organised into the following repositories:

- The relax source code git repository, viewable at <https://sourceforge.net/p/nmr-relax/code/ci/master/tree/>.

- The relax website git repository, viewable at <https://sourceforge.net/p/nmr-relax/website/ci/master/tree/>.
- The relax demonstration file git repository, viewable at <https://sourceforge.net/p/nmr-relax/relax-demo/ci/master/tree/>.
- The archived relax source code and website SVN repository, viewable at <https://sourceforge.net/p/nmr-relax/code-svn-archive/HEAD/tree/>.

13.1.2 Primary relax repository

relax is currently hosted on the SourceForge open source infrastructure. For more information see the free software infrastructure chapter 3 on page 29.

If you are not currently a relax developer you can obtain the repository by typing

```
$ git clone git://git.code.sf.net/p/nmr-relax/code relax
```

Otherwise, if you already are a developer, type

```
$ git clone ssh://[username]@git.code.sf.net/p/nmr-relax/code relax
```

replacing [username] with your registered SourceForge login name. If your version is out of date, it can be updated to the latest commit by typing

```
$ git pull
```

If you are not currently a registered relax developer, changes can be made to forks of the repository (all details are below)

13.1.3 Mirrors of the relax repository

The relax developers are expected to push all changes to the primary relax repository currently located at SourceForge. However, for the future longevity of relax, this repository is mirrored on multiple free software/open source infrastructures. This should ensure access to relax for decades to come. The web interfaces to the current mirrors are:

- [nmr-relax at Bitbucket](#)
- [nmr-relax at GitHub](#)
- [nmr-relax at GitLab](#)
- [nmr-relax at SourceForge](#)

As a relax developer, you can set the mirrored repositories as remotes and push all changes to all mirrors. Firstly rename the default remote name `origin` with

```
$ git remote rename origin sf
```

Then add all remotes not already present by typing

```
$ git remote add bb git@bitbucket.org:nmr-relax/relax.git
$ git remote add gh git@github.com:nmr-relax/relax.git
$ git remote add gl git@gitlab.com:nmr-relax/relax.git
$ git remote add sf ssh://[username]@git.code.sf.net/p/nmr-relax/codet
```

You will need to be registered at these sites to be able to push to them:

- [Bitbucket account registration](#)
- [GitHub account registration](#)
- [GitLab account registration](#)
- [SourceForge account registration](#)

Other mirrors may be added in the future. To push all changes to the mirrors, type

```
$ git push bb
$ git push gh
$ git push gl
$ git push sf
$ git push --tags bb
$ git push --tags gh
$ git push --tags gl
$ git push --tags sf
```

The first commands only push the changes present on the currently checked out branch (creating the remote branch if necessary), and the second set push all tags. This can be simplified by using a git alias set in the git configuration file. For example on POSIX systems, in the file `.gitconfig` add

```
[alias]
pushremotes = !git remote | grep -v local_backup | xargs -L1 git push
```

Then instead type

```
$ git pushremotes
$ git pushremotes --tags
```

To download any changes present in any one of the mirrors, type

```
$ git fetch --all
```

13.2 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 is distributed with relax and is located at `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.

13.2.1 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 GNU/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).

13.2.2 Doc strings

The following are relax’s conventions for docstrings. Many of these are Python conventions.

- 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:

```
1 def delete(self):
2     """Function for deleting all model-free data."""
```

An example of a multiline docstring is:

```
1 def aic(chi2, k, n):
2     """Akaike's Information Criteria (AIC).
3
4     The formula is::
```

```

5
6     AIC = chi2 + 2k
7
8
9     @param chi2:      The minimised chi-squared value.
10    @type chi2:      float
11    @param k:         The number of parameters in the model.
12    @type k:         int
13    @param n:         The dimension of the relaxation data set.
14    @type n:         int
15    @return:        The AIC value.
16    @rtype:         float
17    """
18
19     return chi2 + 2.0*k

```

In addition to the text descriptions, the docstrings use the [Epydoc](#) markup language to describe arguments, return values, and other information about the code. See <http://epydoc.sourceforge.net/fields.html> for a listing of all the epydoc fields allowed. This markup language is important for the creation of the [API documentation](#) and to help developers understand the purpose and operation of the code.

13.2.3 Variable, function, and class names

In relax a mixture of both camel case (e.g. 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:

```

1  def assemble_param_vector(self, spin=None, spin_id=None, sim_index=None, model_type=
2      None):
3      """Assemble the model-free parameter vector (as numpy array).
4
5          If the spin argument is supplied, then the spin_id argument will be ignored.
6
7          @keyword spin:            The spin data container.
8          @type spin:             SpinContainer instance
9          @keyword spin_id:        The spin identification string.
10         @type spin_id:         str
11         @keyword sim_index:     The optional MC simulation index.
12         @type sim_index:       int
13         @keyword model_type:   The optional parameter set, one of 'all', 'diff', 'mf', or
14             'local_tm'.
15         @type model_type:     str or None
16         @return:              An array of the parameter values of the model-free model.
17         @rtype:               numpy array

```

```

16     """
17
18     # Initialise.
19     param_vector = []
20
21     # Determine the model type.
22     if not model_type:
23         model_type = self.determine_param_set_type()
24
25     # Diffusion tensor parameters.
26     if model_type == 'diff' or model_type == 'all':
27         # Normal parameters.
28         if sim_index == None:
29             # Spherical diffusion.
30             if cdp.diff_tensor.type == 'sphere':
31                 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 relax data store code. All the data analysis specific code, data pipe control 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:

```

1  class Model_free_main:
2      """Class containing functions specific to model-free analysis."""
3
4      def are_mf_params_set(self, spin):
5          """Test if the model-free parameter values are set.
6
7          @param spin:    The spin container object.
8          @type spin:    SpinContainer instance
9          @return:       The name of the first parameter in the parameter list in which the
10             corresponding parameter value is None. If all parameters are set, then None is
11             returned.
12             @rtype:        str or None
13
14             """
15
16             # Deselected residue.
17             if spin.select == 0:
18                 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.

13.2.4 Whitespace

The following conventions are for general code cleanliness and readability:

- Trailing whitespace is to be avoided.
- 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, 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:

```

1  class Internal:
2      """The internal relax structural data object.
3
4      The structural data object for this class is a container possessing a number of
5      different arrays corresponding to different structural information. These objects are
6      described in the structural container docstring.
7      """
8
9
10     def _bonded_atom(self, attached_atom, index, mol):
11         """Find the atom named attached_atom directly bonded to the atom located at the
12         index.
13
14         @param attached_atom: The name of the attached atom to return.
15         @type attached_atom: str
16         @param index: The index of the atom which the attached atom is attached
17         to.
18         @type index: int
19         @param mol: The molecule container.
20         @type mol: MolContainer instance
21         @return: A tuple of information about the bonded atom.
22         @rtype: tuple consisting of the atom number (int), atom name (str)
23         , element name (str), and atomic position (Numeric array of len 3)
24         """
25
26
27         # Init.
28         bonded_found = False
29
30
31         # No bonded atoms, so determine the connectivities.

```

```

24     if not mol.bonded[index]:
25         # Determine the molecule type if needed.
26         if not hasattr(mol, 'type'):
27             self._mol_type(mol)

```

13.2.5 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 is 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. When writing code for relax, please attempt to maintain the tradition by aiming towards a 25% comment ratio. The comment should be descriptive and explain the intent of the code below. The script `scripts/code_validator` can be used to check the comment density.

13.3 Committers

13.3.1 Becoming a committer

Anyone can become a relax developer and obtain commit access to the relax repository. Some criteria for selection by the relax developers include:

- To show good judgement.
- Competence in producing good patches.
- Compliance with the coding and commit log conventions.
- Comportment on and usage of the mailing lists.
- Not producing too many bugs.
- Only taking on challenges which can be handled.
- Being able to judge your own abilities.

You will also need to have an understanding of the concepts of version control specifically those relating to git. The git reference manual at <https://git-scm.com/docs> contains all the information you will need. After a number of commits have been accepted and merged, any of the relax developers can propose that you receive commit access to the parent repository. 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 the relax data

store are scrutinised, can be a good starting point for learning the structure of relax. This is because development within the test suite is isolate from normal program execution. Hence the relax test suite is an ideal sandbox.

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.

13.3.2 Register for a relax infrastructure account

The first step in becoming a relax committer is to create a SourceForge account. Simply go to <https://sourceforge.net/user/registration> and fill in the required details (many parts are optional). You will then receive a confirmation email.

13.3.3 Joining the relax project

The second step in becoming a committer is to ask to become a member of the relax project. Simply send an email to the relax development mailing list (see Section 3.3.3 on page 31). Note that as a relax developer, you are expected to subscribe to all of the relax mailing lists.

13.3.4 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 `git log` is legible. In vim, for example, this can be set in the vimrc file with the line `au FileType gitcommit set tw=100`.
- 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`). Also include a link to the tracker.
- 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 the old bug #5559 which is now archived at https://web.archive.org/web/https://gna.org/bugs/?func=detailitem&item_id=5559 and the post to the relax development mailing list describing the fix to that bug which is archived at <http://www.nmr-relax.com/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.

Example commit logs

Note these are old commit logs prior to the switch from SVN to git and prior to the Gna! shutdown (see Section 3.1 on page 29). 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 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.
```

13.3.5 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 development mailing list. If the planned changes have the potential to introduce problems, the creation of a branch may be suggested.

13.4 Submitting changes to the relax project

If you are not currently registered as a relax developer, to have your changes incorporated into relax you will either need to create patches or push your changes from your local repository to a fork of the primary or mirror relax repositories. These changes need to be committed to a dedicated development branch.

13.4.1 Development branches

When making changes to the relax git repository, you should never commit changes to the `master` branch. Instead a dedicated development branch must be used. For example to create a branch from the main development line, the `master` branch, called `molmol_macros` whereby new Molmol macros are to be written, type

```
$ git checkout master
```

This is to firstly switch `master` branch if not already there. Then create and switch to your new development branch by typing

```
$ git branch molmol_macros/r1
$ git checkout molmol_macros/r1
```

The reason for the `*/r1` suffix will be discussed later. The `master` branch should be kept clean, either by tracking the primary or mirror repository's `master` branch directly, or only performing mergers from that branch (as forks should do). Modifications can be made to this copy while normal development continues on the `master` branch.

13.4.2 Keeping the branch up to date

As you develop your branch, changes will be occurring simultaneously within the `master` branch. These changes should be incorporated into your branch on a regular basis to avoid large incompatible changes from forming between the two branches. For your changes to be incorporated into relax's `master` branch, the 'git rebase' concept and not 'git merge' should be used.

Versioning git branches

If you have already pushed your changes to a remote repository, you will need to create a new version of your branch. You should never rebase a branch that is tracking a remote

branch (unless you subsequently rename the rebased branch and untrack the remote). In the above example, the `molmol_macros/r1` branch is using such a manual versioning system. The second version of your branch can be created by typing

```
$ git checkout molmol_macros/r1
$ git branch molmol_macros/r2
$ git checkout molmol_macros/r2
```

You can then leave the `molmol_macros/r1` branch untouched and instead rebase the new `molmol_macros/r2` branch.

Rebasing to the master branch

Assuming you have not already pushed your branch to a remote repository, you can rebase your branch to the `master` branch. Firstly make sure your local copy of the `master` branch is fully up to date, then type

```
$ git rebase -i master
```

Or fetch all changes from the primary SourceForge remote (or a mirror) and rebase to the remote `master` branch with

```
$ git fetch sf
$ git rebase -i sf/master
```

If conflicts have occurred please refer to the [git reference manual](#) for information on how to resolve the problem. To avoid accidental loss of your changes when major conflicts occur, version control of your branch can be used to preserve the current set of changes (see Section 13.4.2). For example if the current branch is named `my_branch/r4`, increment to the fifth version of the branch with

```
$ git checkout my_branch/r4
$ git branch my_branch/r5
$ git checkout my_branch/r5
```

Then you can perform the risky rebase on the new `my_branch/r5` branch, knowing that the changes in `my_branch/r4` remain safe. This process is a lot less stressful than having to deal with ‘git reflog’ when things go wrong.

13.4.3 Submitting patches

Although not recommended, if you wish you can submit your changes as patches. You will need to have a local git repository and have your changes saved as git commits in a dedicated development branch. For details, see the [git reference manual](#). You will need to first make sure that your branch is fully up to date (Section 13.4.2 on page 287). The patches are then created by typing

```
$ git format-patch master
```

This will create one `*.patch` file per branch commit. These files can then be emailed to the `nmr-relax-devel` at `lists.sourceforge.net` mailing list (this requires a [subscription](#)).

13.4.4 Repository forks

A simpler process than submitting patches is to push your development branch to a fork of the relax repository. You can then send an email to the relax development mailing list (see Section 3.3.3 on page 31) describing and naming the branch and providing a web accessible URL to the git repository.

Web interface fork creation

The simplest way to create a relax fork is to use the web interface for the relax repository at SourceForge or one of the mirrors:

- [relax source code repository at Bitbucket](#)
- [relax source code repository at GitHub](#)
- [relax source code repository at GitLab](#)
- [relax source code repository at SourceForge](#)

Simply click on the ‘fork’ button or link and follow the instructions (Bitbucket requires a click on the plus symbol to see the fork link). This requires user registration at one of these sites (see Section 13.1.3 on page 278).

Once the repository is forked, you can clone that repository, create a versioned development branch, and push the branch. Or if you have a preexisting cloned relax repository you have worked on, set the fork as a remote and push the development branch to the new remote.

Pushing to an empty git repository

Alternatively, you can push your already cloned repository to a new git repository anywhere you wish. For example if you have your own server and create an empty bare repository accessible from `git@my_domain.com/relax/relax.git`, in the local clone type

```
$ git remote add my_remote git@my_domain.com/relax/relax.git  
$ git push --all my_remote  
$ git push --tags my_remote
```

This will push all branches and tags to the empty repository.

13.4.5 Merging the branch back into the main line

Once the changes have been accepted, possibly after a few revisions of the branch, it can be merged by one of the relax developers back into the primary relax repository. Taking the MOLMOL macro example again, assuming seven revisions of that branch, the relax developer will add your remote git repository as a remote and pull in the branch. Then it will be rebased and merged to `master` with

```
$ git checkout molmol_macros/r7
$ git rebase -i master
$ git checkout master
$ git merge molmol_macros/r7
```

This ensures a clean and linear history for the `master` branch. If the rebase process results in conflicts that the relax developer cannot fix, you will be asked to create a new `molmol_macros/r8` branch and perform the rebase and conflict resolution yourself.

Note that web interface ‘pull requests’ or ‘merge requests’ will not be used, and that it is better to send an email to the relax developer mailing list rather than to use these features of the web interface (which may be missed by the relax developers, especially when a relax mirror is used).

13.5 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.

13.5.1 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.

13.5.2 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.

13.5.3 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 L^AT_EX 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.

13.5.4 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 L^AT_EX sources.

13.5.5 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 directory `docs/api`.

13.5.6 Making distribution archives

Two types of distribution archive can be created from the current sources – the source and binary distributions. To create the source distribution type

```
$ scons source_dist
```

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 tag from the relax repository. If the current stable release is called 4.0.3, check out that tag by typing

```
$ git checkout 4.0.3
```

This requires you to be in a directory of a local copy of the relax git repository (see Section 13.1 on page 277).

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://sourceforge.net/p/nmr-relax/tickets/search/> 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 free software infrastructure chapter (Chapter 3, page 29).

13.5.7 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`, and any backup files with the extension `*.bak` are removed from all sub-directories. In addition any temporary L^AT_EX compilation files are removed from the `docs/latex` directory.

The more powerful command

```
$ scons clean_all
```

will, in addition to all the files removed by the `clean` target, remove all compiled C shared object files (`*.so`, `*.dylib`, `*.pyd`) and the `build` and `dist` directories created when building the Mac OS X application.

13.6 The core design of relax

To enable flexibility, the internal structure of relax has been designed to be highly 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 data analysis or even by other programs. The core modular design of the program is shown in Figure 13.1.

13.6.1 The divisions of relax's source code

relax's source code can be divided into a few major categories: the initialisation code, the user interface (UI) code, the functional code, the number crunching code, the code storing the program state, the multi-processor, the auto-analyses, and the test suite.

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 current UI modes in relax include the prompt, the script and the GUI modes. These consist of separate code paths, all sitting on top of the underlying functional code. It includes the user function code with is shared by all UIs.

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 data pipe control code, the specific analysis code, and the relax library.

Number crunching: The computationally expensive code belongs in this section. This also includes parts of the relax library as used by the target functions.

Program state: Most of the state of the program is contained within the relax data store which is accessible from all parts of the program as a singleton object. Additional non-persistent state information is held in the relax status singleton object.

Multi-processor: All relax execution passes through this interface which allows for uni-processor and OpenMPI multi-processor operations on code that has been specifically parallelised for this task.

Auto-analyses: Although not shown in Figure 13.1, these are essentially advanced relax scripts to simplify data analyses for users.

Test suite: This is where the majority of relax code is located. The relax test suite covers almost 100% of the functionality of relax. If code paths in relax are not tested in the test suite, they are considered equivalent to not existing. The extensive and almost complete coverage ensures that relax will be future-proof and continue to operate as expected.

13.6.2 The major components of relax

Each of the boxes in Figure 13.1 represents a different grouping of code.

relax: The top level module. This initialises the entire program, tests the dependencies, sets up the multi-processor framework and specific processor fabric, 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 user interface consisting of highly modified Python prompt. The namespace of the interpreter contains the various user functions which are front ends to the functional code. The auto-generated user function front end tests the supplied arguments to make sure they are of the correct type (string, integer, list, or any other type) before sending the values to the functional code. The code for the prompt is located in the `prompt/` directory and the user function front ends in the `user-functions/` directory.

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. The code sits alongside the prompt user interface in the `prompt/` directory.

GUI: The graphical user interface code base is located in the `gui/` directory. It is implemented using `wxPython` so that relax uses the native widgets of the operating system.

Other interfaces: Any number of interfaces (for example a web-based interface or an ncurses interface) could be added to relax with minimal modification of the rest of relax. This must be, without question, developed within the relax source code repository otherwise the code will not be maintainable in the future and will be almost impossible to merge back into relax later on. Due to the almost complete test suite coverage, relax is continually being refactored for modularity, flexibility, and speed hence any out-of-tree code that is not safeguarded by the test suite will quickly suffer bit-rot and depreciate in a short period of time.

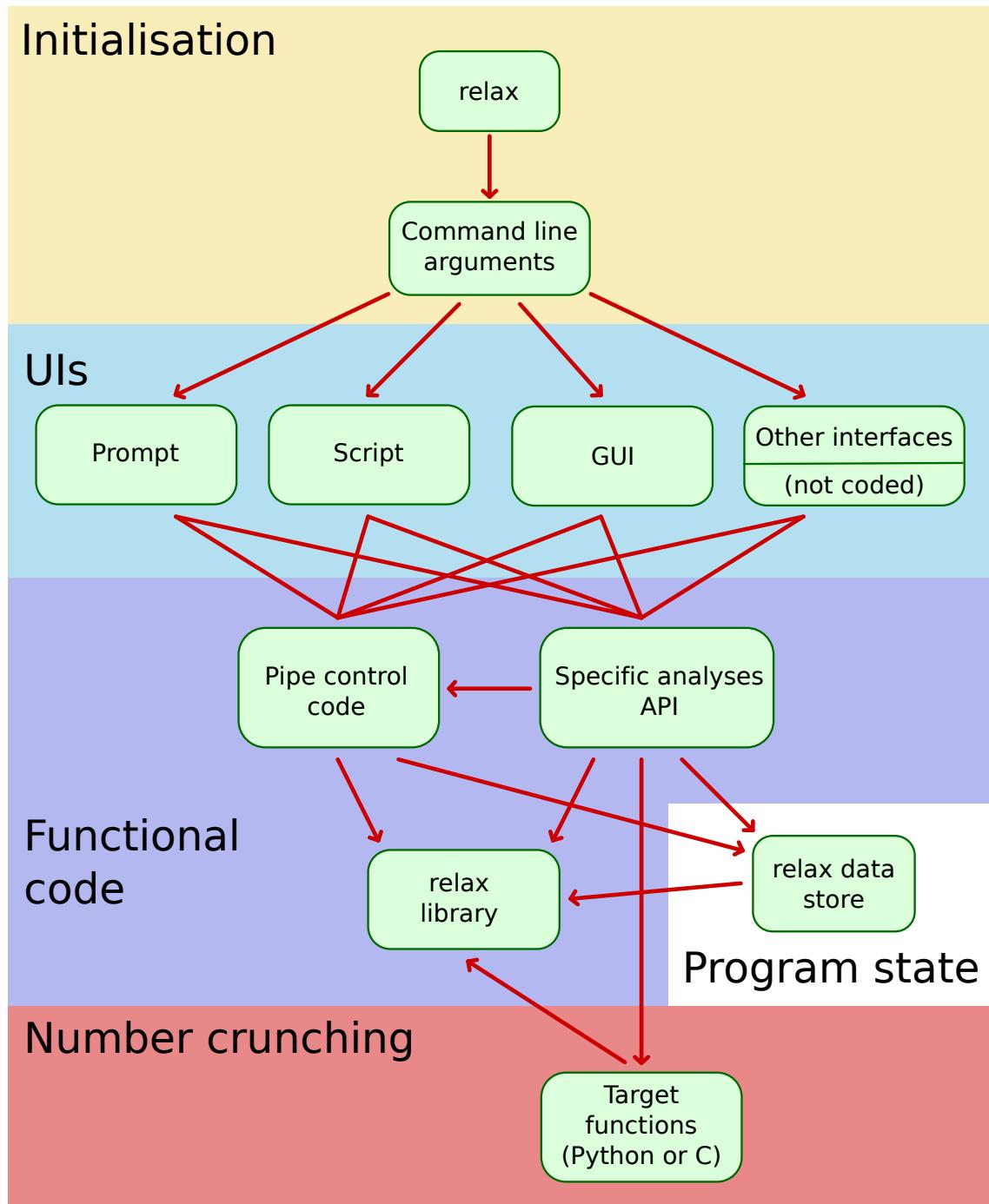


Figure 13.1: The core design of relax.

Pipe control code: This code includes classes and functions which are independent of the UI and not specific to a certain data pipe type or ‘specific analysis’. Pipe control is about managing the relax data store and it is located in the `pipe_control/` directory.

Specific analyses API: This is the code which is specific to the data pipe type – model-free analysis, relaxation curve-fitting, reduced spectral density mapping, the NOE calculation, consistency testing, the N-state or ensemble model analysis, relaxation dispersion, and the frame order analysis. Each type is located in a separate directory (Python package) within the `specific_analyses/` directory.

The relax library: This is a diverse collection of functions located within the `lib/` directory. Most independent functions that do not touch the relax data store or are not part of the internal relax API should be located within the relax library. This standalone library can be used independently of relax and it is used by all parts of relax. The relax data store combined with the relax library creates a development environment rivalling Mathematica, Matlab, Maxima, Octave, etc. but with a strong focus on NMR. This includes support for handling 3D molecular structures (or no structure), spectral data input, NMR phenomenon and many mathematics functions specific for NMR, and data visualization.

Target functions: This is reserved for CPU intensive code involved in optimisations and calculations. Most of relax’s execution time when performing an analysis is spent here. An optimisation algorithm feeds in different parameter values into a ‘target function’ in its attempt to minimise the single value returned by that target function (often a χ^2 value). The code may be written in Python however C code can be used to increase the speed of the calculations. Note that good design and the use of numpy can be orders of magnitude more important for speed than the choice of programming language. For optimisation the code can include function evaluations, calculation of gradients, and calculation of Hessians. These functions are located in the `target_functions/` directory.

Data: The program state is stored in the relax data store singleton object. This class contains all the program data and is accessed solely by the pipe control and specific analysis code. The data structures are located in the `data_store/` directory. Note that some temporary program execution information is stored in the relax status singleton object (located in `status.py`).

13.7 The mailing lists for development

13.7.1 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 copy-and-paste the following message (modifying it as you see fit):

Sorry in advance, but 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 send the message again. For this, please copy-and-paste your message, replying to the original (i.e. no forwarding), and making sure that the mailing list is in the CC field by clicking on “reply-to-all”.

13.8 The bug, task, and support request trackers

relax’s infrastructure includes three different issue trackers. These are the:

- Bug tracker.
- Support request tracker.
- Task tracker.

13.8.1 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: <https://sourceforge.net/p/nmr-relax/tickets/new/>. 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.

13.8.2 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.

13.8.3 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 edited and changed to “closed”. In addition, a message should be included that identifies the commits 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 #1](#), the first bug report on the SourceForge infrastructure, where the closing comment was:

This is fixed with commit [dee4fd3622158fb859c950a163017ac797b61ae9] (<https://sourceforge.net/p/nmr-relax/code/ci/dee4fd3622158fb859c950a163017ac797b61ae9>).

13.9 Links, links, and more links

Creating links throughout the relax infrastructure is important for two major reasons – navigation and search engine indexing.

13.9.1 Navigation

To be able to easily navigate between the relax infrastructure components – the bug tracker, the task tracker, the support request tracker, the mailing lists, the commit logs, and the git 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. When the bug is first added to the bug tracker, a link to the relax-users mailing list archive message 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 commit ID of the fix and a link to the relax-commits archive message. 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. 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.

13.9.2 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://www.nmr-relax.com> web site. The web of links is useful for catching those search engine bots. That way the searching of the mailing list archives will be more up to date (see the [communication web page](#)). However to increase the search engine ranking of the web site, links to <http://www.nmr-relax.com> from external sites is required. This is one reason why relax can be found at a number of sites across the web:

SourceForge: Free software/open source software hosting site. See <https://sourceforge.net/projects/nmr-relax/>.

Bitbucket: Free software/open source software hosting site. See <https://bitbucket.org/nmr-relax/>.

GitHub: Free software/open source software hosting site. See <https://github.com/nmr-relax>.

GitLab: Free software/open source software hosting site. See <https://gitlab.com/nmr-relax>.

The mail archive: This site archives all of the relax mailing lists, including [relax-announce](#), [relax-users](#), [relax-devel](#), and [relax-commits](#).

MARC - Mailing list ARChives: This site archives all of the relax mailing lists, including [relax-announce](#), [relax-users](#), [relax-devel](#), and [relax-commits](#).

Free Software Directory: This is the Free Software Foundation's (FSF) listing of free software. The relax page is <https://directory.fsf.org/wiki/relax>.

OpenHUB: OpenHUB, previously known as Ohloh, is a free public directory of Free and Open Source Software and the contributors who create and maintain it. This site provides a lot of useful statistical information about relax's development. The relax page is <https://www.openhub.net/p/nmr-relax>.

LinuxLinks.com: LinuxLinks.com, the Linux portal, is a website listing many Linux software projects. relax can be found on the [Software:Scientific:Biology:Proteins](#) page.

Softpedia: This is the encyclopedia of free software downloads. The relax page on Softpedia is <http://linux.softpedia.com/get/Science/relax-22351.shtml>. The relax developers pages are: [Edward d'Auvergne](#).

Pro-Linux: Diese ist eine der größten deutschsprachigen Seiten zum Thema Linux. The relax page is <http://www.pro-linux.de/cgi-bin/DBApp/check.cgi?ShowApp..10010.100>.

Open Source Software Directory: Use the Open Source Software Directory to find the best free and open-source software for home and business. relax is currently listed in the [molecule editors category](#).

Part IV

Advanced topics

Chapter 14

Optimisation

14.1 Implementation

14.1.1 The interface

Optimisation or minimisation is available in relax via the `minimise.grid_search` and `minimise.execute` user functions. The mathematical model optimised depends on the current data pipe type – it is implemented differently for each specific analysis. For analyses such as the steady state NOE (Chapter 6) or reduced spectral density mapping (Chapter 8), the solution can be found by direct calculation rather than optimisation. In these cases, the `minimise.calculate` user function should be used instead.

14.1.2 The minfx package

To minimise target functions within relax, the minfx optimisation library is used (<https://sourceforge.net/projects/minfx/>). This Python package is bundled with the official relax distribution archives. If you are using a version of relax checked out directly from the source code repository, you will need to manually install minfx as a standard Python package.

The minfx library originated as one of relax’s packages, but has been spun off as its own project for the benefit of other scientific, analytical, or numerical projects. Minfx is complete, very stable, well tested. Numerous optimisation algorithms are supported and can be clustered into three major categories – the line search methods, the trust-region methods, and the conjugate gradient methods.

The supported line search methods include:

- Steepest descent,
- Back-and-forth coordinate descent,
- Quasi-Newton BFGS,
- Newton,

- Newton-CG.

The supported trust-region methods include:

- Cauchy point,
- Dogleg,
- CG-Steihaug,
- Exact trust region.

The supported conjugate gradient methods include:

- Fletcher-Reeves,
- Polak-Ribière,
- Polak-Ribière +,
- Hestenes-Stiefel.

In addition, the following miscellaneous algorithms are implemented:

- Grid search,
- Nelder-Mead simplex,
- Levenberg-Marquardt.

The step selection subalgorithms include:

- Backtracking line search,
- Nocedal and Wright interpolation based line search,
- Nocedal and Wright line search for the Wolfe conditions,
- More and Thuente line search,
- No line search.

The Hessian modification subalgorithms include:

- Unmodified Hessian,
- Eigenvalue modification,
- Cholesky with added multiple of the identity,
- The Gill, Murray, and Wright modified Cholesky algorithm,

- The Schnabel and Eskow 1999 algorithm.

All methods can be constrained by:

- The Method of Multipliers (also known as the Augmented Lagrangian),
- The logarithmic barrier function.

These lists may be out of date, so please see the minfx website for additional information.

14.2 The optimisation 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

$$\hat{\theta} = \arg \min_{\theta} f(\theta), \quad (14.1)$$

where $\hat{\theta}$ is the parameter vector which is equal to the argument which minimises the function $f(\theta)$. In most analyses in relax, $f(\theta)$ is the chi-squared equation

$$\chi^2(\theta) = \sum_{i=1}^n \frac{(y_i - y_i(\theta))^2}{\sigma_i^2}, \quad (14.2)$$

where i is the summation index over all data, y_i is the experimental data, $y_i(\theta)$ is the back calculated data, and σ_i is the experimental error.

14.3 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.

14.3.1 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 . Conceptually this is the height of the space at the current position. For certain algorithms, such a simplex minimisation, this single value suffices.

14.3.2 The gradient

Most techniques also utilise the gradient at the current position. Although symbolically complex in the case of model-free analysis, for example, the gradient can simply be calculated as the vector of first partial derivatives of the chi-squared equation with respect to each parameter. It is defined as

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \\ \frac{\partial}{\partial \theta_2} \\ \vdots \\ \frac{\partial}{\partial \theta_n} \end{pmatrix} \quad (14.3)$$

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

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

$$f(\theta_k + x) \approx f_k + x^T \nabla f_k, \quad (14.4)$$

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.

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).

14.3.3 The Hessian

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

$$f(\theta_k + x) \approx f_k + x^T \nabla f_k + \frac{1}{2} x^T \nabla^2 f_k x, \quad (14.5)$$

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 . The Hessian is the matrix of second partial derivatives and is defined as

$$\nabla^2 = \begin{pmatrix} \frac{\partial^2}{\partial \theta_1^2} & \frac{\partial^2}{\partial \theta_1 \cdot \partial \theta_2} & \cdots & \frac{\partial^2}{\partial \theta_1 \cdot \partial \theta_n} \\ \frac{\partial^2}{\partial \theta_2 \cdot \partial \theta_1} & \frac{\partial^2}{\partial \theta_2^2} & \cdots & \frac{\partial^2}{\partial \theta_2 \cdot \partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial \theta_n \cdot \partial \theta_1} & \frac{\partial^2}{\partial \theta_n \cdot \partial \theta_2} & \cdots & \frac{\partial^2}{\partial \theta_n^2} \end{pmatrix}. \quad (14.6)$$

The order in which the partial derivatives are calculated is inconsequential, hence the Hessian is symmetric.

As the Hessian is computationally expensive a number of optimisation algorithms try to approximate it, the BFGS algorithm being a notable example. The most powerful minimisation algorithm for model-free analysis – Newton optimisation – requires the value, gradient, and Hessian at the current parameter values.

14.4 Optimisation algorithms

Prior to minimisation, all optimisation algorithms require a starting position within the optimisation 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.

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 \(1999\)](#).

14.4.1 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

$$\theta_{k+1} = \theta_k + \alpha_k p_k. \quad (14.7)$$

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.

The steepest descent 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 algorithm

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 $\theta_1, \theta_2, \theta_3, \theta_2, \theta_1, \theta_2, \dots$, 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 BFGS algorithm

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 (14.5) 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 Newton algorithm

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

$$p_k = -\nabla^2 f_k^{-1} \nabla f_k. \quad (14.8)$$

This direction is obtained from the derivative of (14.5) 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. This algorithm is also known as the Newton-Raphson method.

The Newton conjugate gradient algorithm

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 (14.8) 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.

The auxiliary step-length selection algorithm

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 $\theta_k + \alpha_k p_k$. One 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. Another 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.

14.4.2 Trust region methods

In the trust region class of algorithms the curvature of the space is modelled quadratically by (14.5). This model is assumed to be reliable only within a region of trust defined by the inequality $\|p\| \leq \Delta_k$ 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

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p, \quad \text{s.t. } \|p\| \leq \Delta_k, \quad (14.9)$$

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

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.

The dogleg algorithm

In changing the trust region radius the exact solutions to (14.9) 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 (14.9) can be the BFGS matrix, the unmodified Hessian, or a Hessian modified to be positive definite.

Steihaug's algorithm

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.

The exact trust region

An almost exact solution to (14.9) 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.

14.4.3 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, \dots, p_n which are all conjugate to the Hessian ([Nocedal and Wright, 1999](#)), a property defined as

$$p_i^T \nabla^2 f_k p_j = 0, \quad \text{for all } i \neq j. \quad (14.10)$$

By performing line searches over all directions p_j the solution to the quadratic model (14.5) 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. Preconditioned techniques include the Fletcher-Reeves algorithm which was the original conjugate gradient optimisation technique ([Fletcher and Reeves, 1964](#)), the Polak-Ribière method ([Polak and Ribi  re, 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 Mor   and Thuente auxiliary step-length selection algorithms will be tested with the CG algorithms.

14.4.4 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 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 $\nabla^2 f_k = L D L^T$, 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 spins together with the diffusion tensor parameters. The rate of convergence of the technique is quadratic.

14.4.5 Other methods

Nelder-Mead simplex

Some optimisation algorithms cannot be classified within line search, trust region, or conjugate gradient categories. For example the well known Nelder-Mead 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 amoeboid-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.

Levenberg-Marquardt algorithm

Another algorithm is the commonly used Levenberg-Marquardt algorithm ([Levenberg, 1944](#); [Marquardt, 1963](#)). This is the algorithm used by the model-free analysis programs Modelfree4, Dasha, and Tensor2. This technique is designed for least-squares problems to which the chi-squared equation (14.2) belongs. The key to the algorithm is the replacement of the Hessian with the Levenberg-Marquardt matrix $J^T J + \lambda I$, 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.

The technique does have one weak point though which is often mentioned only in the small print. That is that the algorithm catastrophically fails when the Levenberg-Marquardt matrix is singular. This occurs whenever a parameter is undefined. For example in a model-free analysis if the order parameter is one, then the corresponding internal correlation time can take any value. Performing a grid search with such undefined points greatly amplifies the problem and is the reason why many published model-free papers contain results with order parameters fixed at one ([d'Auvergne and Gooley, 2008b](#)).

14.5 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.

14.5.1 Method of Multipliers algorithm

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

$$\min_{\theta} \mathcal{L}_A(\theta, \lambda^k; \mu_k) \stackrel{\text{def}}{=} f(\theta) + \sum_{i \in \mathfrak{I}} \Psi(c_i(\theta), \lambda_i^k; \mu_k), \quad (14.11)$$

where the function Ψ is defined as

$$\Psi(c_i(\theta), \lambda^k; \mu_k) = \begin{cases} -\lambda^k c_i(\theta) + \frac{1}{2\mu_k} c_i^2(\theta) & \text{if } c_i(\theta) - \mu_k \lambda^k \leq 0, \\ -\frac{\mu_k}{2} (\lambda^k)^2 & \text{otherwise.} \end{cases} \quad (14.12)$$

In (14.11), θ is the parameter vector; \mathcal{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 $k \rightarrow \infty$; \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

$$\lambda_i^{k+1} = \max(\lambda_i^k - c_i(\theta)/\mu_k, 0), \quad \text{for all } i \in \mathfrak{I}. \quad (14.13)$$

The gradient of the Augmented Lagrangian is

$$\nabla \mathcal{L}_A(\theta, \lambda^k; \mu_k) = \nabla f(\theta) - \sum_{i \in \mathfrak{I} | c_i(\theta) \leq \mu_k \lambda_i^k} \left(\lambda_i^k - \frac{c_i(\theta)}{\mu_k} \right) \nabla c_i(\theta), \quad (14.14)$$

and the Hessian is

$$\nabla^2 \mathcal{L}_A(\theta, \lambda^k; \mu_k) = \nabla^2 f(\theta) + \sum_{i \in \mathfrak{I} | c_i(\theta) \leq \mu_k \lambda_i^k} \left[\frac{1}{\mu_k} \nabla c_i^2(\theta) - \left(\lambda_i^k - \frac{c_i(\theta)}{\mu_k} \right) \nabla^2 c_i(\theta) \right]. \quad (14.15)$$

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 $c_i \geq 0$. 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

$$l \leq \theta \leq u, \quad (14.16)$$

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

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} S^2 \\ \tau_e \\ R_{ex} \end{pmatrix} \leq \begin{pmatrix} 1 \\ \infty \\ \infty \end{pmatrix}. \quad (14.17)$$

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

$$A \cdot \theta \geq b, \quad (14.18)$$

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.

In rearranging (14.18) 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.

14.5.2 Logarithmic barrier constraint algorithm

Another constraint method is that of the logarithmic barrier algorithm. As in the Method of Multipliers this method is iterative. The function being minimised is replaced with

$$\Phi(\theta) = \begin{cases} \epsilon \sum_{i=1}^m -\log(b_i - A_i^T \theta) & \text{if } A \cdot \theta < b, \\ +\infty & \text{otherwise.} \end{cases} \quad (14.19)$$

The value of ϵ is increased with each iteration, increase the logarithmic penalty. An advantage of this method over the Method of Multipliers is that gradients are not required.

14.6 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. Again in model-free analyses 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.

Chapter 15

Optimisation of relaxation data – values, gradients, and Hessians

15.1 Introduction to the mathematics behind the optimisation of relaxation data

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, or any theory involving relaxation data. 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 model-free 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. They also apply to all other theories involving the base R_1 , R_2 , and steady-state NOE relaxation rates.

15.2 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.

15.2.1 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

$$\dim \mathfrak{F}_i = k \leqslant 5 \quad (15.1)$$

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.

15.2.2 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

$$\mathfrak{T}_i = \mathfrak{D}_i \cup \mathfrak{F}_i. \quad (15.2)$$

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

$$\dim \mathfrak{T}_i = 1 + k \leqslant 6, \quad (15.3)$$

where k is the number of model-free parameters.

15.2.3 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} ,

$$\mathfrak{D} = \mathfrak{G} \cup \mathfrak{O}. \quad (15.4)$$

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

$$\dim \mathfrak{D} = 1, \quad \dim \mathfrak{D} = 4, \quad \dim \mathfrak{D} = 6, \quad (15.5)$$

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

15.2.4 Optimisation of the global model \mathfrak{S}

The global model is defined as

$$\mathfrak{S} = \mathfrak{D} \cup \left(\bigcup_{i=1}^l \mathfrak{F}_i \right), \quad (15.6)$$

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

$$\dim \mathfrak{S} = \dim \mathfrak{D} + \sum_{i=1}^l k_i \leq 6 + 5l, \quad (15.7)$$

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 .

15.3 Construction of the values, gradients, and Hessians

15.3.1 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 (15.13) on page 321 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,

$$\chi^2 = \sum_{i=1}^l \chi_i^2, \quad (15.8)$$

where i is the residue index and l is the total number of residues used in the analysis. This is equivalent to Equation (15.13) when the index i ranges over the relaxation data of all selected residues.

15.3.2 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 15.1 the construction of the chi-squared gradient $\nabla \chi^2$ for the global model \mathfrak{S} is demonstrated. In this case

$$\nabla \chi^2 = \sum_{i=1}^l \nabla \chi_i^2, \quad (15.9)$$

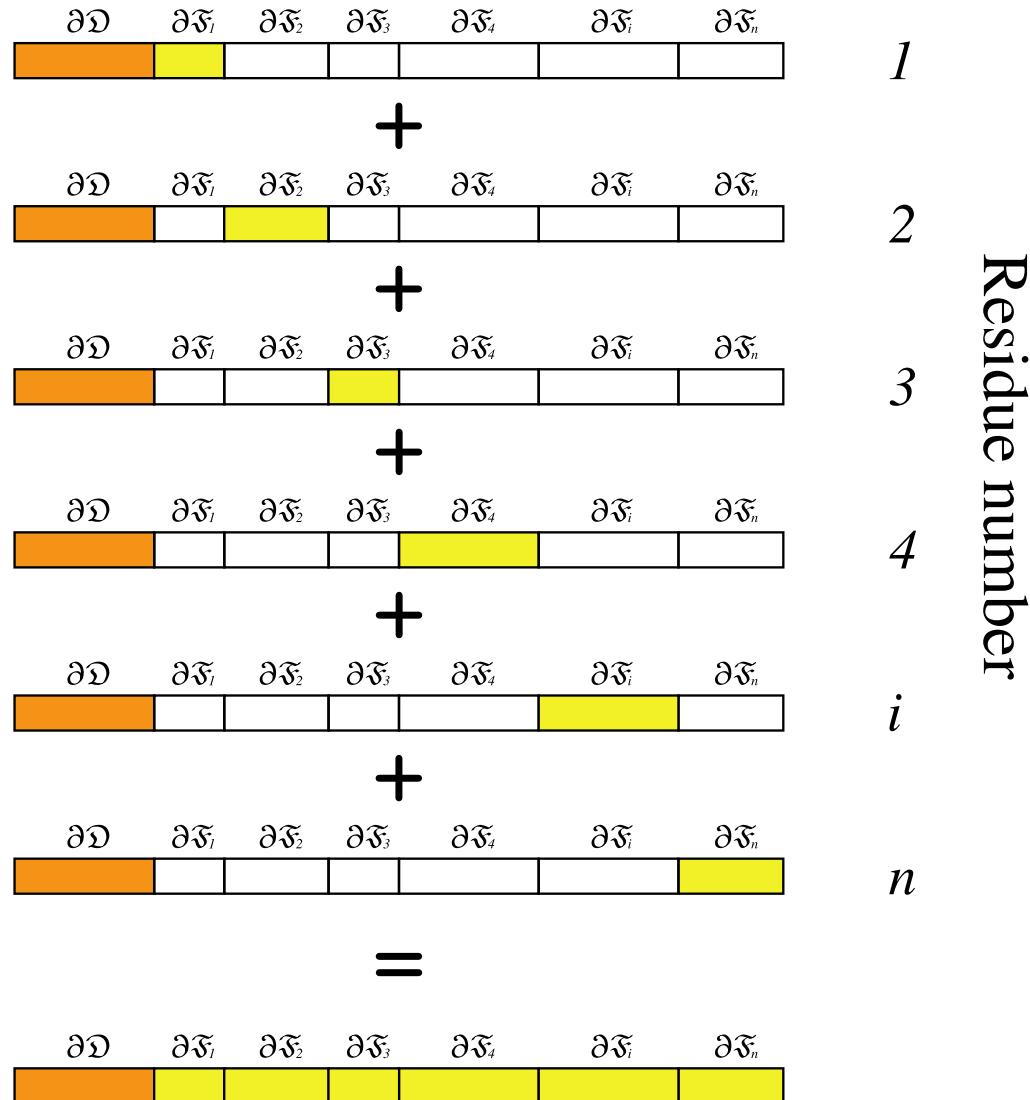


Figure 15.1: The construction of the model-free gradient $\nabla\chi^2$ for the global model \mathfrak{S} . For each residue i a different vector $\nabla\chi^2_i$ is constructed. The first element of the vector represented by the symbol $\partial\mathfrak{D}$ (the orange block) is the sub-vector of chi-squared 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 $\nabla\chi^2_i$ 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 $\nabla\chi^2_i$.

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

$$\|\nabla\chi_i^2\| = \dim \mathfrak{S}, \quad (15.10)$$

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 15.1 by simply taking the component of the gradient $\nabla\chi_i^2$ denoted by the symbol $\partial\mathfrak{D}$ (the orange blocks) and summing these for all residues. This sum is given by (15.9) and

$$\|\nabla\chi^2\| = \dim \mathfrak{D}. \quad (15.11)$$

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 15.1).

The model-free parameter set \mathfrak{F}_i is even simpler. In Figure 15.1 the gradient $\nabla\chi_i^2$ is simply the vector denoted by the single yellow block for the residue i .

15.3.3 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

$$\nabla^2\chi^2 = \sum_{i=1}^l \nabla^2\chi_i^2. \quad (15.12)$$

Figure 15.2 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.

15.4 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 15.3.

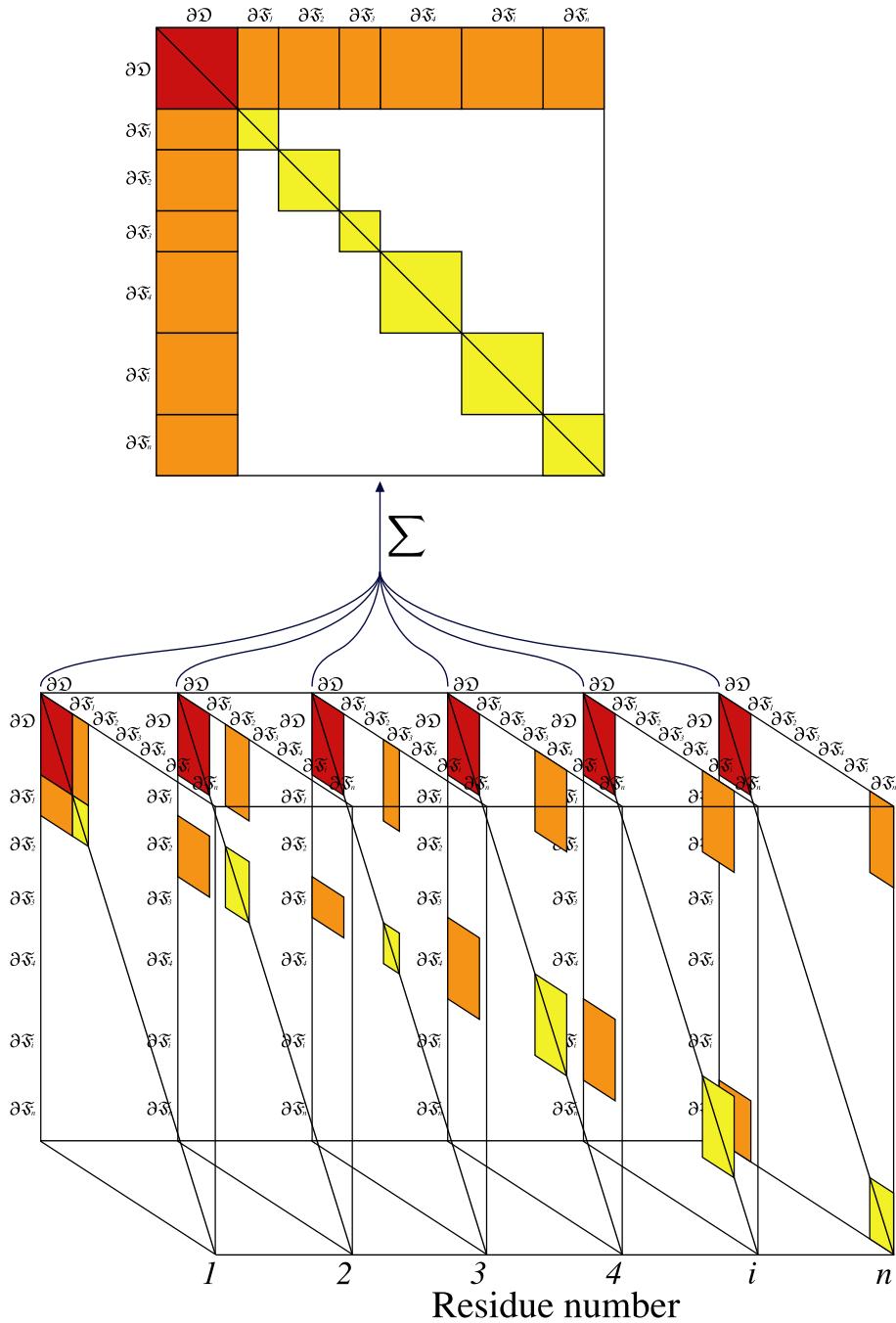


Figure 15.2: The model-free Hessian kite – a demonstration of the construction of the model-free Hessian $\nabla^2\chi^2$ for the global model \mathfrak{G} . For each residue i a different matrix $\nabla^2\chi_i^2$ is constructed. The first element of the matrix represented by the two symbols $\partial\mathfrak{D}$ (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}_i^k where $i \neq l$ are also zero. These blocks of sub-matrices are left uncoloured. The complete Hessian of \mathfrak{G} is the sum of the matrices $\nabla^2\chi_i^2$.

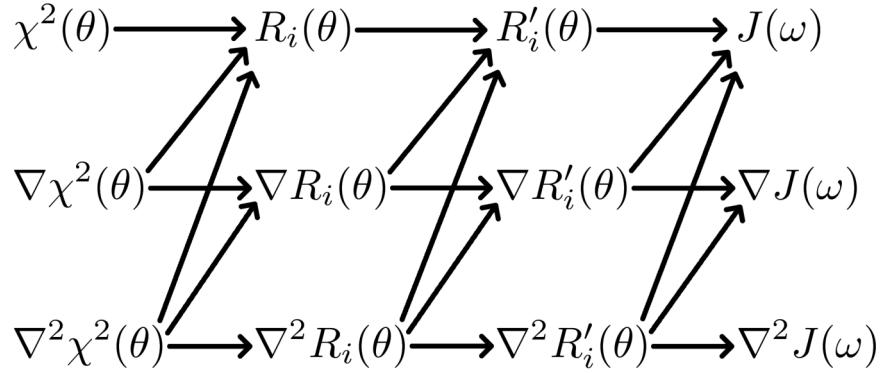


Figure 15.3: Dependencies between the χ^2 , transformed relaxation, relaxation, and spectral density equations, gradients, and Hessians.

15.5 The χ^2 value, gradient, and Hessian

15.5.1 The χ^2 value

The χ^2 value is defined as

$$\chi^2(\theta) = \sum_{i=1}^n \frac{(R_i - R_i(\theta))^2}{\sigma_i^2}, \quad (15.13)$$

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

15.5.2 The χ^2 gradient

The χ^2 gradient in vector notation is

$$\nabla \chi^2(\theta) = 2 \sum_{i=1}^n \frac{(R_i - R_i(\theta))^2}{\sigma_i^2} \nabla R_i(\theta). \quad (15.14)$$

15.5.3 The χ^2 Hessian

The χ^2 Hessian in vector notation is

$$\nabla^2 \chi^2(\theta) = 2 \sum_{i=1}^n \frac{1}{\sigma_i^2} (\nabla R_i(\theta) \cdot \nabla R_i(\theta)^T - (R_i - R_i(\theta)) \nabla^2 R_i(\theta)). \quad (15.15)$$

15.6 The $R_i(\theta)$ values, gradients, and Hessians

15.6.1 The $R_i(\theta)$ values

The $R_i(\theta)$ values are given by

$$R_1(\theta) = R'_1(\theta), \quad (15.16a)$$

$$R_2(\theta) = R'_2(\theta), \quad (15.16b)$$

$$\text{NOE}(\theta) = 1 + \frac{\gamma_H}{\gamma_X} \frac{\sigma_{\text{NOE}}(\theta)}{R_1(\theta)}. \quad (15.16c)$$

15.6.2 The $R_i(\theta)$ gradients

The $R_i(\theta)$ gradients in vector notation are

$$\nabla R_1(\theta) = \nabla R'_1(\theta), \quad (15.17a)$$

$$\nabla R_2(\theta) = \nabla R'_2(\theta), \quad (15.17b)$$

$$\nabla \text{NOE}(\theta) = \frac{\gamma_H}{\gamma_X} \frac{1}{R_1(\theta)^2} \left(R_1(\theta) \nabla \sigma_{\text{NOE}}(\theta) - \sigma_{\text{NOE}}(\theta) \nabla R_1(\theta) \right). \quad (15.17c)$$

15.6.3 The $R_i(\theta)$ Hessians

The $R_i(\theta)$ Hessians in vector notation are

$$\nabla^2 R_1(\theta) = \nabla^2 R'_1(\theta), \quad (15.18a)$$

$$\nabla^2 R_2(\theta) = \nabla^2 R'_2(\theta), \quad (15.18b)$$

$$\begin{aligned} \nabla^2 \text{NOE}(\theta) = & \frac{\gamma_H}{\gamma_X} \frac{1}{R_1(\theta)^3} \left[\sigma_{\text{NOE}}(\theta) \left(2 \nabla R_1(\theta) \cdot \nabla R_1(\theta)^T - R_1(\theta) \nabla^2 R_1(\theta) \right) \right. \\ & \left. - R_1(\theta) \left(\nabla \sigma_{\text{NOE}}(\theta) \cdot \nabla R_1(\theta)^T - R_1(\theta) \nabla^2 \sigma_{\text{NOE}}(\theta) \right) \right]. \end{aligned} \quad (15.18c)$$

15.7 $R'_I(\theta)$ values, gradients, and Hessians

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

15.7.1 Components of the $R'_I(\theta)$ equations

To simplify the calculations of the gradients and Hessians the $R'_I(\theta)$ 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 $\{R_1, R_2, \sigma_{\text{NOE}}\}$. 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

$$d = \frac{1}{4} \left(\frac{\mu_0}{4\pi} \right)^2 \frac{(\gamma_H \gamma_X \hbar)^2}{\langle r^6 \rangle}. \quad (15.19)$$

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

$$d' \equiv \frac{dd}{dr} = -\frac{3}{2} \left(\frac{\mu_0}{4\pi} \right)^2 \frac{(\gamma_H \gamma_X \hbar)^2}{\langle r^7 \rangle}. \quad (15.20)$$

The second derivative with respect to the bond length is

$$d'' \equiv \frac{d^2d}{dr^2} = \frac{21}{2} \left(\frac{\mu_0}{4\pi} \right)^2 \frac{(\gamma_H \gamma_X \hbar)^2}{\langle r^8 \rangle}. \quad (15.21)$$

CSA constant

The CSA constant is defined as

$$c = \frac{(\omega_X \cdot \Delta\sigma)^2}{3}. \quad (15.22)$$

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

$$c' \equiv \frac{dc}{d\Delta\sigma} = \frac{2\omega_X^2 \cdot \Delta\sigma}{3}. \quad (15.23)$$

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

$$c'' \equiv \frac{d^2c}{d\Delta\sigma^2} = \frac{2\omega_X^2}{3}. \quad (15.24)$$

R_{ex} constant

The R_{ex} constant is defined as

$$R_{ex} = \rho_{ex}(2\pi\omega_H)^2. \quad (15.25)$$

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

$$R'_{ex} \equiv \frac{dR_{ex}}{d\rho_{ex}} = (2\pi\omega_H)^2. \quad (15.26)$$

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

$$R''_{ex} \equiv \frac{d^2R_{ex}}{d\rho_{ex}^2} = 0. \quad (15.27)$$

Spectral density terms of the R_1 dipolar component

For the dipolar component of the R_1 equation (7.3a) on page 86 the spectral density terms are

$$J_d^{R_1} = J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H + \omega_X). \quad (15.28)$$

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

$$J_d^{R_1'} \equiv \frac{\partial J_d^{R_1}}{\partial \theta_j} = \frac{\partial J(\omega_H - \omega_X)}{\partial \theta_j} + 3 \frac{\partial J(\omega_X)}{\partial \theta_j} + 6 \frac{\partial J(\omega_H + \omega_X)}{\partial \theta_j}. \quad (15.29)$$

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

$$J_d^{R_1''} \equiv \frac{\partial^2 J_d^{R_1}}{\partial \theta_j \cdot \partial \theta_k} = \frac{\partial^2 J(\omega_H - \omega_X)}{\partial \theta_j \cdot \partial \theta_k} + 3 \frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k} + 6 \frac{\partial^2 J(\omega_H + \omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \quad (15.30)$$

Spectral density terms of the R_1 CSA component

For the CSA component of the R_1 equation (7.3a) on page 86 the spectral density terms are

$$J_c^{R_1} = J(\omega_X). \quad (15.31)$$

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

$$J_c^{R_1'} \equiv \frac{\partial J_c^{R_1}}{\partial \theta_j} = \frac{\partial J(\omega_X)}{\partial \theta_j}. \quad (15.32)$$

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

$$J_c^{R_1''} \equiv \frac{\partial^2 J_c^{R_1}}{\partial \theta_j \cdot \partial \theta_k} = \frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \quad (15.33)$$

Spectral density terms of the R_2 dipolar component

For the dipolar component of the R_2 equation (7.3b) on page 86 the spectral density terms are

$$J_d^{R_2} = 4J(0) + J(\omega_H - \omega_X) + 3J(\omega_X) + 6J(\omega_H) + 6J(\omega_H + \omega_X). \quad (15.34)$$

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

$$J_d^{R_2'} \equiv \frac{\partial J_d^{R_2}}{\partial \theta_j} = 4 \frac{\partial J(0)}{\partial \theta_j} + \frac{\partial J(\omega_H - \omega_X)}{\partial \theta_j} + 3 \frac{\partial J(\omega_X)}{\partial \theta_j} + 6 \frac{\partial J(\omega_H)}{\partial \theta_j} + 6 \frac{\partial J(\omega_H + \omega_X)}{\partial \theta_j}. \quad (15.35)$$

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

$$\begin{aligned} J_d^{R_2''} \equiv \frac{\partial^2 J_d^{R_2}}{\partial \theta_j \cdot \partial \theta_k} &= 4 \frac{\partial^2 J(0)}{\partial \theta_j \cdot \partial \theta_k} + \frac{\partial^2 J(\omega_H - \omega_X)}{\partial \theta_j \cdot \partial \theta_k} + 3 \frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k} \\ &\quad + 6 \frac{\partial^2 J(\omega_H)}{\partial \theta_j \cdot \partial \theta_k} + 6 \frac{\partial^2 J(\omega_H + \omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \end{aligned} \quad (15.36)$$

Spectral density terms of the R_2 CSA component

For the CSA component of the R_2 equation (7.3b) on page 86 the spectral density terms are

$$J_c^{R_2} = 4J(0) + 3J(\omega_X). \quad (15.37)$$

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

$$J_c^{R_2'} \equiv \frac{\partial J_c^{R_2}}{\partial \theta_j} = 4 \frac{\partial J(0)}{\partial \theta_j} + 3 \frac{\partial J(\omega_X)}{\partial \theta_j}. \quad (15.38)$$

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

$$J_c^{R_2''} \equiv \frac{\partial^2 J_c^{R_2}}{\partial \theta_j \cdot \partial \theta_k} = 4 \frac{\partial^2 J(0)}{\partial \theta_j \cdot \partial \theta_k} + 3 \frac{\partial^2 J(\omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \quad (15.39)$$

Spectral density terms of the σ_{NOE} dipolar component

For the dipolar component of the σ_{NOE} equation (7.3c) on page 86 the spectral density terms are

$$J_d^{\sigma_{\text{NOE}}} = 6J(\omega_H + \omega_X) - J(\omega_H - \omega_X). \quad (15.40)$$

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

$$J_d^{\sigma_{\text{NOE}}'} \equiv \frac{\partial J_d^{\sigma_{\text{NOE}}}}{\partial \theta_j} = 6 \frac{\partial J(\omega_H + \omega_X)}{\partial \theta_j} - \frac{\partial J(\omega_H - \omega_X)}{\partial \theta_j}. \quad (15.41)$$

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

$$J_d^{\sigma_{\text{NOE}}''} \equiv \frac{\partial^2 J_d^{\sigma_{\text{NOE}}}}{\partial \theta_j \cdot \partial \theta_k} = 6 \frac{\partial^2 J(\omega_H + \omega_X)}{\partial \theta_j \cdot \partial \theta_k} - \frac{\partial^2 J(\omega_H - \omega_X)}{\partial \theta_j \cdot \partial \theta_k}. \quad (15.42)$$

15.7.2 $R'_i(\theta)$ values

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

$$R_1(\theta) = dJ_d^{R_1} + cJ_c^{R_1}, \quad (15.43a)$$

$$R_2(\theta) = \frac{d}{2}J_d^{R_2} + \frac{c}{6}J_c^{R_2}, \quad (15.43b)$$

$$\sigma_{\text{NOE}}(\theta) = dJ_d^{\sigma_{\text{NOE}}}. \quad (15.43c)$$

15.7.3 $R'_i(\theta)$ 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

$$\frac{\partial R_1(\theta)}{\partial \theta_j} = dJ_d^{R_1'} + cJ_c^{R_1'}, \quad (15.44a)$$

$$\frac{\partial R_2(\theta)}{\partial \theta_j} = \frac{d}{2}J_d^{R_2'} + \frac{c}{6}J_c^{R_2'}, \quad (15.44b)$$

$$\frac{\partial \sigma_{\text{NOE}}(\theta)}{\partial \theta_j} = dJ_d^{\sigma_{\text{NOE}}'}. \quad (15.44c)$$

ρ_{ex} partial derivative

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

$$\frac{\partial R_1(\theta)}{\partial \rho_{ex}} = 0, \quad (15.45a)$$

$$\frac{\partial R_2(\theta)}{\partial \rho_{ex}} = (2\pi\omega_H)^2, \quad (15.45b)$$

$$\frac{\partial \sigma_{\text{NOE}}(\theta)}{\partial \rho_{ex}} = 0. \quad (15.45c)$$

$\Delta\sigma$ partial derivative

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

$$\frac{\partial R_1(\theta)}{\partial \Delta\sigma} = c' J_c^{R_1}, \quad (15.46a)$$

$$\frac{\partial R_2(\theta)}{\partial \Delta\sigma} = \frac{c'}{6} J_c^{R_2}, \quad (15.46b)$$

$$\frac{\partial \sigma_{NOE}(\theta)}{\partial \Delta\sigma} = 0. \quad (15.46c)$$

 r partial derivative

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

$$\frac{\partial R_1(\theta)}{\partial r} = d' J_d^{R_1}, \quad (15.47a)$$

$$\frac{\partial R_2(\theta)}{\partial r} = \frac{d'}{2} J_d^{R_2}, \quad (15.47b)$$

$$\frac{\partial \sigma_{NOE}(\theta)}{\partial r} = d' J_d^{\sigma_{NOE}}. \quad (15.47c)$$

15.7.4 $R'_i(\theta)$ 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 $R'_i(\theta)$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \theta_j \cdot \partial \theta_k} = d J_d^{R_1''} + c J_c^{R_1''}, \quad (15.48a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \theta_j \cdot \partial \theta_k} = \frac{d}{2} J_d^{R_2''} + \frac{c}{6} J_c^{R_2''}, \quad (15.48b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \theta_j \cdot \partial \theta_k} = d J_d^{\sigma_{NOE}''}. \quad (15.48c)$$

$\theta_j - \rho_{ex}$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \theta_j \cdot \partial \rho_{ex}} = 0, \quad (15.49a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \theta_j \cdot \partial \rho_{ex}} = 0, \quad (15.49b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \theta_j \cdot \partial \rho_{ex}} = 0. \quad (15.49c)$$

 $\theta_j - \Delta\sigma$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \theta_j \cdot \partial \Delta\sigma} = c' J_c^{R_1'}, \quad (15.50a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \theta_j \cdot \partial \Delta\sigma} = \frac{c'}{6} J_c^{R_2'}, \quad (15.50b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \theta_j \cdot \partial \Delta\sigma} = 0. \quad (15.50c)$$

 $\theta_j - r$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \theta_j \cdot \partial r} = d' J_d^{R_1'}, \quad (15.51a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \theta_j \cdot \partial r} = \frac{d'}{2} J_d^{R_2'}, \quad (15.51b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \theta_j \cdot \partial r} = d' J_d^{\sigma_{NOE}'}. \quad (15.51c)$$

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

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

$$\frac{\partial^2 R_1(\theta)}{\partial \rho_{ex}^2} = 0, \quad (15.52a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \rho_{ex}^2} = 0, \quad (15.52b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \rho_{ex}^2} = 0. \quad (15.52c)$$

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

$$\frac{\partial^2 R_1(\theta)}{\partial \rho_{ex} \cdot \partial \Delta\sigma} = 0, \quad (15.53a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \rho_{ex} \cdot \partial \Delta\sigma} = 0, \quad (15.53b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \rho_{ex} \cdot \partial \Delta\sigma} = 0. \quad (15.53c)$$

 $\rho_{ex} - r$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \rho_{ex} \cdot \partial r} = 0, \quad (15.54a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \rho_{ex} \cdot \partial r} = 0, \quad (15.54b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \rho_{ex} \cdot \partial r} = 0. \quad (15.54c)$$

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

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

$$\frac{\partial^2 R_1(\theta)}{\partial \Delta\sigma^2} = c'' J_c^{R_1}, \quad (15.55a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \Delta\sigma^2} = \frac{c''}{6} J_c^{R_2}, \quad (15.55b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \Delta\sigma^2} = 0. \quad (15.55c)$$

 $\Delta\sigma - r$ 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

$$\frac{\partial^2 R_1(\theta)}{\partial \Delta\sigma \cdot \partial r} = 0, \quad (15.56a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial \Delta\sigma \cdot \partial r} = 0, \quad (15.56b)$$

$$\frac{\partial^2 \sigma_{NOE}(\theta)}{\partial \Delta\sigma \cdot \partial r} = 0. \quad (15.56c)$$

$r - r$ partial derivative

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

$$\frac{\partial^2 R_1(\theta)}{\partial r^2} = d'' J_d^{R_1}, \quad (15.57a)$$

$$\frac{\partial^2 R_2(\theta)}{\partial r^2} = \frac{d''}{2} J_d^{R_2}, \quad (15.57b)$$

$$\frac{\partial^2 \sigma_{\text{NOE}}(\theta)}{\partial r^2} = d'' J_d^{\sigma_{\text{NOE}}}. \quad (15.57c)$$

15.8 Optimisation equations for the model-free analysis

15.8.1 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 $C_I(\tau)$ from the correlation function of the overall Brownian rotational diffusion $C_O(\tau)$ by the equation

$$C(\tau) = C_O(\tau) \cdot C_I(\tau). \quad (15.58)$$

The overall correlation functions of the diffusion of a sphere, spheroid, and ellipsoid are presented respectively in section [15.9.1](#) on page [350](#), section [15.10.1](#) on page [363](#), and section [15.11.1](#) on page [367](#). 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

$$C_O(\tau) = \frac{1}{5} \sum_{i=-k}^k c_i \cdot e^{-\tau/\tau_i}, \quad (15.59)$$

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

$$C_I(\tau) = S^2 + (1 - S^2)e^{-\tau/\tau_e}, \quad (15.60)$$

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

$$C_I(\tau) = S^2 + (1 - S_f^2)e^{-\tau/\tau_f} + (S_f^2 - S^2)e^{-\tau/\tau_s}, \quad (15.61)$$

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

$$J(\omega) = \frac{2}{5} \sum_{i=-k}^k c_i \cdot \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right). \quad (15.62)$$

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

$$J(\omega) = \frac{2}{5} \sum_{i=-k}^k c_i \cdot \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.63)$$

15.8.2 The original model-free gradient

The model-free gradient of the original spectral density function (15.62) 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 (15.62) with respect to the geometric parameter \mathfrak{G}_j is

$$\frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} + (1 - S^2) \tau_e^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2} \right) + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right) \right). \quad (15.64)$$

\mathfrak{O}_j partial derivative

The partial derivative of (15.62) with respect to the orientational parameter \mathfrak{O}_j is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right). \quad (15.65)$$

S^2 partial derivative

The partial derivative of (15.62) with respect to the order parameter S^2 is

$$\frac{\partial J(\omega)}{\partial S^2} = \frac{2}{5} \sum_{i=-k}^k c_i \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right). \quad (15.66)$$

τ_e partial derivative

The partial derivative of (15.62) with respect to the correlation time τ_e is

$$\frac{\partial J(\omega)}{\partial \tau_e} = \frac{2}{5} (1 - S^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2}. \quad (15.67)$$

15.8.3 The original model-free Hessian

The model-free Hessian of the original spectral density function (15.62) 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 (15.62) with respect to the geometric parameters \mathfrak{G}_j and \mathfrak{G}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(-2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \left(S^2 \omega^2 \tau_i \frac{3 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^3} \right. \right. \\ & \left. \left. + (1 - S^2) \tau_e^2 \frac{(\tau_e + \tau_i)^3 + 3\omega^2 \tau_e^3 \tau_i (\tau_e + \tau_i) - (\omega \tau_e)^4 \tau_i^3}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^3} \right) \right. \\ & \left. + \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_k} + \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_j} + c_i \frac{\partial^2 \tau_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \right) \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \right. \\ & \left. \left. + (1 - S^2) \tau_e^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2} \right) \right. \\ & \left. + \left(\frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right) \right) \right). \quad (15.68) \end{aligned}$$

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

The second partial derivative of (15.62) with respect to the geometric parameter \mathfrak{G}_j and the orientational parameter \mathfrak{O}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \frac{\partial c_i}{\partial \mathfrak{O}_k} \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} + (1 - S^2) \tau_e^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2} \right) \right. \\ & \left. + \frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right) \right). \quad (15.69) \end{aligned}$$

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

The second partial derivative of (15.62) with respect to the geometric parameter \mathfrak{G}_j and the order parameter S^2 is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S^2} = & \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} - \tau_e^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2} \right) \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right) \right). \quad (15.70) \end{aligned}$$

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

The second partial derivative of (15.62) with respect to the geometric parameter \mathfrak{G}_j and the correlation time τ_e is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_e} = \frac{2}{5}(1-S^2) \sum_{i=-k}^k & \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_e \tau_i (\tau_e + \tau_i) \frac{(\tau_e + \tau_i)^2 - 3(\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^3} \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2} \right). \quad (15.71) \end{aligned}$$

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

The second partial derivative of (15.62) with respect to the orientational parameters \mathfrak{O}_j and \mathfrak{O}_k is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1-S^2)(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right). \quad (15.72)$$

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

The second partial derivative of (15.62) with respect to the orientational parameter \mathfrak{O}_j and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S^2} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_e + \tau_i)\tau_e}{(\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2} \right). \quad (15.73)$$

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

The second partial derivative of (15.62) with respect to the orientational parameter \mathfrak{O}_j and the correlation time τ_e is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_e} = \frac{2}{5}(1-S^2) \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2}. \quad (15.74)$$

$S^2 - S^2$ partial derivative

The second partial derivative of (15.62) with respect to the order parameter S^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S^2)^2} = 0. \quad (15.75)$$

$S^2 - \tau_e$ partial derivative

The second partial derivative of (15.62) with respect to the order parameter S^2 and correlation time τ_e is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_e} = -\frac{2}{5} \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_e + \tau_i)^2 - (\omega \tau_e \tau_i)^2}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^2}. \quad (15.76)$$

 $\tau_e - \tau_e$ partial derivative

The second partial derivative of (15.62) with respect to the correlation time τ_e twice is

$$\frac{\partial^2 J(\omega)}{\partial \tau_e^2} = -\frac{4}{5}(1 - S^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_e + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_e (\tau_e + \tau_i) - (\omega \tau_i)^4 \tau_e^3}{((\tau_e + \tau_i)^2 + (\omega \tau_e \tau_i)^2)^3} \quad (15.77)$$

15.8.4 The extended model-free gradient

The model-free gradient of the extended spectral density function (15.63) 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 (15.63) with respect to the geometric parameter \mathfrak{G}_j is

$$\begin{aligned} \frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = & \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & \left. \left. + (S_f^2 - S^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.78)$$

\mathfrak{O}_j partial derivative

The partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.79)$$

S^2 partial derivative

The partial derivative of (15.63) with respect to the order parameter S^2 is

$$\frac{\partial J(\omega)}{\partial S^2} = \frac{2}{5} \sum_{i=-k}^k c_i \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.80)$$

S_f^2 partial derivative

The partial derivative of (15.63) with respect to the order parameter S_f^2 is

$$\frac{\partial J(\omega)}{\partial S_f^2} = -\frac{2}{5} \sum_{i=-k}^k c_i \tau_i \left(\frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.81)$$

τ_f partial derivative

The partial derivative of (15.63) with respect to the correlation time τ_f is

$$\frac{\partial J(\omega)}{\partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.82)$$

 τ_s partial derivative

The partial derivative of (15.63) with respect to the correlation time τ_s is

$$\frac{\partial J(\omega)}{\partial \tau_s} = \frac{2}{5}(S_f^2 - S^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.83)$$

15.8.5 The extended model-free Hessian

The model-free Hessian of the extended spectral density function (15.63) 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 (15.63) with respect to the geometric parameters \mathfrak{G}_j and \mathfrak{G}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(-2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \left(S^2 \omega^2 \tau_i \frac{3 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^3} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_f^3 \tau_i (\tau_f + \tau_i) - (\omega \tau_f)^4 \tau_i^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \\ & + (S_f^2 - S^2) \tau_s^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_s^3 \tau_i (\tau_s + \tau_i) - (\omega \tau_s)^4 \tau_i^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \left. \left. \right) \right. \\ & + \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_k} + \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_j} + c_i \frac{\partial^2 \tau_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \right) \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & + (S_f^2 - S^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \left. \right) \\ & + \left(\frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.84)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the orientational parameter \mathfrak{O}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \frac{\partial c_i}{\partial \mathfrak{O}_k} \left(S^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & + (S_f^2 - S^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \left. \left. \right) \right. \\ & + \frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.85)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S^2} = \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i) \tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \quad (15.86)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_f^2} = -\frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{(\tau_f + \tau_i) \tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} - \frac{(\tau_s + \tau_i) \tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \quad (15.87)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_f} = \frac{2}{5} (1 - S_f^2) \sum_{i=-k}^k \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_f \tau_i (\tau_f + \tau_i) \frac{(\tau_f + \tau_i)^2 - 3(\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \right. \\ \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \right). \quad (15.88)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_s} = \frac{2}{5} (S_f^2 - S^2) \sum_{i=-k}^k \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_s \tau_i (\tau_s + \tau_i) \frac{(\tau_s + \tau_i)^2 - 3(\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \right. \\ \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right). \quad (15.89)$$

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

The second partial derivative of (15.63) with respect to the orientational parameters \mathfrak{O}_j and \mathfrak{O}_k is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(S_f^2 - S^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.90)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the order parameter S^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S^2} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.91)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_f^2} = -\frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.92)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.93)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_s} = \frac{2}{5}(S_f^2 - S^2) \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.94)$$

$S^2 - S^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S^2)^2} = 0. \quad (15.95)$$

 $S^2 - S_f^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameters S^2 and S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial S_f^2} = 0. \quad (15.96)$$

 $S^2 - \tau_f$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_f} = 0. \quad (15.97)$$

 $S^2 - \tau_s$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S^2 \cdot \partial \tau_s} = -\frac{2}{5} \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.98)$$

 $S_f^2 - S_f^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_f^2)^2} = 0. \quad (15.99)$$

 $S_f^2 - \tau_f$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_f} = -\frac{2}{5} \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.100)$$

$S_f^2 - \tau_s$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_s} = \frac{2}{5} \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.101)$$

 $\tau_f - \tau_f$ partial derivative

The second partial derivative of (15.62) with respect to the correlation time τ_f twice is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f^2} = -\frac{4}{5}(1 - S_f^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_f (\tau_f + \tau_i) - (\omega \tau_i)^4 \tau_f^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \quad (15.102)$$

 $\tau_f - \tau_s$ partial derivative

The second partial derivative of (15.62) with respect to the correlation times τ_f and τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f \cdot \partial \tau_s} = 0. \quad (15.103)$$

 $\tau_s - \tau_s$ partial derivative

The second partial derivative of (15.62) with respect to the correlation time τ_s twice is

$$\frac{\partial^2 J(\omega)}{\partial \tau_s^2} = -\frac{4}{5}(S_f^2 - S^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_s (\tau_s + \tau_i) - (\omega \tau_i)^4 \tau_s^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \quad (15.104)$$

15.8.6 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 (15.63) 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 (15.63) with respect to the geometric parameter \mathfrak{G}_j is

$$\begin{aligned} \frac{\partial J(\omega)}{\partial \mathfrak{G}_j} = & \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S_f^2 \cdot S_s^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & \left. \left. + S_f^2 (1 - S_s^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{S_f^2(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.105)$$

\mathfrak{O}_j partial derivative

The partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j is

$$\frac{\partial J(\omega)}{\partial \mathfrak{O}_j} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{S_f^2(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.106)$$

S_f^2 partial derivative

The partial derivative of (15.63) with respect to the order parameter S_f^2 is

$$\frac{\partial J(\omega)}{\partial S_f^2} = \frac{2}{5} \sum_{i=-k}^k c_i \tau_i \left(\frac{S_s^2}{1 + (\omega \tau_i)^2} - \frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.107)$$

S_s^2 partial derivative

The partial derivative of (15.63) with respect to the order parameter S_s^2 is

$$\frac{\partial J(\omega)}{\partial S_s^2} = \frac{2}{5} S_f^2 \sum_{i=-k}^k c_i \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.108)$$

τ_f partial derivative

The partial derivative of (15.63) with respect to the correlation time τ_f is

$$\frac{\partial J(\omega)}{\partial \tau_f} = \frac{2}{5}(1 - S_f^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.109)$$

 τ_s partial derivative

The partial derivative of (15.63) with respect to the correlation time τ_s is

$$\frac{\partial J(\omega)}{\partial \tau_s} = \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.110)$$

15.8.7 The alternative extended model-free Hessian

The model-free Hessian of the extended spectral density function (15.63) 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 (15.63) with respect to the geometric parameters \mathfrak{G}_j and \mathfrak{G}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(-2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \left(S_f^2 \cdot S_s^2 \omega^2 \tau_i \frac{3 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^3} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_f^3 \tau_i (\tau_f + \tau_i) - (\omega \tau_f)^4 \tau_i^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \\ & \left. \left. + S_f^2 (1 - S_s^2) \tau_s^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_s^3 \tau_i (\tau_s + \tau_i) - (\omega \tau_s)^4 \tau_i^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \right) \right. \\ & + \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_k} + \frac{\partial \tau_i}{\partial \mathfrak{G}_k} \cdot \frac{\partial c_i}{\partial \mathfrak{G}_j} + c_i \frac{\partial^2 \tau_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \right) \left(S_f^2 \cdot S_s^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & \left. \left. + S_f^2 (1 - S_s^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ & + \left(\frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{G}_k} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{S_f^2(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.111)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the orientational parameter \mathfrak{O}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} = & \frac{2}{5} \sum_{i=-k}^k \left(\frac{\partial \tau_i}{\partial \mathfrak{G}_j} \frac{\partial c_i}{\partial \mathfrak{O}_k} \left(S_f^2 \cdot S_s^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} \right. \right. \\ & + (1 - S_f^2) \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \\ & \left. \left. + S_f^2 (1 - S_s^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ & + \frac{\partial^2 c_i}{\partial \mathfrak{G}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{S_f^2(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \end{aligned} \quad (15.112)$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the order parameter S_f^2 is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_f^2} = & \frac{2}{5} \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(S_s^2 \frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} - \tau_f^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \right. \right. \\ & + (1 - S_s^2) \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \Big) \\ & + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{S_s^2}{1 + (\omega \tau_i)^2} - \frac{(\tau_f + \tau_i) \tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} \right. \\ & \left. \left. + \frac{(1 - S_s^2)(\tau_s + \tau_i) \tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \quad (15.113) \end{aligned}$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the order parameter S_s^2 is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial S_s^2} = & \frac{2}{5} S_f^2 \sum_{i=-k}^k \left(c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \left(\frac{1 - (\omega \tau_i)^2}{(1 + (\omega \tau_i)^2)^2} - \tau_s^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right) \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i) \tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right) \right). \quad (15.114) \end{aligned}$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the correlation time τ_f is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_f} = & \frac{2}{5} (1 - S_f^2) \sum_{i=-k}^k \left(2 c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_f \tau_i (\tau_f + \tau_i) \frac{(\tau_f + \tau_i)^2 - 3(\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2} \right). \quad (15.115) \end{aligned}$$

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

The second partial derivative of (15.63) with respect to the geometric parameter \mathfrak{G}_j and the correlation time τ_s is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{G}_j \cdot \partial \tau_s} = & \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^k \left(2c_i \frac{\partial \tau_i}{\partial \mathfrak{G}_j} \tau_s \tau_i (\tau_s + \tau_i) \frac{(\tau_s + \tau_i)^2 - 3(\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \right. \\ & \left. + \frac{\partial c_i}{\partial \mathfrak{G}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2} \right). \quad (15.116) \end{aligned}$$

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

The second partial derivative of (15.63) with respect to the orientational parameters \mathfrak{O}_j and \mathfrak{O}_k is

$$\begin{aligned} \frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = & \frac{2}{5} \sum_{i=-k}^k \frac{\partial^2 c_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} \tau_i \left(\frac{S_f^2 \cdot S_s^2}{1 + (\omega \tau_i)^2} + \frac{(1 - S_f^2)(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} \right. \\ & \left. + \frac{S_f^2(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.117) \end{aligned}$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the order parameter S_f^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_f^2} = \frac{2}{5} \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{S_s^2}{1 + (\omega \tau_i)^2} - \frac{(\tau_f + \tau_i)\tau_f}{(\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2} + \frac{(1 - S_s^2)(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.118)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the order parameter S_s^2 is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial S_s^2} = \frac{2}{5} S_f^2 \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i)\tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.119)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_f} = \frac{2}{5} (1 - S_f^2) \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.120)$$

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

The second partial derivative of (15.63) with respect to the orientational parameter \mathfrak{O}_j and the correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \mathfrak{O}_j \cdot \partial \tau_s} = \frac{2}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^k \frac{\partial c_i}{\partial \mathfrak{O}_j} \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.121)$$

 $S_f^2 - S_f^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_f^2)^2} = 0. \quad (15.122)$$

 $S_f^2 - S_s^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameters S_f^2 and S_s^2 is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial S_s^2} = \frac{2}{5} \sum_{i=-k}^k c_i \tau_i \left(\frac{1}{1 + (\omega \tau_i)^2} - \frac{(\tau_s + \tau_i) \tau_s}{(\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2} \right). \quad (15.123)$$

 $S_f^2 - \tau_f$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_f} = -\frac{2}{5} \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^2 - (\omega \tau_f \tau_i)^2}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^2}. \quad (15.124)$$

 $S_f^2 - \tau_s$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_f^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_f^2 \cdot \partial \tau_s} = \frac{2}{5} (1 - S_s^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.125)$$

 $S_s^2 - S_s^2$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_s^2 twice is

$$\frac{\partial^2 J(\omega)}{(\partial S_s^2)^2} = 0. \quad (15.126)$$

$S_s^2 - \tau_f$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_s^2 and correlation time τ_f is

$$\frac{\partial^2 J(\omega)}{\partial S_s^2 \cdot \partial \tau_f} = 0. \quad (15.127)$$

 $S_s^2 - \tau_s$ partial derivative

The second partial derivative of (15.63) with respect to the order parameter S_s^2 and correlation time τ_s is

$$\frac{\partial^2 J(\omega)}{\partial S_s^2 \cdot \partial \tau_s} = -\frac{2}{5} S_f^2 \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^2 - (\omega \tau_s \tau_i)^2}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^2}. \quad (15.128)$$

 $\tau_f - \tau_f$ partial derivative

The second partial derivative of (15.62) with respect to the correlation time τ_f twice is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f^2} = -\frac{4}{5} (1 - S_f^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_f + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_f (\tau_f + \tau_i) - (\omega \tau_i)^4 \tau_f^3}{((\tau_f + \tau_i)^2 + (\omega \tau_f \tau_i)^2)^3} \quad (15.129)$$

 $\tau_f - \tau_s$ partial derivative

The second partial derivative of (15.62) with respect to the correlation times τ_f and τ_s is

$$\frac{\partial^2 J(\omega)}{\partial \tau_f \cdot \partial \tau_s} = 0. \quad (15.130)$$

 $\tau_s - \tau_s$ partial derivative

The second partial derivative of (15.62) with respect to the correlation time τ_s twice is

$$\frac{\partial^2 J(\omega)}{\partial \tau_s^2} = -\frac{4}{5} S_f^2 (1 - S_s^2) \sum_{i=-k}^k c_i \tau_i^2 \frac{(\tau_s + \tau_i)^3 + 3\omega^2 \tau_i^3 \tau_s (\tau_s + \tau_i) - (\omega \tau_i)^4 \tau_s^3}{((\tau_s + \tau_i)^2 + (\omega \tau_s \tau_i)^2)^3} \quad (15.131)$$

15.9 Ellipsoidal diffusion tensor

15.9.1 The diffusion equation of the ellipsoid

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

$$C_O(\tau) = \frac{1}{5} \sum_{i=-2}^2 c_i e^{-\frac{\tau}{\tau_i}}. \quad (15.132)$$

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.

15.9.2 The weights of the ellipsoid

Definitions

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

$$\delta_x = \widehat{XH} \cdot \widehat{\mathfrak{D}}_x, \quad (15.133a)$$

$$\delta_y = \widehat{XH} \cdot \widehat{\mathfrak{D}}_y, \quad (15.133b)$$

$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}}_z. \quad (15.133c)$$

Let the set of geometric parameters be

$$\mathfrak{G} = \{\mathfrak{D}_{iso}, \mathfrak{D}_a, \mathfrak{D}_r\}, \quad (15.134)$$

and the set of orientational parameters be the Euler angles

$$\mathfrak{O} = \{\alpha, \beta, \gamma\}. \quad (15.135)$$

The weights

The five weights c_i in the correlation function of the Brownian rotational diffusion of an ellipsoid (15.132) are

$$c_{-2} = \frac{1}{4}(d - e), \quad (15.136a)$$

$$c_{-1} = 3\delta_y^2\delta_z^2, \quad (15.136b)$$

$$c_0 = 3\delta_x^2\delta_z^2, \quad (15.136c)$$

$$c_1 = 3\delta_x^2\delta_y^2, \quad (15.136d)$$

$$c_2 = \frac{1}{4}(d + e), \quad (15.136e)$$

where

$$d = 3(\delta_x^4 + \delta_y^4 + \delta_z^4) - 1, \quad (15.137)$$

$$e = \frac{1}{\mathfrak{R}} \left[(1 + 3\mathfrak{D}_r)(\delta_x^4 + 2\delta_y^2\delta_z^2) + (1 - 3\mathfrak{D}_r)(\delta_y^4 + 2\delta_x^2\delta_z^2) - 2(\delta_z^4 + 2\delta_x^2\delta_y^2) \right]. \quad (15.138)$$

The factor \mathfrak{R} is defined as

$$\mathfrak{R} = \sqrt{1 + 3\mathfrak{D}_r^2}. \quad (15.139)$$

15.9.3 The weight gradients of the ellipsoid

\mathfrak{O}_i partial derivative

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

$$\frac{\partial c_{-2}}{\partial \mathfrak{O}_i} = 3 \left(\delta_x^3 \frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y^3 \frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_z^3 \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \right) - \frac{\partial e}{\partial \mathfrak{O}_i}, \quad (15.140a)$$

$$\frac{\partial c_{-1}}{\partial \mathfrak{O}_i} = 6\delta_y\delta_z \left(\delta_y \frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \right), \quad (15.140b)$$

$$\frac{\partial c_0}{\partial \mathfrak{O}_i} = 6\delta_x\delta_z \left(\delta_x \frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \right), \quad (15.140c)$$

$$\frac{\partial c_1}{\partial \mathfrak{O}_i} = 6\delta_x\delta_y \left(\delta_x \frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_y \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \right), \quad (15.140d)$$

$$\frac{\partial c_2}{\partial \mathfrak{O}_i} = 3 \left(\delta_x^3 \frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y^3 \frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_z^3 \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \right) + \frac{\partial e}{\partial \mathfrak{O}_i}, \quad (15.140e)$$

where

$$\begin{aligned} \frac{\partial e}{\partial \mathfrak{O}_i} = \frac{1}{\mathfrak{R}} & \left[(1 + 3\mathfrak{D}_r) \left(\delta_x^3 \frac{\partial \delta_x}{\partial \mathfrak{O}_i} + \delta_y \delta_z \left(\delta_y \frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \right) \right) \right. \\ & + (1 - 3\mathfrak{D}_r) \left(\delta_y^3 \frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_x \delta_z \left(\delta_x \frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_z \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \right) \right) \\ & \left. - 2 \left(\delta_z^3 \frac{\partial \delta_z}{\partial \mathfrak{O}_i} + \delta_x \delta_y \left(\delta_x \frac{\partial \delta_y}{\partial \mathfrak{O}_i} + \delta_y \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \right) \right) \right]. \end{aligned} \quad (15.141)$$

τ_m partial derivative

The partial derivatives with respect to the τ_m geometric parameter are

$$\frac{\partial c_{-2}}{\partial \tau_m} = 0, \quad (15.142\text{a})$$

$$\frac{\partial c_{-1}}{\partial \tau_m} = 0, \quad (15.142\text{b})$$

$$\frac{\partial c_0}{\partial \tau_m} = 0, \quad (15.142\text{c})$$

$$\frac{\partial c_1}{\partial \tau_m} = 0, \quad (15.142\text{d})$$

$$\frac{\partial c_2}{\partial \tau_m} = 0. \quad (15.142\text{e})$$

\mathfrak{D}_a partial derivative

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

$$\frac{\partial c_{-2}}{\partial \mathfrak{D}_a} = 0, \quad (15.143\text{a})$$

$$\frac{\partial c_{-1}}{\partial \mathfrak{D}_a} = 0, \quad (15.143\text{b})$$

$$\frac{\partial c_0}{\partial \mathfrak{D}_a} = 0, \quad (15.143\text{c})$$

$$\frac{\partial c_1}{\partial \mathfrak{D}_a} = 0, \quad (15.143\text{d})$$

$$\frac{\partial c_2}{\partial \mathfrak{D}_a} = 0. \quad (15.143\text{e})$$

\mathfrak{D}_r partial derivative

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

$$\frac{\partial c_{-2}}{\partial \mathfrak{D}_r} = -\frac{3}{4} \frac{\partial e}{\partial \mathfrak{D}_r}, \quad (15.144\text{a})$$

$$\frac{\partial c_{-1}}{\partial \mathfrak{D}_r} = 0, \quad (15.144\text{b})$$

$$\frac{\partial c_0}{\partial \mathfrak{D}_r} = 0, \quad (15.144\text{c})$$

$$\frac{\partial c_1}{\partial \mathfrak{D}_r} = 0, \quad (15.144\text{d})$$

$$\frac{\partial c_2}{\partial \mathfrak{D}_r} = \frac{3}{4} \frac{\partial e}{\partial \mathfrak{D}_r}, \quad (15.144\text{e})$$

where

$$\frac{\partial e}{\partial \mathfrak{D}_r} = \frac{1}{\mathfrak{R}^3} \left[(1 - \mathfrak{D}_r) (\delta_x^4 + 2\delta_y^2\delta_z^2) - (1 + \mathfrak{D}_r) (\delta_y^4 + 2\delta_x^2\delta_z^2) + 2\mathfrak{D}_r (\delta_z^4 + 2\delta_x^2\delta_y^2) \right]. \quad (15.145)$$

15.9.4 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

$$\begin{aligned} \frac{\partial^2 c_{-2}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & 3 \left(\delta_x^2 \left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right) \right. \\ & + \delta_y^2 \left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} \right) \\ & \left. + \delta_z^2 \left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} \right) \right) - \frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j}, \end{aligned} \quad (15.146a)$$

$$\begin{aligned} \frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & 6 \delta_y^2 \left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} \right) \\ & + 12 \delta_y \delta_z \left(\frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} \right) \\ & + 6 \delta_z^2 \left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} \right), \end{aligned} \quad (15.146b)$$

$$\begin{aligned} \frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & 6 \delta_x^2 \left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} \right) \\ & + 12 \delta_x \delta_z \left(\frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right) \\ & + 6 \delta_z^2 \left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right), \end{aligned} \quad (15.146c)$$

$$\begin{aligned} \frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & 6 \delta_x^2 \left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} \right) \\ & + 12 \delta_x \delta_y \left(\frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} + \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right) \\ & + 6 \delta_y^2 \left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right), \end{aligned} \quad (15.146d)$$

$$\begin{aligned} \frac{\partial^2 c_2}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & 3 \left(\delta_x^2 \left(\delta_x \frac{\partial^2 \delta_x}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_x}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_x}{\partial \mathfrak{O}_j} \right) \right. \\ & + \delta_y^2 \left(\delta_y \frac{\partial^2 \delta_y}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_y}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_y}{\partial \mathfrak{O}_j} \right) \\ & \left. + \delta_z^2 \left(\delta_z \frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} + 3 \frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} \right) \right) + \frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j}, \end{aligned} \quad (15.146e)$$

where

$$\begin{aligned}
\frac{\partial^2 e}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = & \frac{1}{\mathfrak{R}} \left[(1 + 3\mathfrak{D}_r) \left(\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) \right. \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) \\
& \left. \left. + 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) \right) \right. \\
& + (1 - 3\mathfrak{D}_r) \left(\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) \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) \\
& \left. \left. + 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) \right) \right. \\
& - 2 \left(\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) \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) \\
& \left. \left. + 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) \right) \right]. \quad (15.147)
\end{aligned}$$

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

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

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{O}_i \cdot \partial \tau_m} = 0, \quad (15.148a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \tau_m} = 0, \quad (15.148b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \tau_m} = 0, \quad (15.148c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \tau_m} = 0, \quad (15.148d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_i \cdot \partial \tau_m} = 0. \quad (15.148e)$$

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

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

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_a} = 0, \quad (15.149a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_a} = 0, \quad (15.149b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_a} = 0, \quad (15.149c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_a} = 0, \quad (15.149d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_a} = 0. \quad (15.149e)$$

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

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

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} = -3 \frac{\partial^2 e}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r}, \quad (15.150a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} = 0, \quad (15.150b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} = 0, \quad (15.150c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} = 0, \quad (15.150d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} = 3 \frac{\partial^2 e}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r}, \quad (15.150e)$$

where

$$\begin{aligned} \frac{\partial^2 e}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_r} &= \frac{1}{\mathfrak{R}^3} \left[(1 - \mathfrak{D}_r) \left(\delta_x^3 \frac{\partial \delta_x}{\partial \mathfrak{D}_i} + \delta_y \delta_z \left(\delta_y \frac{\partial \delta_z}{\partial \mathfrak{D}_i} + \delta_z \frac{\partial \delta_y}{\partial \mathfrak{D}_i} \right) \right) \right. \\ &\quad - (1 + \mathfrak{D}_r) \left(\delta_y^3 \frac{\partial \delta_y}{\partial \mathfrak{D}_i} + \delta_x \delta_z \left(\delta_x \frac{\partial \delta_z}{\partial \mathfrak{D}_i} + \delta_z \frac{\partial \delta_x}{\partial \mathfrak{D}_i} \right) \right) \\ &\quad \left. + 2\mathfrak{D}_r \left(\delta_z^3 \frac{\partial \delta_z}{\partial \mathfrak{D}_i} + \delta_x \delta_y \left(\delta_x \frac{\partial \delta_y}{\partial \mathfrak{D}_i} + \delta_y \frac{\partial \delta_x}{\partial \mathfrak{D}_i} \right) \right) \right]. \end{aligned} \quad (15.151)$$

$\tau_m - \tau_m$ partial derivative

The second partial derivatives with respect to the geometric parameter τ_m twice are

$$\frac{\partial^2 c_{-2}}{\partial \tau_m^2} = 0, \quad (15.152\text{a})$$

$$\frac{\partial^2 c_{-1}}{\partial \tau_m^2} = 0, \quad (15.152\text{b})$$

$$\frac{\partial^2 c_0}{\partial \tau_m^2} = 0, \quad (15.152\text{c})$$

$$\frac{\partial^2 c_1}{\partial \tau_m^2} = 0, \quad (15.152\text{d})$$

$$\frac{\partial^2 c_2}{\partial \tau_m^2} = 0. \quad (15.152\text{e})$$

 $\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and \mathfrak{D}_a are

$$\frac{\partial^2 c_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \quad (15.153\text{a})$$

$$\frac{\partial^2 c_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \quad (15.153\text{b})$$

$$\frac{\partial^2 c_0}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \quad (15.153\text{c})$$

$$\frac{\partial^2 c_1}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0, \quad (15.153\text{d})$$

$$\frac{\partial^2 c_2}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 0. \quad (15.153\text{e})$$

 $\tau_m - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and \mathfrak{D}_r are

$$\frac{\partial^2 c_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \quad (15.154\text{a})$$

$$\frac{\partial^2 c_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \quad (15.154\text{b})$$

$$\frac{\partial^2 c_0}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \quad (15.154\text{c})$$

$$\frac{\partial^2 c_1}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \quad (15.154\text{d})$$

$$\frac{\partial^2 c_2}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0. \quad (15.154\text{e})$$

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

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

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_a^2} = 0, \quad (15.155a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_a^2} = 0, \quad (15.155b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_a^2} = 0, \quad (15.155c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_a^2} = 0, \quad (15.155d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_a^2} = 0. \quad (15.155e)$$

 $\mathfrak{D}_a - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters \mathfrak{D}_a and \mathfrak{D}_r are

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \quad (15.156a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \quad (15.156b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \quad (15.156c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \quad (15.156d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0. \quad (15.156e)$$

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

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

$$\frac{\partial^2 c_{-2}}{\partial \mathfrak{D}_r^2} = -\frac{3}{4} \frac{\partial^2 e}{\partial \mathfrak{D}_r^2}, \quad (15.157a)$$

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{D}_r^2} = 0, \quad (15.157b)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{D}_r^2} = 0, \quad (15.157c)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{D}_r^2} = 0, \quad (15.157d)$$

$$\frac{\partial^2 c_2}{\partial \mathfrak{D}_r^2} = \frac{3}{4} \frac{\partial^2 e}{\partial \mathfrak{D}_r^2}, \quad (15.157e)$$

where

$$\begin{aligned} \frac{\partial^2 e}{\partial \mathfrak{D}_r^2} = \frac{1}{\mathfrak{R}^5} & \left[(6\mathfrak{D}_r^2 - 9\mathfrak{D}_r - 1) (\delta_x^4 + 2\delta_y^2\delta_z^2) \right. \\ & + (6\mathfrak{D}_r^2 + 9\mathfrak{D}_r - 1) (\delta_y^4 + 2\delta_x^2\delta_z^2) \\ & \left. - 2(6\mathfrak{D}_r^2 - 1) (\delta_z^4 + 2\delta_x^2\delta_y^2) \right]. \end{aligned} \quad (15.158)$$

15.9.5 The correlation times of the ellipsoid

The five correlation times τ_i in the correlation function of the Brownian rotational diffusion of an ellipsoid (15.132) on page 350 are

$$\tau_{-2} = (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-1}, \quad (15.159a)$$

$$\tau_{-1} = (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-1}, \quad (15.159b)$$

$$\tau_0 = (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-1}, \quad (15.159c)$$

$$\tau_1 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-1}, \quad (15.159d)$$

$$\tau_2 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-1}, \quad (15.159e)$$

where \mathfrak{R} is defined in Equation (15.139) on page 351.

15.9.6 The correlation time gradients of the ellipsoid

τ_m partial derivative

The partial derivatives with respect to the geometric parameter τ_m are

$$\frac{\partial \tau_{-2}}{\partial \tau_m} = \tau_m^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \quad (15.160a)$$

$$\frac{\partial \tau_{-1}}{\partial \tau_m} = \tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \quad (15.160b)$$

$$\frac{\partial \tau_0}{\partial \tau_m} = \tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \quad (15.160c)$$

$$\frac{\partial \tau_1}{\partial \tau_m} = \tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \quad (15.160d)$$

$$\frac{\partial \tau_2}{\partial \tau_m} = \tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \quad (15.160e)$$

\mathfrak{D}_a partial derivative

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

$$\frac{\partial \tau_{-2}}{\partial \mathfrak{D}_a} = 2\mathfrak{R}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \quad (15.161a)$$

$$\frac{\partial \tau_{-1}}{\partial \mathfrak{D}_a} = (1 + 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \quad (15.161b)$$

$$\frac{\partial \tau_0}{\partial \mathfrak{D}_a} = (1 - 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \quad (15.161c)$$

$$\frac{\partial \tau_1}{\partial \mathfrak{D}_a} = -2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \quad (15.161d)$$

$$\frac{\partial \tau_2}{\partial \mathfrak{D}_a} = -2\mathfrak{R}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \quad (15.161e)$$

\mathfrak{D}_r partial derivative

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

$$\frac{\partial \tau_{-2}}{\partial \mathfrak{D}_r} = 6 \frac{\mathfrak{D}_a \mathfrak{D}_r}{\mathfrak{R}} (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a \mathfrak{R})^{-2}, \quad (15.162a)$$

$$\frac{\partial \tau_{-1}}{\partial \mathfrak{D}_r} = 3\mathfrak{D}_a (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \quad (15.162b)$$

$$\frac{\partial \tau_0}{\partial \mathfrak{D}_r} = -3\mathfrak{D}_a (6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \quad (15.162c)$$

$$\frac{\partial \tau_1}{\partial \mathfrak{D}_r} = 0, \quad (15.162d)$$

$$\frac{\partial \tau_2}{\partial \mathfrak{D}_r} = -6 \frac{\mathfrak{D}_a \mathfrak{D}_r}{\mathfrak{R}} (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a \mathfrak{R})^{-2}. \quad (15.162e)$$

15.9.7 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

$$\frac{\partial^2 \tau_{-2}}{\partial \tau_m^2} = 2\tau_m^{-4}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} - 2\tau_m^{-3}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \quad (15.163a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m^2} = 2\tau_m^{-4}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3} - 2\tau_m^{-3}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \quad (15.163b)$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m^2} = 2\tau_m^{-4}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3} - 2\tau_m^{-3}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \quad (15.163c)$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m^2} = 2\tau_m^{-4}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3} - 2\tau_m^{-3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}, \quad (15.163d)$$

$$\frac{\partial^2 \tau_2}{\partial \tau_m^2} = 2\tau_m^{-4}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3} - 2\tau_m^{-3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \quad (15.163e)$$

$\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and \mathfrak{D}_a are

$$\frac{\partial^2 \tau_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 4\mathfrak{R}\tau_m^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \quad (15.164a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2(1 + 3\mathfrak{D}_r)\tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \quad (15.164b)$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2(1 - 3\mathfrak{D}_r)\tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \quad (15.164c)$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4\tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}, \quad (15.164d)$$

$$\frac{\partial^2 \tau_2}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4\mathfrak{R}\tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \quad (15.164e)$$

$\tau_m - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and \mathfrak{D}_r are

$$\frac{\partial^2 \tau_{-2}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 12\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}\tau_m^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \quad (15.165a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 6\mathfrak{D}_a\tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \quad (15.165b)$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = -6\mathfrak{D}_a\tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \quad (15.165c)$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = 0, \quad (15.165d)$$

$$\frac{\partial^2 \tau_2}{\partial \tau_m \cdot \partial \mathfrak{D}_r} = -12\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}}\tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \quad (15.165e)$$

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

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

$$\frac{\partial^2 \tau_{-2}}{\partial \mathfrak{D}_a^2} = 8\mathfrak{R}^2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3}, \quad (15.166a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \mathfrak{D}_a^2} = 2(1 + 3\mathfrak{D}_r)^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \quad (15.166b)$$

$$\frac{\partial^2 \tau_0}{\partial \mathfrak{D}_a^2} = 2(1 - 3\mathfrak{D}_r)^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \quad (15.166c)$$

$$\frac{\partial^2 \tau_1}{\partial \mathfrak{D}_a^2} = 8(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}, \quad (15.166d)$$

$$\frac{\partial^2 \tau_2}{\partial \mathfrak{D}_a^2} = 8\mathfrak{R}^2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3}. \quad (15.166e)$$

$\mathfrak{D}_a - \mathfrak{D}_r$ partial derivative

The second partial derivatives with respect to the geometric parameters \mathfrak{D}_a and \mathfrak{D}_r are

$$\frac{\partial^2 \tau_{-2}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 24\mathfrak{D}_a\mathfrak{D}_r(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} + 6\frac{\mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \quad (15.167a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 6\mathfrak{D}_a(1 + 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3} + 3(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-2}, \quad (15.167b)$$

$$\frac{\partial^2 \tau_0}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = -6\mathfrak{D}_a(1 - 3\mathfrak{D}_r)(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3} - 3(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-2}, \quad (15.167c)$$

$$\frac{\partial^2 \tau_1}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 0, \quad (15.167d)$$

$$\frac{\partial^2 \tau_2}{\partial \mathfrak{D}_a \cdot \partial \mathfrak{D}_r} = 24\mathfrak{D}_a\mathfrak{D}_r(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-3} - 6\frac{\mathfrak{D}_r}{\mathfrak{R}}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \quad (15.167e)$$

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

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

$$\frac{\partial^2 \tau_{-2}}{\partial \mathfrak{D}_r^2} = 72 \left(\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}} \right)^2 (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} + 6\frac{\mathfrak{D}_a}{\mathfrak{R}^3}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-2}, \quad (15.168a)$$

$$\frac{\partial^2 \tau_{-1}}{\partial \mathfrak{D}_r^2} = 18\mathfrak{D}_a^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 + 3\mathfrak{D}_r))^{-3}, \quad (15.168b)$$

$$\frac{\partial^2 \tau_0}{\partial \mathfrak{D}_r^2} = 18\mathfrak{D}_a^2(6\mathfrak{D}_{iso} - \mathfrak{D}_a(1 - 3\mathfrak{D}_r))^{-3}, \quad (15.168c)$$

$$\frac{\partial^2 \tau_1}{\partial \mathfrak{D}_r^2} = 0, \quad (15.168d)$$

$$\frac{\partial^2 \tau_2}{\partial \mathfrak{D}_r^2} = 72 \left(\frac{\mathfrak{D}_a\mathfrak{D}_r}{\mathfrak{R}} \right)^2 (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a\mathfrak{R})^{-3} - 6\frac{\mathfrak{D}_a}{\mathfrak{R}^3}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a\mathfrak{R})^{-2}. \quad (15.168e)$$

15.10 Spheroidal diffusion tensor

15.10.1 The diffusion equation of the spheroid

The correlation function of the Brownian rotational diffusion of a spheroid is

$$C_O(\tau) = \frac{1}{5} \sum_{i=-1}^1 c_i e^{-\frac{\tau}{\tau_i}}. \quad (15.169)$$

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.

15.10.2 The weights of the spheroid

Definitions

The direction cosine defining the XH bond vector within the spheroidal diffusion frame is

$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}_z}. \quad (15.170)$$

Let the set of geometric parameters be

$$\mathfrak{G} = \{\mathfrak{D}_{iso}, \mathfrak{D}_a\}, \quad (15.171)$$

and the set of orientational parameters be the spherical angles

$$\mathfrak{O} = \{\theta, \phi\}. \quad (15.172)$$

The weights

The three spheroid weights c_i in the correlation function of the Brownian rotational diffusion of a spheroid (15.169) are

$$c_{-1} = \frac{1}{4}(3\delta_z^2 - 1)^2, \quad (15.173a)$$

$$c_0 = 3\delta_z^2(1 - \delta_z^2), \quad (15.173b)$$

$$c_1 = \frac{3}{4}(\delta_z^2 - 1)^2. \quad (15.173c)$$

15.10.3 The weight gradients of the spheroid

\mathfrak{O}_i partial derivative

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

$$\frac{\partial c_{-1}}{\partial \mathfrak{O}_i} = 3\delta_z(3\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}, \quad (15.174a)$$

$$\frac{\partial c_0}{\partial \mathfrak{O}_i} = 6\delta_z(1 - 2\delta_z^2)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}, \quad (15.174b)$$

$$\frac{\partial c_1}{\partial \mathfrak{O}_i} = 3\delta_z(\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i}. \quad (15.174c)$$

15.10.4 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

$$\frac{\partial^2 c_{-1}}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3 \left((9\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(3\delta_z^2 - 1)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} \right), \quad (15.175a)$$

$$\frac{\partial^2 c_0}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 6 \left((1 - 6\delta_z^2)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(1 - 2\delta_z^2)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} \right), \quad (15.175b)$$

$$\frac{\partial^2 c_1}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} = 3 \left((3\delta_z^2 - 1)\frac{\partial \delta_z}{\partial \mathfrak{O}_i} \cdot \frac{\partial \delta_z}{\partial \mathfrak{O}_j} + \delta_z(\delta_z^2 - 1)\frac{\partial^2 \delta_z}{\partial \mathfrak{O}_i \cdot \partial \mathfrak{O}_j} \right). \quad (15.175c)$$

15.10.5 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 (15.169) are

$$\tau_{-1} = (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-1}, \quad (15.176a)$$

$$\tau_0 = (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-1}, \quad (15.176b)$$

$$\tau_1 = (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-1}. \quad (15.176c)$$

15.10.6 The correlation time gradients of the spheroid

τ_m partial derivative

The partial derivatives with respect to the geometric parameter τ_m are

$$\frac{\partial \tau_{-1}}{\partial \tau_m} = \tau_m^{-2} (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \quad (15.177a)$$

$$\frac{\partial \tau_0}{\partial \tau_m} = \tau_m^{-2} (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \quad (15.177b)$$

$$\frac{\partial \tau_1}{\partial \tau_m} = \tau_m^{-2} (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \quad (15.177c)$$

\mathfrak{D}_a partial derivative

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

$$\frac{\partial \tau_{-1}}{\partial \mathfrak{D}_a} = 2(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \quad (15.178a)$$

$$\frac{\partial \tau_0}{\partial \mathfrak{D}_a} = (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \quad (15.178b)$$

$$\frac{\partial \tau_1}{\partial \mathfrak{D}_a} = -2(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \quad (15.178c)$$

15.10.7 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

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m^2} = 2\tau_m^{-4} (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3} - 2\tau_m^{-3} (6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-2}, \quad (15.179a)$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m^2} = 2\tau_m^{-4} (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3} - 2\tau_m^{-3} (6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-2}, \quad (15.179b)$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m^2} = 2\tau_m^{-4} (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3} - 2\tau_m^{-3} (6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-2}. \quad (15.179c)$$

$\tau_m - \mathfrak{D}_a$ partial derivative

The second partial derivatives with respect to the geometric parameters τ_m and \mathfrak{D}_a are

$$\frac{\partial^2 \tau_{-1}}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 4\tau_m^{-2}(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3}, \quad (15.180a)$$

$$\frac{\partial^2 \tau_0}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = 2\tau_m^{-2}(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3}, \quad (15.180b)$$

$$\frac{\partial^2 \tau_1}{\partial \tau_m \cdot \partial \mathfrak{D}_a} = -4\tau_m^{-2}(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}. \quad (15.180c)$$

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

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

$$\frac{\partial^2 \tau_{-1}}{\partial \mathfrak{D}_a^2} = 8(6\mathfrak{D}_{iso} - 2\mathfrak{D}_a)^{-3}, \quad (15.181a)$$

$$\frac{\partial^2 \tau_0}{\partial \mathfrak{D}_a^2} = 2(6\mathfrak{D}_{iso} - \mathfrak{D}_a)^{-3}, \quad (15.181b)$$

$$\frac{\partial^2 \tau_1}{\partial \mathfrak{D}_a^2} = 8(6\mathfrak{D}_{iso} + 2\mathfrak{D}_a)^{-3}. \quad (15.181c)$$

15.11 Spherical diffusion tensor

15.11.1 The diffusion equation of the sphere

The correlation function of the Brownian rotational diffusion of a sphere is

$$C_O(\tau) = \frac{1}{5} e^{-\frac{\tau}{\tau_m}}, \quad (15.182)$$

$$= \frac{1}{5} \sum_{i=0}^0 c_i e^{-\frac{\tau}{\tau_i}}. \quad (15.183)$$

where c_i is the weight of the single exponential term and τ_i is the correlation time of the single exponential term.

15.11.2 The weight of the sphere

Definitions

The entire diffusion parameter set consists of a single geometric parameter and is

$$\mathfrak{D} = \{\tau_m\}. \quad (15.184)$$

Summation terms

The summation indices of the correlation function of the Brownian rotational diffusion of a sphere (15.169) range from $k = 0$ to $k = 0$ therefore

$$i \in \{0\}. \quad (15.185)$$

The weights

The single weight c_i in the correlation function of the Brownian rotational diffusion of a sphere (15.169) is

$$c_0 = 1. \quad (15.186)$$

15.11.3 The weight gradient of the sphere

τ_m partial derivative

The partial derivative with respect to the geometric parameter τ_m is

$$\frac{\partial c_0}{\partial \tau_m} = 0. \quad (15.187)$$

15.11.4 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

$$\frac{\partial^2 c_0}{\partial \tau_m^2} = 0. \quad (15.188)$$

15.11.5 The correlation time of the sphere

The single correlation time τ_i of the correlation function of the Brownian rotational diffusion of a sphere (15.169) is

$$\tau_0 = \tau_m. \quad (15.189)$$

15.11.6 The correlation time gradient of the sphere

τ_m partial derivative

The partial derivative with respect to the geometric parameter τ_m is

$$\frac{\partial \tau_0}{\partial \tau_m} = 1. \quad (15.190)$$

15.11.7 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

$$\frac{\partial^2 \tau_0}{\partial \tau_m^2} = 0. \quad (15.191)$$

15.12 Ellipsoidal dot product derivatives

15.12.1 The dot product of the ellipsoid

The dot product is defined as

$$\delta_i = \widehat{XH} \cdot \widehat{\mathfrak{D}}_i, \quad (15.192)$$

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

$$\widehat{\mathfrak{D}}_x = \begin{pmatrix} -\sin \alpha \sin \gamma + \cos \alpha \cos \beta \cos \gamma \\ -\sin \alpha \cos \gamma - \cos \alpha \cos \beta \sin \gamma \\ \cos \alpha \sin \beta \end{pmatrix}, \quad (15.193a)$$

$$\widehat{\mathfrak{D}}_y = \begin{pmatrix} \cos \alpha \sin \gamma + \sin \alpha \cos \beta \cos \gamma \\ \cos \alpha \cos \gamma - \sin \alpha \cos \beta \sin \gamma \\ \sin \alpha \sin \beta \end{pmatrix}, \quad (15.193b)$$

$$\widehat{\mathfrak{D}}_z = \begin{pmatrix} -\sin \beta \cos \gamma \\ \sin \beta \sin \gamma \\ \cos \beta \end{pmatrix}. \quad (15.193c)$$

15.12.2 The dot product gradient of the ellipsoid

The partial derivative of the dot product δ_i with respect to the orientational parameter \mathfrak{D}_j is

$$\frac{\partial \delta_i}{\partial \mathfrak{D}_j} = \frac{\partial}{\partial \mathfrak{D}_j} (\widehat{XH} \cdot \widehat{\mathfrak{D}}_i) = \widehat{XH} \frac{\partial \widehat{\mathfrak{D}}_i}{\partial \mathfrak{D}_j} + \frac{\partial \widehat{XH}}{\partial \mathfrak{D}_j} \widehat{\mathfrak{D}}_i. \quad (15.194)$$

Because \widehat{XH} is constant and not dependent on the Euler angles its derivative is zero. Therefore

$$\frac{\partial \delta_i}{\partial \mathfrak{D}_j} = \widehat{XH} \frac{\partial \widehat{\mathfrak{D}}_i}{\partial \mathfrak{D}_j}. \quad (15.195)$$

The \mathfrak{D}_x gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}}_x$ with respect to the Euler angles are

$$\frac{\partial \widehat{\mathfrak{D}}_x}{\partial \alpha} = \begin{pmatrix} -\cos \alpha \sin \gamma - \sin \alpha \cos \beta \cos \gamma \\ -\cos \alpha \cos \gamma + \sin \alpha \cos \beta \sin \gamma \\ -\sin \alpha \sin \beta \end{pmatrix}, \quad (15.196a)$$

$$\frac{\partial \widehat{\mathfrak{D}}_x}{\partial \beta} = \begin{pmatrix} -\cos \alpha \sin \beta \cos \gamma \\ \cos \alpha \sin \beta \sin \gamma \\ \cos \alpha \cos \beta \end{pmatrix}, \quad (15.196b)$$

$$\frac{\partial \widehat{\mathfrak{D}}_x}{\partial \gamma} = \begin{pmatrix} -\sin \alpha \cos \gamma - \cos \alpha \cos \beta \sin \gamma \\ \sin \alpha \sin \gamma - \cos \alpha \cos \beta \cos \gamma \\ 0 \end{pmatrix}. \quad (15.196c)$$

The \mathfrak{D}_y gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}}_y$ with respect to the Euler angles are

$$\frac{\partial \widehat{\mathfrak{D}}_y}{\partial \alpha} = \begin{pmatrix} -\sin \alpha \sin \gamma + \cos \alpha \cos \beta \cos \gamma \\ -\sin \alpha \cos \gamma - \cos \alpha \cos \beta \sin \gamma \\ \cos \alpha \sin \beta \end{pmatrix}, \quad (15.197a)$$

$$\frac{\partial \widehat{\mathfrak{D}}_y}{\partial \beta} = \begin{pmatrix} -\sin \alpha \sin \beta \cos \gamma \\ \sin \alpha \sin \beta \sin \gamma \\ \sin \alpha \cos \beta \end{pmatrix}, \quad (15.197b)$$

$$\frac{\partial \widehat{\mathfrak{D}}_y}{\partial \gamma} = \begin{pmatrix} \cos \alpha \cos \gamma - \sin \alpha \cos \beta \sin \gamma \\ -\cos \alpha \sin \gamma - \sin \alpha \cos \beta \cos \gamma \\ 0 \end{pmatrix}. \quad (15.197c)$$

The \mathfrak{D}_z gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}}_z$ with respect to the Euler angles are

$$\frac{\partial \widehat{\mathfrak{D}}_z}{\partial \alpha} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad (15.198a)$$

$$\frac{\partial \widehat{\mathfrak{D}}_z}{\partial \beta} = \begin{pmatrix} -\cos \beta \cos \gamma \\ \cos \beta \sin \gamma \\ -\sin \beta \end{pmatrix}, \quad (15.198b)$$

$$\frac{\partial \widehat{\mathfrak{D}}_z}{\partial \gamma} = \begin{pmatrix} \sin \beta \sin \gamma \\ \sin \beta \cos \gamma \\ 0 \end{pmatrix}. \quad (15.198c)$$

15.12.3 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

$$\frac{\partial^2 \delta_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} = \frac{\partial^2}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k} (\widehat{XH} \cdot \widehat{\mathfrak{D}}_i) = \widehat{XH} \frac{\partial^2 \widehat{\mathfrak{D}}_i}{\partial \mathfrak{O}_j \cdot \partial \mathfrak{O}_k}. \quad (15.199)$$

The \mathfrak{D}_x Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}}_x$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \alpha^2} = \begin{pmatrix} \sin \alpha \sin \gamma - \cos \alpha \cos \beta \cos \gamma \\ \sin \alpha \cos \gamma + \cos \alpha \cos \beta \sin \gamma \\ -\cos \alpha \sin \beta \end{pmatrix}, \quad (15.200a)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} \sin \alpha \sin \beta \cos \gamma \\ -\sin \alpha \sin \beta \sin \gamma \\ -\sin \alpha \cos \beta \end{pmatrix}, \quad (15.200b)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} -\cos \alpha \cos \gamma + \sin \alpha \cos \beta \sin \gamma \\ \cos \alpha \sin \gamma + \sin \alpha \cos \beta \cos \gamma \\ 0 \end{pmatrix}, \quad (15.200c)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \beta^2} = \begin{pmatrix} -\cos \alpha \cos \beta \cos \gamma \\ \cos \alpha \cos \beta \sin \gamma \\ -\cos \alpha \sin \beta \end{pmatrix}, \quad (15.200d)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \cos \alpha \sin \beta \sin \gamma \\ \cos \alpha \sin \beta \cos \gamma \\ 0 \end{pmatrix}, \quad (15.200e)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_x}{\partial \gamma^2} = \begin{pmatrix} \sin \alpha \sin \gamma - \cos \alpha \cos \beta \cos \gamma \\ \sin \alpha \cos \gamma + \cos \alpha \cos \beta \sin \gamma \\ 0 \end{pmatrix}. \quad (15.200f)$$

The \mathfrak{D}_y Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}}_y$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \alpha^2} = \begin{pmatrix} -\cos \alpha \sin \gamma - \sin \alpha \cos \beta \cos \gamma \\ -\cos \alpha \cos \gamma + \sin \alpha \cos \beta \sin \gamma \\ -\sin \alpha \sin \beta \end{pmatrix}, \quad (15.201a)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} -\cos \alpha \sin \beta \cos \gamma \\ \cos \alpha \sin \beta \sin \gamma \\ \cos \alpha \cos \beta \end{pmatrix}, \quad (15.201b)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} -\sin \alpha \cos \gamma - \cos \alpha \cos \beta \sin \gamma \\ \sin \alpha \sin \gamma - \cos \alpha \cos \beta \cos \gamma \\ 0 \end{pmatrix}, \quad (15.201c)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \beta^2} = \begin{pmatrix} -\sin \alpha \cos \beta \cos \gamma \\ \sin \alpha \cos \beta \sin \gamma \\ -\sin \alpha \sin \beta \end{pmatrix}, \quad (15.201d)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \sin \alpha \sin \beta \sin \gamma \\ \sin \alpha \sin \beta \cos \gamma \\ 0 \end{pmatrix}, \quad (15.201e)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_y}{\partial \gamma^2} = \begin{pmatrix} -\cos \alpha \sin \gamma - \sin \alpha \cos \beta \cos \gamma \\ -\cos \alpha \cos \gamma + \sin \alpha \cos \beta \sin \gamma \\ 0 \end{pmatrix}. \quad (15.201f)$$

The \mathfrak{D}_z Hessian

The second partial derivatives of the unit vector $\widehat{\mathfrak{D}}_z$ with respect to the Euler angles are

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \alpha^2} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad (15.202a)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \alpha \cdot \partial \beta} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad (15.202b)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \alpha \cdot \partial \gamma} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad (15.202c)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \beta^2} = \begin{pmatrix} \sin \beta \cos \gamma \\ -\sin \beta \sin \gamma \\ -\cos \beta \end{pmatrix}, \quad (15.202d)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \beta \cdot \partial \gamma} = \begin{pmatrix} \cos \beta \sin \gamma \\ \cos \beta \cos \gamma \\ 0 \end{pmatrix}, \quad (15.202e)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_z}{\partial \gamma^2} = \begin{pmatrix} \sin \beta \cos \gamma \\ -\sin \beta \sin \gamma \\ 0 \end{pmatrix}. \quad (15.202f)$$

15.13 Spheroidal dot product derivatives

15.13.1 The dot product of the spheroid

The single dot product of the spheroid is defined as

$$\delta_z = \widehat{XH} \cdot \widehat{\mathfrak{D}}_{\parallel}, \quad (15.203)$$

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

$$\widehat{\mathfrak{D}}_{\parallel} = \begin{pmatrix} \sin \theta \cos \phi \\ \sin \theta \sin \phi \\ \cos \theta \end{pmatrix}. \quad (15.204)$$

15.13.2 The dot product gradient of the spheroid

The partial derivative of the dot product with respect to the orientational parameter \mathfrak{D}_i is

$$\frac{\partial \delta_z}{\partial \mathfrak{D}_i} = \frac{\partial}{\partial \mathfrak{D}_i} (\widehat{XH} \cdot \widehat{\mathfrak{D}}_{\parallel}) = \widehat{XH} \frac{\partial \widehat{\mathfrak{D}}_{\parallel}}{\partial \mathfrak{D}_i} + \frac{\partial \widehat{XH}}{\partial \mathfrak{D}_i} \widehat{\mathfrak{D}}_{\parallel}. \quad (15.205)$$

Because the XH bond vector is constant and not dependent on the spherical angles its derivative is zero. Therefore

$$\frac{\partial \delta_z}{\partial \mathfrak{D}_i} = \widehat{XH} \frac{\partial \widehat{\mathfrak{D}}_{\parallel}}{\partial \mathfrak{D}_i}. \quad (15.206)$$

The \mathfrak{D}_{\parallel} gradient

The partial derivatives of the unit vector $\widehat{\mathfrak{D}}_{\parallel}$ with respect to the spherical angles are

$$\frac{\partial \widehat{\mathfrak{D}}_{\parallel}}{\partial \theta} = \begin{pmatrix} \cos \theta \cos \phi \\ \cos \theta \sin \phi \\ -\sin \theta \end{pmatrix}, \quad (15.207a)$$

$$\frac{\partial \widehat{\mathfrak{D}}_{\parallel}}{\partial \phi} = \begin{pmatrix} -\sin \theta \sin \phi \\ \sin \theta \cos \phi \\ 0 \end{pmatrix}. \quad (15.207b)$$

15.13.3 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{D}_i and \mathfrak{D}_j is

$$\frac{\partial^2 \delta_z}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_j} = \frac{\partial^2}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_j} (\widehat{XH} \cdot \widehat{\mathfrak{D}}_{\parallel}) = \widehat{XH} \frac{\partial^2 \widehat{\mathfrak{D}}_{\parallel}}{\partial \mathfrak{D}_i \cdot \partial \mathfrak{D}_j}. \quad (15.208)$$

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

$$\frac{\partial^2 \widehat{\mathfrak{D}}_{\parallel}}{\partial \theta^2} = \begin{pmatrix} -\sin \theta \cos \phi \\ -\sin \theta \sin \phi \\ -\cos \theta \end{pmatrix}, \quad (15.209a)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_{\parallel}}{\partial \theta \cdot \partial \phi} = \begin{pmatrix} -\cos \theta \sin \phi \\ \cos \theta \cos \phi \\ 0 \end{pmatrix}, \quad (15.209b)$$

$$\frac{\partial^2 \widehat{\mathfrak{D}}_{\parallel}}{\partial \phi^2} = \begin{pmatrix} -\sin \theta \cos \phi \\ -\sin \theta \sin \phi \\ 0 \end{pmatrix}. \quad (15.209c)$$

Chapter 16

The frame order models

16.1 The current frame order models

In this advanced topic chapter, the equations for the various frame order models will be derived and validated using simulations. The models include:

1. Rigid
2. Rotor
3. Free rotor
4. Isotropic cone
5. Isotropic cone, torsionless
6. Isotropic cone, free rotor
7. Pseudo-ellipse
8. Pseudo-ellipse, torsionless
9. Pseudo-ellipse, free rotor
10. Double rotor

For a basic introduction to the frame order concept and the modelling of the tilt and torsion components, see Section 12.3.

16.2 Simulation of the frame order models

To validate the derived frame order matrix equations for the models, the real frame order matrix values for a given parameter set can be simulated. The following script was used for simulating the isotropic cone and pseudo-ellipse frame order matrices. As the other models are parametric restrictions of these two models, the simpler models were not simulated.

The script allows for both an ‘in_frame’ and ‘out_of_frame’ or ‘axis2_1_3’ mode to observe the components both within the motional eigenframe and in a rotated frame. The value of VAR can be set to ‘ISO’ for selecting the isotropic cone θ parameter to vary, or ‘X’, ‘Y’, or ‘Z’ to vary the pseudo-ellipse θ_x , θ_y , and σ_{\max} parameters respectively.

```

1 ######
2 #
3 # Copyright (C) 2014-2015 Edward d'Auvergne
4 #
5 # This file is part of the program relax (http://www.nmr-relax.com).
6 #
7 # This program is free software: you can redistribute it and/or modify
8 # it under the terms of the GNU General Public License as published by
9 # the Free Software Foundation, either version 3 of the License, or
10 # (at your option) any later version.
11 #
12 # This program is distributed in the hope that it will be useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15 # GNU General Public License for more details.
16 #
17 # You should have received a copy of the GNU General Public License
18 # along with this program. If not, see <http://www.gnu.org/licenses/>.
19 #
20 #####
21 #
22 # relax script.
23 #
24 # Python module imports.
25 from math import cos, pi, sin, sqrt
26 from numpy import array, cross, dot, eye, float64, transpose, zeros
27 from numpy.linalg import norm
28 from random import uniform
29 from string import lower
30 import sys
31 #
32 # relax module imports.
33 from lib.errors import RelaxError
34 from lib.geometry.angles import wrap_angles
35 from lib.geometry.rotations import axis_angle_to_R, R_random_hypersphere, R_to_euler_zxy,
   tilt_torsion_to_R
36 from lib.linear_algebra.kronecker_product import kron_prod
37 from lib.text.progress import progress_meter
38 #
39 #
40 # Variables.
41 #MODEL = 'rotor'
42 #MODEL = 'free_rotor'
43 #MODEL = 'iso_cone'
44 #MODEL = 'iso_cone_torsionless'
45 #MODEL = 'iso_cone_free_rotor'
46 #MODEL = 'pseudo-ellipse'
47 #MODEL = 'pseudo-ellipse_torsionless'
48 #MODEL = 'pseudo-ellipse_free_rotor'
49 MODEL = 'double_rotor'
50 #MODEL_TEXT = 'Rotor frame order model'
51 #MODEL_TEXT = 'Free rotor frame order model'
52 #MODEL_TEXT = 'Isotropic cone frame order model'
53 #MODEL_TEXT = 'Torsionless isotropic cone frame order model'
54 #MODEL_TEXT = 'Free rotor isotropic cone frame order model'
```

```

55 #MODEL_TEXT = 'Pseudo-ellipse frame order model'
56 #MODEL_TEXT = 'Torsionless pseudo-ellipse frame order model'
57 #MODEL_TEXT = 'Free rotor pseudo-ellipse frame order model'
58 MODEL_TEXT = 'Double rotor frame order model'
59 SAMPLE_SIZE = 1000000
60 TAG = 'in_frame'
61 #TAG = 'out_of_frame'
62 #TAG = 'axis2_1_3'
63
64 # Angular restrictions.
65 THETA_X = pi / 4
66 THETA_Y = 3 * pi / 8
67 THETA_Z = pi / 6
68 INC = 18
69 VAR = 'X'
70
71 # The frame order eigenframe - I.
72 if TAG == 'in_frame':
73     EIG_FRAME = eye(3)
74
75 # The frame order eigenframe - rotated.
76 if TAG == 'out_of_frame':
77     EIG_FRAME = array([[ 2, -1,  2],
78                         [ 2,  2, -1],
79                         [-1,  2,  2]], float64) / 3.0
80
81 # The frame order eigenframe (and tag) - original isotropic cone axis [2, 1, 3].
82 elif TAG == 'axis2_1_3':
83     # Generate 3 orthogonal vectors.
84     vect_z = array([2, 1, 3], float64)
85     vect_x = cross(vect_z, array([1, 1, 1], float64))
86     vect_y = cross(vect_z, vect_x)
87
88     # Normalise.
89     vect_x = vect_x / norm(vect_x)
90     vect_y = vect_y / norm(vect_y)
91     vect_z = vect_z / norm(vect_z)
92
93     # Build the frame.
94     EIG_FRAME = zeros((3, 3), float64)
95     EIG_FRAME[:, 0] = vect_x
96     EIG_FRAME[:, 1] = vect_y
97     EIG_FRAME[:, 2] = vect_z
98
99
100 class Frame_order:
101     def __init__(self):
102         """Calculate the frame order at infinity.
103
104             This is when the starting positions are random.
105
106         """
107
108         # The file name.
109         file_name = '%s_%s_theta_%s_ens%s.agr' % (MODEL, TAG, lower(VAR), SAMPLE_SIZE)
110
111         # Set the initial storage structures.
112         self.init_storage()
113
114         # Init.
115         index = 0

```

```

116     self.torsion_check = True
117
118     # Pre-transpose the eigenframe for speed.
119     self.eig_frame_T = transpose(EIG_FRAME)
120
121     # Generate the angle data structures.
122     self.angles = []
123     self.angles_deg = []
124     for i in range(INC):
125         # The angle of one increment.
126         inc_angle = pi / INC
127
128         # The angle of the increment.
129         self.angles.append(inc_angle * (i+1))
130
131         # In degrees for the graphs.
132         self.angles_deg.append(self.angles[-1] / (2.0*pi) * 360.0)
133
134     # Alias the bound checking methods.
135     if MODEL == 'rotor':
136         self.inside = self.inside_rotor
137         self.rotation = self.rotation_z_axis
138     elif MODEL == 'free_rotor':
139         self.inside = self.inside_free_rotor
140         self.rotation = self.rotation_z_axis
141     elif MODEL == 'iso_cone':
142         self.inside = self.inside_iso_cone
143         self.rotation = self.rotation_hypersphere
144     elif MODEL == 'iso_cone_torsionless':
145         self.inside = self.inside_iso_cone
146         self.rotation = self.rotation_hypersphere_torsionless
147     elif MODEL == 'iso_cone_free_rotor':
148         self.inside = self.inside_iso_cone
149         self.rotation = self.rotation_hypersphere
150         self.torsion_check = False
151     elif MODEL == 'pseudo-ellipse':
152         self.inside = self.inside_pseudo_ellipse
153         self.rotation = self.rotation_hypersphere
154     elif MODEL == 'pseudo-ellipse_torsionless':
155         self.inside = self.inside_pseudo_ellipse
156         self.rotation = self.rotation_hypersphere_torsionless
157     elif MODEL == 'pseudo-ellipse_free_rotor':
158         self.inside = self.inside_pseudo_ellipse
159         self.rotation = self.rotation_hypersphere
160         self.torsion_check = False
161     elif MODEL == 'double_rotor':
162         self.inside = self.inside_double_rotor
163         self.rotation = self.rotation_double_xy_axes
164     else:
165         raise RelaxError("Unknown model '%s'." % MODEL)
166
167     # Loop over random starting positions.
168     while True:
169         # Printout.
170         progress_meter(index, a=1000, b=100000)
171
172         # Generate the random rotation.
173         theta, phi, sigma = self.rotation()
174
175         # Pre-calculate the R Kronecker outer product for speed.
176         Rx2 = kron_prod(self.rot, self.rot)

```

```

177
178     # Loop over the angle incs.
179     for i in range(INC):
180         # The new limits.
181         max_theta_x, max_theta_y, max_theta_z = self.limits(i)
182
183         # Inside the cone.
184         if not self.full[i] and self.inside(i=i, theta=theta, phi=phi, sigma=sigma
185 , max_theta_x=max_theta_x, max_theta_y=max_theta_y, max_theta_z=max_theta_z):
186
187             # Sum of rotations and cross products.
188             self.first_frame_order[i] += self.rot
189             self.second_frame_order[i] += Rx2
190
191             # Increment the counter.
192             self.count[i] += 1
193
194             # Full.
195             if self.count[i] == SAMPLE_SIZE:
196                 sys.stdout.write("\b"*100 + "The angle restriction of %s deg is
complete.\n" % self.angles_deg[i])
197                 self.full[i] = 1
198
199             # Increment the global index.
200             index += 1
201
202             # Break out.
203             if sum(self.full) == INC:
204                 break
205
206             # Average.
207             self.first_frame_order = self.first_frame_order / float(SAMPLE_SIZE)
208             self.second_frame_order = self.second_frame_order / float(SAMPLE_SIZE)
209
210             # Write the data.
211             self.write_data(file_name=file_name)
212
213             # Final printout.
214             sys.stdout.write("Random rotations required: %i\n\n" % index)
215
216     def init_storage(self):
217         """Initialise the storage structures."""
218
219         # Create the average rotation matrix (first order).
220         self.first_frame_order = zeros((INC, 3, 3), float64)
221
222         # Create the frame order matrix (each element is ensemble averaged and corresponds
223         # to a different time step).
224         self.second_frame_order = zeros((INC, 9, 9), float64)
225
226         # Init the rotation matrix.
227         self.rot = zeros((3, 3), float64)
228         self.rot2 = zeros((3, 3), float64)
229
230         # Some data arrays.
231         self.full = zeros(INC)
232         self.count = zeros(INC)
233
234         # Axes.
235         self.x_axis = array([1, 0, 0], float64)

```

```

235         self.y_axis = array([0, 1, 0], float64)
236         self.z_axis = array([0, 0, 1], float64)
237
238
239     def inside_double_rotor(self, i=None, theta=None, phi=None, sigma=None, max_theta_x=
240                             None, max_theta_y=None, max_theta_z=None):
241         """Determine if the frame is inside the limits."""
242
243         # Alias the angles.
244         sigma1, sigma2 = theta, phi
245
246         # Check for torsion angle violations.
247         if sigma1 < -max_theta_y or sigma1 > max_theta_y:
248             return False
249         if sigma2 < -max_theta_x or sigma2 > max_theta_x:
250             return False
251
252         # Inside.
253         return True
254
255
256     def inside_free_rotor(self, i=None, theta=None, phi=None, sigma=None, max_theta_x=None
257                           , max_theta_y=None, max_theta_z=None):
258         """Determine if the frame is inside the limits, which for the free rotor is always
259         true."""
260
261         # Inside.
262         return True
263
264
265     def inside_iso_cone(self, i=None, theta=None, phi=None, sigma=None, max_theta_x=None,
266                         max_theta_y=None, max_theta_z=None):
267         """Determine if the frame is inside the limits."""
268
269         # Check for a torsion angle violation.
270         if self.torsion_check and (sigma < -max_theta_z or sigma > max_theta_z):
271             return False
272
273         # Check for a tilt angle violation.
274         if theta > max_theta_x:
275             return False
276
277         # Inside.
278         return True
279
280
281     def inside_pseudo_ellipse(self, i=None, theta=None, phi=None, sigma=None, max_theta_x=
282                             None, max_theta_y=None, max_theta_z=None):
283         """Determine if the frame is inside the limits."""
284
285         # Check for a torsion angle violation.
286         if self.torsion_check and (sigma < -max_theta_z or sigma > max_theta_z):
287             return False
288
289         # Check for a tilt angle violation.
290         max_theta = 1.0 / sqrt(cos(phi)**2 / max_theta_x**2 + sin(phi)**2 / max_theta_y
291                     **2)
292         if theta > max_theta:
293             return False
294
295         # Inside.

```

```

290         return True
291
292
293     def inside_rotor(self, i=None, theta=None, phi=None, sigma=None, max_theta_x=None,
294                      max_theta_y=None, max_theta_z=None):
295         """Determine if the frame is inside the limits."""
296
297         # Check for a torsion angle violation.
298         if sigma < -max_theta_z or sigma > max_theta_z:
299             return False
300
301         # Inside.
302         return True
303
304
305     def limits(self, i):
306         """Determine the angular restrictions for the increment i."""
307
308         # Alias the angle for the increment.
309         theta = self.angles[i]
310
311         # The different angles to vary.
312         if VAR == 'X':
313             return theta, THETA_Y, THETA_Z
314         elif VAR == 'Y':
315             return THETA_X, theta, THETA_Z
316         elif VAR == 'Z':
317             return THETA_X, THETA_Y, theta
318
319
320     def rotation_double_xy_axes(self):
321         """Random double rotation around the x- and y-axes and return of torsion-tilt
322         angles"""
323
324         # First a random angle between -pi and pi for the y-axis.
325         sigma1 = uniform(-pi, pi)
326         axis_angle_to_R(self.y_axis, sigma1, self.rot)
327
328         # Second a random angle between -pi and pi for the x-axis.
329         sigma2 = uniform(-pi, pi)
330         axis_angle_to_R(self.x_axis, sigma2, self.rot2)
331
332         # Construct the frame.
333         frame = dot(self.rot2, self.rot)
334
335         # Rotate the frame.
336         self.rot = dot(EIG_FRAME, dot(frame, self.eig_frame_T))
337
338         # Return the two torsion angles, and zero.
339         return sigma1, sigma2, 0.0
340
341
342     def rotation_hypersphere(self):
343         """Random rotation using 4D hypersphere point picking and return of torsion-tilt
344         angles."""
345
346         # Generate a random rotation.
347         R_random_hypersphere(self.rot)
348
349         # Rotate the frame.
350         frame = dot(self.eig_frame_T, dot(self.rot, EIG_FRAME))

```

```

348
349     # Decompose the frame into the zyz Euler angles.
350     alpha, beta, gamma = R_to_euler_zyz(frame)
351
352     # Convert to tilt and torsion angles (properly wrapped) and return them.
353     theta = beta
354     phi = wrap_angles(gamma, -pi, pi)
355     sigma = wrap_angles(alpha + gamma, -pi, pi)
356     return theta, phi, sigma
357
358
359 def rotation_hypersphere_torsionless(self):
360     """Random rotation using 4D hypersphere point picking and return of torsion-tilt
361     angles."""
362
363     # Obtain the random torsion-tilt angles from the random hypersphere method.
364     theta, phi, sigma = self.rotation_hypersphere()
365
366     # Reconstruct a rotation matrix, setting the torsion angle to zero.
367     tilt_torsion_to_R(phi, theta, 0.0, self.rot)
368
369     # Rotate the frame.
370     self.rot = dot(EIG_FRAME, dot(self.rot, self.eig_frame_T))
371
372     # Return the angles.
373     return theta, phi, 0.0
374
375
376 def rotation_z_axis(self):
377     """Random rotation around the z-axis and return of torsion-tilt angles"""
378
379     # Random angle between -pi and pi.
380     angle = uniform(-pi, pi)
381
382     # Generate the rotation matrix.
383     axis_angle_to_R(self.z_axis, angle, self.rot)
384
385     # Decompose the rotation into the zyz Euler angles.
386     alpha, beta, gamma = R_to_euler_zyz(self.rot)
387
388     # Rotate the frame.
389     self.rot = dot(EIG_FRAME, dot(self.rot, self.eig_frame_T))
390
391     # Convert to tilt and torsion angles (properly wrapped) and return them.
392     theta = beta
393     phi = wrap_angles(gamma, -pi, pi)
394     sigma = wrap_angles(alpha + gamma, -pi, pi)
395     return theta, phi, sigma
396
397
398 def write_data(self, file_name=None):
399     """Dump the data to files.
400
401     @keyword file_name:      The end part of the files to create. This will be
402     prepended by either 'Sij' or 'Sijkl'.
403     @type file_name:         str
404     """
405
406     # Open the files.
407     file_1st = open("Sij" + file_name, 'w')
408     file_2nd = open("Sijkl" + file_name, 'w')

```

```

407     files = [file_1st, file_2nd]
408
409     # The headers.
410     for i in range(2):
411         # Alias the file.
412         file = files[i]
413
414         # The titles.
415         file.write("@with g0\n")
416         if i == 0:
417             file.write("@    world 0, -0.2, 180, 1\n")
418         else:
419             file.write("@    world 0, -0.7, 180, 1\n")
420         file.write("@    title \"Simulated frame order matrix elements\"\n")
421         if i == 0:
422             file.write("@    subtitle \"%s, 1\\Sst\\N degree matrix, %i simulations\"\
423             n" % (MODEL_TEXT, SAMPLE_SIZE))
424         else:
425             file.write("@    subtitle \"%s, 2\\Snd\\N degree matrix, %i simulations\"\
426             n" % (MODEL_TEXT, SAMPLE_SIZE))
427
428         # Legend.
429         if i == 0:
430             file.write("@    legend 0.23, 0.55\n")
431         else:
432             file.write("@    legend off\n")
433
434         # Plot data.
435         file.write("@    xaxis bar linewidth 0.5\n")
436         file.write("@    xaxis label \"Cone half-angle \\xq\\f{}\\s%s\\N (deg.)\"\n")
437         # VAR)
438         file.write("@    xaxis label char size 1.000000\n")
439         file.write("@    xaxis tick major 45\n")
440         file.write("@    xaxis tick major linewidth 0.5\n")
441         file.write("@    xaxis tick minor ticks 3\n")
442         file.write("@    xaxis tick minor linewidth 0.5\n")
443         file.write("@    yaxis bar linewidth 0.5\n")
444         if i == 0:
445             file.write("@    yaxis label \"Order parameter \\qS\\sij\"\n")
446         else:
447             file.write("@    yaxis label \"Order parameter \\qS\\sijkl\"\n")
448             file.write("@    yaxis label char size 1.000000\n")
449             file.write("@    yaxis tick major 0.2\n")
450             file.write("@    yaxis tick major linewidth 0.5\n")
451             file.write("@    yaxis tick minor ticks 1\n")
452             file.write("@    yaxis tick minor linewidth 0.5\n")
453
454         # Header for first order matrix.
455         graph_num = 0
456         for i in range(3):
457             for j in range(3):
458                 file_1st.write("@    s%i legend \"\\q<c\\s%s\\N\"\\n" % (graph_num, i+1,
459                             j+1))
460                 file_1st.write("@    s%i linewidth 0.5\\n" % graph_num)
461                 graph_num += 1
462
463         # Header for second order matrix.
464         graph_num = 0
465         for i in range(3):
466             for j in range(3):
467                 for k in range(3):

```

```

464         for l in range(3):
465             file_2nd.write("@    s%i legend \"<\\qc\\$%$s\\N.c\\$%$s\\N>\"\n"
466             " % (graph_num, i+1, j+1, k+1, l+1))
467             file_2nd.write("@    s%i linewidth 0.5\n" % graph_num)
468             graph_num += 1
469
470     # Loop over the first rotation matrix index.
471     graph_num = 0
472     for i in range(3):
473         # Loop over the second rotation matrix index.
474         for j in range(3):
475             # Header.
476             file_1st.write("@target GO.S%i\n" % graph_num)
477             file_1st.write("@type xy\n")
478
479             # Loop over each time point.
480             for k in range(INC):
481                 file_1st.write("%s %s\n" % (self.angles_deg[k], self.first_frame_order
482 [k, i, j]))
483
484                 # Footer.
485                 file_1st.write("&\n")
486
487                 # Inc.
488                 graph_num += 1
489
490             # Loop over the first frame order index.
491             graph_num = 0
492             for i in range(9):
493                 # Loop over the second frame order index.
494                 for j in range(9):
495                     # Header.
496                     file_2nd.write('@target GO.S%i\n' % graph_num)
497                     file_2nd.write('@type xy\n')
498
499                     # Loop over each time point.
500                     for k in range(INC):
501                         file_2nd.write('%s %s\n' % (self.angles_deg[k],
502 self.second_frame_order[k, i, j]))
503
504                         # Footer.
505                         file_2nd.write('&\n')
506
507                         # Inc.
508                         graph_num += 1
509
510             # No autoscaling.
511             file_1st.write("@autoscale onread none\n")
512             file_2nd.write("@autoscale onread none\n")
513
514             # Close the files.
515             file_1st.close()
516             file_2nd.close()
517
518     # Calculate the frame order.
519     Frame_order()

```

16.3 Rigid frame order model

16.3.1 Rigid model parameterisation

This model consists solely of the six parameters of translation and rotation of the ‘moving’ domain to its average position. The parameter set is therefore

$$\mathfrak{M} = \mathfrak{P} = \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\}, \quad (16.1)$$

where P_i are the average domain position translations and rotations.

16.3.2 Rigid model equations

Rigid model rotation matrices

For the rigid model, there is no motion so the rotation matrix is simply the identity matrix

$$R = \begin{pmatrix} 1 & . & . \\ . & 1 & . \\ . & . & 1 \end{pmatrix}. \quad (16.2)$$

Rigid frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = R^{\otimes n}, \quad (16.3)$$

where surface normalisation factor is 1.

Rigid 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is the identity matrix

$$\mathbb{M}^{(1)} = R^{\otimes 1}, \quad (16.4a)$$

$$= \begin{pmatrix} 1 & . & . \\ . & 1 & . \\ . & . & 1 \end{pmatrix}. \quad (16.4b)$$

Rigid 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 is the identity matrix

$$\mathbb{M}^{(2)} = R^{\otimes 2}, \quad (16.5a)$$

$$= \begin{pmatrix} 1 & & & & & & & & \\ . & 1 & . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & 1 & . \\ . & . & . & . & . & . & . & . & 1 \end{pmatrix}. \quad (16.5b)$$

16.4 Rotor frame order model

The rotor model is defined by its rotation axis and a pivot point to position the center of the rotation.

16.4.1 Rotor parameterisation

The natural way to parameterise the rotation axis of rotor frame order model is to use a 3D point, the pivot point, and a unit vector using the spherical angle basis set. This would result in the parameter set

$$\mathfrak{M} = \mathfrak{E}_{\text{ax}} + \mathfrak{p}_1, \quad (16.6a)$$

$$= \{E_\theta, E_\phi\} + \{p_x, p_y, p_z\}. \quad (16.6b)$$

However this is an overparameterisation as the point $\{p_x, p_y, p_z\}$ can lie anywhere on the line defined by itself and the unit vector. Due to computational truncation artifacts, this results in the pivot point shooting out to infinity along the line during optimisation.

The minimal set of independent parameters for the rotor model is four. There are many parameterisations of lines in 3D using this minimal set of four parameters, however many of these suffer from singularity problems which can be fatal for optimisation. By using geometry of the model together with information about the molecular system, a parameterisation can be constructed which avoid singularities:

- A point of reference is the PDB frame close to the molecule is chosen, in this case the centre of mass (CoM) of the total system.
- The point on the line defining the rotation axis closest to the reference point defines an orthogonal system of the rotation axis to the vector of the CoM to closest point. This point can be optimised with the parameters $\mathfrak{p}_1 = \{p_x, p_y, p_z\}$ with no singularity problems and minimising truncation artifacts in the numerical PCS calculation.

- As the rotation axis is perpendicular to the CoM- \mathbf{p}_1 vector, it can be defined using a single angle of rotation (E_α^{ax}). The reference starting point for the angle is arbitrarily set to the xy-plane. The rotation of a unit vector about the CoM- \mathbf{p}_1 centred at \mathbf{p}_1 is smooth, hence singularities are also avoided.

The full parameter set for the rotor model implementation is therefore

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E}_{\text{ax}}^\alpha + \mathbf{p}_1 + \mathfrak{S}, \quad (16.7\text{a})$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\alpha^{\text{ax}}\} + \{p_x, p_y, p_z\} + \{\sigma_{\max}\}, \quad (16.7\text{b})$$

where P_i are the average domain position translations and rotations, E_α^{ax} is the single angle defining the rotation axis, p_i are the coordinates of the pivot point, and σ_{\max} is the torsion half-angle of the rotor motion.

The rotation axis vector in the system is calculated as

$$\hat{r}_{ax} = |R_\alpha \cdot (\hat{r}_{\text{CoM}-\mathbf{p}_1} \times \hat{z})|. \quad (16.8)$$

The angle E_α^{ax} is obtained from \hat{r}_{ax} as

$$E_\alpha^{\text{ax}} = \frac{\pi}{2} - \arccos(\hat{r}_{ax} \cdot \hat{z}), \quad (16.9\text{a})$$

$$= \arcsin(\hat{r}_{ax} \cdot \hat{z}), \quad (16.9\text{b})$$

16.4.2 Rotor equations

The only motion is in the torsion angle about the rotation axis.

Rotor rotation matrices

Assuming the rotation axis is the z-axis, the rotation matrix is defined as

$$R(\sigma) = \begin{pmatrix} \cos \sigma & -\sin \sigma & 0 \\ \sin \sigma & \cos \sigma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (16.10)$$

Rotor frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS \Big/ \int_S dS, \quad (16.11\text{a})$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} R^{\otimes n} d\sigma \Big/ \int_S dS. \quad (16.11\text{b})$$

The surface normalisation factor is

$$\int_S dS = \int_{-\sigma_{\max}}^{\sigma_{\max}} d\sigma, \quad (16.12\text{a})$$

$$= 2\sigma_{\max}. \quad (16.12\text{b})$$

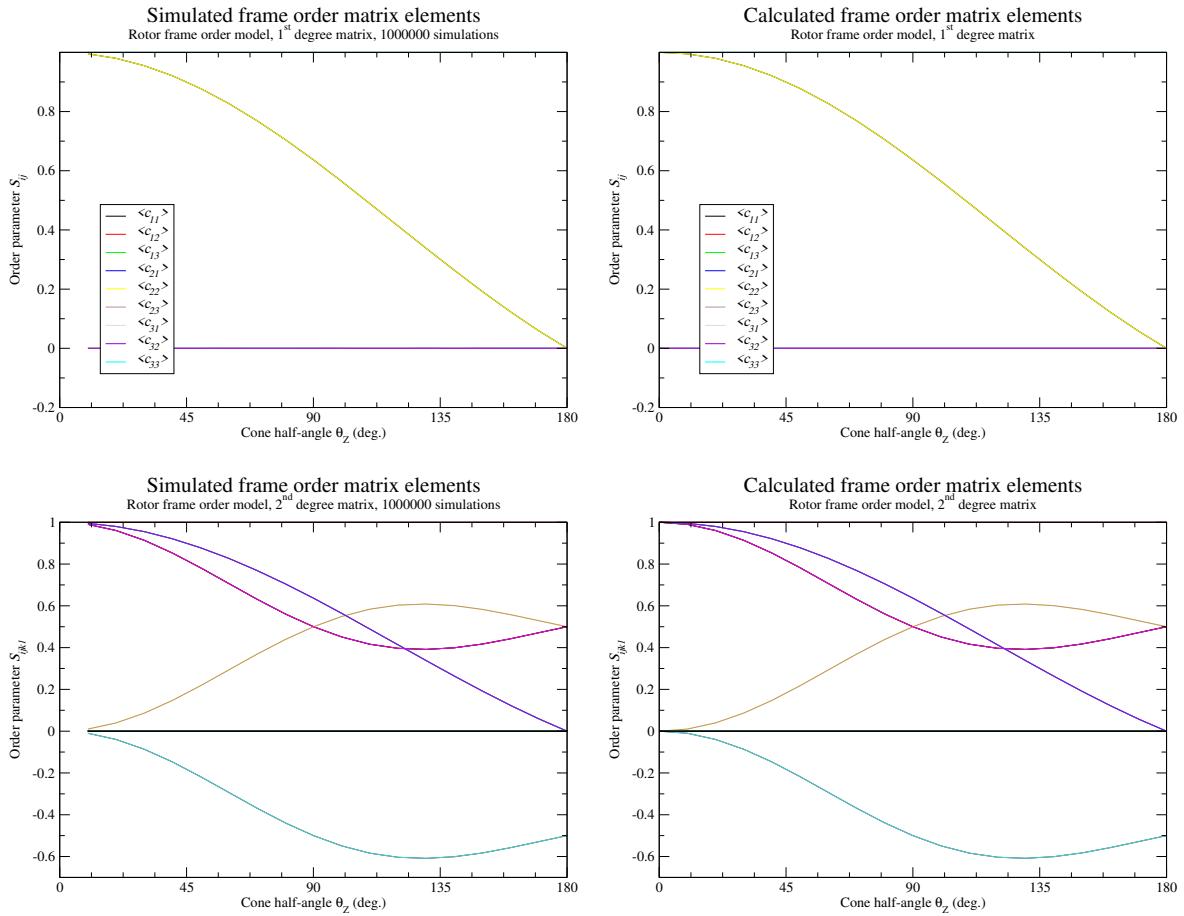


Figure 16.1: The rotor model simulated and calculated in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_Z corresponds to the torsion half-angle σ_{\max} . Frame order matrix values have been calculated every 10 degrees.

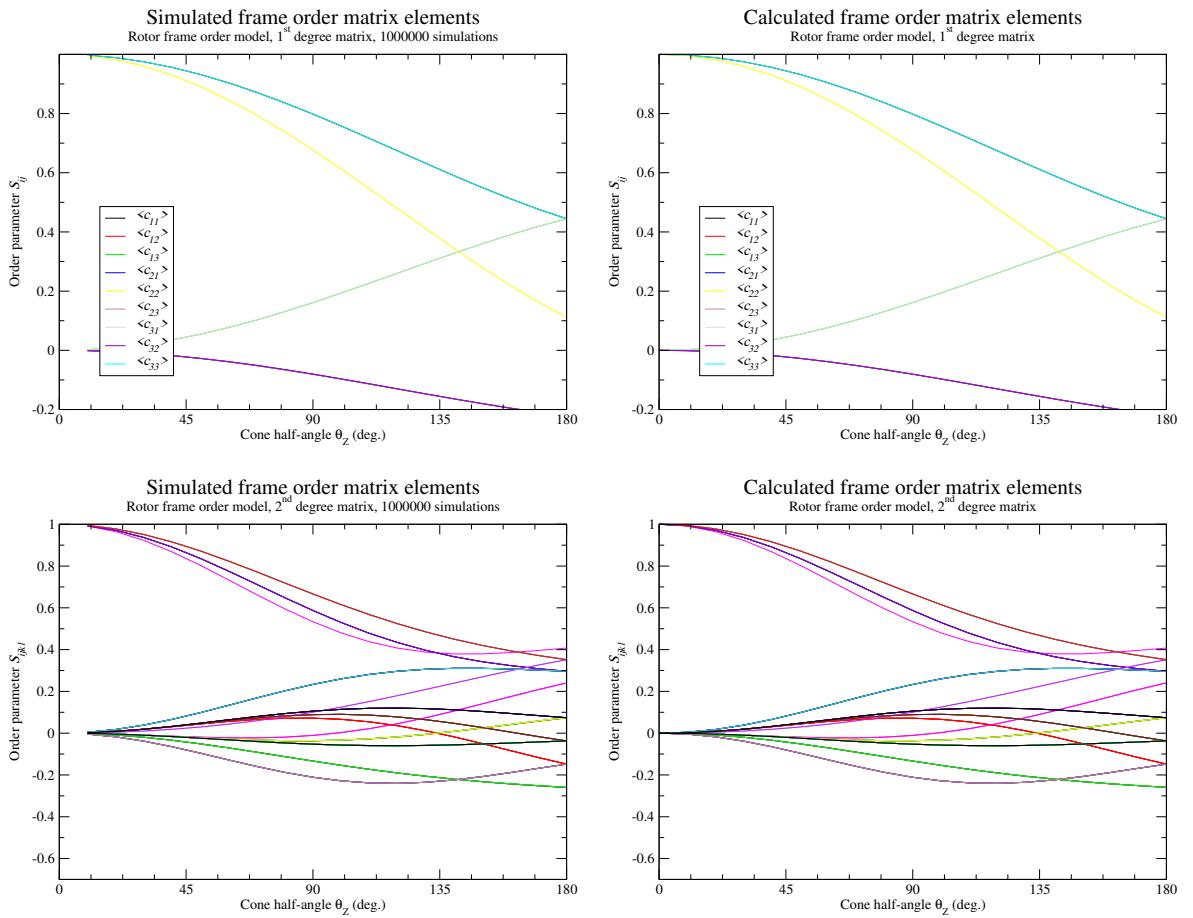


Figure 16.2: The rotor model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_Z corresponds to the torsion half-angle σ_{\max} . Frame order matrix values have been calculated every 10 degrees.

Rotor 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS / \int_S dS, \quad (16.13a)$$

$$= \int_S R dS / 2\sigma_{\max}, \quad (16.13b)$$

$$= \begin{pmatrix} \text{sinc } \sigma_{\max} & \cdot & \cdot \\ \cdot & \text{sinc } \sigma_{\max} & \cdot \\ \cdot & \cdot & 1 \end{pmatrix}. \quad (16.13c)$$

Rotor 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 is

$$\mathbb{M}^{(2)} = \int_S R^{\otimes 2} dS / \int_S dS, \quad (16.14a)$$

$$= \int_S R \otimes R dS / 2\sigma_{\max}, \quad (16.14b)$$

$$= \frac{1}{2} \begin{pmatrix} s_{2\sigma} + 1 & \cdot & \cdot & \cdot & -s_{2\sigma} + 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & s_{2\sigma} + 1 & \cdot & s_{2\sigma} - 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 2s_{\sigma} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & s_{2\sigma} - 1 & \cdot & s_{2\sigma} + 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ -s_{2\sigma} + 1 & \cdot & \cdot & \cdot & s_{2\sigma} + 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 2s_{\sigma} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 2s_{\sigma} & \cdot & \cdot \\ \cdot & 2s_{\sigma} & \cdot \\ \cdot & 1 \end{pmatrix}, \quad (16.14c)$$

where $s_{\sigma} = \text{sinc } \sigma_{\max}$ and $s_{2\sigma} = \text{sinc}(2\sigma_{\max})$. The active matrix elements which are not zero due to symmetries, in Kronecker product double indices from 0 to 8, are

$$\mathbb{M}_{00} = \frac{1}{2} \text{sinc}(2\sigma_{\max}) + \frac{1}{2}, \quad (16.15a)$$

$$\mathbb{M}_{11} = \mathbb{M}_{00}, \quad (16.15b)$$

$$\mathbb{M}_{22} = \text{sinc } \sigma_{\max}, \quad (16.15c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{00}, \quad (16.15d)$$

$$\mathbb{M}_{44} = \mathbb{M}_{00}, \quad (16.15e)$$

$$\mathbb{M}_{55} = \mathbb{M}_{22}, \quad (16.15f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.15g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{22}, \quad (16.15h)$$

$$\mathbb{M}_{88} = 1, \quad (16.15i)$$

$$\mathbb{M}_{04} = -\frac{1}{2} \text{sinc}(2\sigma_{\max}) + \frac{1}{2}, \quad (16.15j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{04}, \quad (16.15k)$$

$$\mathbb{M}_{08} = 0, \quad (16.15l)$$

$$\mathbb{M}_{80} = 0, \quad (16.15m)$$

$$\mathbb{M}_{48} = 0, \quad (16.15n)$$

$$\mathbb{M}_{84} = 0, \quad (16.15o)$$

$$\mathbb{M}_{13} = -\mathbb{M}_{04}, \quad (16.15p)$$

$$\mathbb{M}_{31} = -\mathbb{M}_{04}, \quad (16.15q)$$

$$\mathbb{M}_{26} = 0, \quad (16.15r)$$

$$\mathbb{M}_{62} = 0, \quad (16.15s)$$

$$\mathbb{M}_{57} = 0, \quad (16.15t)$$

$$\mathbb{M}_{75} = 0. \quad (16.15u)$$

Rotor frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.13 and 16.15 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.1. The out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.2.

16.5 Free rotor frame order model

This is similar to the rotor model but with no torsional restriction ($\sigma_{\max} = \pi$).

16.5.1 Free rotor parameterisation

The full parameter set for the free-rotor model implementation is

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E}_{\text{ax}}^{\alpha} + \mathfrak{p}_1, \quad (16.16a)$$

$$= \{P_x, P_y, P_z, P_{\alpha}, P_{\beta}, P_{\gamma}\} + \{E_{\alpha}^{\text{ax}}\} + \{p_x, p_y, p_z\}, \quad (16.16b)$$

where P_i are the average domain position translations and rotations, E_{α}^{ax} is the single angle defining the rotation axis, and p_i are the coordinates of the pivot point.

16.5.2 Free rotor equations

Free rotor rotation matrices

The rotation matrix is defined in equation 16.10.

Free rotor frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS / \int_S dS, \quad (16.17a)$$

$$= \int_{-\pi}^{\pi} R^{\otimes n} d\sigma / \int_S dS. \quad (16.17b)$$

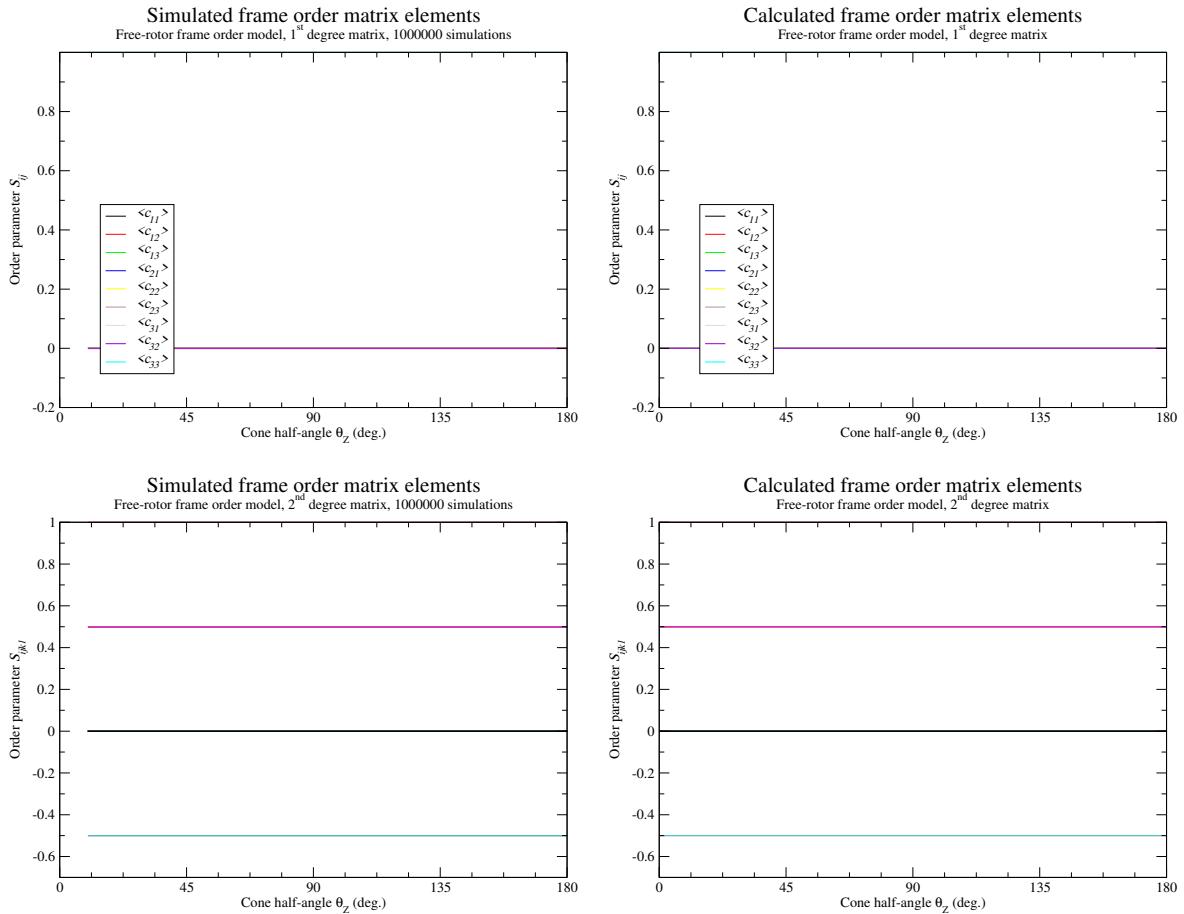


Figure 16.3: The free rotor model simulated and calculated in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, as no motional order parameters exist for the model, θ_z corresponds to nothing. Frame order matrix values have been calculated every 10 degrees.

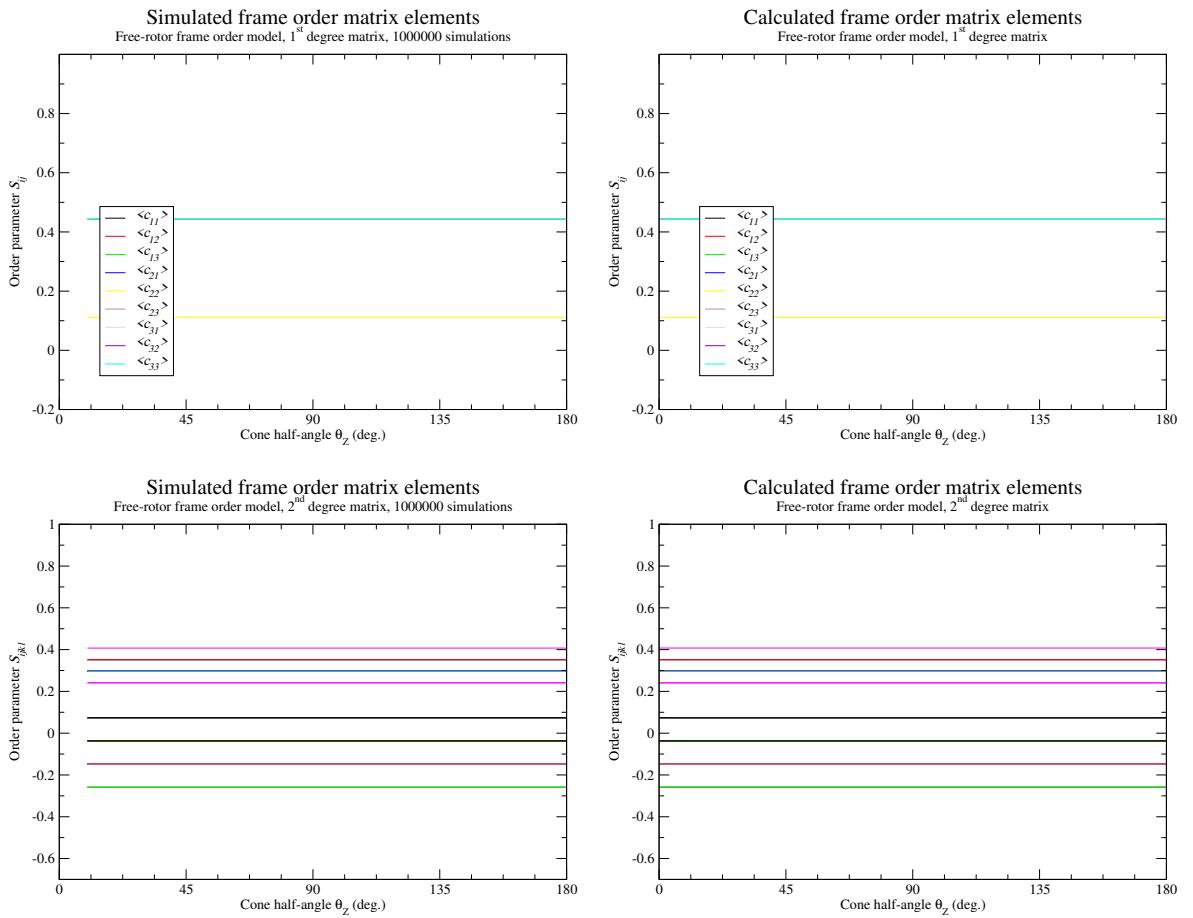


Figure 16.4: The free rotor model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, as no motional order parameters exist for the model, θ_z corresponds to nothing. Frame order matrix values have been calculated every 10 degrees.

The surface normalisation factor is

$$\int_S dS = \int_{-\pi}^{\pi} d\sigma, \quad (16.18a)$$

$$= 2\pi. \quad (16.18b)$$

Free rotor 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS / \int_S dS, \quad (16.19a)$$

$$= \int_S R dS / 2\pi, \quad (16.19b)$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 \end{pmatrix}. \quad (16.19c)$$

Free rotor 2nd degree frame order The frame order matrix in Kronecker product notation is fixed as

$$\mathbb{M}^{(2)} = \begin{pmatrix} \frac{1}{2} & \cdot & \cdot & \cdot & \frac{1}{2} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \frac{1}{2} & \cdot & -\frac{1}{2} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot \\ \cdot & -\frac{1}{2} & \cdot & \frac{1}{2} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{1}{2} & \cdot & \cdot & \cdot & \frac{1}{2} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & 1 \end{pmatrix}. \quad (16.20)$$

Free rotor frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.19 and 16.20 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.3. The out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.4.

16.6 Isotropic cone frame order model

16.6.1 Isotropic cone parameterisation

In this model, the tilt component of the tilt and torsion angle system is modelled simply as

$$\theta \leq \theta_{\max}. \quad (16.21)$$

The torsion angle restriction of 12.76 on page 253 is used for the modelling of the torsion component. A uniform distribution of rigid body positions within these bounds is assumed.

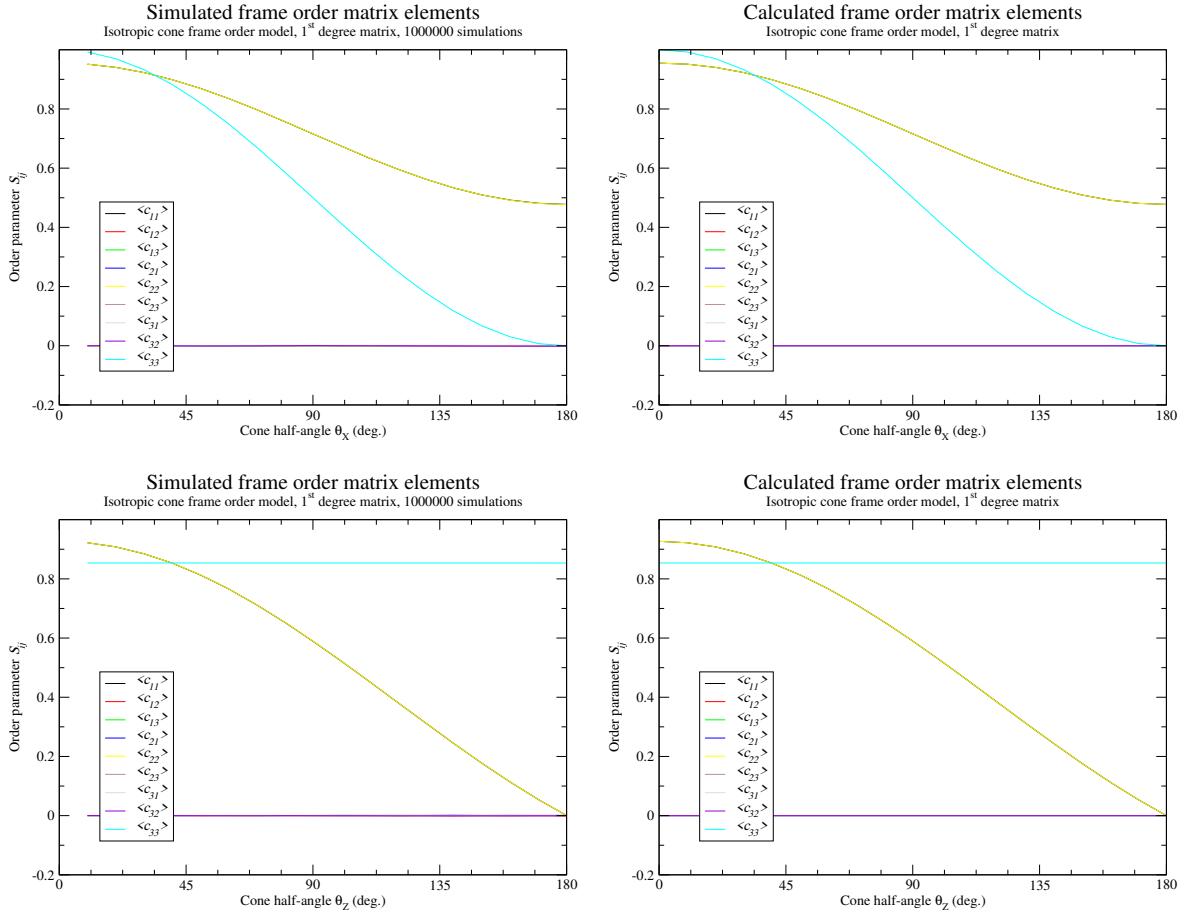


Figure 16.5: The isotropic cone model simulated and calculated in-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ and θ_Z to the torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta = \pi/4$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees.

The isotropic cone model of the ball and socket joint consists of a single pivot point, a single unit z-vector of the motional eigenframe defining the cone axis, and the maximum cone opening and torsion half-angles. Including the average domain position, the parameter set is therefore

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E}_{\text{ax}} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.22a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\theta, E_\phi\} + \{p_x, p_y, p_z\} + \{\theta_{\max}, \sigma_{\max}\}, \quad (16.22b)$$

where P_i are the average domain position translations and rotations, E_i are the spherical angles defining the cone axis, p_i are the coordinates of the pivot point, and θ_{\max} and σ_{\max} are the maximum cone opening and torsion half-angles respectively.

16.6.2 Isotropic cone equations

Isotropic cone rotation matrices

The rotation matrix is the full torsion-tilt rotation matrix of equation 12.74c on page 252.

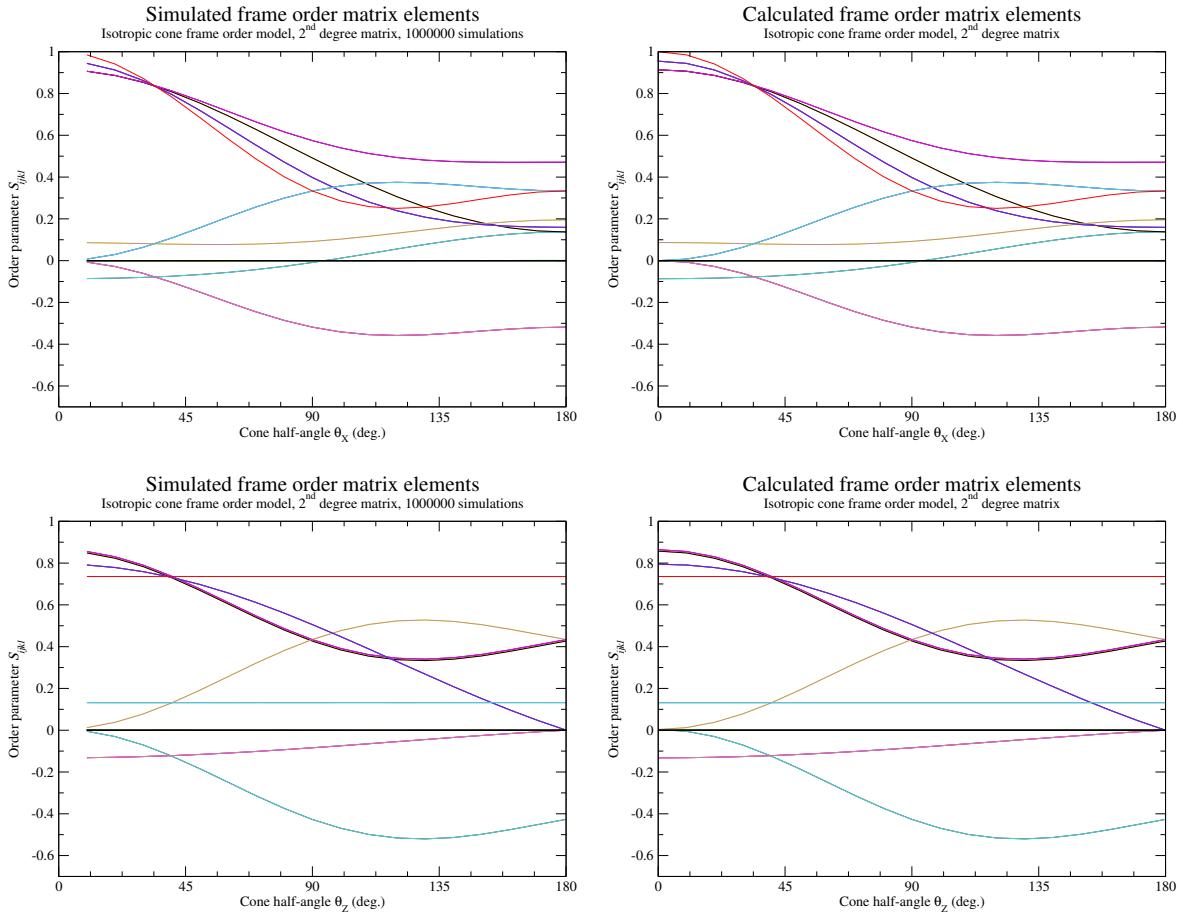


Figure 16.6: The isotropic cone model simulated and calculated in-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ and θ_Z to the torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta = \pi/4$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees.

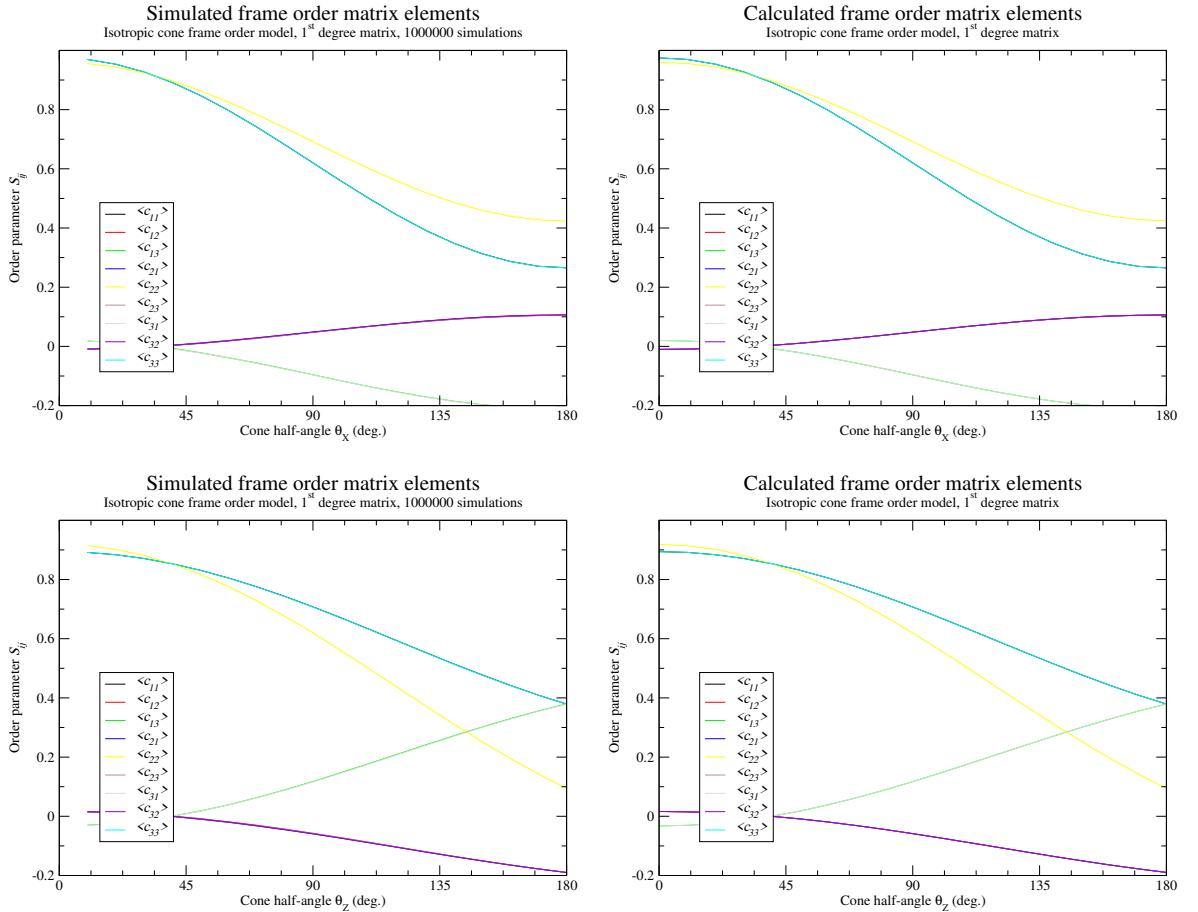


Figure 16.7: The isotropic cone model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ and θ_Z to the torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta = \pi/4$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees.

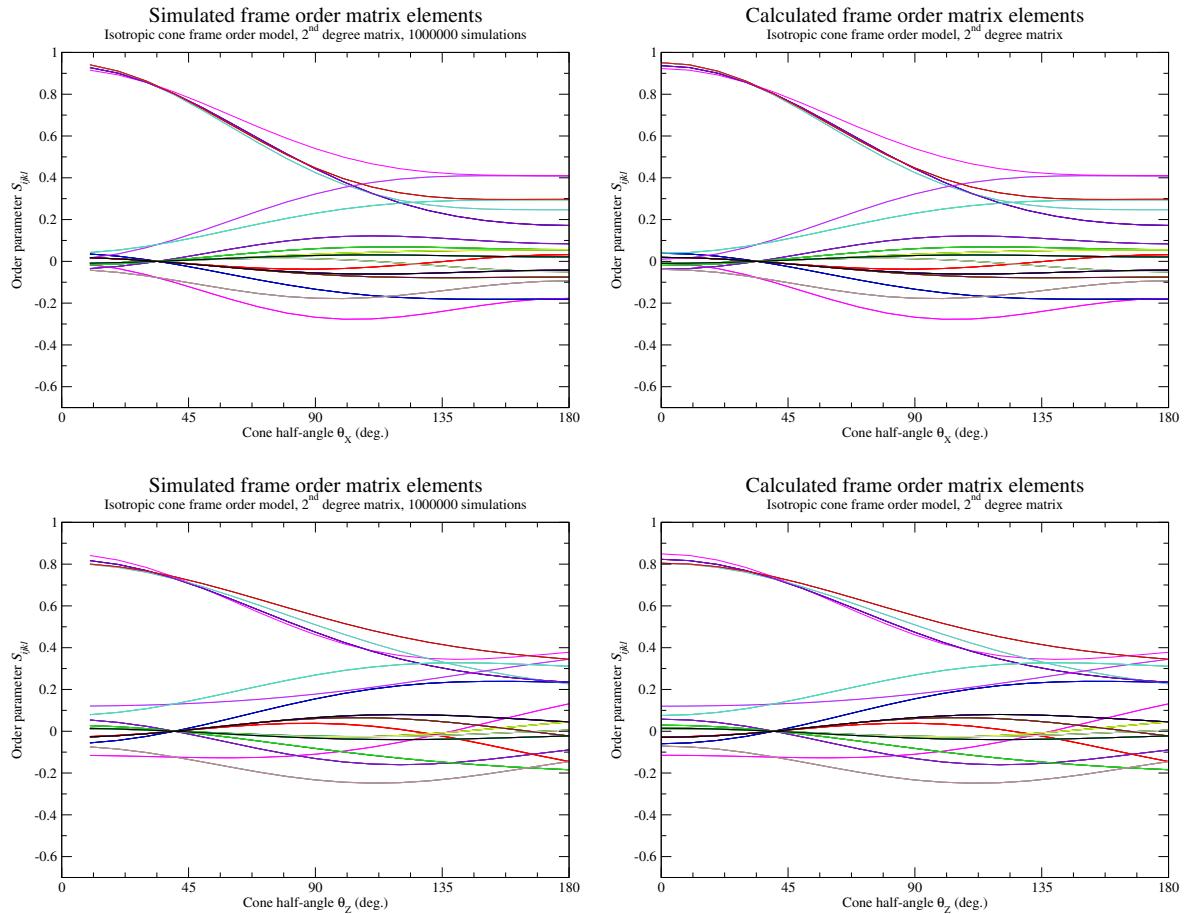


Figure 16.8: The isotropic cone model simulated and calculated out-of-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ and θ_Z to the torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta = \pi/4$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees.

Isotropic cone frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS \Big/ \int_S dS, \quad (16.23a)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi d\sigma \Big/ \int_S dS. \quad (16.23b)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi d\sigma, \quad (16.24a)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi d\sigma, \quad (16.24b)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} 2\pi (1 - \cos \theta_{\max}) d\sigma, \quad (16.24c)$$

$$= 4\pi \sigma_{\max} (1 - \cos \theta_{\max}). \quad (16.24d)$$

Isotropic cone 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS \Big/ \int_S dS, \quad (16.25a)$$

$$= \int_S R dS \Big/ 4\pi \sigma_{\max} (1 - \cos \theta_{\max}), \quad (16.25b)$$

$$= \frac{1}{4} \begin{pmatrix} \text{sinc } \sigma_{\max} (\cos \theta_{\max} + 3) & & & \\ & \ddots & & \\ & & \text{sinc } \sigma_{\max} (\cos \theta_{\max} + 3) & \\ & & & \ddots \\ & & & & 2 \cos \theta_{\max} + 2 \end{pmatrix}. \quad (16.25c)$$

Isotropic cone 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\mathbb{M}_{00} = \frac{1}{24} \text{sinc}(2\sigma_{\max}) (\cos^2 \theta_{\max} + 4 \cos \theta_{\max} + 7) + \frac{1}{12} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 4), \quad (16.26a)$$

$$\mathbb{M}_{11} = \frac{1}{24} \text{sinc}(2\sigma_{\max}) (\cos^2 \theta_{\max} + 4 \cos \theta_{\max} + 7) + \frac{1}{4} (\cos \theta_{\max} + 1), \quad (16.26b)$$

$$\mathbb{M}_{22} = \frac{1}{12} \text{sinc}(2\sigma_{\max}) (2 \cos^2 \theta_{\max} + 5 \cos \theta_{\max} + 5), \quad (16.26c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.26d)$$

$$\mathbb{M}_{44} = \mathbb{M}_{00}, \quad (16.26e)$$

$$\mathbb{M}_{55} = \mathbb{M}_{22}, \quad (16.26f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.26g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{22}, \quad (16.26h)$$

$$\mathbb{M}_{88} = \frac{1}{3} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 1), \quad (16.26i)$$

$$\mathbb{M}_{04} = -\frac{1}{24} \text{sinc}(2\sigma_{\max}) (\cos^2 \theta_{\max} + 4 \cos \theta_{\max} + 7) + \frac{1}{12} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 4), \quad (16.26j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{04}, \quad (16.26k)$$

$$\mathbb{M}_{08} = -\frac{1}{6} (\cos^2 \theta_{\max} + \cos \theta_{\max} - 2), \quad (16.26l)$$

$$\mathbb{M}_{80} = \mathbb{M}_{08}, \quad (16.26m)$$

$$\mathbb{M}_{48} = \mathbb{M}_{08}, \quad (16.26n)$$

$$\mathbb{M}_{84} = \mathbb{M}_{08}, \quad (16.26o)$$

$$\mathbb{M}_{13} = \frac{1}{24} \operatorname{sinc}(2\sigma_{\max}) (\cos^2 \theta_{\max} + 4 \cos \theta_{\max} + 7) - \frac{1}{4} (\cos \theta_{\max} + 1), \quad (16.26p)$$

$$\mathbb{M}_{31} = \mathbb{M}_{13}, \quad (16.26q)$$

$$\mathbb{M}_{26} = \frac{1}{6} \operatorname{sinc}(2\sigma_{\max}) (\cos^2 \theta_{\max} + \cos \theta_{\max} - 2), \quad (16.26r)$$

$$\mathbb{M}_{62} = \mathbb{M}_{26}, \quad (16.26s)$$

$$\mathbb{M}_{57} = \mathbb{M}_{26}, \quad (16.26t)$$

$$\mathbb{M}_{75} = \mathbb{M}_{26}. \quad (16.26u)$$

Isotropic cone frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.25 and 16.26 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.5 and $\mathbb{M}^{(2)}$ in figure 16.6. The out-of-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.7 and $\mathbb{M}^{(2)}$ in figure 16.8.

16.7 Torsionless isotropic cone frame order model

16.7.1 Torsionless isotropic cone parameterisation

As this is simply the isotropic cone model with the torsion angle set to zero, $\sigma_{\max} = 0$, the models parameters are

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E}_{\text{ax}} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.27a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\theta, E_\phi\} + \{p_x, p_y, p_z\} + \{\theta_{\max}\}, \quad (16.27b)$$

where P_i are the average domain position translations and rotations, E_i are the spherical angles defining the cone axis, p_i are the coordinates of the pivot point, and θ_{\max} is the maximum cone opening half-angle.

16.7.2 Torsionless isotropic cone equations

Torsionless isotropic cone rotation matrices

The torsionless rotation matrix is defined in equation 12.77c.

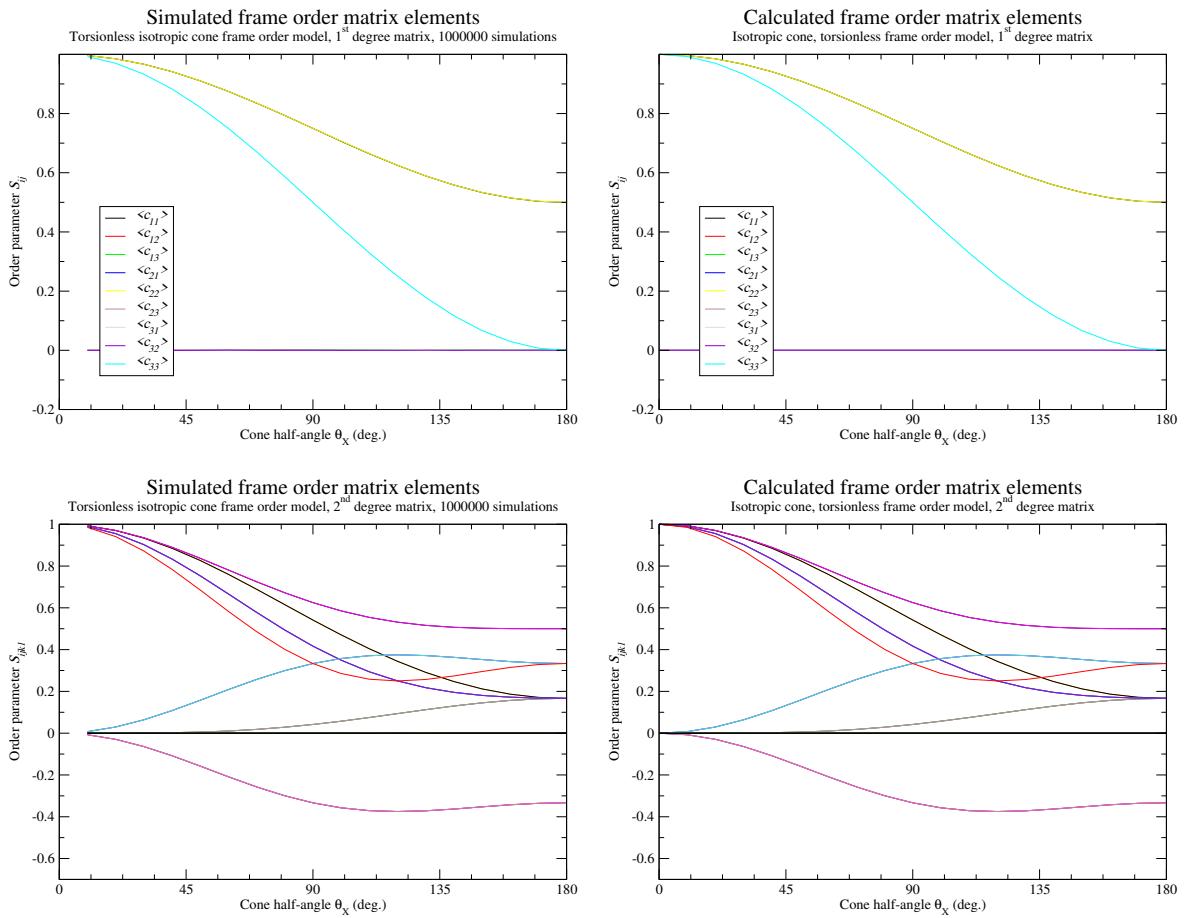


Figure 16.9: The torsionless isotropic cone model simulated and calculated in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_X corresponds to the cone opening half-angle θ . Frame order matrix values have been calculated every 10 degrees.

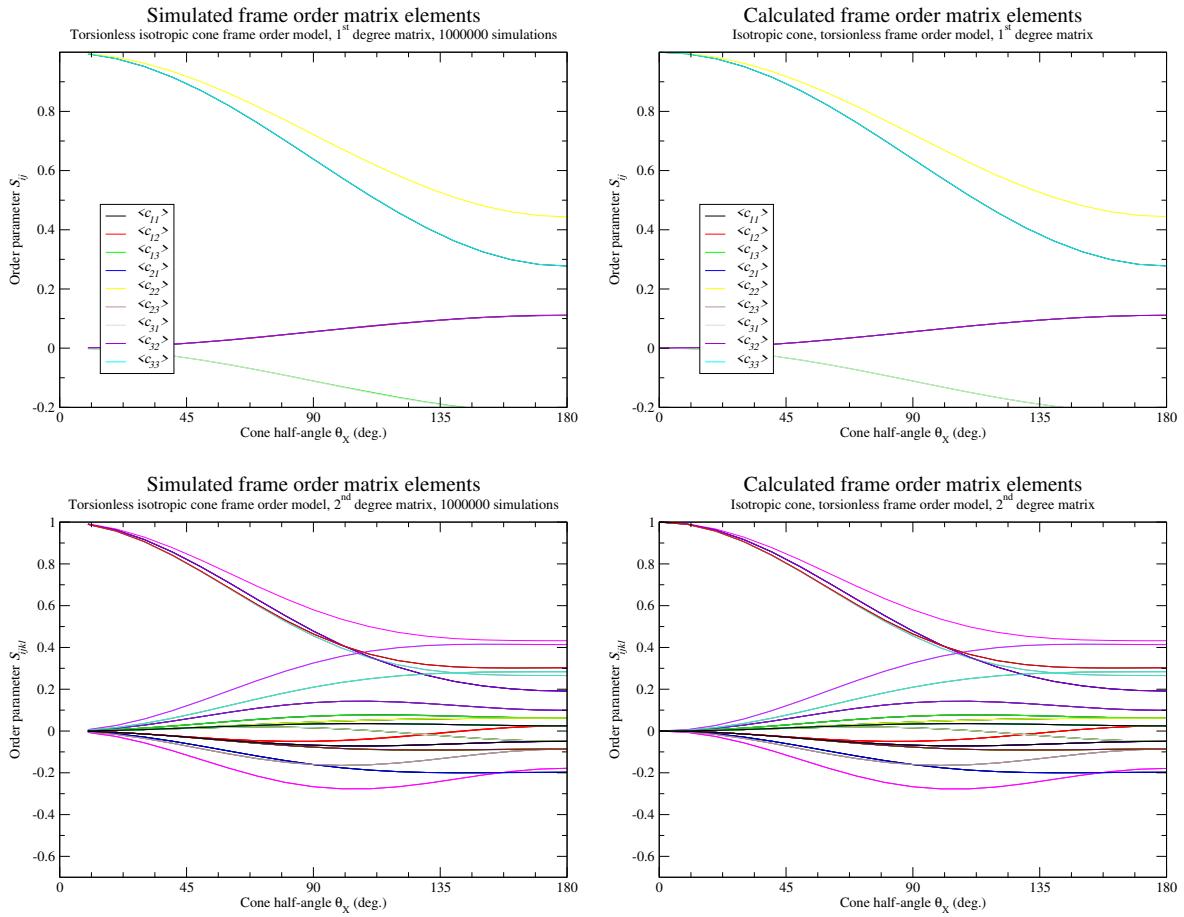


Figure 16.10: The torsionless isotropic cone model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_X corresponds to the cone opening half-angle θ . Frame order matrix values have been calculated every 10 degrees.

Torsionless isotropic cone frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS \Big/ \int_S dS, \quad (16.28a)$$

$$= \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi \Big/ \int_S dS. \quad (16.28b)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi, \quad (16.29a)$$

$$= \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi, \quad (16.29b)$$

$$= 2\pi (1 - \cos \theta_{\max}). \quad (16.29c)$$

Torsionless isotropic cone 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS \Big/ \int_S dS, \quad (16.30a)$$

$$= \int_S R dS \Big/ 2\pi (1 - \cos \theta_{\max}), \quad (16.30b)$$

$$= \frac{1}{4} \begin{pmatrix} \cos \theta_{\max} + 3 & . & . & . \\ . & \cos \theta_{\max} + 3 & . & . \\ . & . & \cos \theta_{\max} + 3 & . \\ . & . & . & 2(\cos \theta_{\max} + 1) \end{pmatrix}. \quad (16.30c)$$

Torsionless isotropic cone 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\mathbb{M}_{00} = \frac{1}{24} (3 \cos^2 \theta_{\max} + 6 \cos \theta_{\max} + 15), \quad (16.31a)$$

$$\mathbb{M}_{11} = \frac{1}{24} (\cos^2 \theta_{\max} + 10 \cos \theta_{\max} + 13), \quad (16.31b)$$

$$\mathbb{M}_{22} = \frac{1}{24} (4 \cos^2 \theta_{\max} + 10 \cos \theta_{\max} + 10), \quad (16.31c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.31d)$$

$$\mathbb{M}_{44} = \mathbb{M}_{00}, \quad (16.31e)$$

$$\mathbb{M}_{55} = \mathbb{M}_{22}, \quad (16.31f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.31g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{22}, \quad (16.31h)$$

$$\mathbb{M}_{88} = \frac{1}{3} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 1), \quad (16.31i)$$

$$\mathbb{M}_{04} = \frac{1}{24} (\cos^2 \theta_{\max} - 2 \cos \theta_{\max} + 1), \quad (16.31j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{04}, \quad (16.31k)$$

$$\mathbb{M}_{08} = -\frac{1}{6} (\cos^2 \theta_{\max} + \cos \theta_{\max} - 2), \quad (16.31l)$$

$$\mathbb{M}_{80} = \mathbb{M}_{08}, \quad (16.31\text{m})$$

$$\mathbb{M}_{48} = \mathbb{M}_{08}, \quad (16.31\text{n})$$

$$\mathbb{M}_{84} = \mathbb{M}_{08}, \quad (16.31\text{o})$$

$$\mathbb{M}_{13} = \mathbb{M}_{04}, \quad (16.31\text{p})$$

$$\mathbb{M}_{31} = \mathbb{M}_{04}, \quad (16.31\text{q})$$

$$\mathbb{M}_{26} = -\mathbb{M}_{08}, \quad (16.31\text{r})$$

$$\mathbb{M}_{62} = -\mathbb{M}_{08}, \quad (16.31\text{s})$$

$$\mathbb{M}_{57} = -\mathbb{M}_{08}, \quad (16.31\text{t})$$

$$\mathbb{M}_{75} = -\mathbb{M}_{08}. \quad (16.31\text{u})$$

After factorisation, the equations are

$$\mathbb{M}_{00} = \frac{1}{8} (\cos \theta_{\max} + 1)^2 + \frac{1}{2}, \quad (16.32\text{a})$$

$$\mathbb{M}_{11} = \frac{1}{24} (\cos \theta_{\max} + 5)^2 - \frac{1}{2}, \quad (16.32\text{b})$$

$$\mathbb{M}_{22} = \frac{1}{96} \left((4 \cos \theta_{\max} + 5)^2 + 15 \right), \quad (16.32\text{c})$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.32\text{d})$$

$$\mathbb{M}_{44} = \mathbb{M}_{00}, \quad (16.32\text{e})$$

$$\mathbb{M}_{55} = \mathbb{M}_{22}, \quad (16.32\text{f})$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.32\text{g})$$

$$\mathbb{M}_{77} = \mathbb{M}_{22}, \quad (16.32\text{h})$$

$$\mathbb{M}_{88} = \frac{1}{12} (2 \cos \theta_{\max} + 1)^2 + \frac{1}{4}, \quad (16.32\text{i})$$

$$\mathbb{M}_{04} = \frac{1}{24} (\cos \theta_{\max} - 1)^2, \quad (16.32\text{j})$$

$$\mathbb{M}_{40} = \mathbb{M}_{04}, \quad (16.32\text{k})$$

$$\mathbb{M}_{08} = -\frac{1}{6} (\cos \theta_{\max} + 2) (\cos \theta_{\max} - 1), \quad (16.32\text{l})$$

$$\mathbb{M}_{80} = \mathbb{M}_{08}, \quad (16.32\text{m})$$

$$\mathbb{M}_{48} = \mathbb{M}_{08}, \quad (16.32\text{n})$$

$$\mathbb{M}_{84} = \mathbb{M}_{08}, \quad (16.32\text{o})$$

$$\mathbb{M}_{13} = \mathbb{M}_{04}, \quad (16.32\text{p})$$

$$\mathbb{M}_{31} = \mathbb{M}_{04}, \quad (16.32\text{q})$$

$$\mathbb{M}_{26} = -\mathbb{M}_{08}, \quad (16.32\text{r})$$

$$\mathbb{M}_{62} = -\mathbb{M}_{08}, \quad (16.32\text{s})$$

$$\mathbb{M}_{57} = -\mathbb{M}_{08}, \quad (16.32\text{t})$$

$$\mathbb{M}_{75} = -\mathbb{M}_{08}. \quad (16.32\text{u})$$

Torsionless isotropic cone frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.30 and 16.32 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.9. The out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.10.

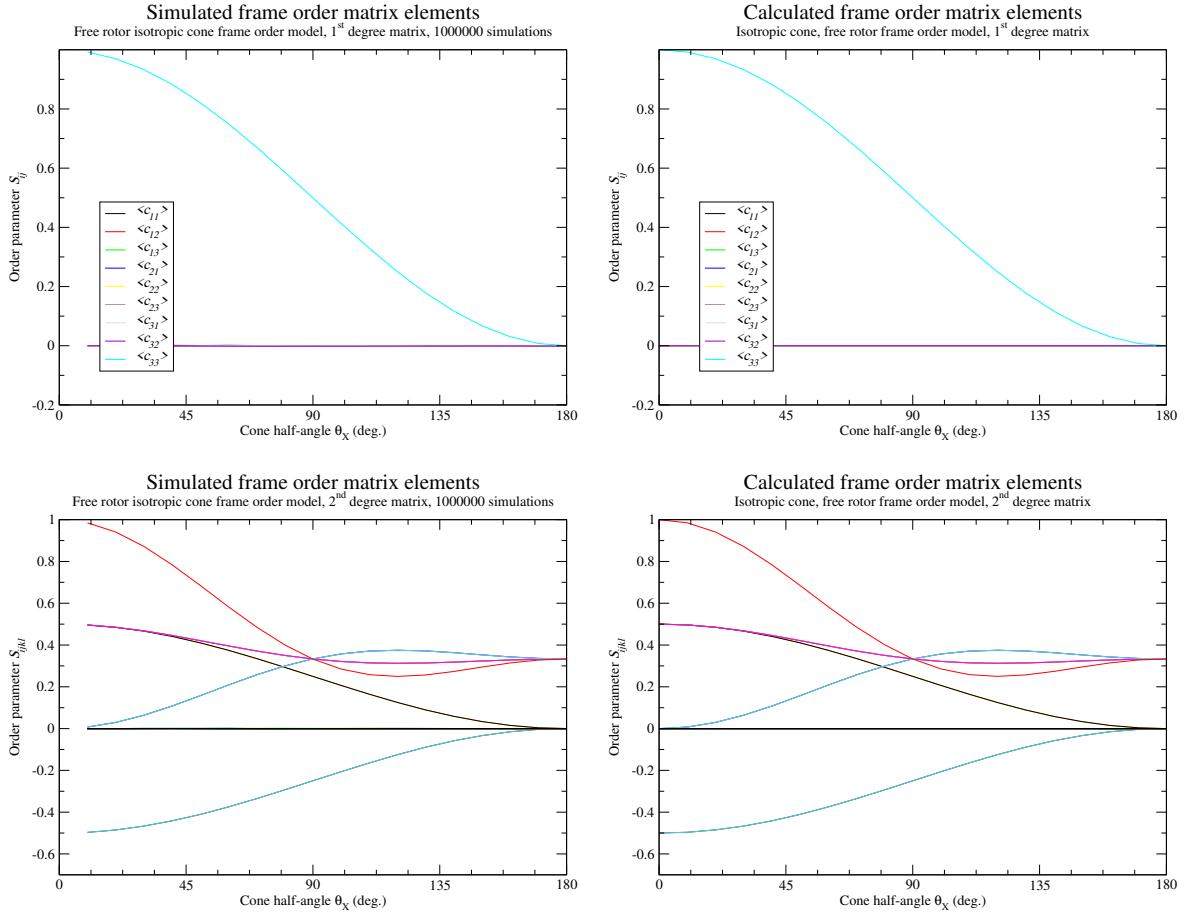


Figure 16.11: The free rotor isotropic cone model simulated and calculated in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_X corresponds to the cone opening half-angle θ . Frame order matrix values have been calculated every 10 degrees.

16.8 Free rotor isotropic cone frame order model

16.8.1 Free rotor isotropic cone parameterisation

For the free rotor variation of the isotropic cone model, the torsion angle restriction is absent and the model parameters are

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E}_{\text{ax}} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.33a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\theta, E_\phi\} + \{p_x, p_y, p_z\} + \{\theta_{\max}\}, \quad (16.33b)$$

where P_i are the average domain position translations and rotations, E_i are the spherical angles defining the cone axis, p_i are the coordinates of the pivot point, and θ_{\max} is the maximum cone opening half-angle.

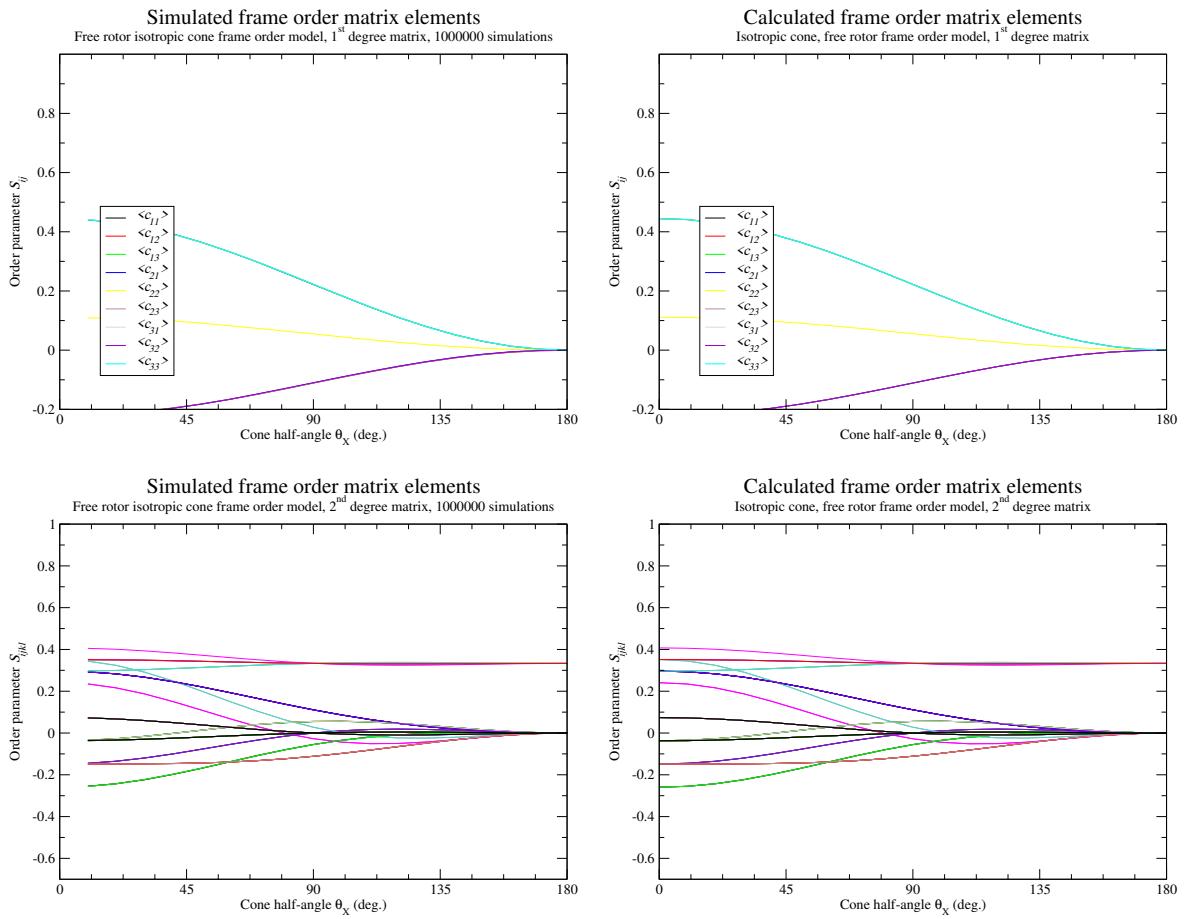


Figure 16.12: The free rotor isotropic cone model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ frame order matrix elements. The top row corresponds to $\mathbb{M}^{(1)}$ and the bottom to $\mathbb{M}^{(2)}$. In these plots, θ_X corresponds to the cone opening half-angle θ . Frame order matrix values have been calculated every 10 degrees.

16.8.2 Free rotor isotropic cone equations

Free rotor isotropic cone rotation matrices

The rotation matrix is the full torsion-tilt rotation matrix of equation 12.74c on page 252.

Free rotor isotropic cone frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS \Big/ \int_S dS, \quad (16.34a)$$

$$= \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi d\sigma \Big/ \int_S dS. \quad (16.34b)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi d\sigma, \quad (16.35a)$$

$$= \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi d\sigma, \quad (16.35b)$$

$$= \int_{-\pi}^{\pi} 2\pi (1 - \cos \theta_{\max}) d\sigma, \quad (16.35c)$$

$$= 4\pi^2 (1 - \cos \theta_{\max}). \quad (16.35d)$$

Free rotor isotropic cone 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS \Big/ \int_S dS, \quad (16.36a)$$

$$= \int_S R dS \Big/ 2\pi (1 - \cos \theta_{\max}), \quad (16.36b)$$

$$= \frac{1}{2} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cos \theta_{\max} + 1 \end{pmatrix}. \quad (16.36c)$$

Free rotor isotropic cone 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product

double indices from 0 to 8

$$\mathbb{M}_{00} = \frac{1}{12} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 4), \quad (16.37a)$$

$$\mathbb{M}_{11} = \frac{1}{4} (\cos \theta_{\max} + 1), \quad (16.37b)$$

$$\mathbb{M}_{22} = 0, \quad (16.37c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.37d)$$

$$\mathbb{M}_{44} = \mathbb{M}_{00}, \quad (16.37e)$$

$$\mathbb{M}_{55} = 0, \quad (16.37f)$$

$$\mathbb{M}_{66} = 0, \quad (16.37g)$$

$$\mathbb{M}_{77} = 0, \quad (16.37h)$$

$$\mathbb{M}_{88} = \frac{1}{3} (\cos^2 \theta_{\max} + \cos \theta_{\max} + 1), \quad (16.37i)$$

$$\mathbb{M}_{04} = \mathbb{M}_{00}, \quad (16.37j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{00}, \quad (16.37k)$$

$$\mathbb{M}_{08} = -\frac{1}{6} (\cos^2 \theta_{\max} + \cos \theta_{\max} - 2), \quad (16.37l)$$

$$\mathbb{M}_{80} = \mathbb{M}_{08}, \quad (16.37m)$$

$$\mathbb{M}_{48} = \mathbb{M}_{08}, \quad (16.37n)$$

$$\mathbb{M}_{84} = \mathbb{M}_{08}, \quad (16.37o)$$

$$\mathbb{M}_{13} = -\mathbb{M}_{11}, \quad (16.37p)$$

$$\mathbb{M}_{31} = -\mathbb{M}_{11}, \quad (16.37q)$$

$$\mathbb{M}_{26} = 0, \quad (16.37r)$$

$$\mathbb{M}_{62} = 0, \quad (16.37s)$$

$$\mathbb{M}_{57} = 0, \quad (16.37t)$$

$$\mathbb{M}_{75} = 0. \quad (16.37u)$$

Free rotor isotropic cone frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.36 and 16.37 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.11. The out-of-frame $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values are shown in figure 16.12.

16.9 Pseudo-ellipse frame order model

16.9.1 Pseudo-ellipse parameterisation

To extend to the next level of motional complexity above the isotropic cone models, an anisotropic cone can be modelled. This cone is defined via the ball and socket joint pivoted mechanics with an angular restriction in all three angles. The simplest anisotropic distribution would be to create an ellipse using the standard quadric surface formula for an elliptic cone

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0. \quad (16.38)$$

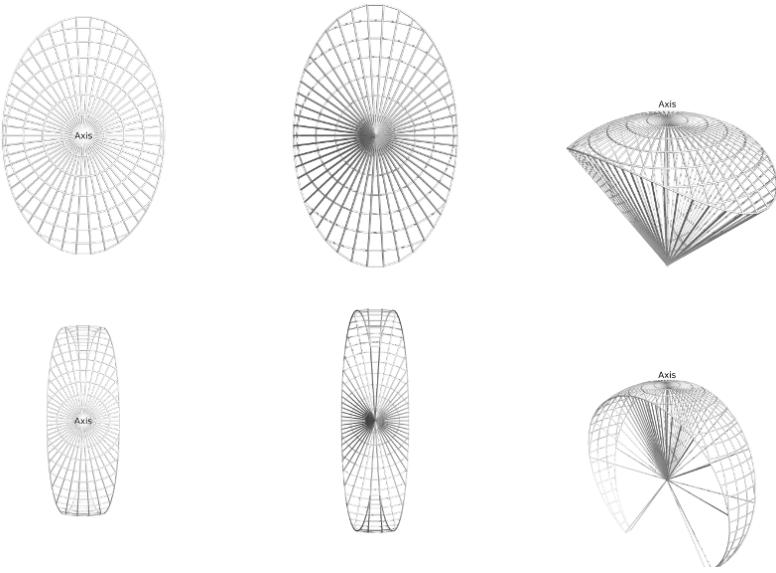


Figure 16.13: The pseudo-elliptic cone. The top three representations are for $\theta_x = 30^\circ$ and $\theta_y = 50^\circ$. The bottom three representations are for $\theta_x = 20^\circ$ and $\theta_y = 160^\circ$.

Let the two cone opening half-angles of the ellipse be θ_x and θ_y . For a sphere of radius $z = 1$ and using the tilt angles θ and ϕ , a boundary polar angle θ_{\max} can be modelled as

$$\frac{1}{\theta_{\max}^2} = \frac{\cos^2 \phi}{\theta_x^2} + \frac{\sin^2 \phi}{\theta_y^2}. \quad (16.39)$$

As the quadric constants a , b and c are angles rather than axis lengths, this is not a true ellipse. It will therefore instead be called a pseudo-ellipse. The form of this pseudo-elliptic cone is shown in figure 16.13.

The model consists of the average domain position, a single pivot point, the full motional eigenframe, and the maximum cone opening and torsion half-angles

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.40a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\alpha, E_\beta, E_\gamma\} + \{p_x, p_y, p_z\} + \{\theta_x, \theta_y, \sigma_{\max}\}, \quad (16.40b)$$

where P_i are the average domain position translations and rotations, E_i are the Euler angles defining the motional eigenframe, p_i are the coordinates of the pivot point, θ_x and θ_y are the maximum cone opening half-angles, and σ_{\max} is the torsion half-angle.

16.9.2 Derivation of a 2D trigonometric function - the pseudo-elliptic cosine

For the surface normalisation factor of the pseudo-elliptic cone, the integral from equation 16.51b on page 412 is

$$\int_S dS = \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi d\sigma. \quad (16.41)$$

When combined with the pseudo-ellipse of equation 16.39, this becomes the intractable integral

$$\int_S dS = \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} \left(1 - \cos \left(\frac{1}{\sqrt{\frac{\cos^2 \phi}{\theta_x^2} + \frac{\sin^2 \phi}{\theta_y^2}}} \right) \right) d\phi d\sigma, \quad (16.42)$$

Instead the cosine series expansion will be used

$$\cos \theta_{\max} = 1 - \frac{\theta_{\max}^2}{2!} + \frac{\theta_{\max}^4}{4!} - \frac{\theta_{\max}^6}{6!} + \frac{\theta_{\max}^8}{8!} - \frac{\theta_{\max}^{10}}{10!} + \dots, \quad (16.43)$$

$$= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} \theta_{\max}^{2n}. \quad (16.44)$$

Integrating each element of the sum over the ϕ parameter and using the assumption that $\theta_x, \theta_y \geq 0$ gives

$$\begin{aligned} \int_{-\pi}^{\pi} \frac{\theta_{\max}^2}{2!} d\phi &= \pi \theta_x \theta_y, \\ \int_{-\pi}^{\pi} \frac{\theta_{\max}^4}{4!} d\phi &= \frac{\pi \theta_x \theta_y}{24} (\theta_x^2 + \theta_y^2), \\ \int_{-\pi}^{\pi} \frac{\theta_{\max}^6}{6!} d\phi &= \frac{\pi \theta_x \theta_y}{2880} (3\theta_x^4 + 2\theta_x^2 \theta_y^2 + 3\theta_y^4), \\ \int_{-\pi}^{\pi} \frac{\theta_{\max}^8}{8!} d\phi &= \frac{\pi \theta_x \theta_y}{322560} (5\theta_x^6 + 3\theta_x^4 \theta_y^2 + 3\theta_x^2 \theta_y^4 + 5\theta_y^6), \\ \int_{-\pi}^{\pi} \frac{\theta_{\max}^{10}}{10!} d\phi &= \frac{\pi \theta_x \theta_y}{232243200} (35\theta_x^8 + 20\theta_x^6 \theta_y^2 + 18\theta_x^4 \theta_y^4 + 20x^2 \theta_y^6 + 35\theta_y^8), \\ &\dots \end{aligned} \quad (16.45)$$

Therefore a new two dimension trigonometric function, the pseudo-elliptic cosine, can be defined as

$$\text{pec}(\theta_x, \theta_y) = \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi, \quad (16.46a)$$

$$= 2\pi \theta_x \theta_y \sum_{n=0}^{\infty} \frac{(-1)^n}{4^n (2n+2)!} f_n(\theta_x, \theta_y). \quad (16.46b)$$

Let

$$\begin{aligned} a &= \theta_x^2, \\ b &= \theta_y^2, \end{aligned} \quad (16.47)$$

then the first of the $f_n(\theta_x, \theta_y)$ equations are

$$\begin{aligned}
f_0 &= 1, \\
f_1 &= 2a + 2b, \\
f_2 &= 6a^2 + 4ab + 6b^2, \\
f_3 &= 20a^3 + 12a^2b + 12ab^2 + 20b^3, \\
f_4 &= 70a^4 + 40a^3b + 36a^2b^2 + 40ab^3 + 70b^4, \\
f_5 &= 252a^5 + 140a^4b + 120a^3b^2 + 120a^2b^3 + 140ab^4 + 252b^5, \\
f_6 &= 924a^6 + 504a^5b + 420a^4b^2 + 400a^3b^3 + 420a^2b^4 + 504ab^5 + 924b^6, \\
f_7 &= 3432a^7 + 1848a^6b + 1512a^5b^2 + 1400a^4b^3 + 1400a^3b^4 + 1512a^2b^5 + 1848ab^6 + 3432b^7, \\
&\dots
\end{aligned} \tag{16.48}$$

Or

$$\begin{aligned}
f_0 &= 1, \\
f_1 &= 2(a + b), \\
f_2 &= 6(a^2 + b^2) + 4ab, \\
f_3 &= 20(a^3 + b^3) + 12(a^2b + ab^2), \\
f_4 &= 70(a^4 + b^4) + 40(a^3b + ab^3) + 36a^2b^2, \\
f_5 &= 252(a^5 + b^5) + 140(a^4b + ab^4) + 120(a^3b^2 + a^2b^3), \\
f_6 &= 924(a^6 + b^6) + 504(a^5b + ab^5) + 420(a^4b^2 + a^2b^4) + 400a^3b^3, \\
f_7 &= 3432(a^7 + b^7) + 1848(a^6b + ab^6) + 1512(a^5b^2 + a^2b^5) + 1400(a^4b^3 + a^3b^4), \\
f_8 &= 12870(a^8 + b^8) + 6864(a^7b + ab^7) + 5544(a^6b^2 + a^2b^6) + 5040(a^5b^3 + a^3b^5) + 4900a^4b^4, \\
f_9 &= 48620(a^9 + b^9) + 25740(a^8b + ab^8) + 20592(a^7b^2 + a^2b^7) + 18480(a^6b^3 + a^3b^6) + 17640(a^5b^4 + a^4b^5), \\
f_{10} &= 184756(a^{10} + b^{10}) + 97240(a^9b + ab^9) + 77220(a^8b^2 + a^2b^8) + 68640(a^7b^3 + a^3b^7) + 64680(a^6b^4 + a^4b^6) + 63504a^5b^5, \\
&\dots
\end{aligned} \tag{16.49}$$

This series expansion up to $n = 10$ is sufficient for writing a fast and accurate pec function implementation in computer code. The numerical representation of this function is shown in figure 16.14.

16.9.3 Pseudo-ellipse equations

Pseudo-ellipse rotation matrices

For the pseudo-ellipse model, the full torsion-tilt system is used. The full rotation matrix is given in equation 12.74c on page 252.

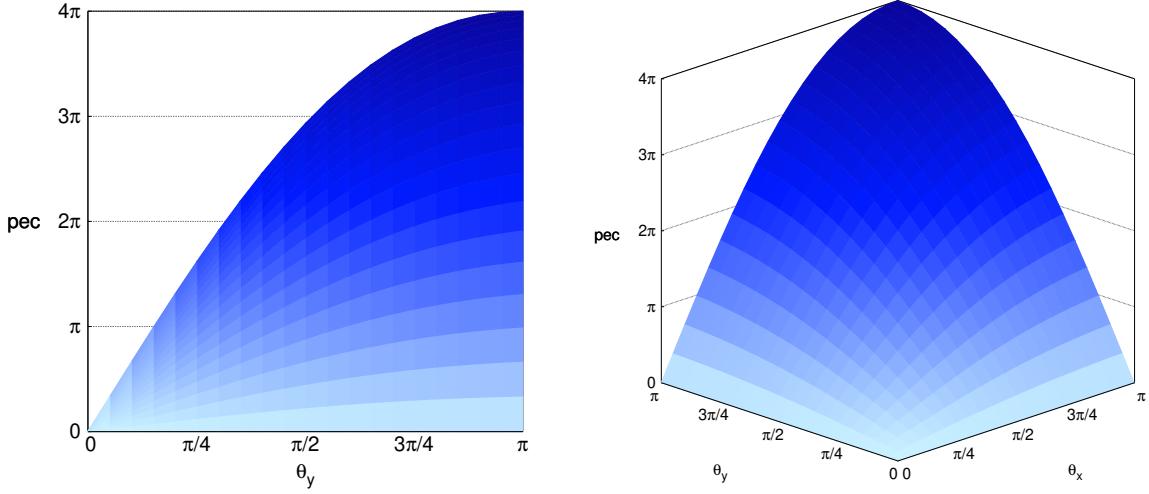


Figure 16.14: The pseudo-ellipse cosine 2D trigonometric function. This is the surface area on a unit sphere bounded by the pseudo-elliptic cone.

Pseudo-ellipse frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS / \int_S dS, \quad (16.50a)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi d\sigma / \int_S dS. \quad (16.50b)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi d\sigma, \quad (16.51a)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi d\sigma, \quad (16.51b)$$

$$= \int_{-\sigma_{\max}}^{\sigma_{\max}} \text{pec}(\theta_x, \theta_y) d\sigma, \quad (16.51c)$$

$$= 2\sigma_{\max} \text{pec}(\theta_x, \theta_y). \quad (16.51d)$$

Pseudo-ellipse 1st degree frame order The 1st degree frame order matrix with tensor rank-2 consists of the following elements

$$\mathbb{M}_{00} = \frac{\text{sinc } \sigma_{\max}}{2 \text{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} (\cos^2 \phi \sin^2 \theta_{\max} - 2 \sin^2 \phi \cos \theta_{\max}) d\phi \right], \quad (16.52a)$$

$$\mathbb{M}_{11} = \frac{\text{sinc } \sigma_{\max}}{2 \text{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} (\sin^2 \phi \sin^2 \theta_{\max} - 2 \cos^2 \phi \cos \theta_{\max}) d\phi \right], \quad (16.52b)$$

$$\mathbb{M}_{22} = \frac{1}{2\sigma_{\max} \text{pec}(\theta_x, \theta_y)} \int_{-\pi}^{\pi} \sin^2 \theta_{\max} d\phi. \quad (16.52c)$$

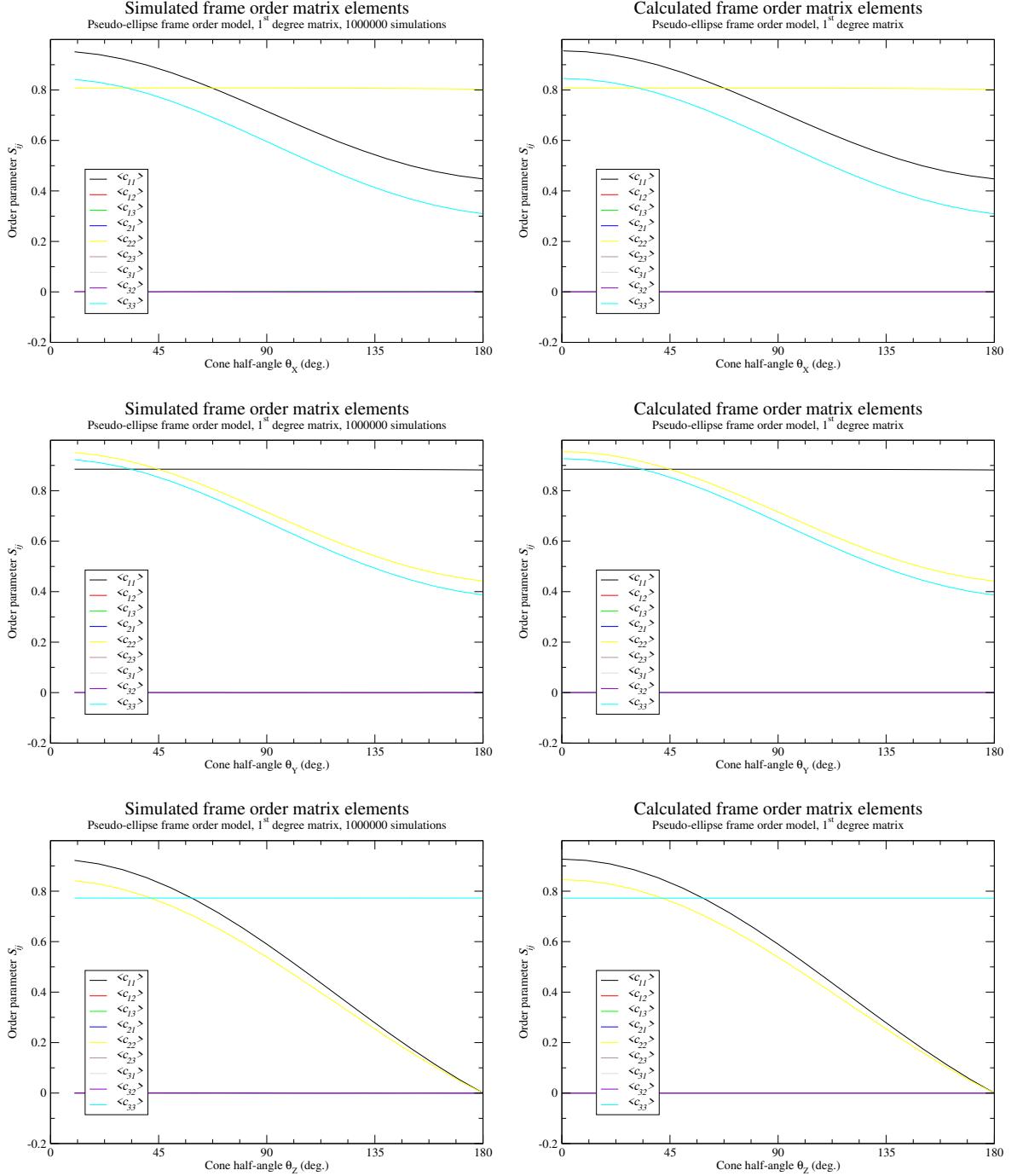


Figure 16.15: The pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x , θ_Y to the cone opening half-angle θ_y and θ_Z to torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$, $\theta_y = 3\pi/8$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated θ_X and θ_Y graphs is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

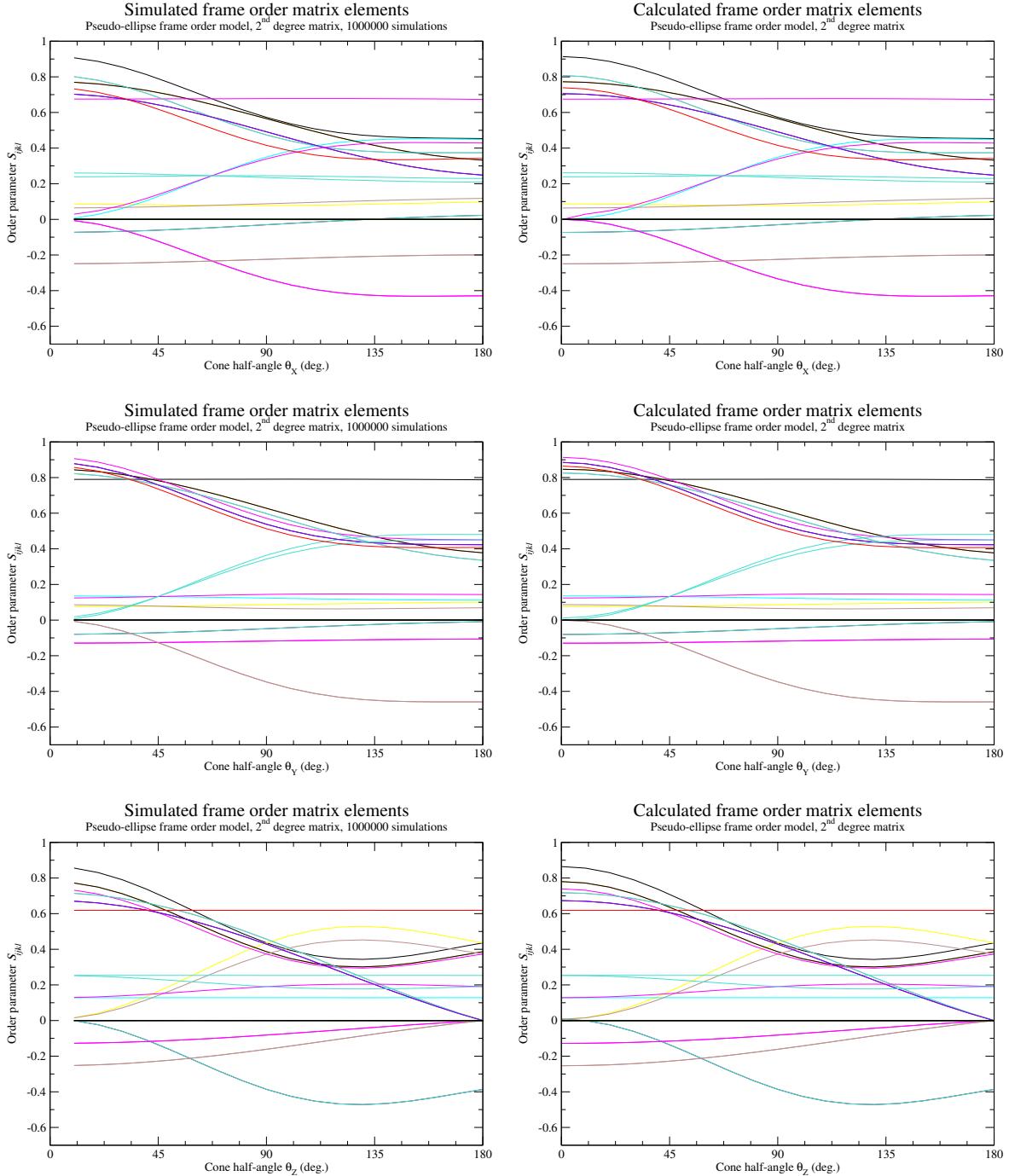


Figure 16.16: The pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x , θ_Y to the cone opening half-angle θ_y and θ_Z to torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$, $\theta_y = 3\pi/8$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated θ_X and θ_Y graphs is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

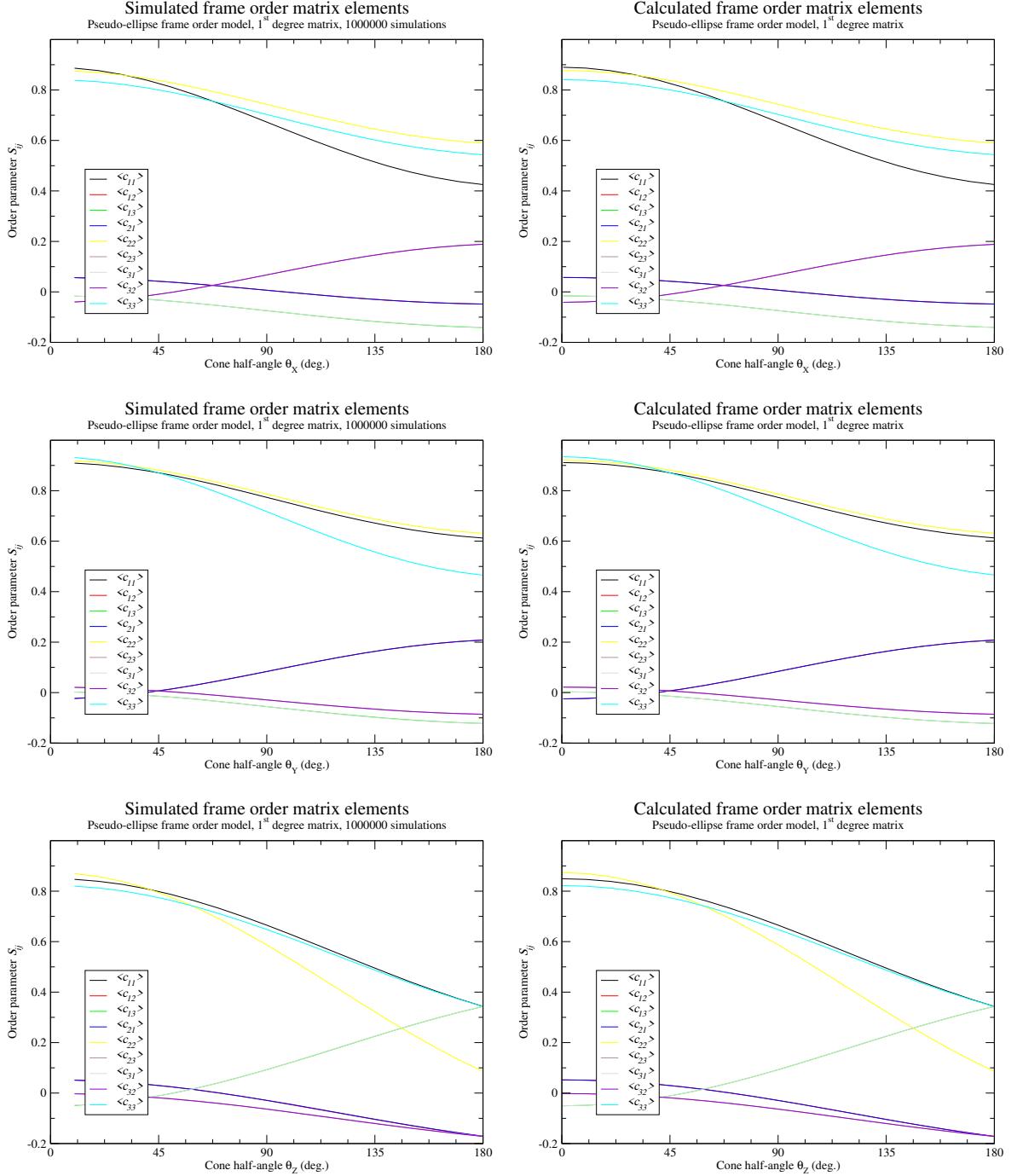


Figure 16.17: The pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x , θ_Y to the cone opening half-angle θ_y and θ_Z to torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$, $\theta_y = 3\pi/8$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated θ_X and θ_Y graphs is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

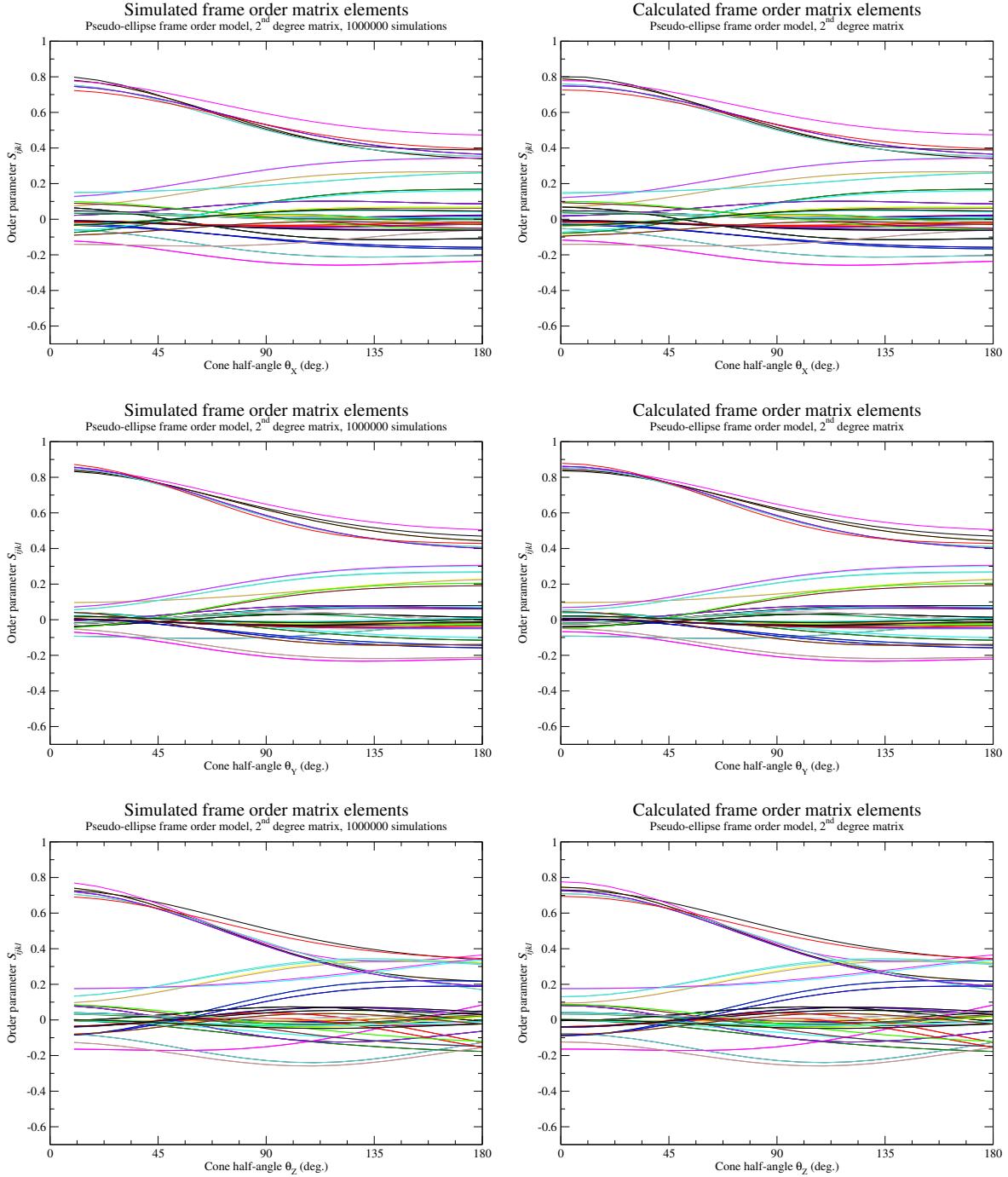


Figure 16.18: The pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x , θ_Y to the cone opening half-angle θ_y and θ_Z to torsion half-angle σ_{\max} . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$, $\theta_y = 3\pi/8$ or $\sigma_{\max} = \pi/6$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated θ_X and θ_Y graphs is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

As the trigonometric functions of θ_{\max} cannot be integrated, these components must be numerically integrated.

Pseudo-ellipse 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\begin{aligned} \mathbb{M}_{00} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[4\pi (\operatorname{sinc}(2\sigma_{\max}) + 2) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 \sin^2 \theta_{\max} \cos^2 \phi \left((2 \cos^2 \phi - 1) \cos \theta_{\max} \right. \right. \right. \\ & \left. \left. \left. + 6 \sin^2 \phi \right) - 2 \cos \theta_{\max} \left(2 \cos^2 \phi (4 \cos^2 \phi - 5) + 3 \right) \right) \right. \\ & \left. + 2 \cos^2 \phi \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) - 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53a)$$

$$\begin{aligned} \mathbb{M}_{11} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[4\pi \operatorname{sinc}(2\sigma_{\max}) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(\sin^2 \theta_{\max} \left(4 \cos^2 \phi \sin^2 \phi (\cos \theta_{\max} \right. \right. \right. \\ & \left. \left. \left. - 3) + 3 \right) - 16 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} \right) \right. \\ & \left. + 3 \sin^2 \theta_{\max} d\phi \right], \end{aligned} \quad (16.53b)$$

$$\mathbb{M}_{22} = \frac{\operatorname{sinc}(\sigma_{\max})}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[5\pi - \int_{-\pi}^{\pi} 2 \cos^2 \phi \cos^3 \theta_{\max} + 3 \sin^2 \phi \cos^2 \theta_{\max} d\phi \right], \quad (16.53c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.53d)$$

$$\begin{aligned} \mathbb{M}_{44} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[4\pi (\operatorname{sinc}(2\sigma_{\max}) + 2) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 \sin^2 \theta_{\max} \sin^2 \phi \left((2 \sin^2 \phi - 1) \cos \theta_{\max} \right. \right. \right. \\ & \left. \left. \left. + 6 \cos^2 \phi \right) - 2 \cos \theta_{\max} \left(2 \sin^2 \phi (4 \sin^2 \phi - 5) + 3 \right) \right) \right. \\ & \left. + 2 \sin^2 \phi \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) - 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53e)$$

$$\mathbb{M}_{55} = \frac{\operatorname{sinc}(\sigma_{\max})}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[5\pi - \int_{-\pi}^{\pi} 2 \sin^2 \phi \cos^3 \theta_{\max} + 3 \cos^2 \phi \cos^2 \theta_{\max} d\phi \right], \quad (16.53f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.53g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{55}, \quad (16.53\text{h})$$

$$\mathbb{M}_{88} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi - \int_{-\pi}^{\pi} \cos^3 \theta_{\max} d\phi \right], \quad (16.53\text{i})$$

$$\begin{aligned} \mathbb{M}_{04} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[4\pi (2 - \operatorname{sinc}(2\sigma_{\max})) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 \sin^2 \theta_{\max} \cos^2 \phi \left((2 \sin^2 \phi - 1) \cos \theta_{\max} \right. \right. \right. \\ & \left. \left. \left. - 6 \sin^2 \phi \right) + 2 \cos \theta_{\max} \left(2 \cos^2 \phi (4 \cos^2 \phi - 5) + 3 \right) \right) \right. \\ & \left. + 2 \cos^2 \phi \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) - 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53\text{j})$$

$$\begin{aligned} \mathbb{M}_{40} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[4\pi (2 - \operatorname{sinc}(2\sigma_{\max})) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 \sin^2 \theta_{\max} \sin^2 \phi \left((2 \cos^2 \phi - 1) \cos \theta_{\max} \right. \right. \right. \\ & \left. \left. \left. - 6 \cos^2 \phi \right) + 2 \cos \theta_{\max} \left(2 \sin^2 \phi (4 \sin^2 \phi - 5) + 3 \right) \right) \right. \\ & \left. + 2 \sin^2 \phi \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) - 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53\text{k})$$

$$\mathbb{M}_{08} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi - \int_{-\pi}^{\pi} \cos \theta_{\max} \cos^2 \phi (\sin^2 \theta_{\max} + 2) d\phi \right], \quad (16.53\text{l})$$

$$\begin{aligned} \mathbb{M}_{80} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[8\pi + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 (1 - 2 \cos^2 \phi) \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) \right) \right. \\ & \left. + 2 \cos^3 \theta_{\max} - 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53\text{m})$$

$$\mathbb{M}_{48} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi - \int_{-\pi}^{\pi} \cos \theta_{\max} \sin^2 \phi (\sin^2 \theta_{\max} + 2) d\phi \right], \quad (16.53\text{n})$$

$$\begin{aligned} \mathbb{M}_{84} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} & \left[8\pi - \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(2 (1 - 2 \cos^2 \phi) \cos \theta_{\max} (\sin^2 \theta_{\max} + 2) \right) \right. \\ & \left. - 2 \cos^3 \theta_{\max} + 6 \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.53\text{o})$$

$$\mathbb{M}_{13} = \frac{1}{12 \operatorname{pec}(\theta_x, \theta_y)} \left[4\pi \operatorname{sinc}(2\sigma_{\max}) + \int_{-\pi}^{\pi} \operatorname{sinc}(2\sigma_{\max}) \left(\sin^2 \theta_{\max} (4 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} - 12 \cos^2 \phi \sin^2 \phi + 3) - 16 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} \right) - 3 \sin^2 \theta_{\max} d\phi \right], \quad (16.53p)$$

$$\mathbb{M}_{31} = \mathbb{M}_{13}, \quad (16.53q)$$

$$\mathbb{M}_{26} = -\frac{\operatorname{sinc}(\sigma_{\max})}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \cos^2 \phi (\cos^3 \theta_{\max} - 3 \cos \theta_{\max}) d\phi \right], \quad (16.53r)$$

$$\mathbb{M}_{62} = \mathbb{M}_{26}, \quad (16.53s)$$

$$\mathbb{M}_{57} = -\frac{\operatorname{sinc}(\sigma_{\max})}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \sin^2 \phi (\cos^3 \theta_{\max} - 3 \cos \theta_{\max}) d\phi \right], \quad (16.53t)$$

$$\mathbb{M}_{75} = \mathbb{M}_{57}. \quad (16.53u)$$

As the trigonometric functions of θ_{\max} cannot be integrated, these components must be numerically integrated. All other frame order matrix elements can be numerically shown to be zero.

Pseudo-ellipse frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.52 and 16.53 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.15 and $\mathbb{M}^{(2)}$ in figure 16.16. The out-of-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.17 and $\mathbb{M}^{(2)}$ in figure 16.18.

16.10 Torsionless pseudo-ellipse frame order model

The first simplification of the pseudo-ellipse frame order model, which can be very useful for restricted motion systems such as CaM complexed with target peptides, would be to have no torsional motions.

16.10.1 Torsionless pseudo-ellipse parameterisation

This model is the pseudo-ellipse model with the torsion angle set to zero, $\sigma_{\max} = 0$. The model parameters are therefore

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.54a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\alpha, E_\beta, E_\gamma\} + \{p_x, p_y, p_z\} + \{\theta_x, \theta_y\}, \quad (16.54b)$$

where P_i are the average domain position translations and rotations, E_i are the Euler angles defining the motional eigenframe, p_i are the coordinates of the pivot point, and θ_x and θ_y are the maximum cone opening half-angles.

16.10.2 Torsionless pseudo-ellipse equations

Torsionless pseudo-ellipse rotation matrices

Setting the torsion angle σ to zero in the full torsion-tilt rotation matrix of equation 12.74c, the matrix becomes

$$R(\theta, \phi) = \begin{pmatrix} \cos^2 \phi \cos \theta + \sin^2 \phi & \cos \phi \sin \phi \cos \theta - \cos \phi \sin \phi & \cos \phi \sin \theta \\ \cos \phi \sin \phi \cos \theta - \cos \phi \sin \phi & \sin^2 \phi \cos \theta + \cos^2 \phi & \sin \phi \sin \theta \\ -\cos \phi \sin \theta & -\sin \phi \sin \theta & \cos \theta \end{pmatrix}. \quad (16.55)$$

Torsionless pseudo-ellipse frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS / \int_S dS, \quad (16.56a)$$

$$= \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi / \int_S dS. \quad (16.56b)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi, \quad (16.57a)$$

$$= \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi, \quad (16.57b)$$

$$= \text{pec}(\theta_x, \theta_y). \quad (16.57c)$$

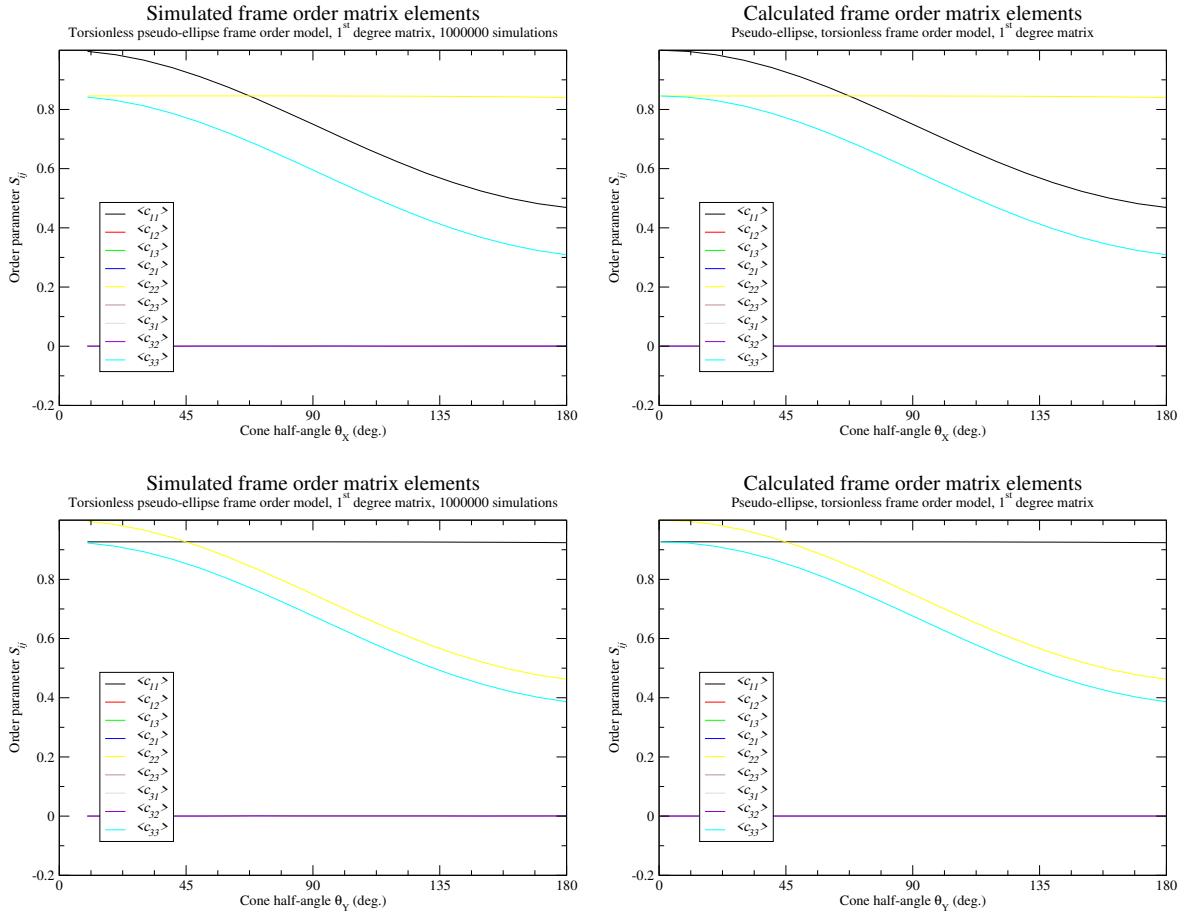


Figure 16.19: The torsionless pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

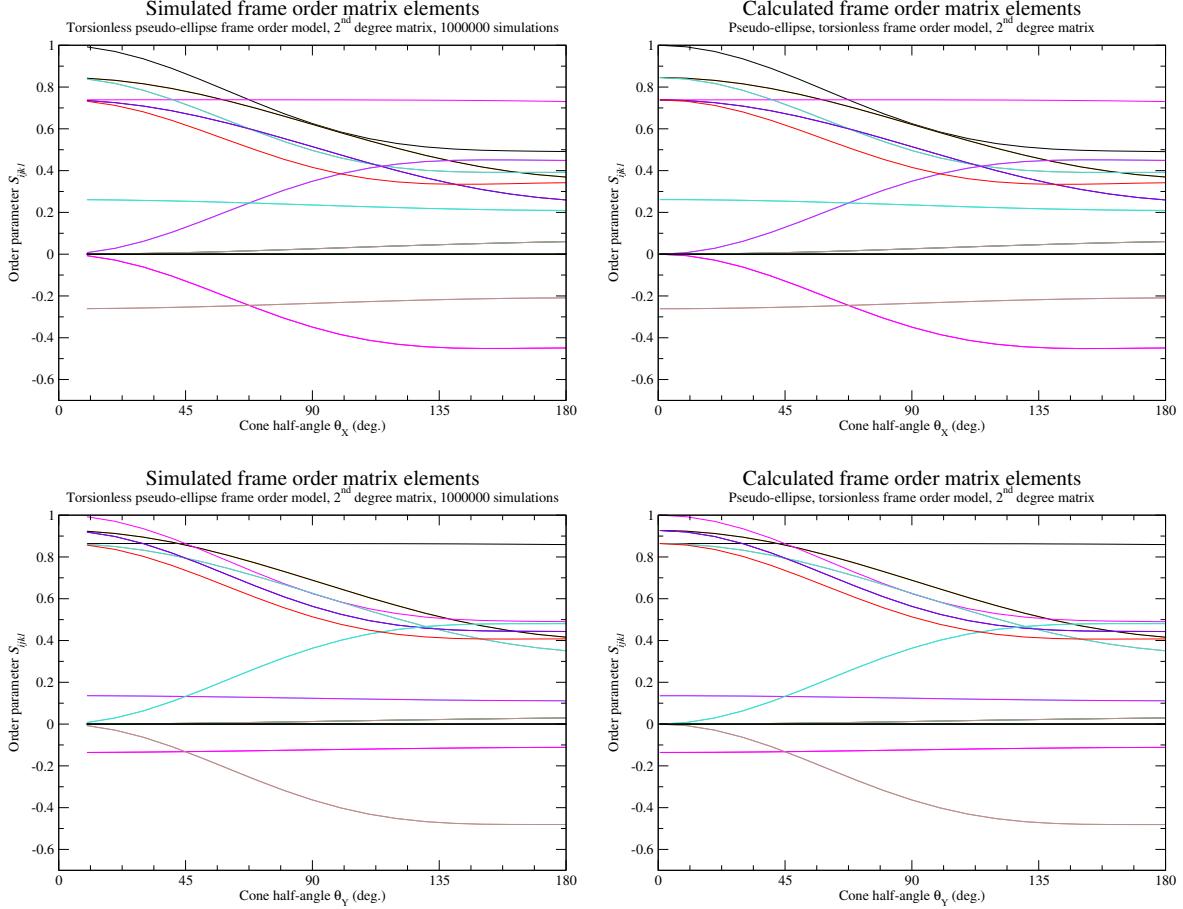


Figure 16.20: The torsionless pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_x corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

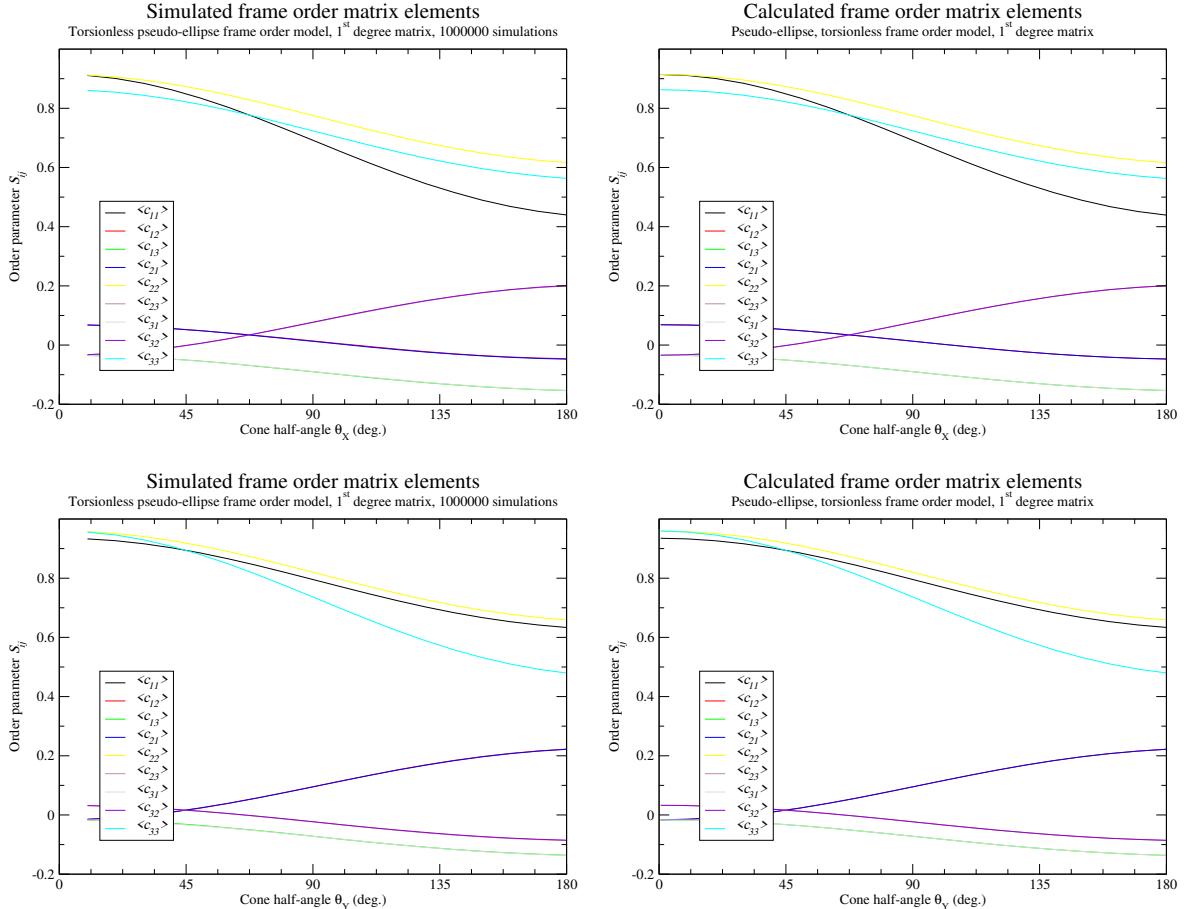


Figure 16.21: The torsionless pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

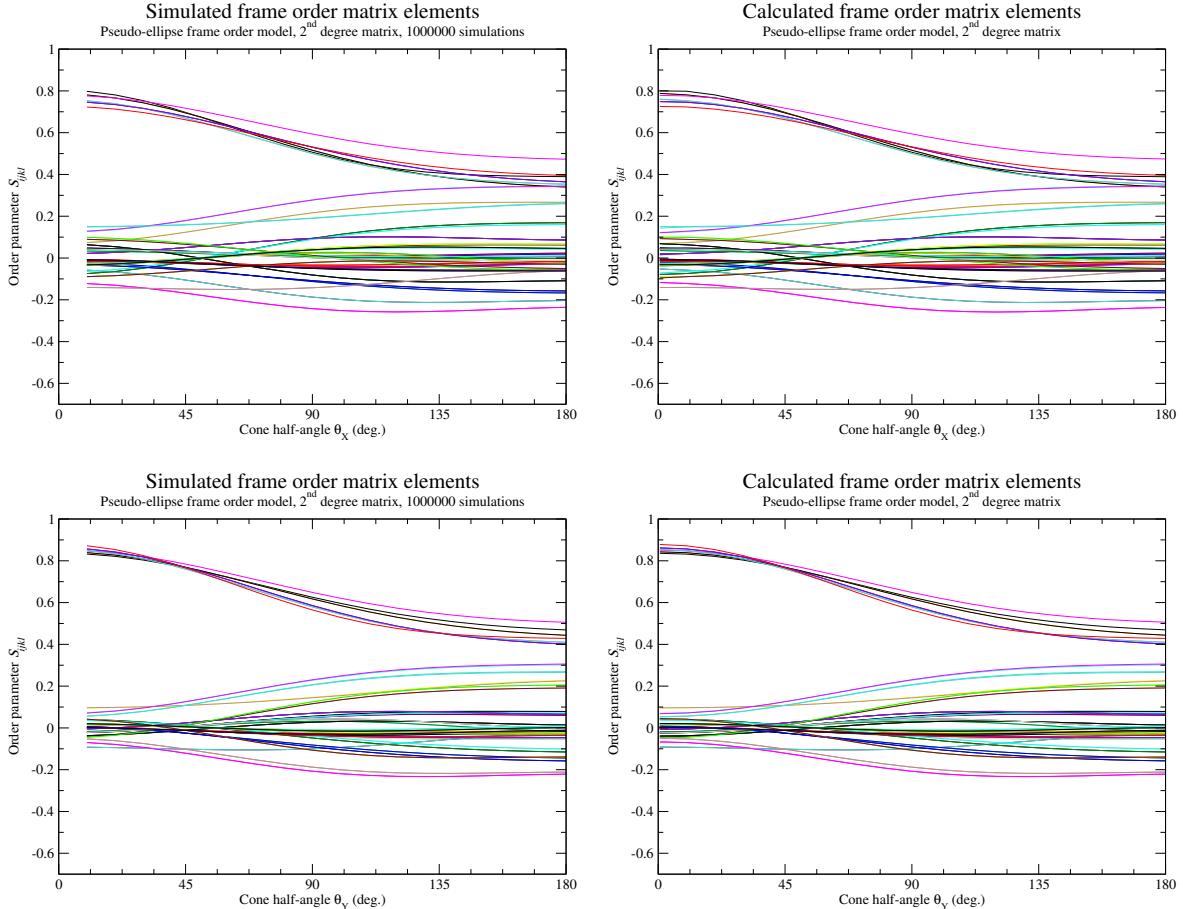


Figure 16.22: The torsionless pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

Torsionless pseudo-ellipse 1st degree frame order The 1st degree frame order matrix with tensor rank-2 consists of the following elements

$$\mathbb{M}_{00} = \frac{1}{2 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} (\cos^2 \phi \sin^2 \theta_{\max} - 2 \sin^2 \phi \cos \theta_{\max}) d\phi \right], \quad (16.58a)$$

$$\mathbb{M}_{11} = \frac{1}{2 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} (\sin^2 \phi \sin^2 \theta_{\max} - 2 \cos^2 \phi \cos \theta_{\max}) d\phi \right], \quad (16.58b)$$

$$\mathbb{M}_{22} = \frac{1}{2 \operatorname{pec}(\theta_x, \theta_y)} \int_{-\pi}^{\pi} \sin^2 \theta_{\max} d\phi. \quad (16.58c)$$

As the trigonometric functions of θ_{\max} cannot be symbolically integrated, these components must be numerically integrated.

Torsionless pseudo-ellipse 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\begin{aligned} \mathbb{M}_{00} = & \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[3\pi + \int_{-\pi}^{\pi} (\cos^4 \phi \cos \theta_{\max} + 3 \cos^2 \phi \sin^2 \phi) \sin^2 \theta_{\max} \right. \\ & \left. - (3 \sin^4 \phi + \cos^4 \phi) \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.59a)$$

$$\begin{aligned} \mathbb{M}_{11} = & \frac{1}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} (2 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} + 3 \sin^4 \phi + 3 \cos^4 \phi) \sin^2 \theta_{\max} \right. \\ & \left. - 8 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.59b)$$

$$\mathbb{M}_{22} = \frac{1}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[5\pi - \int_{-\pi}^{\pi} 2 \cos^2 \phi \cos^3 \theta_{\max} + 3 \sin^2 \phi \cos^2 \theta_{\max} d\phi \right], \quad (16.59c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.59d)$$

$$\begin{aligned} \mathbb{M}_{44} = & \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[3\pi + \int_{-\pi}^{\pi} (\sin^4 \phi \cos \theta_{\max} + 3 \cos^2 \phi \sin^2 \phi) \sin^2 \theta_{\max} \right. \\ & \left. - (\sin^4 \phi + 3 \cos^4 \phi) \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.59e)$$

$$\mathbb{M}_{55} = \frac{1}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[5\pi - \int_{-\pi}^{\pi} 2 \sin^2 \phi \cos^3 \theta_{\max} + 3 \cos^2 \phi \cos^2 \theta_{\max} d\phi \right], \quad (16.59f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.59g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{55}, \quad (16.59h)$$

$$\mathbb{M}_{88} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi - \int_{-\pi}^{\pi} \cos^3 \theta_{\max} d\phi \right], \quad (16.59i)$$

$$\begin{aligned} \mathbb{M}_{04} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} & \left[\pi + \int_{-\pi}^{\pi} (\cos^2 \phi \sin^2 \phi \cos \theta_{\max} - 3 \cos^2 \phi \sin^2 \phi) \sin^2 \theta_{\max} \right. \\ & \left. - 4 \cos^2 \phi \sin^2 \phi \cos \theta_{\max} d\phi \right], \end{aligned} \quad (16.59j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{04}, \quad (16.59k)$$

$$\mathbb{M}_{08} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \cos^2 \phi \cos^3 \theta_{\max} - 3 \cos^2 \phi \cos \theta_{\max} d\phi \right], \quad (16.59l)$$

$$\mathbb{M}_{80} = \mathbb{M}_{08}, \quad (16.59m)$$

$$\mathbb{M}_{48} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \sin^2 \phi \cos^3 \theta_{\max} - 3 \sin^2 \phi \cos \theta_{\max} d\phi \right], \quad (16.59n)$$

$$\mathbb{M}_{84} = \mathbb{M}_{48}, \quad (16.59o)$$

$$\mathbb{M}_{13} = \mathbb{M}_{04}, \quad (16.59p)$$

$$\mathbb{M}_{31} = \mathbb{M}_{04}, \quad (16.59q)$$

$$\mathbb{M}_{26} = -\mathbb{M}_{80}, \quad (16.59r)$$

$$\mathbb{M}_{62} = -\mathbb{M}_{80}, \quad (16.59s)$$

$$\mathbb{M}_{57} = -\mathbb{M}_{48}, \quad (16.59t)$$

$$\mathbb{M}_{75} = -\mathbb{M}_{48}. \quad (16.59u)$$

Torsionless pseudo-ellipse frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.58 and 16.59 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.19 and $\mathbb{M}^{(2)}$ in figure 16.20. The out-of-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.21 and $\mathbb{M}^{(2)}$ in figure 16.22.

16.11 Free rotor pseudo-ellipse frame order model

16.11.1 Free rotor pseudo-ellipse parameterisation

The free rotor model is the pseudo-ellipse model with the torsion angle restriction absent. Its parameters are

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E} + \mathfrak{p}_1 + \mathfrak{S}, \quad (16.60\text{a})$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\alpha, E_\beta, E_\gamma\} + \{p_x, p_y, p_z\} + \{\theta_x, \theta_y\}, \quad (16.60\text{b})$$

where P_i are the average domain position translations and rotations, E_i are the Euler angles defining the motional eigenframe, p_i are the coordinates of the pivot point, and θ_x and θ_y are the maximum cone opening half-angles.

16.11.2 Free rotor pseudo-ellipse equations

Free rotor pseudo-ellipse rotation matrices

The rotation matrix is the full torsion-tilt rotation matrix of equation 12.74c on page 252.

Free rotor pseudo-ellipse frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R^{\otimes n} dS \Big/ \int_S dS, \quad (16.61\text{a})$$

$$= \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} R^{\otimes n} \sin \theta d\theta d\phi d\sigma \Big/ \int_S dS. \quad (16.61\text{b})$$

The surface normalisation factor is

$$\int_S dS = \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_0^{\theta_{\max}} \sin \theta d\theta d\phi d\sigma, \quad (16.62\text{a})$$

$$= \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} (1 - \cos \theta_{\max}) d\phi d\sigma, \quad (16.62\text{b})$$

$$= \int_{-\pi}^{\pi} \text{pec}(\theta_x, \theta_y) d\sigma, \quad (16.62\text{c})$$

$$= 2\pi \text{pec}(\theta_x, \theta_y). \quad (16.62\text{d})$$

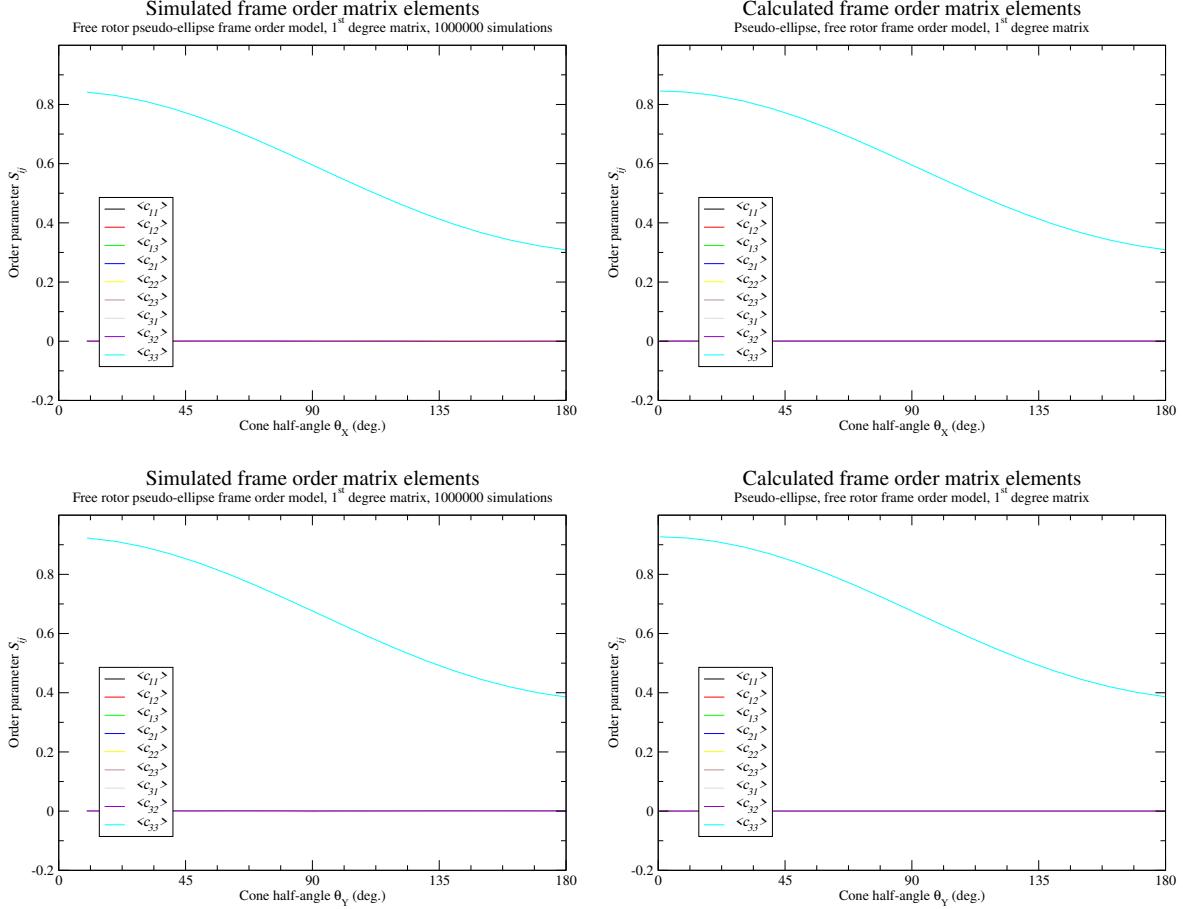


Figure 16.23: The free rotor pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

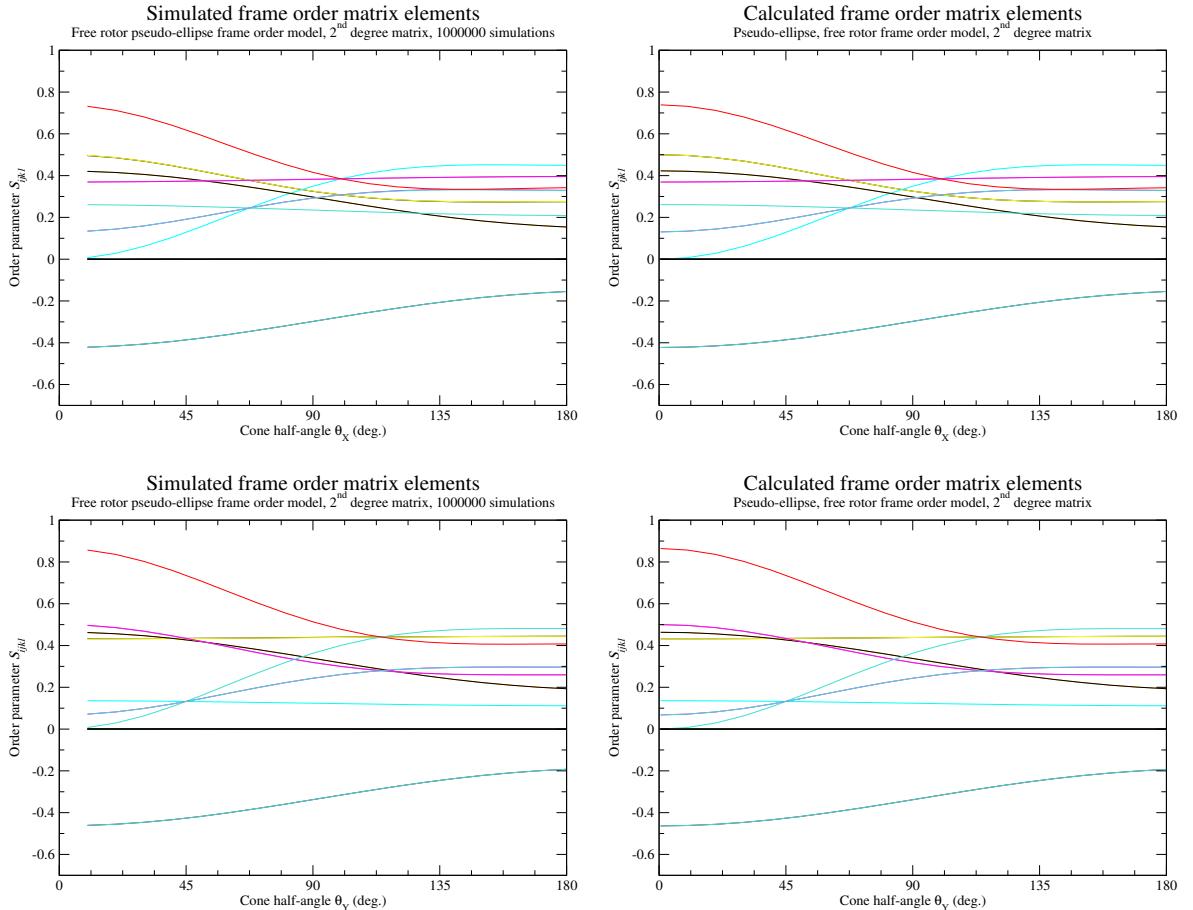


Figure 16.24: The free rotor pseudo-ellipse model simulated and calculated in-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

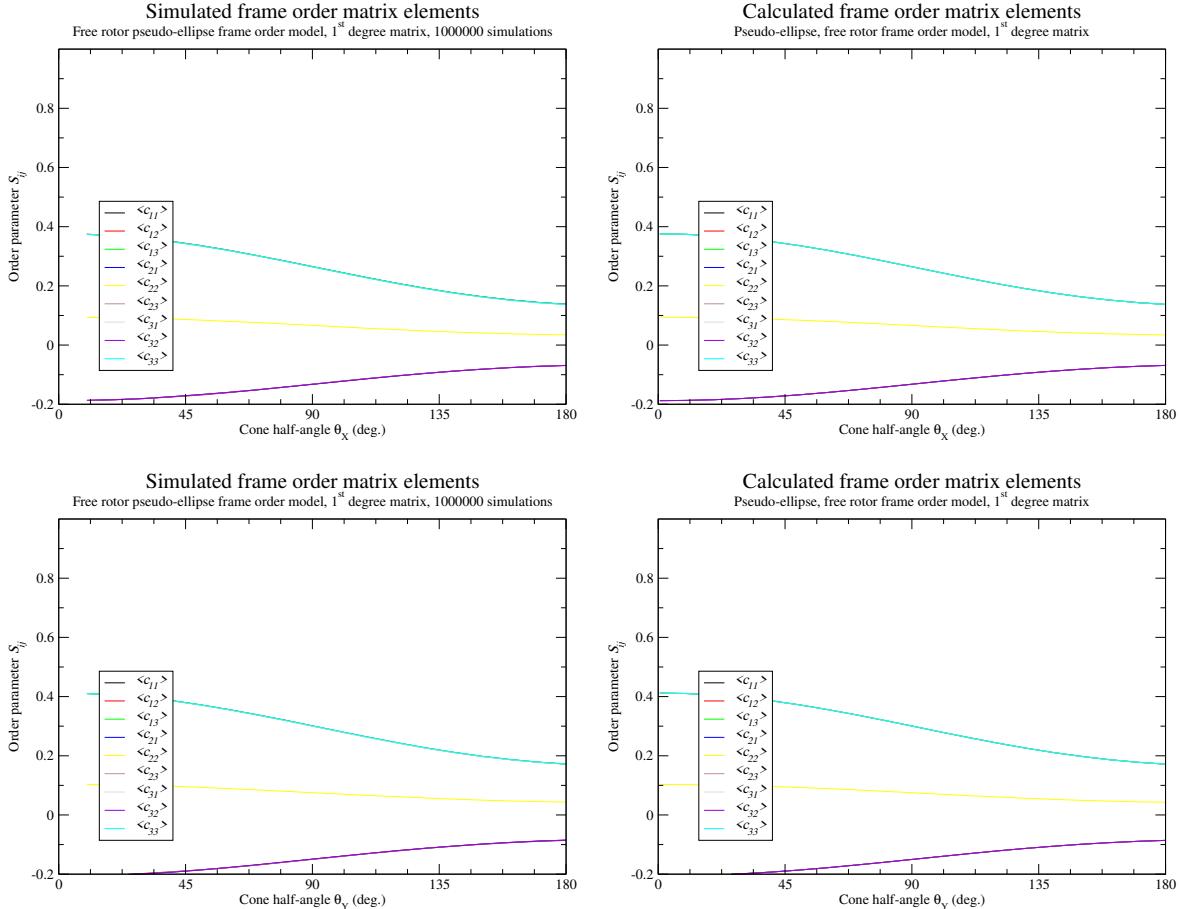


Figure 16.25: The free rotor pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

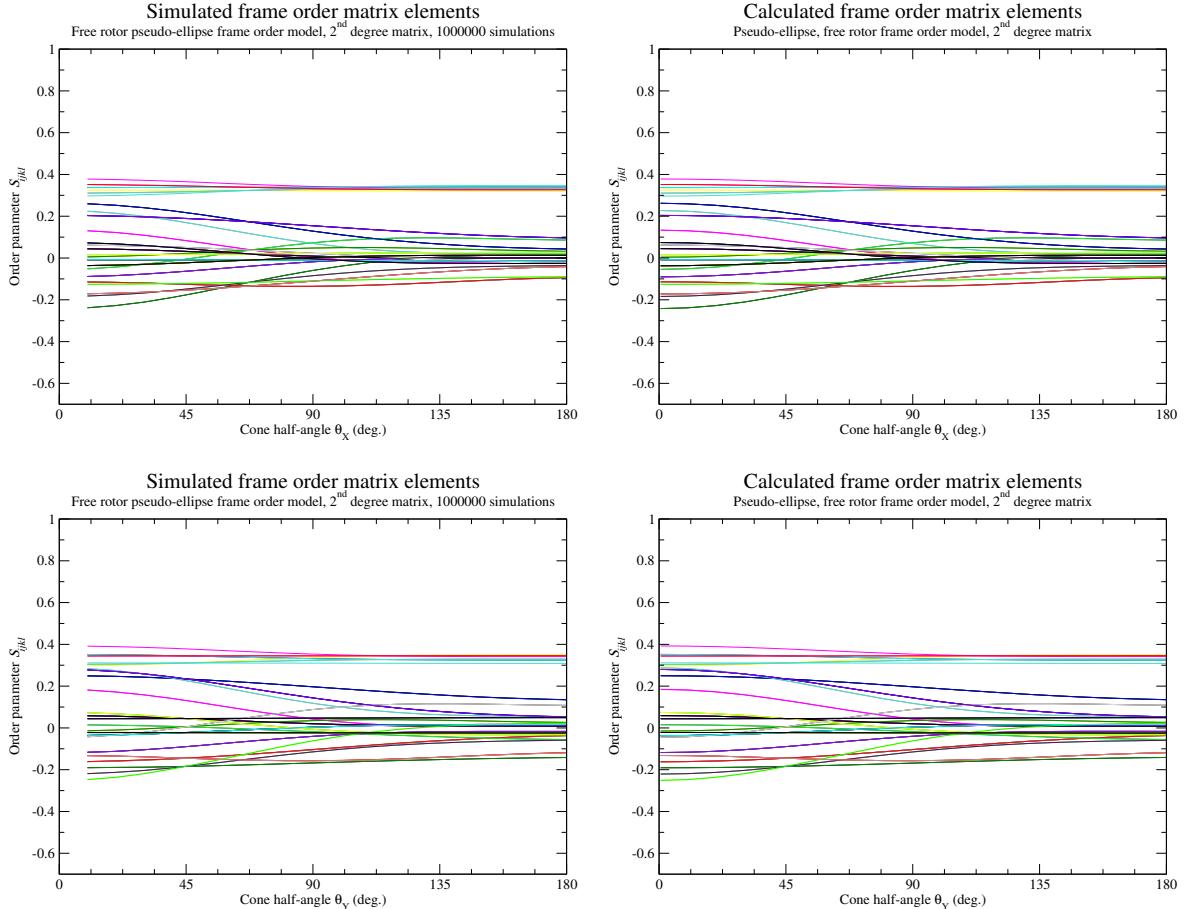


Figure 16.26: The free rotor pseudo-ellipse model simulated and calculated out-of-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the cone opening half-angle θ_x and θ_Y to the cone opening half-angle θ_y . When the half-angle is not varied, the angle is fixed to either $\theta_x = \pi/4$ or $\theta_y = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees. The first angle for the calculated elements is set to 0.01 degrees as a pseudo-ellipse cone opening angle of 0.0 cannot be correctly handled by the numerical integration.

Free rotor pseudo-ellipse 1st degree frame order The 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)} = \int_S R^{\otimes 1} dS / \int_S dS, \quad (16.63a)$$

$$= \int_S R dS / 2\pi \text{pec}(\theta_x, \theta_y), \quad (16.63b)$$

$$= \frac{1}{2 \text{pec}(\theta_x, \theta_y)} \int_{-\pi}^{\pi} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \sin^2 \theta_{\max} \end{pmatrix} d\phi. \quad (16.63c)$$

Free rotor pseudo-ellipse 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\mathbb{M}_{00} = \frac{1}{6 \text{pec}(\theta_x, \theta_y)} \left[4\pi - \int_{-\pi}^{\pi} \cos^2 \phi \cos^3 \theta_{\max} + 3 \sin^2 \phi \cos \theta_{\max} d\phi \right], \quad (16.64a)$$

$$\mathbb{M}_{11} = \frac{1}{4 \text{pec}(\theta_x, \theta_y)} \int_{-\pi}^{\pi} \sin^2 \theta_{\max} d\phi, \quad (16.64b)$$

$$\mathbb{M}_{22} = 0, \quad (16.64c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.64d)$$

$$\mathbb{M}_{44} = \frac{1}{6 \text{pec}(\theta_x, \theta_y)} \left[4\pi - \int_{-\pi}^{\pi} \sin^2 \phi \cos^3 \theta_{\max} + 3 \cos^2 \phi \cos \theta_{\max} d\phi \right], \quad (16.64e)$$

$$\mathbb{M}_{55} = 0, \quad (16.64f)$$

$$\mathbb{M}_{66} = 0, \quad (16.64g)$$

$$\mathbb{M}_{77} = 0, \quad (16.64h)$$

$$\mathbb{M}_{88} = \frac{1}{3 \text{pec}(\theta_x, \theta_y)} \left[2\pi - \int_{-\pi}^{\pi} \cos^3 \theta_{\max} d\phi \right], \quad (16.64i)$$

$$\mathbb{M}_{04} = \mathbb{M}_{00}, \quad (16.64j)$$

$$\mathbb{M}_{40} = \mathbb{M}_{44}, \quad (16.64k)$$

$$\mathbb{M}_{08} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \cos^2 \phi (\cos^3 \theta_{\max} - 3 \cos \theta_{\max}) d\phi \right], \quad (16.64l)$$

$$\mathbb{M}_{80} = \frac{1}{6 \operatorname{pec}(\theta_x, \theta_y)} \left[4\pi + \int_{-\pi}^{\pi} \cos^3 \theta_{\max} - 3 \cos \theta_{\max} d\phi \right], \quad (16.64m)$$

$$\mathbb{M}_{48} = \frac{1}{3 \operatorname{pec}(\theta_x, \theta_y)} \left[2\pi + \int_{-\pi}^{\pi} \sin^2 \phi (\cos^3 \theta_{\max} - 3 \cos \theta_{\max}) d\phi \right], \quad (16.64n)$$

$$\mathbb{M}_{84} = \mathbb{M}_{80}, \quad (16.64o)$$

$$\mathbb{M}_{13} = -\mathbb{M}_{11}, \quad (16.64p)$$

$$\mathbb{M}_{31} = -\mathbb{M}_{11}, \quad (16.64q)$$

$$\mathbb{M}_{26} = 0, \quad (16.64r)$$

$$\mathbb{M}_{62} = 0, \quad (16.64s)$$

$$\mathbb{M}_{57} = 0, \quad (16.64t)$$

$$\mathbb{M}_{75} = 0. \quad (16.64u)$$

Free rotor pseudo-ellipse frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.63 and 16.64 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.23 and $\mathbb{M}^{(2)}$ in figure 16.24. The out-of-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.25 and $\mathbb{M}^{(2)}$ in figure 16.26.

16.12 Double rotor frame order model

The eigenframe of the motion of the double rotor model is characterised by two pivot points and two rotor axes. To simplify the modelling, the two axes are assumed to be orthogonal. Due to the nature of the RDC and PCS data used, it may not be possible to distinguish deviations from orthogonality from the noise.

16.12.1 Double rotor parameterisation

Assuming the axes are orthogonal for the model, the size of the set of non-redundant parameters is 15. To eliminate the redundant parameters, the geometry of the system can be used to construct a 3D eigenframe of the motion consisting of three Euler angles:

x-axis This axis of the eigensystem can be defined as a vector parallel to the 1st rotor axis.

y-axis This axis of the eigensystem can be defined as a vector parallel to the 2nd rotor axis.

z-axis This can be defined as a vector parallel to the line of shortest distance connecting the two rotor axes. As x and y are orthogonal by definition of the model, the line of shortest distance will be orthogonal to both x and y.

The two pivot points defining the position in space of the two rotor axes define the system. Using the above eigenframe, these can be parameterised using only four parameters:

1st pivot point This is defined using three coordinates and is located at the intersection of the 1st rotor axis and the line of shortest distance between the axes.

2nd pivot point This is defined as the intersection of the 2nd rotor axis and the line of shortest distance. Using the z-axis of the eigenframe and the 1st pivot, the 3D position can be defined as a simple displacement, p_d .

The set of all parameters of the system is therefore

$$\mathfrak{M} = \mathfrak{P} + \mathfrak{E} + \mathfrak{p}_1 + \mathfrak{p}_2 + \mathfrak{S}, \quad (16.65a)$$

$$= \{P_x, P_y, P_z, P_\alpha, P_\beta, P_\gamma\} + \{E_\alpha, E_\beta, E_\gamma\} + \{p_x, p_y, p_z\} + \{p_d\} + \{\sigma_{\max}, \sigma_{\max,2}\}, \quad (16.65b)$$

where P_i are the average domain position translations and rotations, E_i are the eigenframe Euler angles, p_i are the coordinates of the 1st pivot point, p_d is the displacement for the 2nd pivot point, and $\sigma_{\max,i}$ are the two torsion half-angles of the rotors.

16.12.2 Double rotor equations

The double rotor model consists of two standard rotations, the first about the x-axis and the second about the y-axis. Hence the frame order matrix is simply the integration over both torsion angles of the Kronecker product of the product of the R_x and R_y rotation matrices, divided by the surface area normalisation factor.

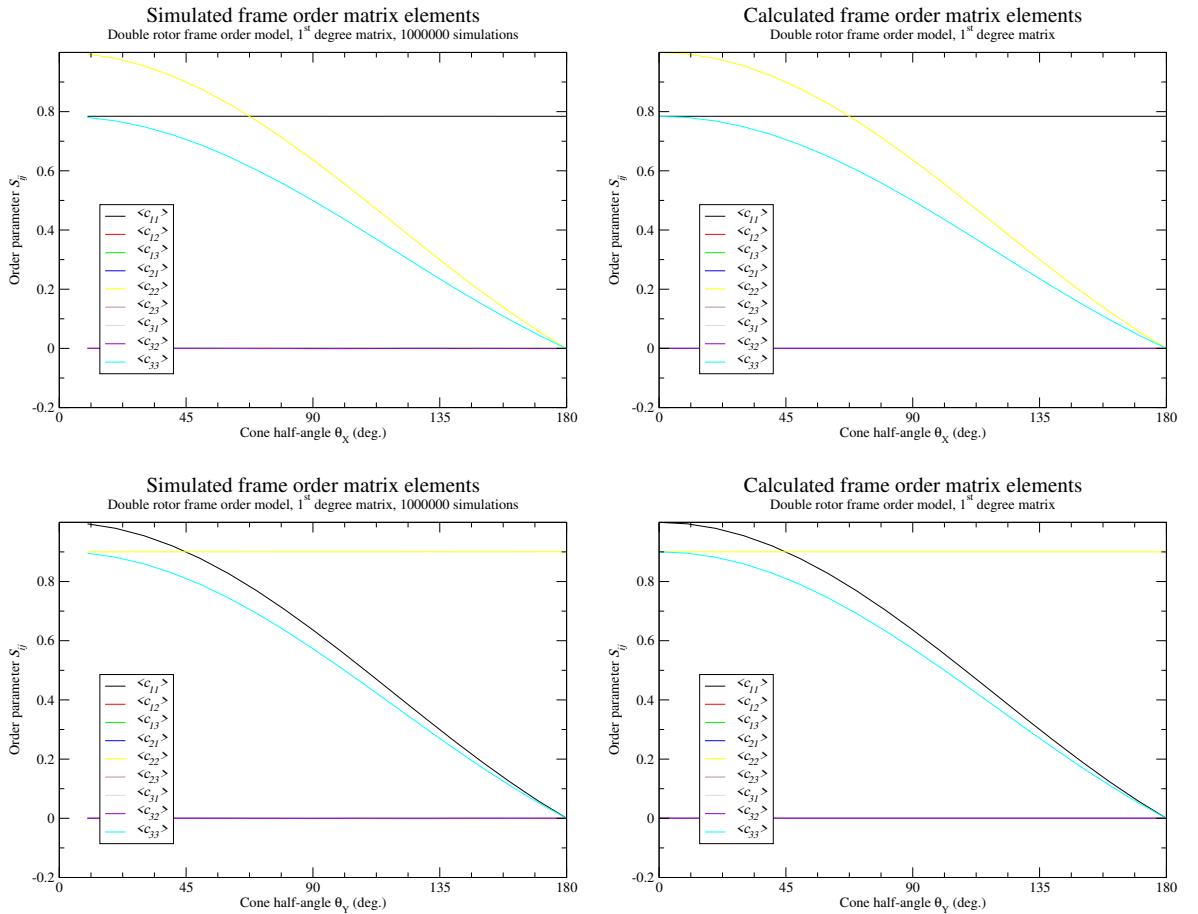


Figure 16.27: The double rotor model simulated and calculated in-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the torsion half-angle $\sigma_{\max,1}$ and θ_Y to the torsion half-angle $\sigma_{\max,2}$. When the half-angle is not varied, the angle is fixed to either $\sigma_{\max,1} = \pi/4$ or $\sigma_{\max,2} = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees.

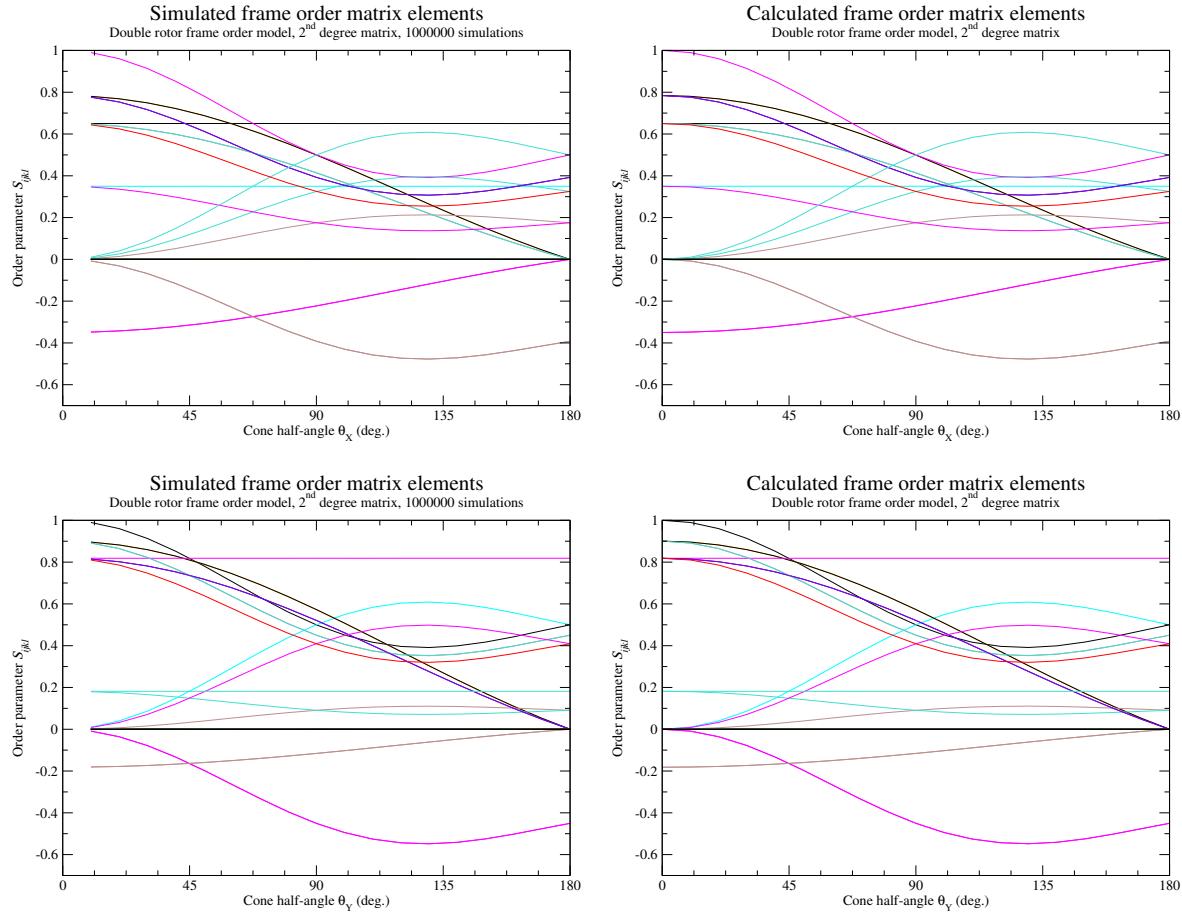


Figure 16.28: The double rotor model simulated and calculated in-frame $\mathbb{M}^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the torsion half-angle $\sigma_{\max,1}$ and θ_Y to the torsion half-angle $\sigma_{\max,2}$. When the half-angle is not varied, the angle is fixed to either $\sigma_{\max,1} = \pi/4$ or $\sigma_{\max,2} = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees.

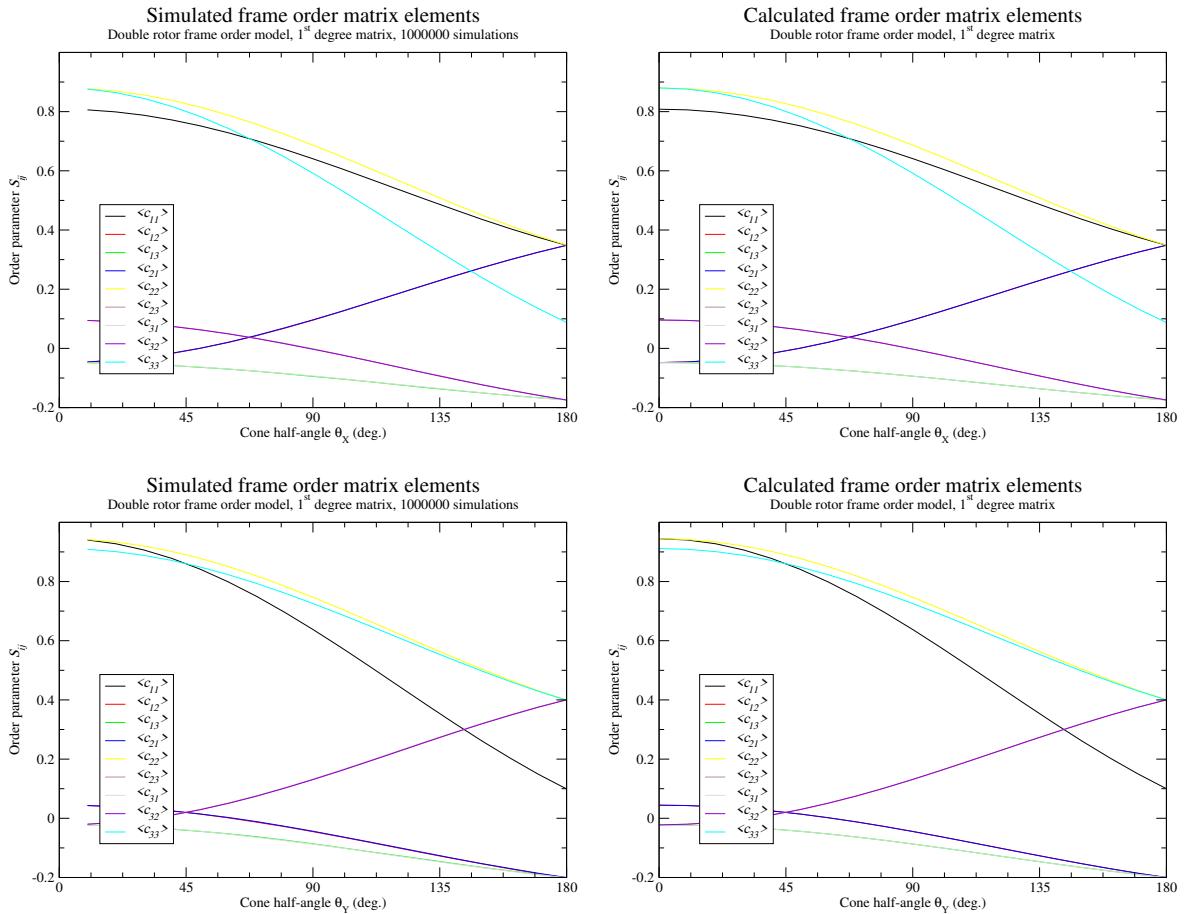


Figure 16.29: The double rotor model simulated and calculated out-of-frame $\mathbb{M}^{(1)}$ frame order matrix elements. In these plots, θ_X corresponds to the torsion half-angle $\sigma_{\max,1}$ and θ_Y to the torsion half-angle $\sigma_{\max,2}$. When the half-angle is not varied, the angle is fixed to either $\sigma_{\max,1} = \pi/4$ or $\sigma_{\max,2} = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees.

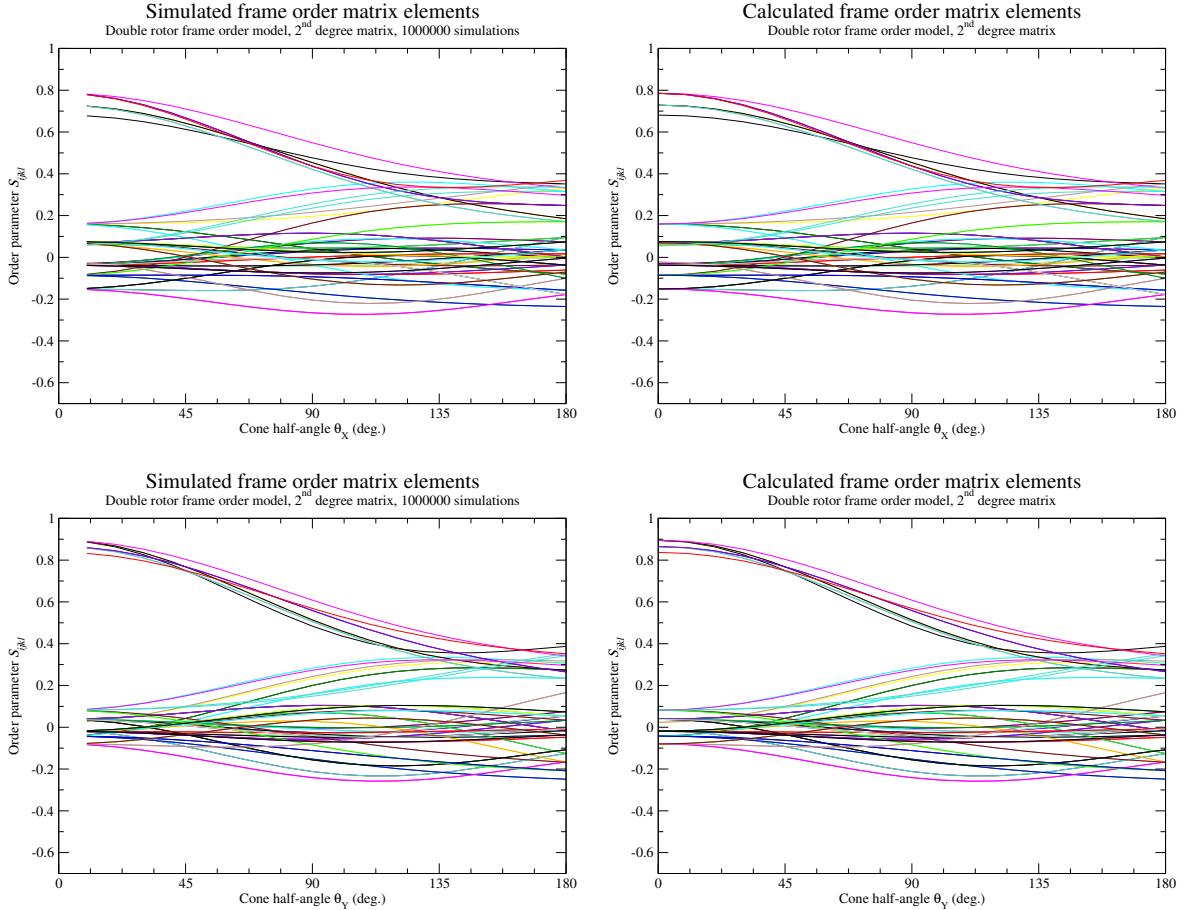


Figure 16.30: The double rotor model simulated and calculated out-of-frame $\aleph^{(2)}$ frame order matrix elements. In these plots, θ_X corresponds to the torsion half-angle $\sigma_{\max,1}$ and θ_Y to the torsion half-angle $\sigma_{\max,2}$. When the half-angle is not varied, the angle is fixed to either $\sigma_{\max,1} = \pi/4$ or $\sigma_{\max,2} = 3\pi/8$. Frame order matrix values have been calculated every 10 degrees.

Double rotor rotation matrices

The individual rotations are

$$R_y(\sigma_1) = \begin{pmatrix} \cos \sigma_1 & 0 & \sin \sigma_1 \\ 0 & 1 & 0 \\ -\sin \sigma_1 & 0 & \cos \sigma_1 \end{pmatrix}, \quad (16.66a)$$

$$R_x(\sigma_2) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \sigma_2 & -\sin \sigma_2 \\ 0 & \sin \sigma_2 & \cos \sigma_2 \end{pmatrix}. \quad (16.66b)$$

The full rotation is then

$$R(\sigma_1, \sigma_2) = R_x(\sigma_2) \cdot R_y(\sigma_1), \quad (16.67a)$$

$$= \begin{pmatrix} \cos \sigma_1 & 0 & \sin \sigma_1 \\ \sin \sigma_1 \sin \sigma_2 & \cos \sigma_2 & -\cos \sigma_1 \sin \sigma_2 \\ -\sin \sigma_1 \cos \sigma_2 & \sin \sigma_2 & \cos \sigma_1 \cos \sigma_2 \end{pmatrix}. \quad (16.67b)$$

Double rotor frame order matrix

The frame order matrix is

$$\mathbb{M}^{(n)} = \int_S R(\sigma_1, \sigma_2)^{\otimes n} dS / \int_S dS, \quad (16.68)$$

$$= \int_{-\sigma_{\max,2}}^{\sigma_{\max,2}} \int_{-\sigma_{\max,1}}^{\sigma_{\max,1}} R(\sigma_1, \sigma_2)^{\otimes n} d\sigma_{\max,1} d\sigma_{\max,2} / \int_S dS. \quad (16.69)$$

The surface normalisation factor is

$$\int_S dS = \int_{-\sigma_{\max,2}}^{\sigma_{\max,2}} \int_{-\sigma_{\max,1}}^{\sigma_{\max,1}} d\sigma_{\max,1} d\sigma_{\max,2}, \quad (16.70a)$$

$$= 2\sigma_{\max,1} \int_{-\sigma_{\max,2}}^{\sigma_{\max,2}} d\sigma_{\max,2}, \quad (16.70b)$$

$$= 4\sigma_{\max,1}\sigma_{\max,2}. \quad (16.70c)$$

Double rotor 1st degree frame order The un-normalised 1st degree frame order matrix with tensor rank-2 is

$$\mathbb{M}^{(1)'} = \int_S R(\sigma_1, \sigma_2)^{\otimes 1} dS, \quad (16.71a)$$

$$= \int_{-\sigma_{\max,2}}^{\sigma_{\max,2}} \int_{-\sigma_{\max,1}}^{\sigma_{\max,1}} R(\sigma_1, \sigma_2) d\sigma_{\max,1} d\sigma_{\max,2}, \quad (16.71b)$$

$$= \begin{pmatrix} 4\sin(\sigma_{\max,1})\sigma_{\max,2} & \cdot & \cdot \\ \cdot & 4\sin(\sigma_{\max,2})\sigma_{\max,1} & \cdot \\ \cdot & \cdot & 4\sin \sigma_{\max,1} \sin \sigma_{\max,2} \end{pmatrix}. \quad (16.71c)$$

After normalisation, the full frame order matrix is

$$\mathbb{M}^{(1)} = \begin{pmatrix} \text{sinc } \sigma_{\max,1} & & & & \\ & \ddots & & & \\ & & \text{sinc } \sigma_{\max,2} & & \\ & & & \ddots & \\ & & & & \text{sinc } \sigma_{\max,1} \text{ sinc } \sigma_{\max,2} \end{pmatrix}. \quad (16.72)$$

Double rotor 2nd degree frame order The 2nd degree frame order matrix with tensor rank-4 consists of the following elements, using Kronecker product double indices from 0 to 8

$$\mathbb{M}_{00} = \frac{1}{2}(\text{sinc}(2\sigma_{\max,1}) + 1), \quad (16.73a)$$

$$\mathbb{M}_{11} = \text{sinc } \sigma_{\max,1} \text{ sinc } \sigma_{\max,2}, \quad (16.73b)$$

$$\mathbb{M}_{22} = \frac{1}{2} \text{sinc } \sigma_{\max,2} (\text{sinc}(2\sigma_{\max,1}) + 1), \quad (16.73c)$$

$$\mathbb{M}_{33} = \mathbb{M}_{11}, \quad (16.73d)$$

$$\mathbb{M}_{44} = \frac{1}{2}(\text{sinc}(2\sigma_{\max,2}) + 1), \quad (16.73e)$$

$$\mathbb{M}_{55} = \frac{1}{2} \text{sinc } \sigma_{\max,1} (\text{sinc}(2\sigma_{\max,2}) + 1), \quad (16.73f)$$

$$\mathbb{M}_{66} = \mathbb{M}_{22}, \quad (16.73g)$$

$$\mathbb{M}_{77} = \mathbb{M}_{55}, \quad (16.73h)$$

$$\mathbb{M}_{88} = \frac{1}{4}(\text{sinc}(2\sigma_{\max,1}) + 1)(\text{sinc}(2\sigma_{\max,2}) + 1), \quad (16.73i)$$

$$\mathbb{M}_{04} = 0, \quad (16.73j)$$

$$\mathbb{M}_{40} = \frac{1}{4}(\text{sinc}(2\sigma_{\max,1}) - 1)(\text{sinc}(2\sigma_{\max,2}) - 1), \quad (16.73k)$$

$$\mathbb{M}_{08} = -\frac{1}{2}(\text{sinc}(2\sigma_{\max,1}) - 1), \quad (16.73l)$$

$$\mathbb{M}_{80} = -\frac{1}{4}(\text{sinc}(2\sigma_{\max,1}) - 1)(\text{sinc}(2\sigma_{\max,2}) + 1), \quad (16.73m)$$

$$\mathbb{M}_{48} = -\frac{1}{4}(\text{sinc}(2\sigma_{\max,1}) + 1)(\text{sinc}(2\sigma_{\max,2}) - 1), \quad (16.73n)$$

$$\mathbb{M}_{84} = -\frac{1}{2}(\text{sinc}(2\sigma_{\max,2}) - 1), \quad (16.73o)$$

$$\mathbb{M}_{13} = 0, \quad (16.73p)$$

$$\mathbb{M}_{31} = 0, \quad (16.73q)$$

$$\mathbb{M}_{26} = \frac{1}{2} \text{sinc } \sigma_{\max,2} (\text{sinc}(2\sigma_{\max,1}) - 1), \quad (16.73r)$$

$$\mathbb{M}_{62} = \mathbb{M}_{26}, \quad (16.73s)$$

$$\mathbb{M}_{57} = \frac{1}{2} \text{sinc } \sigma_{\max,1} (\text{sinc}(2\sigma_{\max,2}) - 1), \quad (16.73t)$$

$$\mathbb{M}_{75} = \mathbb{M}_{57}. \quad (16.73u)$$

Double rotor frame order matrix simulation and calculation The frame order matrix element simulation script from Section 16.2, page 375 was used to compare the implementation of equations 16.72 and 16.73 above. Frame order matrix $\mathbb{M}^{(1)}$ and $\mathbb{M}^{(2)}$ values were both simulated and calculated, both within and out of the motional eigenframe. The in-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.27 and $\mathbb{M}^{(2)}$ in figure 16.28. The out-of-frame $\mathbb{M}^{(1)}$ values are shown in figure 16.29 and $\mathbb{M}^{(2)}$ in figure 16.30.

Part V

Reference

Chapter 17

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)`.

17.1 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.

17.2 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.

17.2.1 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.

17.2.2 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.

17.2.3 Docstring sectioning

All other sections are created from the sectioning of the user function docstring.

```
relax> align_tensor.copy(tensor_from='Otting',
                           pipe_to='new')
```

17.2.4 align_tensor.copy



Synopsis

Copy alignment tensor data.

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')
```

Defaults

```
align_tensor.copy(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 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 source or destination data pipes are not supplied, then both will default to the current data pipe. Both the source and destination tensor IDs must be supplied.

Prompt 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')
```

17.2.5 align_tensor.delete**Synopsis**

Delete alignment tensor data from the relax data store.

Defaults

`align_tensor.delete(tensor=None)`

Keyword arguments

`tensor`: The alignment tensor identification string.

Description

This will delete the specified alignment tensor data from the current data pipe. If no tensor is specified, all tensors will be deleted.

17.2.6 align_tensor.display**Synopsis**

Display the alignment tensor information in full detail.

Defaults

`align_tensor.display(tensor=None)`

Keyword arguments

`tensor`: The alignment tensor identification string.

Description

This will show all information relating to the alignment tensor, including the different tensor forms:

Probability tensor.

Saupe order matrix.

Alignment tensor.

Magnetic susceptibility tensor.

All possible tensor parameters and information will also be shown (Eigensystem, GDO, Aa, Ar, \Re , eta, chi_ax, chi_rh, etc). The printout will be extensive.

If no tensor is specified, all tensors will be displayed.

17.2.7 align_tensor.fix



Synopsis

Fix all alignment tensors so that they do not change during optimisation.

Defaults

```
align_tensor.fix(id=None, fixed=True)
```

Keyword arguments

id: The alignment tensor identification string.

fixed: The flag specifying if the tensors should be fixed or variable.

Description

If the ID string is left unset, then all alignment tensors will be fixed.

17.2.8 align_tensor.init



Synopsis

Initialise an alignment tensor.

Defaults

```
align_tensor.init(tensor=None, align_id=None, domain=None, params=None, scale=1.0, angle_units='deg', param_types=2, errors=False)
```

Keyword arguments

tensor: The optional alignment tensor ID string, required if multiple tensors exist per alignment.

align_id: The alignment ID string that the tensor corresponds to.

domain: The optional domain ID string that the tensor corresponds to.

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

The tensor ID is only required if there are multiple unique tensors per alignment. An example is if internal domain motions cause multiple parts of the molecule to align differently. The tensor ID is optional and in the case of only a single tensor per alignment, the tensor can be identified using the alignment ID instead.

The alignment tensor parameters should be a tuple of floating point numbers (a list surrounded by round brackets). These correspond to the parameters of the tensor which can be specified by the parameter types whereby 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.

Prompt examples

To set a rhombic tensor for the domain labelled ‘domain 1’ with the alignment named ‘super media’, type one of:

```
relax> align_tensor.init('domain 1', '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='domain 1',
align_id='super media', params=(-8
.6322e-05, -5.5786e-04, -3.1732e-05,
.2927e-05, 2.8599e-04), param_types=1)
```

17.2.9 align_tensor.matrix_angles



Synopsis

Calculate the angles between all alignment tensors.

Defaults

```
align_tensor.matrix_angles(basis_set='matrix', tensors=
None, angle_units='deg', precision=1)
```

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.

angle_units: The units for the angle parameters, either ‘deg’ or ‘rad’.

precision: The precision of the printed out angles. The number corresponds to the number of figures to print after the decimal point.

Description

This will calculate the inter-matrix angles between all loaded alignment tensors for the current data pipe. For the vector basis sets, the matrices are first mapped to vector form and then the inter-vector angles rather than inter-matrix angles are calculated. The angles are dependent upon the basis set - linear maps produce identical results whereas non-linear maps result in different angles. The basis set can be one of:

‘matrix’ – The standard inter-matrix angles. This default option is a linear map, hence angles are preserved. The angle is calculated via the arccos of the Euclidean inner product of the alignment matrices in rank-2, 3D form divided by the Frobenius norm —A—F of the matrices.

‘irreducible 5D’ – The inter-tensor vector angles for the irreducible spherical tensor 5D basis set {A-2, A-1, A0, A1, A2}. This is a linear map, hence angles are preserved. These are the spherical harmonic decomposition coefficients.

‘unitary 9D’ – The inter-tensor vector angles for the unitary 9D basis set {Sxx, Sxy, Sxz, Syx, Syy, Syz, Szx, Szy, Szz}. This is a linear map, hence angles are preserved.

'unitary 5D' – The inter-tensor vector angles for the unitary 5D basis set {Sxx, Syy, Sxy, Sxz, Syz}. This is a non-linear map, hence angles are not preserved.

'geometric 5D' – The inter-tensor vector angles for the geometric 5D basis set {Szz, Sxxy, Sxy, Sxz, Syz}. This is a non-linear map, hence angles are not preserved. This is also the Pales standard notation.

The full matrix angle via the Euclidean inner product is defined as

$$\theta = \arccos \left| \frac{\langle A_1 | A_2 \rangle}{\sqrt{|A_1|^2} \cdot \sqrt{|A_2|^2}} \right|,$$

where $\langle A_1 | A_2 \rangle$ is the Euclidean inner product and $\sqrt{|A|}$ is the Frobenius norm of the matrix. For the irreducible spherical tensor 5D basis set, the Am components are defined as

$$\begin{aligned} A_0 &= \frac{1}{5} \langle A_1 | S_{zz} | A_2 \rangle, \\ A_{+-1} &= \pm \frac{1}{\sqrt{15}} \langle A_1 | (S_{xz} \pm iS_{yz}) | A_2 \rangle, \\ A_{+-2} &= \pm \frac{1}{\sqrt{15}} \langle A_1 | (S_{xx} - S_{yy} \pm 2iS_{xy}) | A_2 \rangle, \end{aligned}$$

and, for this complex notation, the angle is

$$\theta = \arccos \left| \frac{\operatorname{Re}(\langle A_1 | A_2 \rangle)}{\sqrt{|A_1|^2} \cdot \sqrt{|A_2|^2}} \right|,$$

where the inner product is defined as

$$\langle A_1 | A_2 \rangle = \sum_{m=-2}^2 \sum_{m'=-2}^2 A_m^{*} A_{m'},$$

and where $A_m^* = (-1)^m A_{-m}$, and the norm is defined as $\sqrt{|A|} = \operatorname{Re}(\sqrt{\langle A | A \rangle})$. For all other basis sets whereby the map is real matrix -i.e. real vector, the inter-tensor angle is defined as

$$\theta = \arccos \left| \frac{\langle A_1 | A_2 \rangle}{\sqrt{|A_1|^2} \cdot \sqrt{|A_2|^2}} \right|,$$

where the inner product $\langle A_1 | A_2 \rangle$ is simply the vector dot product and $\sqrt{|A|}$ is the vector length.

17.2.10 align_tensor.reduction



Synopsis

Specify that one tensor is a reduction of another.

Defaults

`align_tensor.reduction(full_tensor=None, red_tensor=None)`

Keyword arguments

`full_tensor`: The full alignment tensor.

`red_tensor`: The reduced 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.

Prompt 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')
```

17.2.11 align_tensor.set_domain



Synopsis

Set the domain label for the alignment tensor.

Defaults

```
align_tensor.set_domain(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.

Prompt 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')
```

17.2.12 align_tensor.svd



Synopsis

Calculate the singular values and condition number for all alignment tensors.

Defaults

```
align_tensor.svd(basis_set='irreducible 5D', tensors=None,
precision=4)
```

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.

precision: The precision of the printed out singular values and condition numbers. The number corresponds to the number of figures to print after the decimal point.

Description

This will perform a singular value decomposition for all alignment tensors and calculate the condition number. The singular values and condition number are dependent on the basis set - linear maps produce identical results whereas non-linear maps result in different values. The basis set can be one of:

‘irreducible 5D’ – The irreducible spherical tensor 5D basis set {A-2, A-1, A0, A1, A2}. This is a linear map, hence angles, singular values, and condition number are preserved. These are the spherical harmonic decomposition coefficients.

‘unitary 9D’ – The unitary 9D basis set {Sxx, Sxy, Sxz, Syx, Syy, Syz, Szx, Szy, Szz}. This is a linear map, hence angles, singular values, and condition number are preserved.

‘unitary 5D’ – The unitary 5D basis set {Sxx, Syy, Sxy, Sxz}. This is a non-linear map, hence angles, singular values, and condition number are not preserved.

‘geometric 5D’ – The geometric 5D basis set {Szz, Sxxy, Sxy, Sxz, Syz}. This is a non-linear map, hence angles, singular values, and condition number are not preserved. This is also the Pales standard notation.

If the selected basis set is the default of ‘irreducible 5D’, the matrix on which SVD will be performed will be:

$$\begin{array}{|c c c c c|} \hline & & & & \backslash 15 / \\ & & & & / 2\pi \backslash 1/2 \\ A+/-2 = & | --- | & (S_{xx} - S_{yy} +/- 2iS_{xy}) . & & \\ & & \backslash 15 / & & \\ \hline | A-2(1) & A-1(1) & A0(1) & A1(1) & A2(1) | & & & & \\ | A-2(2) & A-1(2) & A0(2) & A1(2) & A2(2) | & & & & \\ | A-2(3) & A-1(3) & A0(3) & A1(3) & A2(3) | & & & & \\ | . & . & . & . & . | & & & & \\ | . & . & . & . & . | & & & & \\ | . & . & . & . & . | & & & & \\ | A-2(N) & A-1(N) & A0(N) & A1(N) & A2(N) | & & & & \\ \hline \end{array}$$

The relationships between the geometric and unitary basis sets are

$$S_{zz} = - S_{xx} - S_{yy},$$

$$S_{xxyy} = S_{xx} - S_{yy}.$$

If the selected basis set is ‘unitary 9D’, the matrix on which SVD will be performed will be:

$$\begin{array}{|c c c c c c c c c|} \hline | S_{xx1} & S_{xy1} & S_{xz1} & S_{yx1} & S_{yy1} & S_{yz1} & S_{zx1} & S_{zy1} & S_{zz1} | \\ | S_{xx2} & S_{xy2} & S_{xz2} & S_{yx2} & S_{yy2} & S_{yz2} & S_{zx2} & S_{zy2} & S_{zz2} | \\ | S_{xx3} & S_{xy3} & S_{xz3} & S_{yx3} & S_{yy3} & S_{yz3} & S_{zx3} & S_{zy3} & S_{zz3} | \\ | . & . & . & . & . & . & . & . & . | \\ | . & . & . & . & . & . & . & . & . | \\ | . & . & . & . & . & . & . & . & . | \\ | S_{xxN} & S_{xyN} & S_{xzN} & S_{yxN} & S_{yyN} & S_{yzN} & S_{zxN} & S_{zyN} & S_{zzN} | \\ \hline \end{array}$$

Otherwise if the selected basis set is ‘unitary 5D’, the matrix for SVD is:

$$\begin{array}{|c c c c|} \hline | S_{xx1} & S_{yy1} & S_{xy1} & S_{xz1} & S_{yz1} | \\ | S_{xx2} & S_{yy2} & S_{xy2} & S_{xz2} & S_{yz2} | \\ | S_{xx3} & S_{yy3} & S_{xy3} & S_{xz3} & S_{yz3} | \\ | . & . & . & . & . | \\ | . & . & . & . & . | \\ | . & . & . & . & . | \\ | S_{xxN} & S_{yyN} & S_{xyN} & S_{xzN} & S_{yzN} | \\ \hline \end{array}$$

Or if the selected basis set is ‘geometric 5D’, the stretching and skewing parameters Szz and Sxx-yy will be used instead and the matrix is:

$$\begin{array}{|c c c c|} \hline | S_{zz1} & S_{xxyy1} & S_{xy1} & S_{xz1} & S_{yz1} | \\ | S_{zz2} & S_{xxyy2} & S_{xy2} & S_{xz2} & S_{yz2} | \\ | S_{zz3} & S_{xxyy3} & S_{xy3} & S_{xz3} & S_{yz3} | \\ | . & . & . & . & . | \\ | . & . & . & . & . | \\ | . & . & . & . & . | \\ | S_{zzN} & S_{xxyyN} & S_{xyN} & S_{xzN} & S_{yzN} | \\ \hline \end{array}$$

For the irreducible spherical tensor basis set, the Am components are defined as

$$\begin{aligned} A0 &= \frac{1}{4\pi} \frac{1}{5} S_{zz}, \\ A+/-1 &= \frac{1}{8\pi} \frac{1}{5} (S_{xz} +/- iS_{yz}), \end{aligned}$$

17.2.13 angles.diff_frame



Synopsis

Calculate the angles defining the XH bond vector within the diffusion frame.

Defaults

`angles.diff_frame()`

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.

17.2.14 bmrbcitation



Synopsis

Specify a citation to be added the BMRB data file.

Defaults

```
bmrbcitation(cite_id=None, authors=None, doi=None,
pubmed_id=None, full_citation=None, title=None,
status='published', type='journal', journal_abbrev=None,
journal_full=None, volume=None, issue=None,
page_first=None, page_last=None, year=None)
```

Keyword arguments

`cite_id`: The citation ID string.

`authors`: The list of authors. Each author element is a list of four elements (the first name, last name, first initial, and middle initials).

`doi`: The DOI number, e.g. ‘10.1000/182’.

`pubmed_id`: The identification code assigned to the publication by PubMed.

`full_citation`: The full citation as given in a reference list.

`title`: The title of the publication.

`status`: The status of the publication. This can be a value such as ‘published’, ‘submitted’, etc.

`type`: The type of publication, for example ‘journal’.

`journal_abbrev`: The standard journal abbreviation.

`journal_full`: The full journal name.

`volume`: The volume number.

`issue`: The issue number.

`page_first`: The first page number.

`page_last`: The last page number.

`year`: The publication year.

Description

The full_citation should be in a format similar to that used in a journal article by either cutting and pasting from another document or by typing. Please include author names, title, journal, page numbers, and year or equivalent information for the type of publication given.

The journal status can only be one of:

```
'preparation',
'in press',
'published',
'retracted',
'submitted'.
```

The citation type can only be one of:

```
'abstract',
'BMRB only',
'book',
'book chapter',
'internet',
'journal',
'personal communication',
'thesis'.
```

The standard journal abbreviation is that defined by the Chemical Abstract Services for the journal where the data are or will be published. If the data in the deposition are related to a J. Biomol. NMR paper, the value must be 'J. Biomol. NMR' to alert the BMRB annotators so that the deposition is properly processed. If the depositor truly does not know the journal, a value of 'not known' or 'na' is acceptable.

Prompt examples

To add the citation "d'Auvergne E. J., Gooley P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. Mol. Biosyst., 3(7), 483-494.", type:

```
relax> bmrbcitation(authors=[["Edward", "d' Auvergne", "E.", "J."], ["Paul", "Gooley", "P.", "R."]], doi="10.1039/b702202f", pubmed_id="17579774",
full_citation="d'Auvergne E. J., Gooley P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. Mol. Biosyst., 3(7), 483-494.", title="
```

```
Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm.", status="published", type="journal", journal_abbrev="Mol. Biosyst.", journal_full="Molecular Biosystems", volume=3, issue=7, page_first=483, page_last=498, year=2007)
```

17.2.15 bmrdb.display**Synopsis**

Display the BMRB data in NMR-STAR format.

Defaults

```
bmrdb.display(version='3.1')
```

Keyword arguments

version: The version of the BMRB NMR-STAR format to display.

Description

This will print the BMRB NMR-STAR formatted data to STDOUT.

17.2.16 bmrdb.read**Synopsis**

Read BMRB files in the NMR-STAR format.

Defaults

```
bmrdb.read(file=None, dir=None, version=None, sample_conditions=None)
```

Keyword arguments

file: The name of the BMRB NMR-STAR formatted file to read.

dir: The directory where the file is located.

version: The version of the BMRB NMR-STAR format to read. This is not necessary as the version is normally auto-detected.

sample_conditions: The sample conditions label in the NMR-STAR file to restrict loading to.

Description

This will allow most of the data from a BMRB NMR-STAR formatted file to be loaded into the relax data store. Note that an empty data pipe should be created for storing the data, and that currently only model-free data pipes can be used. Also, only one sample condition can be read per relax data pipe. Therefore if one of the sample conditions is not specified and multiple conditions exist in the NMR-STAR file, an error will be raised.

17.2.17 bmrbscript



Synopsis

Specify the scripts used in the analysis.

Defaults

```
bmrbscript(file=None, dir=None, analysis_type=None,
model_selection=None, engine='relax', model_elim=False,
universal_solution=False)
```

Keyword arguments

file: The name of the script file.

dir: The directory where the file is located.

analysis_type: The type of analysis performed.

model_selection: The model selection technique used, if relevant. For example ‘AIC’ model selection.

engine: The software engine used in the analysis.

model_elim: A model-free specific flag specifying if model elimination was performed.

universal_solution: A model-free specific flag specifying if the universal solution was sought after.

Description

This user function allows scripts used in the analysis to be included in the BMRB deposition. The following addition information may need to be specified with the script.

The analysis type must be set. Allowable values include all the data pipe types used in relax, ie:

‘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,

The model selection technique only needs to be set if the script selects between different mathematical models. This can be anything, but the following are recommended:



‘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).

The engine is the software used in the calculation, optimisation, etc. This can be anything, but those recognised by relax (automatic program info, citations, etc. added) include:

‘relax’ – hence relax was used for the full analysis.

‘modelfree4’ – Art Palmer’s Modelfree4 program was used for optimising the model-free parameter values.

‘dasha’ – The Dasha program was used for optimising the model-free parameter values.

‘curvefit’ – Art Palmer’s curvefit program was used to determine the R₁ or R₂ values.

The model_elim flag is model-free specific and should be set if the methods from ”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.” were used. This should be set to True for the full_analysis.py script.

The universal_solution flag is model-free specific and should be set if the methods from ”d’Auvergne E. J., Gooley P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. Mol. Biosyst., 3(7), 483-494.” were used. This should be set to True for the full_analysis.py script.

Prompt examples

For BMRB deposition, to specify that the full_analysis.py script was used, type one of:

```
relax> bmrbl.script('full_analysis.py',
    model-free', 'AIC', 'relax', True, True
)

relax> bmrbl.script(file='full_analysis.py',
    dir=None, analysis_type='model-free',
    model_selection='AIC', engine='relax',
    model_elim=True, universal_solution=
    True)
```

17.2.18 bmrbl.software



Synopsis

Specify the software used in the analysis.

Defaults

```
bmrbl.software(name=None, version=None, url=None,
    vendor_name=None, cite_ids=None, tasks=None)
```

Keyword arguments

name: The name of the software program utilised.

version: The version of the software, if applicable.

url: The web address of the software.

vendor_name: The name of the company or person behind the program.

cite_ids: A list of the BMRB citation ID numbers.

tasks: A list of all the tasks performed by the software.

Description

This user function allows the software used in the analysis to be specified in full detail.

For the tasks list, this should be a python list of strings (e.g. ['spectral processing']). Although not restricted to these, the values suggested by the BMRB are:

```
'chemical shift assignment',
'chemical shift calculation',
'collection',
'data analysis',
'geometry optimization',
'peak picking',
'processing',
'refinement',
'structure solution'
```

Prompt examples

For BMRB deposition, to say that Sparky was used in the analysis, type:

```
relax> cite_id = bmrbb.citation(authors=[["Tom", "Goddard", "T.", "D."], ["D.", "Kneller", "D.", "G."]], title="Goddard, T. D. and Kneller, D. G., SPARKY 3, University of California, San Francisco."
```

```
relax> bmrbb.software("Sparky", version="3.110", url="http://www.cgl.ucsf.edu/home/sparky/", vendor_name="Goddard, T. D.", cite_ids=[cite_id], tasks=["spectral analysis"])
```

17.2.19 bmrbb.software_select



Synopsis

Select the software used in the analysis.

Defaults

`bmrbb.software_select(name=None, version=None)`

Keyword arguments

`name`: The name of the software program utilised.

`version`: The version of the software, if applicable.

Description

Rather than specifying all the information directly, this user function allows the software packaged used in the analysis to be selected by name. The programs currently supported are:

‘NMRPipe’ – <http://spin.niddk.nih.gov/NMRPipe/>
 ‘Sparky’ – <http://www.cgl.ucsf.edu/home/sparky/>

More can be added if all relevant information (program name, description, website, original citation, purpose, etc.) is emailed to relax-users@gna.org.

Note that relax is automatically added to the BMRB file.

Prompt examples

For BMRB deposition, to say that both NMRPipe and Sparky were used prior to relax, type:

```
relax> bmrbb.software_select('NMRPipe')

relax> bmrbb.software_select('Sparky',
    version='3.113')
```

17.2.20 bmrb.thiol_state**Synopsis**

Select the thiol state of the system.

Defaults

```
bmrb.thiol_state(state=None)
```

Keyword arguments

`state`: The thiol state.

Description

The thiol state can be any text, thought the BMRB suggests the following:

```
'all disulfide bound',
'all free',
'all other bound',
'disulfide and other bound',
'free and disulfide bound',
'free and other bound',
'free disulfide and other bound',
'not available',
'not present',
'not reported',
'unknown'.
```

Alternatively the pure states ‘reduced’ or ‘oxidised’ could be specified.

Prompt examples

For BMRB deposition, to say that the protein studied is in the oxidised state, ttype one of:

```
relax> bmrb.thiol_state('oxidised')

relax> bmrb.thiol_state(state='oxidised')
```

17.2.21 bmrb.write**Synopsis**

Write the results to a BMRB NMR-STAR formatted file.

Defaults

```
bmrb.write(file=None, dir='pipe_name', version='3.1', force=False)
```

Keyword arguments

`file`: The name of the BMRB file to output results to. Optionally this can be a file object, or any object with a `write()` method.

`dir`: The directory name.

`version`: The NMR-STAR dictionary format version to create.

`force`: A flag which if True will cause the any pre-existing file to be overwritten.

Description

This will create a NMR-STAR formatted file of the data in the current data pipe for BMRB deposition.

In the prompt/script UI modes, to place the BMRB 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.

17.2.22 bruker.read



Synopsis

Read a Bruker Dynamics Center (DC) relaxation data file.

Defaults

bruker.read(ri_id=None, file=None, dir=None)

Keyword arguments

ri_id: The relaxation data ID string. This must be a unique identifier.

file: The name of the Bruker Dynamics Center file containing the relaxation data.

dir: The directory where the file is located.

Description

This user function is used to load all of the data out of a Bruker Dynamics Center (DC) relaxation data file for subsequent analysis within relax. Currently the R₁ and R₂ relaxation rates and steady-state NOE data is supported.



17.2.23 chemical_shift.read

δ



Synopsis

Read chemical shifts from a file.

Defaults

chemical_shift.read(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 peak list of generic formatted file containing the chemical shifts.

dir: The directory where the file is located.

spin_id_col: The spin ID string column used by the generic 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 file format (alternative to the spin ID column).

res_num_col: The residue number column used by the generic file format (alternative to the spin ID column).

res_name_col: The residue name column used by the generic file format (alternative to the spin ID column).

spin_num_col: The spin number column used by the generic file format (alternative to the spin ID column).

spin_name_col: The spin name column used by the generic file format (alternative to the spin ID column).

sep: The column separator used by the generic format (the default is white space).

spin_id: The spin ID string used to restrict the loading of data to certain spin subsets.

Description

This will read chemical shifts from a peak list or a generic column formatted file.

Prompt examples

The following commands will read the chemical shifts out of the Sparky peak list ‘10ms.list’:

```
relax> chemical_shift.read('10ms.list')
```

17.2.24 consistency_tests.set_frq


 ω

Synopsis

Select which relaxation data to use in the consistency tests by NMR spectrometer frequency.

Defaults

```
consistency_tests.set_frq(frq=None)
```

Keyword arguments

frq: The spectrometer frequency in Hz. This must match the currently loaded data to the last decimal point. See the ‘sfrq’ parameter in the Varian procpar file or the ‘SF01’ parameter in the Bruker acqus file.

Description

This will select the relaxation data to use in the consistency tests corresponding to the given frequencies. The data is selected by the spectrometer frequency in Hertz, which should be set to the exact value (see the ‘sfrq’ parameter in the Varian procpar file or the ‘SF01’ parameter in the Bruker acqus file). Note thought that the R₁, R₂ and NOE are all expected to have the exact same frequency in the $J(\omega)$ mapping analysis (to the last decimal point).

Prompt examples

```
relax> consistency_tests.set_frq(600.0 * 1e6
)
relax> consistency_tests.set_frq(frq=600.0 *
1e6)
```

17.2.25 dasha.create



Synopsis

Create the Dasha script.

Defaults

```
dasha.create(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 algorithm 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.

17.2.26 dasha.execute



Synopsis

Perform a model-free optimisation using Dasha.

Defaults

```
dasha.execute(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.

Description

Dasha will be executed as

```
$ dasha < dasha_script | tee dasha_results
```

If you would like to use a different Dasha executable file, change the binary name 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.

17.2.27 dasha.extract



Synopsis

Extract data from the Dasha results file.

Defaults

```
dasha.extract(dir=None)
```

Keyword arguments

dir: The directory where the file 'dasha_results' is found.

Description

The model-free results will be extracted from the Dasha results file 'dasha_results' located in the given directory.

17.2.28 deselect.all**Synopsis**

```
deselect.all()
```

Defaults

`deselect.all()`

Description

This will deselect all spins, regardless of their current state.

Prompt examples

To deselect all spins, simply type:

```
relax> deselect.all()
```

17.2.29 deselect.interatom**Synopsis**

```
deselect.interatom(spin_id1=None, spin_id2=None, boolean='AND', change_all=False)
```

Defaults

`deselect.interatom(spin_id1=None, spin_id2=None, boolean='AND', change_all=False)`

Keyword arguments

`spin_id1`: The spin ID string of the first spin of the interatomic data container.

`spin_id2`: The spin ID string of the second spin of the interatomic data container.

`boolean`: The boolean operator specifying how interatomic data containers should be selected.

`change_all`: A flag specifying if all other interatomic data containers should be changed.

Description

This is used to deselect specific interatomic data containers which store information about spin pairs such as RDCs, NOEs, dipole-dipole pairs involved in relaxation, etc. The ‘`change_all`’ flag default is False meaning that all interatomic data containers currently either selected or deselected will remain that way. Setting this to True will cause all interatomic data containers not specified by the spin ID strings to be deselected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Table 17.1: Boolean operators and their effects on selections

Spin system or interatomic data container	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

Prompt examples

To deselect all N-H backbone bond vectors of a protein, assuming these interatomic data containers have been already set up, type one of:

```
relax> deselect.interatom('ON', 'OH')
```

```
relax> deselect.interatom(spin_id1='ON',
                           spin_id2='OH')
```

To deselect all H-H interatomic vectors of a small organic molecule, type one of:

```
relax> deselect.interatom('OH*', 'OH*')
```

```
relax> deselect.interatom(spin_id1='OH*',
                           spin_id2='OH*')
```

17.2.30 deselect.read



Synopsis

Deselect the spins contained in a file.

Defaults

```
deselect.read(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).

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.

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 molecule name, residue number, residue name, spin number and/or spin name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID string 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 default is False meaning that all spins currently either selected or deselected will remain that way. Setting this to True will cause all spins not specified in the file to be selected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Prompt examples

To deselect all overlapped residues listed with residue numbers in the first column of the file ‘`unresolved`’, type one of:

```
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)
```

17.2.31 deselect.reverse



Synopsis

Reversal of the spin selection for the given spins.

Defaults

```
deselect.reverse(spin_id=None)
```

Keyword arguments

spin_id: The spin ID string.

Description

By supplying the spin ID string, a subset of spins can have their selection status reversed.

Description

To deselect all currently selected spins and select those which are deselected type:

```
relax> deselect.reverse()
```

17.2.32 deselect.sn_ratio

**Synopsis**

Deselect spins with signal to noise ratio higher or lower than the given ratio.

Defaults

```
deselect.sn_ratio(ratio=10.0, operation='<', all_sn=False)
```

Keyword arguments

ratio: The signal to noise ratio to compare to.

operation: The comparison operation by which to deselect the spins.

all_sn: A flag specifying if all the signal to noise ratios per spin should match the comparison operator, or if just a single comparison match is enough.

Description

The comparison operation is the method which to deselect spins according to: operation(sn_ratio, ratio).

The possible operations are: '<':strictly less than, '<=':less than or equal, '>':strictly greater than, '>=':greater than or equal, '==':equal, '!=':not equal.

The 'all_sn' flag default is False, meaning that if any of the spin's signal to noise levels evaluates to True in the comparison, the spin is deselected.

Prompt examples

To deselect all spins with a signal to noise ratio lower than 10.0:

```
relax> deselect.sn_ratio(ratio=10.0,
                           operation='<')

relax> deselect.sn_ratio(ratio=10.0,
                           operation='<', all_sn=True)
```

17.2.33 deselect.spin

**Synopsis**

Deselect specific spins.

Defaults

```
deselect.spin(spin_id=None, boolean='AND', change_all=False)
```

Keyword arguments

spin_id: The spin ID string.

boolean: The boolean operator specifying how spins should be deselected.

change_all: A flag specifying if all other spins should be changed.

Description

The 'change_all' flag default is False meaning that all spins currently either selected or deselected will remain that way. Setting this to True will cause all spins not specified by the spin ID string to be deselected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Prompt 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')
```

17.2.34 diffusion_tensor.copy**Synopsis**

Copy diffusion tensor data from one data pipe to another.

Defaults

```
diffusion_tensor.copy(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 will copy the diffusion tensor data between data pipes. The destination data pipe must not contain any diffusion tensor data. If the source or destination data pipes are not supplied, then both will default to the current data pipe (hence specifying at least one is essential).

Prompt 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')
```

17.2.35 diffusion_tensor.delete**Synopsis**

Delete the diffusion tensor data from the relax data store.

Defaults

```
diffusion_tensor.delete()
```

Description

This will delete all diffusion tensor data from the current data pipe.

17.2.36 diffusion_tensor.display
**Synopsis**

Display the diffusion tensor information.

Defaults

`diffusion_tensor.display()`

Description

This will display all of the diffusion tensor information of the current data pipe.

17.2.37 diffusion_tensor.init
**Synopsis**

Initialise the diffusion tensor.

Defaults

`diffusion_tensor.init(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

$$C(\tau) = \frac{1 - e^{-\tau / \tau_m}}{5},$$

To select isotropic diffusion, the parameter 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 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

```

      _1_
      1 \           - tau / tau_i
C(tau) = - > ci . e
      5 /__
      i=-1

```

where

$$\begin{aligned} c-1 &= 1/4 (3 \delta_z^2 - 1)^2, \\ c0 &= 3 \delta_z^2 (1 - \delta_z^2), \\ c1 &= 3/4 (\delta_z^2 - 1)^2, \end{aligned}$$

and

$$\begin{aligned} 1 / \tau -1 &= 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a, \\ 1 / \tau 0 &= 6\mathfrak{D}_{iso} - \mathfrak{D}_a, \\ 1 / \tau 1 &= 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a. \end{aligned}$$

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 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 – $\{\mathfrak{D}_{||}, \mathfrak{D}_{\perp}, \theta, \phi\}$,

4 – $\{\mathfrak{D}_{iso}, \mathfrak{D}_{ratio}, \theta, \phi\}$,

where

$$\tau_m = 1 / 6\mathfrak{D}_{iso},$$

$$\mathfrak{D}_{iso} = 1/3 (\mathfrak{D}_{||} + 2\mathfrak{D}_{\perp}),$$

$$\mathfrak{D}_a = \mathfrak{D}_{||} - \mathfrak{D}_{\perp},$$

$$\mathfrak{D}_{ratio} = \mathfrak{D}_{||} / \mathfrak{D}_{\perp}.$$

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 should be ‘oblate’, ‘prolate’, or None. This 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 boundary $\mathfrak{D}_a = 0$.

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

```


$$\text{C}(\tau) = - \frac{1}{\sqrt{5}} \frac{\tau - \tau_i / \tau_{i-1}}{\sqrt{i-2}}$$


```

where the weights on the exponentials are

$$\begin{aligned} c_{-2} &= 1/4 (d + e), \\ c_{-1} &= 3 \delta_y^2 \delta_z^2, \\ c_0 &= 3 \delta_x^2 \delta_z^2, \\ c_1 &= 3 \delta_x^2 \delta_y^2, \\ c_2 &= 1/4 (d + e). \end{aligned}$$

Let

$$\mathfrak{R} = \sqrt{1 + 3\mathfrak{D}_r},$$

then

$$\begin{aligned} d &= 3 (\delta_x^4 + \delta_y^4 + \delta_z^4) - 1, \\ e &= -1 / \mathfrak{R} ((1 + 3\mathfrak{D}_r)(\delta_x^4 + 2\delta_y^2 \delta_z^2) + \\ &\quad (1 - 3\mathfrak{D}_r)(\delta_y^4 + 2\delta_x^2 \delta_z^2) - 2(\delta_z^4 + 2\delta_x^2 \delta_y^2)). \end{aligned}$$

The correlation times are

$$\begin{aligned} 1 / \tau_{-2} &= 6\mathfrak{D}_{iso} - 2\mathfrak{D}_a \cdot \mathfrak{R}, \\ 1 / \tau_{-1} &= 6\mathfrak{D}_{iso} - \mathfrak{D}_a (1 + 3\mathfrak{D}_r), \\ 1 / \tau_0 &= 6\mathfrak{D}_{iso} - \mathfrak{D}_a (1 - 3\mathfrak{D}_r), \\ 1 / \tau_1 &= 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a, \\ 1 / \tau_2 &= 6\mathfrak{D}_{iso} + 2\mathfrak{D}_a \cdot \mathfrak{R}. \end{aligned}$$

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 $[\delta_x, \delta_y, \delta_z]$.

To select fully anisotropic diffusion, the parameters 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

$$\begin{aligned} \mathbf{0} &= \{\tau_m, \mathfrak{D}_a, \mathfrak{D}_r, \alpha, \beta, \gamma\} \text{ (Default),} \\ \mathbf{1} &= \{\mathfrak{D}_{iso}, \mathfrak{D}_a, \mathfrak{D}_r, \alpha, \beta, \gamma\}, \\ \mathbf{2} &= \{\mathfrak{D}_x, \mathfrak{D}_y, \mathfrak{D}_z, \alpha, \beta, \gamma\}, \end{aligned}$$

3 – { \mathfrak{D}_{xx} , \mathfrak{D}_{yy} , \mathfrak{D}_{zz} , \mathfrak{D}_{xy} , \mathfrak{D}_{xz} , \mathfrak{D}_{yz} },

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 correlation time scaling value should be a floating point number. The only parameter affected by this value is τ_m .

The diffusion tensor eigenvalue scaling value 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 units for the angle parameters should be either ‘deg’ or ‘rad’. Parameters affected are θ , ϕ , α , β , and γ .

Prompt 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)
```

17.2.38 domain**Synopsis**

Definition of structural domains.

Defaults

domain(id=None, spin_id=None)

Keyword arguments

id: The ID string used to identify molecular domains.

spin_id: The spin ID string of all atomic members of the domain.

Description

This is used to define structural domains. Multiple domains can be defined, and these can overlap. Rather than labelling the currently loaded spins with the ID string, the spin ID string is stored for later use. This allows new spins to be loaded later and still be included within the same domain.

17.2.39 dx.execute



Synopsis

Execute an OpenDX program.

Defaults

```
dx.execute(file_prefix='map', dir='dx', dx_exe='dx',
vp_exec=True)
```

Keyword arguments

file_prefix: 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.

dx_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.

Description

This will execute OpenDX to display the space maps created previously by the δ_x .map user function. This will work for any type of OpenDX map.

17.2.40 dx.map



Synopsis

Create a map of the given space in OpenDX format.

Defaults

```
dx.map(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', chi_surface=None, create_par_file=False)
```

Keyword arguments

params: The parameters to be mapped. This 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 ID string.

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: This argument allows specific points in the optimisation space to be displayed as coloured spheres. This can be used to highlight a minimum or other any other feature of the space. Either a single point or a list of



points can be supplied. Each point is a list of floating point numbers in the form $[x, y, z]$

point_file: The name of that the point output files will be prefixed with.

chi_surface: A list of 4 numbers, setting the level for the 4 isosurfaces. Useful in scripting if you create a set of OpenDX maps with all the same contour levels. Ideal for comparisons.

create_par_file: A flag specifying whether to create a file with parameters and associated chi2 value. The default of False causes the file not to be created.

```
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)
```

Description

This will map the space corresponding to the spin identifier and create the OpenDX files. The map type can be changed to one of the following supported map types:

Please see Table 17.2 on page 473.

Model-free parameters

Please see Table 17.3 on page 473.

N-state model parameters

Please see Table 17.4 on page 473.

Relaxation dispersion parameters

Please see Table 17.5 on page 474.

Frame order parameters

Please see Table 17.6 on page 474.

Prompt 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')
```

Table 17.2: OpenDx mapping types.

Surface type	Name
3D isosurface	'Iso3D'

Table 17.3: Model-free parameters.

Name	Description
tm	Global correlation time
Diso	Isotropic component of the diffusion tensor
Dx	Eigenvalue associated with the x-axis of the diffusion tensor
Dy	Eigenvalue associated with the y-axis of the diffusion tensor
Dz	Eigenvalue associated with the z-axis of the diffusion tensor
Dpar	Diffusion coefficient parallel to the major axis of the spheroid diffusion tensor
Dper	Diffusion coefficient perpendicular to the major axis of the spheroid diffusion tensor
Da	Anisotropic component of the diffusion tensor
Dr	Rhombic component of the diffusion tensor
Dratio	Ratio of the parallel and perpendicular components of the spheroid diffusion tensor
alpha	The first Euler angle of the ellipsoid diffusion tensor
beta	The second Euler angle of the ellipsoid diffusion tensor
gamma	The third Euler angle of the ellipsoid diffusion tensor
theta	The polar angle defining the major axis of the spheroid diffusion tensor
phi	The azimuthal angle defining the major axis of the spheroid diffusion tensor
s2	S^2 , the model-free generalised order parameter ($S^2 = S_f^2/S_{2s}$)
s2f	S_f^2 , the faster motion model-free generalised order parameter
s2s	S_{2s}^2 , the slower motion model-free generalised order parameter
local_tm	The spin specific global correlation time (seconds)
te	Single motion effective internal correlation time (seconds)
tf	Faster motion effective internal correlation time (seconds)
ts	Slower motion effective internal correlation time (seconds)
rex	Chemical exchange relaxation ($\sigma_{ex} = R_{ex} / \omega^{**2}$)
csa	Chemical shift anisotropy (unitless)

Table 17.4: N-state model parameters.

Name	Description	Type
Axx	The Axx component of the alignment tensor	float
Ayy	The Ayy component of the alignment tensor	float
Axy	The Axy component of the alignment tensor	float
Axz	The Axz component of the alignment tensor	float
Ayz	The Ayz component of the alignment tensor	float
probs	The probabilities of each state	list
alpha	The α Euler angles (for the rotation of each state)	list
beta	The β Euler angles (for the rotation of each state)	list
gamma	The γ Euler angles (for the rotation of each state)	list
paramagnetic_centre	The paramagnetic centre	list

Table 17.5: Relaxation dispersion parameters.

Name	Description	Type
r2eff	The effective transversal relaxation rate	dict
i0	The initial intensity	dict
r1	The longitudinal relaxation rate	dict
r2	The transversal relaxation rate	dict
r2a	The transversal relaxation rate for state A in the absence of exchange	dict
r2b	The transversal relaxation rate for state B in the absence of exchange	dict
pA	The population for state A	float
pB	The population for state B	float
pC	The population for state C	float
phi_ex	The $\phi_{ex} = pA.pB.dw^{**2}$ value (ppm ²)	float
phi_ex_B	The fast exchange factor between sites A and B (ppm ²)	float
phi_ex_C	The fast exchange factor between sites A and C (ppm ²)	float
padw2	The $pA.dw^{**2}$ value (ppm ²)	float
dw	The chemical shift difference between states A and B (in ppm)	float
dw_AB	The chemical shift difference between states A and B for 3-site exchange (in ppm)	float
dw_AC	The chemical shift difference between states A and C for 3-site exchange (in ppm)	float
dw_BC	The chemical shift difference between states B and C for 3-site exchange (in ppm)	float
dwH	The proton chemical shift difference between states A and B (in ppm)	float
dwH_AB	The proton chemical shift difference between states A and B for 3-site exchange (in ppm)	float
dwH_AC	The proton chemical shift difference between states A and C for 3-site exchange (in ppm)	float
dwH_BC	The proton chemical shift difference between states B and C for 3-site exchange (in ppm)	float
kex	The exchange rate	float
kex_AB	The exchange rate between sites A and B for 3-site exchange with $k_{ex_AB} = k_{AB} + k_{BA}$ (rad.s ⁻¹)	float
kex_AC	The exchange rate between sites A and C for 3-site exchange with $k_{ex_AC} = k_{AC} + k_{CA}$ (rad.s ⁻¹)	float
kex_BC	The exchange rate between sites B and C for 3-site exchange with $k_{ex_BC} = k_{BC} + k_{CB}$ (rad.s ⁻¹)	float
kB	Approximate chemical exchange rate constant between sites A and B (rad.s ⁻¹)	float
kC	Approximate chemical exchange rate constant between sites A and C (rad.s ⁻¹)	float
tex	The time of exchange (tex = 1/kex)	float
k_AB	The exchange rate from state A to state B	float
k_BA	The exchange rate from state B to state A	float

Table 17.6: Frame order parameters.

Name	Description
pivot_x	The pivot point position x coordinate
pivot_y	The pivot point position y coordinate
pivot_z	The pivot point position z coordinate
pivot_disp	The 2 nd pivot point displacement - the minimum distance between the two rotor axes
ave_pos_x	The average position x translation
ave_pos_y	The average position y translation
ave_pos_z	The average position z translation
ave_pos_alpha	The average position α Euler angle
ave_pos_beta	The average position β Euler angle
ave_pos_gamma	The average position γ Euler angle
eigen_alpha	The Eigenframe α Euler angle
eigen_beta	The Eigenframe β Euler angle
eigen_gamma	The Eigenframe γ Euler angle
axis_theta	The cone axis polar angle (for the isotropic cone model)
axis_phi	The cone axis azimuthal angle (for the isotropic cone model)
axis_alpha	The rotor axis α angle (the rotation angle out of the xy plane)
cone_theta_x	The pseudo-ellipse cone opening half-angle for the x-axis
cone_theta_y	The pseudo-ellipse cone opening half-angle for the y-axis
cone_theta	The isotropic cone opening half-angle
cone_sigma_max	The torsion angle
cone_sigma_max_2	The torsion angle of the 2 nd motional mode

17.2.41 eliminate



Synopsis

Elimination or rejection of models.

Defaults

`eliminate(function=None, args=None)`

Keyword arguments

function: An optional user supplied function for model elimination.

args: A tuple of arguments used by the optional function for model elimination.

Description

This 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.

Empirical rules are used for model rejection and are listed below. However these can be overridden by supplying a function in the prompt and scripting modes. 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 function arguments 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 tm 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:

`tm >= 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 te, tf, ts 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 ($\tau_e, \tau_f, \tau_s \leq 2\tau_m$). These failed models are eliminated using the rule:

`te, tf, ts >= c . tm.`

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).

17.2.42 error_analysis.covariance_matrix

Synopsis

Parameter error estimation via the covariance matrix.

Defaults

`error_analysis.covariance_matrix(epsrel=0.0, verbosity=1)`

Keyword arguments

`epsrel`: The parameter to remove linear-dependent columns when J is rank deficient.

`verbosity`: The higher the value, the greater the verbosity.

Description

This is a new experimental feature from version 3.3.

This will estimate parameter errors by using the exponential decay Jacobian matrix ‘ J ’ to compute the covariance matrix of the best-fit parameters.

This can be used to for comparison to Monte-Carlo simulations.

This method is inspired from the GNU Scientific Library (GSL).

The covariance matrix is given by: $\text{covar} = Q_{xx} = (J^T W J)^{-1}$, where the weight matrix W is constructed by the multiplication of an Identity matrix I and a weight array w . The weight array is $1/\text{errors}^2$, which then gives $W = I \cdot w = I / \text{errors}^2$.

Q_{xx} is computed by QR decomposition, $J^T W J = QR$, $Q_{xx} = R^{-1} Q^T$. The columns of R which satisfy: $|R_{\{kk\}}| \leq \text{epsrel} |R_{\{11\}}|$ are considered linearly-dependent and are excluded from the covariance matrix (the corresponding rows and columns of the covariance matrix are set to zero).

The parameter ‘`epsrel`’ is used to remove linear-dependent columns when J is rank deficient.

17.2.43 fix



Synopsis

Fix or allow parameter values to change during optimisation.

Defaults

`fix(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 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 parameters 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.

17.2.44 frame_order.count_sobol_points



Synopsis

Count the number of Sobol' points used for the current parameter values.

Defaults

`frame_order.count_sobol_points()`

Description

This allows the number of Sobol' integration points used during the Frame Order target function optimisation to be counted. This uses the current parameter values to determine how many are used for the PCS calculation compared to the total number.

17.2.45 frame_order.decompose



Synopsis

Structural representation of the individual frame order motional components.

Defaults

`frame_order.decompose(root='decomposed', dir=None, atom_id=None, model=1, total=None, reverse=False, mirror=False, force=False)`

Keyword arguments

`root`: The file root for the PDB files created. Each motional component will be represented by a different PDB file appended with '`_mode1.pdb`', '`_mode2.pdb`', '`_mode3.pdb`', etc.

`dir`: The directory where the files are to be saved.

`atom_id`: The atom identification string to allow the representation to be applied to a subset of all atoms.

`model`: Only one model from an analysed ensemble of structures can be used for the representation, as the decomposition PDB files consist of one model per state.

`total`: The total number of structures to distribute along the motional modes. This overrides the fixed angle value.

`reverse`: Set this to reverse the ordering of the models distributed along the motional mode.

`mirror`: Set this to have the models distributed along the motional mode shift from the negative angle to positive angle, and then return to the negative angle.

`force`: A flag which, if set to True, will overwrite the any pre-existing file.

Description

An alternative way to visualise the frame order motions is to decompose the motions and visualise each mode separately. This user function will create a uniform distribution of structures shifted from the original position and rotated around the eigenvector for that motional mode. Each distribution will be output to a PDB file appended with '`_modeX.pdb`', where X are the discrete motional modes ordered from largest to smallest. The curved line of positions will extend over the full distribution of structures.

17.2.46 frame_order.distribute



represent the components of the distribution present in the data, as modelled by the frame order models.

As the distribution consists of one model per state, if an ensemble of structures has been analysed, only one model from the ensemble can be used for the representation.

Synopsis

Structural distribution of the frame order motions.

Defaults

```
frame_order.distribute(file='distribution.pdb.gz', dir=
None, atom_id=None, total=1000, max_rotations=
100000, model=1, force=False)
```

Keyword arguments

file: The PDB file for storing the frame order motional distribution. The compression is determined automatically by the file extensions ‘*.pdb’, ‘*.pdb.gz’, and ‘*.pdb.bz2’.

dir: The directory where the files are to be located.

atom_id: The atom identification string to allow the distribution to be a subset of all atoms.

total: The total number of structures to include in the uniform distribution.

max_rotations: The maximum number of rotations to generate the distribution from. This prevents the user function from executing for an infinite amount of time. This occurs whenever a frame order amplitude parameter (cone opening angle or torsion angle) is zero so that the subset of all rotations within the motional distribution is also zero.

model: Only one model from an analysed ensemble of structures can be used for the distribution, as the distribution PDB file consists of one model per state.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

To visualise the frame order motions, this user function generates a distribution of structures randomly within the bounds of the uniform distribution of the frame order model. The original structure is rotated randomly and only accepted for the distribution if it is within the bounds. This is a more faithful representation of the dynamics than the pseudo-Brownian simulation user function.

Note that the RDC and PCS data does not contain information about all parts of the real distribution of structures. Therefore the structures in this distribution only

17.2.47 frame_order.pdb_model



Synopsis

Create a PDB file representation of the frame order dynamics.

Defaults

```
frame_order.pdb_model(ave_pos='ave_pos', rep='frame_order', dir=None, compress_type=0, size=30.0, inc=36, model=1, force=False)
```

Keyword arguments

ave_pos: The file root of the 3D structure PDB file for the molecular structure with the moving domains shifted to the average position.

rep: The file root of the PDB file for the geometric object representation of the frame order dynamics.

dir: The directory where the files are to be located.

compress_type: The type of compression to use when creating the files.

size: The size of the geometric object in Å.

inc: The number of increments used to create the geometric object.

model: Only one model from an analysed ensemble can be used for the PDB representation of the Monte Carlo simulations of the average domain position, as these consists of one model per simulation.

force: A flag which, if set to True, will overwrite the any pre-existing files.

Description

This function creates a set of PDB files for representing the frame order cone models. This includes a file for the average position of the molecule and a file containing a geometric representation of the frame order motions.

The three files are specified via the file root whereby the extensions ‘.pdb’, ‘.pdb.gz’, etc. should not be provided. This is important for the geometric representation whereby different files are created for the positive and negative representations (due to symmetry in the NMR data, these cannot be differentiated), and for the Monte Carlo simulations. For example if the file root is ‘frame_order’, the positive and negative representations will be placed in the ‘frame_order_pos.pdb.gz’

and ‘frame_order_neg.pdb.gz’ files and the Monte Carlo simulations in the ‘frame_order_sim_pos.pdb.gz’ and ‘frame_order_sim_neg.pdb.gz’ files. For models where there is no difference in representation between the positive and negative directions, the files ‘frame_order.pdb.gz’ and ‘frame_order_sim.pdb.gz’ will be produced.

There are four different types of residue within the PDB. The pivot point is represented 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 Å away on the cone axis (set by the geometric object size). 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.

The PDB representation of the Monte Carlo simulations consists of one model per simulation. Therefore if an ensemble of structures has been analysed, only one model from the ensemble can be used for the representation. This defaults to model number 1, but this can be changed.

17.2.48 frame_order.permute_axes**Synopsis**

Permute the axes of the motional eigenframe to switch between local minima.

Defaults

```
frame_order.permute_axes(permuation='A')
```

Keyword arguments

permuation: Which of the two permutations ‘A’ or ‘B’ to create. Three permutations are possible, and ‘A’ and ‘B’ select those which are not the starting combination.

Description

The isotropic and pseudo-elliptic cone frame order models consist of multiple solutions as the optimisation space contains multiple local minima. Because of the constraint $\text{cone_theta_x} \leq \text{cone_theta_y}$ in the pseudo-ellipse model, there are exactly three local minima (out of 6 possible permutations). However the $\text{cone_theta_x} == \text{cone_theta_y}$ condition of the isotropic cone collapses this to two minima. The multiple minima correspond to permutations of the motional system - the eigenframe x , y and z -axes as well as the cone opening angles cone_theta_x , cone_theta_y , and cone_sigma_max associated with these axes. But as the mechanics of the cone angles is not identical to that of the torsion angle, only one of the three local minima is the global minimum.

When optimising the pseudo-elliptic models, specifically the ‘pseudo-ellipse’ and ‘pseudo-ellipse, torsionless’ model, any of the three local minima can be found. Convergence to the global minimum is not guaranteed. Therefore this user function can be used to permute the motional system to jump from one local minimum to the other. Optimisation will be required as the permuted parameters will not be exactly at the minimum.

Please see Table 17.7 on page 481.

In this table, the condition and cone angle values $[x, y, z]$ correspond to cone_theta_x , cone_theta_y , and cone_sigma_max .

Prompt examples

For combination ‘A’, simply type:

```
relax> frame_order.permute_axes('A')
```

17.2.49 frame_order.pivot**Synopsis**

Set the pivot points for the two body motion in the structural coordinate system.

Defaults

```
frame_order.pivot(pivot=None, order=1, fix=False)
```

Keyword arguments

pivot: The pivot point for the motion (e.g. the position between the 2 domains in PDB coordinates).

order: The ordinal number of the pivot point. The value of 1 is for the first pivot point, the value of 2 for the second pivot point, and so on.

fix: A flag specifying if the pivot point should be fixed during optimisation.

Description

This will set the pivot points for the two domain system within the PDB coordinate system. This is required for interpreting PCS data as well as for the generation of cone or other PDB representations of the domain motions.

This user function can also be used to change the optimisation status of an already set pivot point. By simply providing the fixed flag and not the pivot point values, the pivot can be changed to be either fixed during optimisation or that it will be optimised.

Prompt 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])
```

To change an already set and fixed pivot point so that it can now be optimised, type:

```
relax> frame_order.pivot(fix=False)
```

Table 17.7: The motional eigenframe axis permutations for the frame order models.

Condition	Permutation name	Cone angles	Axes
$x < y < z$	Self	$[x, y, z]$	$[x, y, z]$
	A	$[x, z, y]$	$[-z, y, x]$
	B	$[y, z, x]$	$[z, x, y]$
$x < z < y$	Self	$[x, y, z]$	$[x, y, z]$
	A	$[x, z, y]$	$[-z, y, x]$
	B	$[z, y, x]$	$[x, -z, y]$
$z < x < y$	Self	$[x, y, z]$	$[x, y, z]$
	A	$[z, x, y]$	$[y, z, x]$
	B	$[z, y, x]$	$[x, -z, y]$

17.2.50 frame_order.quad_int



Synopsis

Turn the high precision quadratic integration on or off.

Defaults

```
frame_order.quad_int(flag=True)
```

Keyword arguments

flag: The flag with if True will perform high precision numerical integration via the `scipy.integrate quad()`, `dblquad()` and `tplquad()` integration methods rather than the rough quasi-random numerical integration.

Description

This allows the high precision numerical integration of the Scipy `quad()` and related functions to be used instead of the lower precision quasi-random Sobol' sequence integration. This is for the optimisation of the Frame Order target functions. The quadratic integration is orders of magnitude slower than the Sobol' sequence integration, but the precision is much higher.

17.2.51 frame_order.ref_domain



Synopsis

Set the reference non-moving domain for the 2-domain frame order theories.

Defaults

```
frame_order.ref_domain(ref=None)
```

Keyword arguments

ref: The non-moving domain which will act as the frame of reference.

Description

Prior to optimisation of the frame order model, the frame of reference non-moving domain must be specified. This is essential for determining which spins will be used in the analysis, which will be shifted to the average position, etc.

Prompt 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')
```

17.2.52 frame_order.select_model



Synopsis

Select and set up the Frame Order model.

Defaults

```
frame_order.select_model(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 2nd 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.

'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.

'double rotor' – Restricted motions about two independent but orthogonal rotor axes. The first rotation is about the y-axis and the second is about the x-axis.

Prompt examples

To select the isotropic cone model, type:

```
relax> frame_order.select_model(model='iso
cone')
```

17.2.53 frame_order.simulate



Synopsis

Pseudo-Brownian dynamics simulation of the frame order motions.

Defaults

```
frame_order.simulate(file='simulation.pdb.gz', dir=None,
step_size=2.0, snapshot=10, total=1000, model=1,
force=False)
```

Keyword arguments

file: The PDB file for storing the frame order pseudo-Brownian dynamics simulation. The compression is determined automatically by the file extensions ‘*.pdb’, ‘*.pdb.gz’, and ‘*.pdb.bz2’.

dir: The directory where the files are to be located.

step_size: The rotation will be of a random direction but with this fixed angle. The value is in degrees.

snapshot: The number of steps in the simulation when snapshots will be taken.

total: The total number of snapshots to take before stopping the simulation.

model: Only one model from an analysed ensemble of structures can be used for the pseudo-Brownian simulation, as the simulation and corresponding PDB file consists of one model per simulation.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

To visualise the frame order motions, this user function performs a type of simulation whereby structures are randomly rotated by a fixed angle within the bounds of the uniform distribution of the frame order model. This can be thought of as a pseudo-Brownian dynamics simulation. It is in no way a real molecular or Brownian dynamics simulation.

Note that the RDC and PCS data does not contain information about all parts of the real distribution of structures. Therefore the snapshots in this simulation only represent the components of the distribution present in the data, as modelled by the frame order models.

The simulation algorithm is as follows. The current state is initially defined as the identity matrix I. The maximum opening angle θ or the torsion angle sigma are defined by the parameter values of the frame order model. The algorithm for one step of the simulation is:

- 1 – Generate a random vector in 3D.
- 2 – Construct a rotation matrix from the random vector and the fixed rotation angle.
- 3 – Pre-multiply the current state by the rotation matrix.
- 4 – Decompose the new state into the torsion-tilt angles.
- 5 – If θ or sigma are greater than model parameter values, set them to these maximum values.
- 6 – Back convert the modified torsion-tilt angles to a rotation matrix - this is the current state.
- 7 – Store a snapshot if the correct number of iterations has been reached. This consists of rotating a new model about the pivot(s), as defined by the frame order model.
- 8 – Terminate the loop if the maximum number of snapshots has been reached.

The setting of the steps outside of the distribution to the maximum parameter values is specifically to allow for models with parameter values close to zero. Without this, the simulation would take a huge amount of time to complete.

As the simulation consists of one model per snapshot, if an ensemble of structures has been analysed, only one model from the ensemble can be used for the representation. This defaults to model number 1, but this can be changed.

17.2.54 frame_order.sobol_setup



Synopsis

Set up the quasi-random Sobol' sequence points for numerical PCS integration.

Defaults

`frame_order.sobol_setup(max_num=200, oversample=1)`

Keyword arguments

max_num: The maximum number of integration points to use in the Sobol' sequence during optimisation. This can be considered as the number of molecular structures in an ensemble used form a uniform distribution of the dynamics.

oversample: The generation of the Sobol' sequence oversamples as $N * Ov * 10^{**M}$, where N is the maximum number of points, Ov is the oversampling value, and M is the number of dimensions or torsion-tilt angles used in the system.

Description

This allows the maximum number of integration points N used during the frame order target function optimisation to be specified. This is used in the quasi-random Sobol' sequence for the numerical integration of the PCS. The formula used to find the total number of Sobol' points is:

```
total_num = N * Ov * 10**M,
```

where:

N is the maximum number of Sobol' integration points,

Ov is the oversampling factor.

M is the number of dimensions or torsion-tilt angles used in the system.

The aim of the oversampling is to try to reach the maximum number of points. However if the system is not very dynamic, the maximum number of points may not be reached. In this case, simply increase the oversampling factor. The algorithm used for uniformly sampling the motional space is:

Generate the Sobol' sequence for the total number of points.

Convert all points to the torsion-tilt angle system.

Skip all Sobol' points with angles greater than the current parameter values.

Terminate the loop over the Sobol' points once the maximum number of points has been reached.

17.2.55 grace.view

 Grace

Synopsis

Visualise the file within Grace.

Defaults

```
grace.view(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 can be used to view the specified Grace ‘*.agr’ file by opening it with the Grace program.

Prompt 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')
      )
```

17.2.56 grace.write

 Grace



Synopsis

Create a grace ‘.agr’ file to visualise the 2D data.

Defaults

```
grace.write(x_data_type='res_num', y_data_type=None,
spin_id=None, plot_data='value', norm_type='first', 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 ID string.

`plot_data`: The data to use for the plot.

`norm_type`: How the graph should be normalised, if the norm flag is set.

`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.

`norm`: A flag which, if set to True, will cause all graphs to be normalised to 1. This is for the normalisation of series type data. The point for normalisation is set with the `norm_type` argument.

Description

This 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 information can be used to create various types of plot. These include the x-axis and y-axis data types, the spin ID string, and the type of data plot.

The x-axis and y-axis data types should be plain strings, regular expression is not allowed. The two axes of the Grace plot can be any of the data types listed in the tables below. The only limitation is that the data must belong to the same data pipe.

If the x-axis data type is not given, the plot will default to having the residue numbering along the x-axis. Two special data types for the axes are:

'res_num' – The axis will consist of the residue numbering.

'spin_num' – The axis will consist of the spin numbering.

The spin ID string can be used to limit which spins are used in the plot. The default is that all spins will be used, however, the ID string 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 setting. This 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 R₂ 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.

Relaxation curve fitting parameters

Please see Table 17.8 on page 487.

Steady-state NOE parameters

Please see Table 17.9 on page 487.

Model-free parameters

Please see Table 17.10 on page 487.

Reduced spectral density mapping parameters

Please see Table 17.11 on page 487.

Consistency testing parameters

Please see Table 17.12 on page 488.

Relaxation dispersion parameters

Please see Table 17.13 on page 488.

Prompt examples

To write the NOE values for all spins to the Grace file 'noe.agr', type one of:

```
relax> grace.write('res_num', 'noe', file='noe.agr')
```

```
relax> grace.write(y_data_type='noe', file='noe.agr')
```

```
relax> grace.write(x_data_type='res_num',
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 versus 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)
```

Table 17.8: Relaxation curve fitting parameters and minimisation statistics.

Name	Description
rx	Either the R_1 or R_2 relaxation rate
i0	The initial intensity
iinf	The intensity at infinity
chi2	Chi-squared value
iter	Optimisation iterations
f_count	Number of function calls
g_count	Number of gradient calls
h_count	Number of Hessian calls
warning	Optimisation warning

Table 17.9: Steady-state NOE parameters.

Name	Description
noe	The steady-state NOE value

Table 17.10: Model-free parameters and minimisation statistics.

Name	Description
s2	S^2 , the model-free generalised order parameter ($S^2 = S_f^2 \cdot S_{2s}$)
s2f	S_f^2 , the faster motion model-free generalised order parameter
s2s	S_s^2 , the slower motion model-free generalised order parameter
local_tm	The spin specific global correlation time (seconds)
te	Single motion effective internal correlation time (seconds)
tf	Faster motion effective internal correlation time (seconds)
ts	Slower motion effective internal correlation time (seconds)
rex	Chemical exchange relaxation ($\sigma_{ex} = R_{ex} / \omega^{**2}$)
csa	Chemical shift anisotropy (unitless)

Table 17.11: Reduced spectral density mapping parameters.

Name	Description
j0	Spectral density value at 0 MHz - $J(0)$
jwx	Spectral density value at the frequency of the heteronucleus - $J(\omega_X)$
jwh	Spectral density value at the frequency of the proton - $J(\omega_H)$
csa	Chemical shift anisotropy (unitless)

Table 17.12: Consistency testing parameters.

Name	Description
j0	Spectral density value at 0 MHz (from Farrow et al. (1995) JBNMR, 6: 153-162)
f_eta	Eta-test (from Fushman et al. (1998) JACS, 120: 10947-10952)
f_r2	R2-test (from Fushman et al. (1998) JACS, 120: 10947-10952)
csa	Chemical shift anisotropy (unitless)
orientation	Angle between the $^{15}\text{N}-\text{H}$ vector and the principal axis of the ^{15}N chemical shift tensor
tc	The single global correlation time estimate/approximation

Table 17.13: Relaxation dispersion parameters and minimisation statistics.

Name	Description
r2eff	The effective transversal relaxation rate
i0	The initial intensity
r1	The longitudinal relaxation rate
r2	The transversal relaxation rate
r2a	The transversal relaxation rate for state A in the absence of exchange
r2b	The transversal relaxation rate for state B in the absence of exchange
pA	The population for state A
pB	The population for state B
pC	The population for state C
phi_ex	The $\phi_{\text{ex}} = pA.pB.dw^{**2}$ value (ppm 2)
phi_ex_B	The fast exchange factor between sites A and B (ppm 2)
phi_ex_C	The fast exchange factor between sites A and C (ppm 2)
padw2	The $pA.dw^{**2}$ value (ppm 2)
dw	The chemical shift difference between states A and B (in ppm)
dw_AB	The chemical shift difference between states A and B for 3-site exchange (in ppm)
dw_AC	The chemical shift difference between states A and C for 3-site exchange (in ppm)
dw_BC	The chemical shift difference between states B and C for 3-site exchange (in ppm)
dwh	The proton chemical shift difference between states A and B (in ppm)
dwh_AB	The proton chemical shift difference between states A and B for 3-site exchange (in ppm)
dwh_AC	The proton chemical shift difference between states A and C for 3-site exchange (in ppm)
dwh_BC	The proton chemical shift difference between states B and C for 3-site exchange (in ppm)
kex	The exchange rate
kex_AB	The exchange rate between sites A and B for 3-site exchange with $k_{\text{ex_AB}} = k_{\text{AB}} + k_{\text{BA}}$ (rad.s $^{-1}$)
kex_AC	The exchange rate between sites A and C for 3-site exchange with $k_{\text{ex_AC}} = k_{\text{AC}} + k_{\text{CA}}$ (rad.s $^{-1}$)
kex_BC	The exchange rate between sites B and C for 3-site exchange with $k_{\text{ex_BC}} = k_{\text{BC}} + k_{\text{CB}}$ (rad.s $^{-1}$)
kB	Approximate chemical exchange rate constant between sites A and B (rad.s $^{-1}$)
kC	Approximate chemical exchange rate constant between sites A and C (rad.s $^{-1}$)
tex	The time of exchange ($\text{tex} = 1/kex$)
k_AB	The exchange rate from state A to state B
k_BA	The exchange rate from state B to state A
chi2	Chi-squared value
iter	Optimisation iterations
f_count	Number of function calls
g_count	Number of gradient calls
h_count	Number of Hessian calls
warning	Optimisation warning

17.2.57 interatom.copy



Synopsis

Copy all data associated with a interatomic data container.

Defaults

```
interatom.copy(pipe_from=None, pipe_to=None,
               spin_id1=None, spin_id2=None)
```

Keyword arguments

pipe_from: The data pipe containing the interatomic data container from which the data will be copied. This defaults to the current data pipe.

pipe_to: The data pipe to copy the interatomic data container to. This defaults to the current data pipe.

spin_id1: The spin ID of the first spin.

spin_id2: The spin ID of the second spin.

Description

This will copy all the data associated with the identified interatomic data container to a different data pipe. The new interatomic data container must not already exist.

Prompt examples

To copy the interatomic data container between ':2@C' and ':2@H', from the 'orig' data pipe to the current data pipe, type one of:

```
relax> interatom.copy('orig', spin_id1=':2@C',
                       spin_id2=':2@H')
```

```
relax> interatom.copy(pipe_from='orig',
                       spin_id1=':2@C', spin_id2=':2@H')
```

17.2.58 interatom.define



Synopsis

Define interatomic interactions between pairs of spins.

Defaults

```
interatom.define(spin_id1='@N', spin_id2='@H',
                 direct_bond=True, spin_selection=True, pipe=None)
```

Keyword arguments

spin_id1: The spin ID string for the first spin of the interatomic interaction.

spin_id2: The spin ID string for the second spin of the interatomic interaction.

direct_bond: This is a flag which if True means that the two spins are directly bonded. This flag is useful to simplify the set up of the main heteronuclear relaxation mechanism or one-bond residual dipolar couplings.

spin_selection: Define the interatomic data container selection based on the spin selection. If either spin is deselected, the interatomic container will also be deselected. Otherwise the container will be selected.

pipe: The data pipe to create the interatomic data container for. This defaults to the current data pipe if not supplied.

Description

To analyse relaxation or residual dipolar coupling (RDC) data, for example, pairs of spins which are coupled need to be defined. This can be via the magnetic dipole-dipole interaction or scalar coupling interaction. This function will create an interatomic data object connecting two existing spins. This data container will be used to store all information about the interatomic interaction including interatomic vectors and distances.

For analyses which use relaxation data, simply defining the interatomic interaction will indicate that there is a dipolar relaxation mechanism operating between the two spins. Note that for model-free analyses or reduced spectral density mapping, only a single relaxation mechanism can be handled. For RDC dependent analyses, the presence of the interatomic interaction indicates that dipolar coupling is expected between the two spins.

If the spin selection flag is set, then the newly created interatomic data container will be selected based on the

current selection status of the two spins defining the interaction. If either of the spins are deselected, then the new interatomic data container will also be deselected. If both spins are selected, then the interatomic data container will also be selected.

Prompt examples

To connect the spins ':1@N' to ':1@H', type one of:

```
relax> interatom.define(':1@N', ':1@H')

relax> interatom.define(spin_id1=':1@N',
    spin_id2=':1@H')
```

To define the protein 15N heteronuclear relaxation mechanism for a model-free analysis, type one of the following:

```
relax> interatom.define('@N', '@H', True)

relax> interatom.define(spin_id1='@N',
    spin_id2='@H', direct_bond=True)
```

17.2.59 interatom.read_dist



Synopsis

Read inter-spin distances from a file.

Defaults

```
interatom.read_dist(file=None, dir=None, unit='meter',
spin_id1_col=1, spin_id2_col=2, data_col=3, sep=None)
```

Keyword arguments

file: The name of the file containing the averaged distance data.

dir: The directory where the file is located.

unit: The unit of distance. The default is meter, but Å can also be specified.

spin_id1_col: The spin ID string column for the first spin.

spin_id2_col: The spin ID string column for the second spin.

data_col: The distance data column.

sep: The column separator (the default is white space).

Description

This allows interatomic distances to be read from a file. This is useful in the case when the distances vary, avoiding having to tediously use the interatom.set_dist user function for each spin-pair separately. The format of the file should be columnar, with the two spin ID strings in two separate columns and the distances in any other. The default measurement unit is meter but this can be changed to Å.

For RDC and relaxation based analyses, as the magnetic dipole-dipole interaction is averaged in NMR over the interatomic distance to the inverse third power, the interatomic distances within a 3D structural file are of no use for defining the interaction. Therefore these r^{-3} average distances must be explicitly defined.

Prompt examples

To load the distances in meters from the fifth column of the ‘distances’ file, and where the spin IDs are in the first and second columns, type one of the following:

```
relax> interatom.read_dist('distances', 1,
2, 5)
```



```
relax> interatom.read_dist(file='distances',
unit='meter', spin_id1_col=1,
spin_id2_col=2, data_col=5)
```

17.2.60 interatom.set_dist



Synopsis

Set the inter-spin distances.

Defaults

```
interatom.set_dist(spin_id1='@N', spin_id2='@H',
ave_dist=1.020000000000001e-10, unit='meter')
```

Keyword arguments

spin_id1: The spin identification string for the first spin of the dipole pair.

spin_id2: The spin identification string for the second spin of the dipole pair.

ave_dist: The r^{-3} averaged distance between the two spins to be used in the magnetic dipole constant, defaulting to meters.

unit: The unit of distance (the default is ‘meter’).

Description

For many NMR interactions, the distance between the spin of interest and another spin or atom must be defined. This information can be extracted from a 3D structure but, in many cases, these distances are not of interest. For example the empirical or fixed distance calculation of proton positions in X-ray crystallographic structures will often not correspond to the real interatomic distances.

Another example is the magnetic dipole-dipole interaction which is averaged over the interatomic distance to the inverse third power. In this case, the interatomic distances from any 3D structural file can be of no use for defining the interaction. The average distances must be explicitly supplied. This user function allows these distances to be set up. The default measurement unit is meter but this can be changed to Å. Alternatively the distances can be read from a file using other user functions in this class.

Prompt examples

To set the N-H distance for protein the 15N heteronuclear relaxation mechanism to 1.02 Å, type one of the following:

```
relax> interatom.set_dist('@N', '@H', 1.02 *
1e-10)
```

```
relax> interatom.set_dist(spin_id1='@N',
   spin_id2='@H', ave_dist=1.02 * 1e-10,
   unit='meter')

relax> interatom.set_dist(spin_id1='@N',
   spin_id2='@H', ave_dist=1.02, unit='
   Angstrom')
```

17.2.61 interatom.unit_vectors



Synopsis

Calculate the unit vectors for all interatomic interactions.

Defaults

interatom.unit_vectors(ave=True)

Keyword arguments

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 will have no effect.

Description

For an orientational dependent analysis, such as model-free analysis with the spheroidal and ellipsoidal global diffusion tensors or any analysis using RDCs, the unit vectors between the two dipoles must be calculated prior to starting the analysis. For the unit vector extraction, the two interacting spins should already possess positional information and the dipole-dipole interaction should already be defined via the interatom.define user function. This information will be used to calculate unit vectors between the two spins. Without positional information from a 3D structure, no vectors can be calculated and an orientational dependent analysis will not be possible.

The number of unit vectors per interaction will be defined by the number of positions each spin possesses together with the averaging flag. If both spins have N and M positions loaded, the number of positions for both must match (N=M). In this case, as well as when one spin has N positions and the other a single position, then N unit vectors will be calculated. This is unless the averaging flag is set in which case an averaged vector of unit length will be calculated.

Prompt examples

To calculate the unit vectors prior to a model-free analysis, type one of the following:

```
relax> interatom.unit_vectors(True)
```

```
relax> interatom.unit_vectors(ave=True)
```

17.2.62 j_coupling.copy**Synopsis**

Copy J coupling data from one data pipe to another.

Defaults

```
j_coupling.copy(pipe_from=None, pipe_to=None)
```

Keyword arguments

`pipe_from`: The name of the pipe to copy the J coupling data from.

`pipe_to`: The name of the pipe to copy the J coupling data to.

Description

This function will copy J coupling data from one pipe to another.

Prompt examples

To copy all J coupling data from pipe ‘DMSO’ to pipe ‘CDC13’, type one of:

```
relax> j_coupling.copy('DMSO', 'CDC13')
```

```
relax> j_coupling.copy(pipe_from='DMSO',
    pipe_to='CDC13')
```

17.2.63 j_coupling.delete**Synopsis**

Delete the J coupling values.

Defaults

```
j_coupling.delete()
```

Description

This will delete all J coupling data in the current data pipe.

Prompt examples

To delete all J coupling data, type:

```
relax> j_coupling.delete()
```

17.2.64 j_coupling.display**Synopsis**

Display the J coupling data in the current data pipe.

Defaults

`j_coupling.display()`

Description

This will display all of the J coupling data in the current data pipe.

Prompt examples

To display all J coupling data, type:

`relax> j_coupling.display()`

17.2.65 j_coupling.read**Synopsis**

Read the J coupling data from file.

Defaults

`j_coupling.read(file=None, dir=None, spin_id1_col=1, spin_id2_col=2, data_col=None, error_col=None, sign_col=None, sep=None)`

Keyword arguments

`file`: The name of the file containing the J coupling data.

`dir`: The directory where the file is located.

`spin_id1_col`: The spin ID string column for the first spin.

`spin_id2_col`: The spin ID string column for the second spin.

`data_col`: The J coupling data column.

`error_col`: The experimental error column.

`sign_col`: A special column holding the sign of the J coupling, being either 1 or -1, in case this data is obtained separately.

`sep`: The column separator (the default is white space).

Description

This will read J coupling data from a file. If the sign of the J coupling has been determined by a different experiment, this information can be present in a different column having either the value of 1 or -1.

Prompt examples

The following commands will read the J coupling data out of the file 'J.txt' where the columns are separated by the symbol ',':

`relax> j_coupling.read('J.txt', sep=',')`

If the individual spin J coupling errors are located in the file 'j_err.txt' in column number 5 then, to read these values into relax, type one of:



```
relax> j_coupling.read('j_err.txt',
    error_col=5)

relax> j_coupling.read(file='j_err.txt',
    error_col=5)
```

17.2.66 j_coupling.write



Synopsis

Write the J coupling data to file.

Defaults

`j_coupling.write(file=None, dir=None, force=False)`

Keyword arguments

`file`: The name of the file.

`dir`: The directory name.

`force`: A flag which if True will cause the file to be overwritten.

Description

This will write the J coupling values to file. If no directory name is given, the file will be placed in the current working directory.

17.2.67 jw_mapping.set_frq

$J(\omega)$

ω

17.2.68 minimise.calculate



Synopsis

Select which relaxation data to use in the $J(\omega)$ mapping by NMR spectrometer frequency.

Defaults

```
jw_mapping.set_frq(frq=None)
```

Keyword arguments

frq: The spectrometer frequency in Hz. This must match the currently loaded data to the last decimal point. See the ‘**sfrq**’ parameter in the Varian procpar file or the ‘**SF01**’ parameter in the Bruker acqus file.

Description

This will select the relaxation data to use in the reduced spectral density mapping corresponding to the given frequency. The data is selected by the spectrometer frequency in Hertz, which should be set to the exact value (see the ‘**sfrq**’ parameter in the Varian procpar file or the ‘**SF01**’ parameter in the Bruker acqus file). Note thought that the R_1 , R_2 and NOE are all expected to have the exact same frequency in the $J(\omega)$ mapping analysis (to the last decimal point).

Prompt examples

```
relax> jw_mapping.set_frq(600.0 * 1e6)
relax> jw_mapping.set_frq(frq=600.0 * 1e6)
```

Synopsis

Calculate the model parameters or the current target function value.

Defaults

```
minimise.calculate(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.

Description

The operation of this user function is two-fold and depends on whether the solution for the models of the current analysis are found by direct calculation or by optimisation. The dual operations are:

Direct calculation models – For these models, the parameters will be directly calculated from the base data. This will be the exact solution and the user function will store the parameter values. The grid search and optimisation user functions are not implemented for this analysis type.

Optimised models – This will call the target function normally used for optimisation for each model using the current parameter values. This can be used to manually find the chi-squared value for different parameter values. The parameter values will not be affected.

17.2.69 minimise.execute



Synopsis

Perform an optimisation.

Defaults

```
minimise.execute(min_algor='newton', line_search=None,
hessian_mod=None, hessian_type=None, func_tol=1e-25,
grad_tol=None, max_iter=10000000, constraints=True,
scaling=True, verbosity=1)
```

Keyword arguments

min_algor: The optimisation algorithm to use.

line_search: The line search algorithm which will only be used in combination with the line search and conjugate gradient methods. This will default to the More and Thuente line search.

hessian_mod: The Hessian modification. This will only be used in the algorithms which use the Hessian, and defaults to Gill, Murray, and Wright modified Cholesky algorithm.

hessian_type: The Hessian type. This will only be used in a few trust region algorithms, and defaults to BFGS.

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_iter: 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.

Description

This will perform an optimisation starting from the current parameter values. This is only suitable for data pipe types which have target functions and hence support optimisation.

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 $(2 * \pi * 600,000,000)^{**2}$ to make it field strength independent. The scaling vector for this model may be something like [1.0, 1e-9, 1/(2 * $\pi * 6e8^{**2}$)]. 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.

Minimisation algorithms

A minimisation function is selected if the minimisation algorithm 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, use a string which matches one of the following patterns given in the tables.

Unconstrained line search methods:

Please see Table 17.14 on page 498.

Unconstrained trust-region methods:

Please see Table 17.15 on page 498.

Table 17.14: Minimisation algorithms – unconstrained line search methods.

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 17.15: Minimisation algorithms – unconstrained trust-region methods.

Minimisation algorithm	Patterns
Cauchy point	'^[[Cc]auchy'
Dogleg	'^[[Dd]ogleg'
CG-Steihaug	'^[[Cc][Gg][-][Ss]teihaug' or '^[[Ss]teihaug'
Exact trust region	'^[[Ee]xact'

Unconstrained conjugate gradient methods:

Please see Table 17.16 on page 499.

Miscellaneous unconstrained methods:

Please see Table 17.17 on page 499.

Global minimisation methods:

Please see Table 17.18 on page 499.

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. The algorithms are:

Please see Table 17.19 on page 499.

Hessian modifications. These are used in the Newton, Dogleg, and Exact trust region algorithms:

Please see Table 17.20 on page 499.

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:

Please see Table 17.21 on page 499.

For Newton minimisation, the default line search algorithm is the More and Thuente line search, while the default Hessian modification is the GMW algorithm.

Prompt 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.execute('newton')

relax> minimise.execute('Newton')

relax> minimise.execute('newton', 'gmw')

relax> minimise.execute('newton', 'mt')

relax> minimise.execute('newton', 'gmw', 'mt')

relax> minimise.execute('newton', 'mt', 'gmw')

relax> minimise.execute('newton', func_tol=1
    e-25)

relax> minimise.execute('newton', func_tol=1
    e-25, grad_tol=None)

relax> minimise.execute('newton', max_iter=1
    e7)

relax> minimise.execute('newton',
    constraints=True, max_iter=1e7)

relax> minimise.execute('newton',
    verbosity =1)
```

Table 17.16: Minimisation algorithms – unconstrained conjugate gradient methods.

Minimisation algorithm	Patterns
Fletcher-Reeves	'^[[Ff][Rr]\$' or '^[[Ff]letcher[-_-][Rr]eeves\$'
Polak-Ribiere	'^[[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 17.17: Minimisation algorithms – miscellaneous unconstrained methods.

Minimisation algorithm	Patterns
Simplex	'^[[Ss]implex\$'
Levenberg-Marquardt	'^[[Ll][Mm]\$' or '^[[Ll]evenburg-[Mm]arquardt\$'

Table 17.18: Minimisation algorithms – global minimisation methods.

Minimisation algorithm	Patterns
Simulated Annealing	'^[[Ss][Aa]\$' or '^[[Ss]imulated [Aa]nnealing\$'

Table 17.19: Minimisation sub-algorithms – line search algorithms.

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 17.20: Minimisation sub-algorithms – Hessian modifications.

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 17.21: Minimisation sub-algorithms – Hessian type.

Hessian type	Patterns
Quasi-Newton BFGS	'^[[Bb][Ff][Gg][Ss]\$'
Newton	'^[[Nn]ewton\$'

To use constrained Simplex minimisation with a maximum of 5000 iterations, type:

```
relax> minimise.execute('simplex',
    constraints=True, max_iter=5000)
```

17.2.70 minimise.grid_search



Synopsis

Perform a grid search to find an initial non-biased parameter set for optimisation.

Defaults

```
minimise.grid_search(lower=None, upper=None, inc=21,
    verbosity=1, constraints=True, skip_preset=True)
```

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.

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.

constraints: A boolean flag specifying whether the parameters should be constrained. The default is to turn constraints on (constraints=True).

skip_preset: This argument, when True, allows any parameter which already has a value set to be skipped in the grid search. This value will be overridden and turned off when a zooming grid search is active.

Description

The optimisation of a mathematical model normally consists of two parts - a coarse grid search across the parameter space to find an initial set of parameter values followed by the use of a high precision optimisation algorithm to exactly find the local or global solution. The grid search is an essential tool as it allows a non-biased initial optimisation position to be found. It avoids the statistical bias and preconditioning introduced by using a self chosen initial parameter set. The high computational cost of the grid search is almost always favourable to the statistical bias of a user defined starting position.

The region of the parameter space that the grid search covers is defined by the lower and upper grid bounds. These will generally default to the entire parameter space except for when the parameter is non-bounded, for example a 3D position in the PDB space. This user function will print out the grid bounds used and, if the default bounds are deemed to be insufficient, then the lower, upper or both bounds can supplied. This only works if all active models have the same parameters. The coarseness or fineness of the grid is defined by the number of increments to search across between the bounds. For an alternative to using large numbers of increments, see the zooming grid search.

It is possible to decrease the dimensionality of the grid search, and hence drop the computational cost by orders of magnitude, if certain parameter values are known a priori. For example if the values are determined via a different experiment. Such parameters can be set with the value setting user function. Then, when the skip preset flag is set, these parameters will be skipped in the grid search. This feature should not be abused and statistical bias should be avoided at all cost.

The parameter skipping logic is as follows. Firstly setting the increments argument to a list with None elements causes the corresponding parameters to be skipped in the grid search, or an error to be raised if no preset parameter is present. This overrides all other settings. Secondly the preset skipping flag only allows parameters to be skipped if the zooming grid search is non-active and a value is preset.

17.2.71 `minimise.grid_zoom`



Synopsis

Activate the zooming grid search by setting the zoom level.

Defaults

`minimise.grid_zoom(level=0)`

Keyword arguments

`level`: The zooming grid search level. This can be any number, positive or negative.

Description

The optimisation of a mathematical model normally consists of two parts - a coarse grid search to find an initial set of parameter values followed by the use of a high precision optimisation algorithm to exactly find the local or global solution. But in certain situations, for example where a parallelised grid search is advantageous, a finer grid may be desired. The zooming grid search provides a more efficient alternative to simply increasing the number of increments used in the initial grid search. Note that in most situations, standard optimisation algorithms will be far computationally less expensive.

The zooming grid search can be activated via this user function. After setting the desired zoom level, the original grid search user function should be called again. The zoom level is used to decrease the total area of the grid search. The grid width for each dimension of the parameter space will be divided by $2^{**\text{zoom_level}}$. So a level of 1 will halve all dimensions, a level of 2 will quarter the widths, a level of 3 will be an eighth of the widths, etc.

The zooming algorithm proceeds as follows. The new zoomed grid will be centred at the current parameter values. However if the new grid is outside of the bounds of the original grid, the entire grid will be translated so that it lies entirely within the original bounds. This is to avoid grid points lying within undefined regions of the space. An exception is when the zoom factor is negative, hence the new grid will be larger than the original.

An example of using the zooming grid search is to first perform a standard initial grid search, then set the zoom level to 1 and perform a second grid search. Continue for zoom levels 2, 3, etc. until the desired fineness is obtained. Note that convergence is not guaranteed - as the zoom level is increased to infinity, the parameter values do not necessarily converge to the local minimum. Therefore performing standard optimisation is recommended after completing a zooming grid search.

'te' – The effective correlation time.

17.2.72 model_free.create_model

$S_{\tau_e}^2$



Synopsis

Create a model-free model.

Defaults

`model_free.create_model(model=None, equation=None, params=None, spin_id=None)`

Keyword arguments

model: The new 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.

Description

This user function should almost never be used. It is provided for academic reasons for the study of old analyses and published results. If you are looking for a normal model-free model, use the `model_free.select_model` user function instead.

Model-free equation

The model-free equation can be one of the following:

'mf_orig' selects the original model-free equations with parameters $\{S^2, \tau_e\}$.

'mf_ext' selects the extended model-free equations with parameters $\{S_f^2, \tau_f, S_s^2, \tau_s\}$.

'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.

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 $\langle r \rangle$.

'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.

Prompt 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')
```

17.2.73 model_free.delete

$S^2\tau_e$



Synopsis

Delete all model-free data from the current data pipe.

Defaults

model_free.delete()

Description

This will delete all of the model-free data - parameters, model, etc. - from the current data pipe.

Prompt examples

To delete all model-free data, type:

```
relax> model_free.delete()
```

17.2.74 model_free.remove_tm S^2, τ_e **Synopsis**

Remove the local τ_m parameter from a model.

Defaults

```
model_free.remove_tm(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.

Prompt examples

The following command will remove the parameter ‘tm’:

```
relax> model_free.remove_tm()
```

17.2.75 model_free.select_model S^2, τ_e **Synopsis**

Select a preset model-free model.

Defaults

```
model_free.select_model(model=None, spin_id=None)
```

Keyword arguments

model: The name of the preset model.

spin_id: The spin identification string.

Description

This allows a standard model-free model to be selected from a long list of models.

The preset models

The standard preset model-free models are

```
'm0' - {},  
'm1' - {S2},  
'm2' - {S2, τe},  
'm3' - {S2, Rex},  
'm4' - {S2, τe, Rex},  
'm5' - {Sf2, S2, τs},  
'm6' - {Sf2, τf, S2, τs},  
'm7' - {Sf2, S2, τs, Rex},  
'm8' - {Sf2, τf, S2, τs, Rex},  
'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 flexibility of these models where a change in the parameter ‘r’ is compensated by a corresponding change in the parameter ‘csa’ and vice versa.

The preset local tm models

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’ – $\{\tau_m, r, S_f^2, \tau_f, S^2, \tau_s\}$,

‘tm27’ – $\{\tau_m, r, S_f^2, S^2, \tau_s, R_{ex}\}$,

‘tm28’ – $\{\tau_m, r, S_f^2, \tau_f, S^2, \tau_s, R_{ex}\}$,

‘tm29’ – $\{\tau_m, r, \text{CSA}, R_{ex}\}$.

The preset model-free models with both optimisation of the bond length and CSA are

‘tm30’ – $\{\tau_m, r, \text{CSA}\}$,

‘tm31’ – $\{\tau_m, r, \text{CSA}, S^2\}$,

‘tm32’ – $\{\tau_m, r, \text{CSA}, S^2, \tau_e\}$,

‘tm33’ – $\{\tau_m, r, \text{CSA}, S^2, R_{ex}\}$,

‘tm34’ – $\{\tau_m, r, \text{CSA}, S^2, \tau_e, R_{ex}\}$,

‘tm35’ – $\{\tau_m, r, \text{CSA}, S_f^2, S^2, \tau_s\}$,

‘tm36’ – $\{\tau_m, r, \text{CSA}, S_f^2, \tau_f, S^2, \tau_s\}$,

‘tm37’ – $\{\tau_m, r, \text{CSA}, S_f^2, S^2, \tau_s, R_{ex}\}$,

‘tm38’ – $\{\tau_m, r, \text{CSA}, S_f^2, \tau_f, S^2, \tau_s, R_{ex}\}$,

‘tm39’ – $\{\tau_m, r, \text{CSA}, R_{ex}\}$.

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.

Prompt examples

To pick model ‘m1’ for all selected spins, type:

```
relax> model_free.select_model('m1')
```

```
relax> model_free.select_model(model='m1')
```

17.2.76 model_selection



Synopsis

Select the best model from a set of optimised models.

Defaults

model_selection(method=‘AIC’, modsel_pipe=None, bundle=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.

bundle: The optional pipe bundle is a special grouping or clustering of data pipes. If this is specified, the newly created data pipe will be added to this bundle.

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 Monte Carlo simulations should have previously been executed with the monte_carlo.create_data method set to Bootstrapping to modify its behaviour.

If the data pipes have not been specified, then all data pipes will be used for model selection.

Prompt examples

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'])
```

17.2.77 molecule.copy



Synopsis

Copy all data associated with a molecule.

Defaults

`molecule.copy(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 name of the molecule from which to copy data from.

`pipe_to`: The data pipe to copy the data to. This defaults to the current data pipe.

`mol_to`: The name of the new molecule. If left blank, the new molecule will have the same name as the old. This needs to be a molecule ID string, starting with '#'.

Description

This will copy all the data associated with a molecule to a second molecule. This includes all residue and spin system information. The new molecule name must be unique in the destination data pipe.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Prompt 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')
```

17.2.78 molecule.create



Synopsis

Create a new molecule.

Defaults

```
molecule.create(mol_name=None, mol_type=None)
```

Keyword arguments

`mol_name`: The name of the new molecule.

`mol_type`: The type of molecule.

Description

This adds 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 given, it should be one of 'protein', 'DNA', 'RNA', 'organic molecule', or 'inorganic molecule'.

Prompt examples

To create the molecules 'Ap4Aase', 'ATP', and 'MgF4', type:

```
relax> molecule.create('Ap4Aase')
```

```
relax> molecule.create('ATP')
```

```
relax> molecule.create('MgF4')
```

17.2.79 molecule.delete



Synopsis

Deleting molecules from the relax data store.

Defaults

molecule.delete(mol_id=None)

Keyword arguments

mol_id: The molecule ID string.

Description

This can be used to delete a single or sets of molecules from the relax data store. The molecule will be deleted from the current data pipe.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

17.2.80 molecule.display



Synopsis

Display the molecule information.

Defaults

molecule.display(mol_id=None)

Keyword arguments

mol_id: The molecule ID string.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

17.2.81 molecule.name



Synopsis

Name a molecule.

Defaults

`molecule.name(mol_id=None, name=None, force=False)`

Keyword arguments

`mol_id`: The molecule ID 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 simply allows molecules to be named (or renamed).

Prompt 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.

Spin ID 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 - one per spin - 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>,

...]] @<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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string ‘@H*’ will select the protons ‘H’, ‘H2’, ‘H98’.

Spin ID string documentation

17.2.82 molecule.type



Synopsis

Set the molecule type.

Defaults

```
molecule.type(mol_id=None, type=None, force=False)
```

Keyword arguments

mol_id: The molecule ID string corresponding to one or more molecules.

type: The molecule type.

force: A flag which if True will cause the molecule to type to be overwritten.

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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Description

This allows the type of the molecule to be specified. It can be one of:

```
'protein',
'DNA',
'RNA',
'organic molecule',
'inorganic molecule'.
```

Prompt examples

To set the molecule 'Ap4Aase' to the 'protein' type, type one of:

```
relax> molecule.type('#Ap4Aase', 'protein',
True)

relax> molecule.type(mol_id='#Ap4Aase', type
='protein', force=True)
```

17.2.83 molmol.clear_history**Synopsis**

Clear the Molmol command history.

Defaults

molmol.clear_history()

Description

This will clear the Molmol history from memory.

17.2.84 molmol.command**Synopsis**

Execute a user supplied Molmol command.

Defaults

molmol.command(command=None)

Keyword arguments

command: The Molmol command to execute.

Description

This allows Molmol commands to be passed to the program. This can be useful for automation or scripting.

Prompt examples

To reinitialise the Molmol instance, type:

relax> molmol.command("InitAll yes")

Colour

17.2.85 molmol.macro_apply



Synopsis

Execute Molmol macros.

Defaults

```
molmol.macro_apply(data_type=None, style='classic',
colour_start_name=None, colour_start_rgb=None,
colour_end_name=None, colour_end_rgb=None,
colour_list=None)
```

Keyword arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start_name: The name of the starting colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the starting colour RGB colour array cannot be given.

colour_start_rgb: The starting colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the starting colour name cannot be given.

colour_end_name: The name of the ending colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the ending colour RGB colour array cannot be given.

colour_end_rgb: The ending colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the ending colour name cannot be given.

colour_list: The colour list to search for the colour names. This can be either ‘`molmol`’ or ‘`x11`’.

Description

This allows spin specific values to be mapped to a structure through Molmol macros. Currently only the ‘classic’ style, which is described below, is available.

The values are coloured based on a linear colour gradient which is specified through starting and ending colours. These 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, ‘white’ and [1.0, 1.0, 1.0] both select the same colour. Leaving both colours unset 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 name. To explicitly select these lists, set the colour list to either ‘`molmol`’ or ‘`x11`’.

Model-free classic style

Creator: Edward d’Auvergne

Argument string: “classic”

Description: The classic style draws the backbone of a protein in a cylindrical bond 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:

Please see Table 17.22 on page 514.

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.

Please see Table 17.23 on page 515.

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.

Please see Table 17.24 on page 516.

Table 17.22: The model-free classic style for mapping model spin specific data onto 3D molecular structures using either PyMOL or Molmol.

Data type	String	Description
S^2 .	's2'	The standard model-free order parameter, equal to $S_f^2 S_2$ s 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 amplitude 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 amplitude 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 17.23: Molmol colour names and corresponding RGB colour values (from 0 to 1)

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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
burlwood 1	255	211	155
burlwood 2	238	197	145
burlwood 3	205	170	125
burlwood 4	139	115	85
wheat 1	255	231	186

Table 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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 17.24: X11 colour names and corresponding RGB colour values

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

Prompt examples

To map the order parameter values, S^2 , onto the structure using the classic style, type:

```
relax> molmol.macro_apply('s2')  
  
relax> molmol.macro_apply(data_type='s2')  
  
relax> molmol.macro_apply(data_type='s2',  
    style="classic")
```

17.2.86 molmol.macro_run



Synopsis

Open and execute the Molmol macro file.

Defaults

```
molmol.macro_run(file=None, dir='molmol')
```

Keyword arguments

file: The name of the Molmol macro file.

dir: The directory name.

Description

This user function is for opening and running a Molmol macro located within a text file.

Prompt examples

To execute the macro file ‘s2.mac’ located in the directory ‘molmol’, type:

```
relax> molmol.macro_run(file='s2.mac')  
  
relax> molmol.macro_run(file='s2.mac', dir='molmol')
```

Colour

17.2.87 molmol.macro_write




Synopsis

Create Molmol macros.

Defaults

```
molmol.macro_write(data_type=None, style='classic',
colour_start_name=None, colour_start_rgb=None,
colour_end_name=None, colour_end_rgb=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_name: The name of the starting colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the starting colour RGB colour array cannot be given.

colour_start_rgb: The starting colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the starting colour name cannot be given.

colour_end_name: The name of the ending colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the ending colour RGB colour array cannot be given.

colour_end_rgb: The ending colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the ending colour name cannot be given.

colour_list: The colour list to search for the colour names. This can be either ‘molmol’ or ‘x11’.

file: The optional name of the file.

dir: The optional directory to save the file to.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

This 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.

The values are coloured based on a linear colour gradient which is specified through starting and ending colours. These 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, ‘white’ and [1.0, 1.0, 1.0] both select the same colour. Leaving both colours unset 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 name. To explicitly select these lists, set the colour list to either ‘molmol’ or ‘x11’.

Model-free classic style

Creator: Edward d’Auvergne

Argument string: “classic”

Description: The classic style draws the backbone of a protein in a cylindrical bond 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:

Please see Table 17.22 on page 514.

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.

Please see Table 17.23 on page 515.

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.

Please see Table 17.24 on page 516.

Prompt examples

To create a Molmol macro mapping the order parameter values, S^2 , onto the structure using the classic style:

```
relax> molmol.macro_write('s2')
```

```
relax> molmol.macro_write(data_type='s2')

relax> molmol.macro_write(data_type='s2',
    style="classic", file='s2.mac', dir='
molmol')
```

17.2.88 molmol.ribbon

MOLMOL

Synopsis

Apply the Molmol ribbon style.

Defaults

molmol.ribbon()

Description

This 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
```

Prompt examples

To apply the ribbon style to the PDB file loaded, type:

```
relax> molmol.ribbon()
```

Then only the atoms and bonds of the geometric object are selected and the ‘ball/stick’ style applied:

17.2.89 molmol.tensor_pdb

SelectAtom ‘0’

SelectBond ‘0’

SelectAtom ‘:TNS’

SelectBond ‘:TNS’

XMacStand ball_stick.mac

Synopsis

Display the diffusion tensor PDB geometric object over the loaded PDB.

The appearance is finally touched up:

Defaults

molmol.tensor_pdb(file=None)

RadiusAtom 1

SelectAtom ‘:TNS@C*’

RadiusAtom 1.5

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 overlaid with the loaded PDB file and displayed within Molmol. The PDB file containing the geometric object must be created using the complementary structure.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
```

17.2.90 molmol.view

Synopsis

View the collection of molecules from the loaded PDB file.

Defaults

molmol.view()

Description

This will simply launch Molmol.

Prompt examples

```
relax> molmol.view()
```

17.2.91 monte_carlo.create_data

Synopsis

Create the Monte Carlo simulation data.

Defaults

monte_carlo.create_data(method='back_calc', distribution='measured', fixed_error=None)

Keyword arguments

method: The simulation method.

distribution: The error distribution method.

fixed_error: The fixed value to use when distribution is set to 'fixed'.

Description

The method can either be set to back calculation (Monte Carlo) or direct (bootstrapping), the choice of which determines the simulation type. If the values or parameters are calculated rather than minimised, this option will have no effect. Errors should only be propagated via Monte Carlo simulations if errors have been measured.

For error analysis, the method should be set to back calculation 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 calculation, this function should only be called after the model is fully minimised.

The simulation type can be changed by setting the method to direct. This will result in bootstrapping 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 user 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.

The errors generated per simulation can either be generated individual per datapoint and drawn from a gauss distribution described by the standard deviation of the individual point, or it can be generated from a overall gauss distribution described by the standard deviation of the goodness of fit, where $SD_{fit} = \sqrt{\chi^2/(N-p)}$. The

last possibility is to supply a fixed value of the standard deviation, from which gauss distribution to draw errors from.

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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```

relax> minimise.grid_search(inc=11)
          # Step 2.

relax> minimise.execute('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.execute('newton')
          # Step 6.

relax> eliminate()
          # Step 7.

relax> monte_carlo.error_analysis()
          # Step 8.

An example for reduced spectral density mapping is:

relax> minimise.calculate()
          # Step 2.

relax> monte_carlo.setup(number=500)
          # Step 3.

relax> monte_carlo.create_data(method='back_calc')      # Step 4.

relax> minimise.calculate()
          # Step 6.

relax> monte_carlo.error_analysis()
          # Step 8.

```

17.2.92 monte_carlo.error_analysis



Synopsis

Calculate parameter errors from the Monte Carlo simulations.

Defaults

`monte_carlo.error_analysis()`

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.

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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```
relax> minimise.grid_search(inc=11)
# Step 2.

relax> minimise.execute('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.execute('newton')
# Step 6.

relax> eliminate()
# Step 7.

relax> monte_carlo.error_analysis()
# Step 8.
```

An example for reduced spectral density mapping is:

```
relax> minimise.calculate()
# Step 2.

relax> monte_carlo.setup(number=500)
# Step 3.

relax> monte_carlo.create_data(method='back_calc')
# Step 4.

relax> minimise.calculate()
# Step 6.

relax> monte_carlo.error_analysis()
# Step 8.
```

17.2.93 monte_carlo.initial_values



Synopsis

Set the initial simulation parameter values.

Defaults

monte_carlo.initial_values()

Description

This 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 user functions `minimise.grid_search` and `minimise.execute` 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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```
relax> minimise.grid_search(inc=11)
          # Step 2.

relax> minimise.execute('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.execute('newton')
          # Step 6.

relax> eliminate()
          # Step 7.

relax> monte_carlo.error_analysis()
          # Step 8.
```

An example for reduced spectral density mapping is:

```
relax> minimise.calculate()
          # Step 2.
```

```

relax> monte_carlo.setup(number=500)
      # Step 3.

relax> monte_carlo.create_data(method='
back_calc')           # Step 4.

relax> minimise.calculate()
      # Step 6.

relax> monte_carlo.error_analysis()
      # Step 8.

```

17.2.94 monte_carlo.off



Synopsis

Turn the Monte Carlo simulations off.

Defaults

monte_carlo.off()

Description

This will turn off the Monte Carlo simulations so that subsequent optimisation will operate directly on the model parameters and not on the simulations.

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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```
relax> minimise.grid_search(inc=11)
          # Step 2.

relax> minimise.execute('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.execute('newton')
          # Step 6.

relax> eliminate()
          # Step 7.

relax> monte_carlo.error_analysis()
          # Step 8.
```

An example for reduced spectral density mapping is:

```
relax> minimise.calculate()
          # Step 2.

relax> monte_carlo.setup(number=500)
          # Step 3.

relax> monte_carlo.create_data(method='back_calc')
          # Step 4.

relax> minimise.calculate()
          # Step 6.

relax> monte_carlo.error_analysis()
          # Step 8.
```

17.2.95 monte_carlo.on



Synopsis

Turn the Monte Carlo simulations on.

Defaults

monte_carlo.on()

Description

This will turn on the Monte Carlo simulations so that subsequent optimisation will operate on the simulations rather than on the real model parameters.

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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```
relax> minimise.grid_search(inc=11)
          # Step 2.

relax> minimise.execute('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.execute('newton')
          # Step 6.

relax> eliminate()
          # Step 7.

relax> monte_carlo.error_analysis()
          # Step 8.
```

An example for reduced spectral density mapping is:

```
relax> minimise.calculate()
          # Step 2.

relax> monte_carlo.setup(number=500)
          # Step 3.

relax> monte_carlo.create_data(method='back_calc')      # Step 4.

relax> minimise.calculate()
          # Step 6.

relax> monte_carlo.error_analysis()
          # Step 8.
```

17.2.96 monte_carlo.setup



Synopsis

Set up the Monte Carlo simulations.

Defaults

monte_carlo.setup(number=500)

Keyword arguments

number: The number of Monte Carlo simulations.

Description

This 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 using the prompt UI mode which includes only the functions required for implementing the above steps is:

```
relax> minimise.grid_search(inc=11)
          # Step 2.

relax> minimise.execute('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.execute('newton')
          # Step 6.

relax> eliminate()
          # Step 7.

relax> monte_carlo.error_analysis()
          # Step 8.
```

An example for reduced spectral density mapping is:

```
relax> minimise.calculate()
          # Step 2.

relax> monte_carlo.setup(number=500)
          # Step 3.

relax> monte_carlo.create_data(method='back_calc')
          # Step 4.

relax> minimise.calculate()
          # Step 6.

relax> monte_carlo.error_analysis()
          # Step 8.
```

17.2.97 n_state_model.CoM



Synopsis

The defunct centre of mass (CoM) analysis.

Defaults

```
n_state_model.CoM(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: Manually specify the CoM of the initial position prior to the N rotations to the positions of the N states. This is optional.

Description

WARNING: This analysis is now defunct!

This 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 of mass is supplied, it must also be a vector of floating point numbers (of length 3). If the centre of mass is not supplied, then the CoM will be calculated from the selected parts of a previously loaded structure.

Prompt 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])
```

17.2.98 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(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

WARNING: This analysis is now defunct!

This 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 to one of these values. 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 multiplied by the scaling factor. 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 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 scaling factor). 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.

17.2.99 n_state_model.elim_no_prob



Synopsis

Eliminate the structures or states with no probability.

Defaults

`n_state_model.elim_no_prob()`

Description

This will simply remove the structures from the N-state analysis which have an optimised probability of zero.

Prompt examples

Simply type:

`relax> n_state_model.elim_no_prob(N=8)`

17.2.100 n_state_model.number_of_states
**Synopsis**

Set the number of states in the N-state model.

Defaults

`n_state_model.number_of_states(N=1)`

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 - the ensemble size.

Prompt examples

To set up an 8-state model, type:

```
relax> n_state_model.number_of_states(N=8)
```

17.2.101 n_state_model.ref_domain
**Synopsis**

Set the reference domain for the ‘2-domain’ N-state model.

Defaults

`n_state_model.ref_domain(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.

Prompt 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')
```

17.2.102 n_state_model.select_model



Synopsis

Select the N-state model type and set up the model.

Defaults

```
n_state_model.select_model(model='population')
```

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:

'population' – The N-state model whereby only populations are optimised. The structures loaded into relax are assumed to be fixed, *i.e.* the orientations are not optimised, or 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 tensors - {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 parameters of each alignment tensor {Axx, Ayy, Axy, Axz, Ayz, ...}.

'2-domain' – The N-state model for a system of two domains, where one domain experiences a reduced tensor.

Prompt examples

To analyse populations of states, type:

```
relax> n_state_model.select_model(model='populations')
```



17.2.103 noe.read_restraints



Synopsis

Read NOESY or ROESY restraints from a file.

Defaults

```
noe.read_restraints(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 restraint 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

The format of the file will be automatically determined, 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 column numbers. The proton names need to be in the spin ID string format.

Prompt 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)
```

17.2.104 noe.spectrum_type**Synopsis**

Set the steady-state NOE spectrum type for pre-loaded peak intensities.

Defaults

```
noe.spectrum_type(spectrum_type=None, spectrum_id=None)
```

Keyword arguments

spectrum_type: The type of steady-state NOE spectrum, one of ‘ref’ for the reference spectrum or ‘sat’ for the saturated spectrum.

spectrum_id: The spectrum ID string.

Description

The spectrum type can be one of the following:

The steady-state NOE reference spectrum.

The steady-state NOE spectrum with proton saturation turned on.

Peak intensities should be loaded before this user function via the spectrum.read_intensities user function. The intensity values will then be associated with a spectrum ID string which can be used here.

17.2.105 palmer.create**Synopsis**

Create the Modelfree4 input files.

Defaults

```
palmer.create(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 binary name 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.

17.2.106 palmer.execute



Synopsis

Perform a model-free optimisation using Modelfree4.

Defaults

```
palmer.execute(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 binary name 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.

17.2.107 palmer.extract**Synopsis**

Extract data from the Modelfree4 ‘mfout’ star formatted file.

Defaults

```
palmer.extract(dir=None)
```

Keyword arguments

dir: The directory where the file ‘mfout’ is found.

Description

The model-free results will be extracted from the Modelfree4 results file ‘mfout’ located in the given directory.

17.2.108 paramag.centre**Synopsis**

Specify which atom is the paramagnetic centre.

Defaults

```
paramag.centre(pos=None, atom_id=None, pipe=None,
               verbosity=1, fix=True, ave_pos=True, force=False)
```

Keyword arguments

pos: The atomic position of the paramagnetic centre.

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 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.

If the average position 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.

Prompt 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)
```

17.2.109 pcs.back_calc



Synopsis

Back calculate the pseudo-contact shifts.

Defaults

```
pcs.back_calc(align_id=None)
```

Keyword arguments

align_id: The alignment ID string.

Description

This will back calculate the pseudo-contact shifts if the paramagnetic centre, temperature and magnetic field strength has been specified, an alignment tensor is present, and atomic positions have been loaded into the relax data store.

17.2.110 `pcs.calc_q_factors`



Synopsis

Calculate the PCS Q factor for the selected spins.

Defaults

```
pcs.calc_q_factors(spin_id=None, verbosity=1)
```

Keyword arguments

`spin_id`: The spin ID string for restricting to subset of all selected spins.

`verbosity`: The amount of information to print out. Set to zero to silence the user function, or one to see all messages.

Description

For this to work, the back-calculated PCS data must first be generated by the analysis specific code. Otherwise a warning will be given.

Prompt 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')
```

17.2.111 `pcs.copy`



Synopsis

Copy PCS data from one data pipe to another.

Defaults

```
pcs.copy(pipe_from=None, pipe_to=None, align_id=None,
         back_calc=True)
```

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.

`back_calc`: A flag which if True will cause any back-calculated PCSs present to also be copied with the real values and errors.

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.

Prompt 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')
```

17.2.112 pcs.corr_plot**Synopsis**

Generate a correlation plot of the measured vs. the back-calculated PCSs.

Defaults

```
pcs.corr_plot(format='grace', title=None, subtitle=None,
file='pcs_corr_plot.agr', dir=None, force=False)
```

Keyword arguments

format: The format of the plot data.

title: The title for the plot, overriding the default.

subtitle: The subtitle for the plot, overriding the default.

file: The name of the Grace file to create.

dir: The directory name.

force: A flag which if True will cause the file to be overwritten.

Description

Two formats are currently supported. If format is set to ‘grace’, then a Grace plot file will be created. If the format is not set then a plain text list of the measured and back-calculated data will be created.

Prompt 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)
```

17.2.113 pcs.delete**Synopsis**

Delete the PCS data corresponding to the alignment ID.

Defaults

```
pcs.delete(align_id=None)
```

Keyword arguments

align_id: The alignment ID string of the data to delete.

Description

This will delete all PCS data associated with the alignment ID in the current data pipe.

Prompt examples

To delete the PCS data corresponding to align_id=‘PH_gel’, type:

```
relax> pcs.delete('PH_gel')
```

17.2.114 pcs.display**Synopsis**

Display the PCS data corresponding to the alignment ID.

Defaults

`pcs.display(align_id=None, bc=False)`

Keyword arguments

`align_id`: The alignment ID string.

`bc`: A flag which if set will display the back-calculated rather than measured RDCs.

Description

This will display all of the PCS data associated with the alignment ID in the current data pipe.

Prompt examples

To display the ‘phage’ PCS data, type:

```
relax> pcs.display('phage')
```

17.2.115 pcs.read**Synopsis**

Read the PCS data from file.

Defaults

`pcs.read(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

This will read PCS data from a file and associate it with an alignment ID, either a new ID or a preexisting one with no PCS data.

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 molecule name, residue number and name, and spin number and name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Prompt 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 4th column and the 1H in the 9th, 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)
```

17.2.116 pcs.set_errors



Synopsis

Set the errors for the PCSs.

Defaults

```
pcs.set_errors(align_id=None, spin_id=None, sd=0.1)
```

Keyword arguments

align_id: The optional alignment ID string.

spin_id: The optional spin ID string.

sd: The PCS standard deviation value in ppm.

Description

If the PCS errors have not already been read from a PCS data file or if they need to be changed, then the errors can be set via this user function.

17.2.117 pcs.structural_noise
**Synopsis**

Determine the PCS error due to structural noise via simulation.

Defaults

```
pcs.structural_noise(align_id=None, rmsd=0.2, sim_num=1000, file=None, dir=None, force=False)
```

Keyword arguments

align_id: The optional alignment ID string.

rmsd: The atomic position RMSD, in Å, to randomise the spin positions with for the simulations.

sim_num: The number of simulations, N, to perform to determine the structural noise component of the PCS errors.

file: The optional name of the Grace file to plot the structural errors verses the paramagnetic centre to spin distances.

dir: The directory name to place the Grace file into.

force: A flag which if True will cause the file to be overwritten.

Description

The analysis of the pseudo-contact shift is influenced by two significant sources of noise - that of the NMR experiment and structural noise from the 3D molecular structure used. The closer the spin to the paramagnetic centre, the greater the influence of structural noise. This distance dependence is governed by the equation:

$$\sigma_{\text{dist}} = \frac{\sqrt{3} * |\delta| * \text{RMSD}}{r},$$

where σ_{dist} is the distance component of the structural noise as a standard deviation, δ is the PCS value, RMSD is the atomic position root-mean-square deviation, and r is the paramagnetic centre to spin distance. When close to the paramagnetic centre, this error source

can exceed that of the NMR experiment. The equation for the angular component of the structural noise is more complicated. The PCS error is influenced by distance, angle in the alignment frame, and the magnetic susceptibility tensor.

For the simulation the following must already be set up in the current data pipe:

The position of the paramagnetic centre.

The alignment and magnetic susceptibility tensor.

The protocol for the simulation is as follows:

The lanthanide or paramagnetic centre position will be fixed. Its motion is assumed to be on the femto- to pico- and nanosecond timescales. Hence the motion is averaged over the evolution of the PCS and can be ignored.

The positions of the nuclear spins will be randomised N times. For each simulation a random unit vector will be generated. Then a random distance along the unit vector will be generated by sampling from a Gaussian distribution centered at zero, the original spin position, with a standard deviation set to the given RMSD. Both positive and negative displacements will be used.

The PCS for the randomised position will be back calculated.

The PCS standard deviation will be calculated from the N randomised PCS values.

The standard deviation will both be stored in the spin container data structure in the relax data store as well as being added to the already present PCS error (using variance addition). This will then be used in any optimisations involving the PCS.

If the alignment ID string is not supplied, the procedure will be applied to the PCS data from all alignments.

17.2.118 pcs.weight**Synopsis**

`Set optimisation weights on the PCS data.`

Defaults

`pcs.weight(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 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.

17.2.119 pcs.write**Synopsis**

`Write the PCS data to file.`

Defaults

`pcs.write(align_id=None, file=None, dir=None, bc=False, force=False)`

Keyword arguments

- `align_id`: The alignment ID string.
- `file`: The name of the file.
- `dir`: The directory name.

`bc`: A flag which if set will write out the back-calculated rather than measured RDCs.

`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 alignment ID is required for selecting which PCS data set will be written to file.

17.2.120 pipe.bundle**Synopsis**

The grouping of data pipes into a bundle.

Defaults

```
pipe.bundle(bundle=None, pipe=None)
```

Keyword arguments

bundle: The pipe bundle is a special grouping or clustering of data pipes.

pipe: The name of the data pipe to add to the bundle.

Description

Data pipes can be grouped or clustered together through special structures known as pipe bundles. If the specified data pipe bundle does not currently exist, it will be created.

Prompt examples

To add the data pipes ‘test 1’, ‘test 2’, and ‘test 3’ to the bundle ‘first analysis’, type the following:

```
relax> pipe.bundle('first analysis 1', 'test 1')
```

```
relax> pipe.bundle('first analysis 1', 'test 2')
```

```
relax> pipe.bundle('first analysis 1', 'test 3')
```

17.2.121 pipe.change_type**Synopsis**

Change the type of the current data pipe.

Defaults

```
pipe.change_type(pipe_type=None)
```

Keyword arguments

pipe_type: The type of data pipe.

Description

The data pipe type must be one of the following:

‘ct’ – Consistency testing.

‘frame_order’ – Frame Order theories.

‘jw’ – Reduced spectral density mapping.

‘hybrid’ – Special hybrid pipe.

‘mf’ – Model-free analysis.

‘N-state’ – N-state model or ensemble analysis.

‘noe’ – Steady state NOE calculation.

‘relax_disp’ – Relaxation dispersion.

‘relax_fit’ – Relaxation curve fitting.

Prompt examples

To change the type of the current ‘frame_order’ data pipe to the N-state model, type one of:

```
relax> pipe.change_type('N-state')
```

```
relax> pipe.change_type(pipe_type='N-state')
```

17.2.122 pipe.copy



Synopsis

Copy a data pipe.

Defaults

```
pipe.copy(pipe_from=None, pipe_to=None, bundle_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.

bundle_to: If given, the new data pipe will be grouped into this bundle.

Description

This allows the contents of a data pipe to be copied. If the source data pipe is not set, the current data pipe will be assumed. The target data pipe must not yet exist.

The optional bundling allows the newly created data pipe to be placed into either a new or existing data pipe bundle. If not specified, then the copied data pipe will not be associated with a bundle.

Prompt 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')
```

17.2.123 pipe.create



Synopsis

Add a new data pipe to the relax data store.

Defaults

```
pipe.create(pipe_name=None, pipe_type=None, bundle=None)
```

Keyword arguments

pipe_name: The name of the data pipe.

pipe_type: The type of data pipe.

bundle: The optional pipe bundle is a special grouping or clustering of data pipes. If this is specified, the newly created data pipe will be added to this bundle.

Description

The data pipe name can be any string however the data pipe type can only be one of the following:

‘ct’ – Consistency testing,

‘frame_order’ – The Frame Order theories,

‘jw’ – Reduced spectral density mapping,

‘hybrid’ – A special hybrid pipe,

‘mf’ – Model-free analysis,

‘N-state’ – N-state model or ensemble analysis,

‘noe’ – Steady state NOE calculation,

‘relax_disp’ – Relaxation dispersion curve fitting,

‘relax_fit’ – Relaxation curve fitting,

The pipe bundling concept is simply a way of grouping data pipes together. This is useful for a number of purposes:

The grouping or categorisation of data pipes, for example when multiple related analyses are performed.

In the auto-analyses, as all the data pipes that they spawn are bound together within the original bundle.

In the graphical user interface mode as analysis tabs are linked to specific pipe bundles.

Prompt examples

To set up a model-free analysis data pipe with the name 'm5', type:

```
relax> pipe.create('m5', 'mf')
```

17.2.124 pipe.current



Synopsis

Print the name of the current data pipe.

Defaults

```
pipe.current()
```

Prompt examples

To run the user function, type:

```
relax> pipe.current()
```

17.2.125 pipe.delete**Synopsis**

Delete a data pipe from the relax data store.

Defaults

`pipe.delete(pipe_name=None)`

Keyword arguments

`pipe_name`: The name of the data pipe to delete.

Description

This will permanently remove the data pipe and all of its contents from the relax data store. If the pipe name is not given, then all data pipes will be deleted.

17.2.126 pipe.display**Synopsis**

Print a list of all the data pipes.

Defaults

`pipe.display()`

Prompt examples

To run the user function, type:

`relax> pipe.display()`



17.2.127 pipe.hybridise

**Synopsis**

Create a hybrid data pipe by fusing a number of other data pipes.

Defaults

```
pipe.hybridise(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. 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.

Prompt examples

The two data pipes ‘`N_sphere`’ and ‘`C_ellipsoid`’ could 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'])
```

17.2.128 pipe.switch

**Synopsis**

Switch between the data pipes of the relax data store.

**Defaults**

```
pipe.switch(pipe_name=None)
```

Keyword arguments

pipe_name: The name of the data pipe.

Description

This will switch between the various data pipes within the relax data store.

Prompt examples

To switch to the ‘`ellipsoid`’ data pipe, type:

```
relax> pipe.switch('ellipsoid')
```

```
relax> pipe.switch(pipe_name='ellipsoid')
```

17.2.129 pymol.cartoon**Synopsis**

Apply the PyMOL cartoon style and colour by secondary structure.

Defaults

pymol.cartoon()

Description

This 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.

Prompt examples

To apply this user function, type:

relax> pymol.cartoon()

17.2.130 pymol.clear_history**Synopsis**

Clear the PyMOL command history.

Defaults

pymol.clear_history()

Description

This will clear the Pymol history from memory.

17.2.131 pymol.command**Synopsis**

Execute a user supplied PyMOL command.

Defaults

```
pymol.command(command=None)
```

Keyword arguments

command: The PyMOL command to execute.

Description

This allows PyMOL commands to be passed to the program. This can be useful for automation or scripting.

Prompt examples

To reinitialise the PyMOL instance, type:

```
relax> pymol.command("reinitialise")
```

17.2.132 pymol.cone_pdb**Synopsis**

Display the cone PDB geometric object.

Defaults

```
pymol.cone_pdb(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’
```

17.2.133 pymol.frame_order



The user function will not only search for these files, but also all *.gz and *.bz2 versions of the average position and frame order representations. This is to support all output files from the frame_order.pdb_model user function.

Synopsis

Display the frame order results from the frame_order.pdb_model and frame_order.simulate user functions.

Defaults

```
pymol.frame_order(ave_pos='ave_pos', rep='frame_order',
sim='simulation.pdb.gz', dir=None)
```

Keyword arguments

ave_pos: The file root of the 3D structure PDB file for the molecular structure with the moving domains shifted to the average position.

rep: The file root of the PDB file for the geometric object representation of the frame order dynamics.

sim: The full name the Brownian simulation PDB file.

dir: The directory where the files are located.

Description

This user function is designed to be combined with the frame_order.pdb_model and frame_order.simulate user functions. It will take the two PDB representations created by frame_order.pdb_model, the molecular structure with the averaged domain positions and the frame order dynamics representation files, and the Brownian simulation PDB file and display them in PyMOL. Rather than loading the three representations into PyMOL manually, this user function will change the representation to improve visualisation.

For the PDB files, if the file roots are left to the defaults then the following files will be loaded:

Average position – The default is to load the ‘ave_pos.pdb’ and ‘ave_pos_sim.pdb’ files.

Frame order motional representation –
The is to load the ‘frame_order.pdb’, ‘frame_order_A.pdb’, ‘frame_order_B.pdb’, ‘frame_order_sim.pdb’, ‘frame_order_sim_A.pdb’ and ‘frame_order_sim_B.pdb’ files, if present.

Brownian simulation – The default is to load the ‘simulation.pdb.gz’ file.

Colour

17.2.134 pymol.macro_apply



Synopsis

Execute PyMOL macros.

Defaults

```
pymol.macro_apply(data_type=None, style='classic',
colour_start_name=None, colour_start_rgb=None,
colour_end_name=None, colour_end_rgb=None,
colour_list=None)
```

Keyword arguments

data_type: The data type to map to the structure.

style: The style of the macro.

colour_start_name: The name of the starting colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the starting colour RGB colour array cannot be given.

colour_start_rgb: The starting colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the starting colour name cannot be given.

colour_end_name: The name of the ending colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the ending colour RGB colour array cannot be given.

colour_end_rgb: The ending colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the ending colour name cannot be given.

colour_list: The colour list to search for the colour names. This can be either ‘molmol’ or ‘x11’.

Description

This allows spin specific values to be mapped to a structure through PyMOL macros. Currently only the ‘classic’ style, which is described below, is available.

The values are coloured based on a linear colour gradient which is specified through starting and ending colours. These 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, ‘white’ and [1.0, 1.0, 1.0] both select the same colour. Leaving both colours unset 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 name. To explicitly select these lists, set the colour list to either ‘molmol’ or ‘x11’.

Model-free classic style

Creator: Edward d’Auvergne

Argument string: “classic”

Description: The classic style draws the backbone of a protein in a cylindrical bond 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:

Please see Table 17.22 on page 514.

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.

Please see Table 17.23 on page 515.

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.

Please see Table 17.24 on page 516.

Prompt examples

To map the order parameter values, S^2 , onto the structure using the classic style, type:

```
relax> pymol.macro_apply('s2')
```

```
relax> pymol.macro_apply(data_type='s2')  
relax> pymol.macro_apply(data_type='s2',  
    style="classic")
```

17.2.135 pymol.macro_run



Synopsis

Open and execute the PyMOL macro file.

Defaults

```
pymol.macro_run(file=None, dir='pymol')
```

Keyword arguments

file: The name of the PyMOL macro file.

dir: The directory name.

Description

This user function is for opening and running a PyMOL macro located within a text file.

Prompt examples

To execute the macro file ‘s2.pml’ located in the directory ‘pymol’, type:

```
relax> pymol.macro_run(file='s2.pml')
```

```
relax> pymol.macro_run(file='s2.pml', dir='  
    pymol')
```

17.2.136 pymol.macro_write



Synopsis

Create PyMOL macros.

Defaults

```
pymol.macro_write(data_type=None, style='classic',
colour_start_name=None, colour_start_rgb=None,
colour_end_name=None, colour_end_rgb=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_name: The name of the starting colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the starting colour RGB colour array cannot be given.

colour_start_rgb: The starting colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the starting colour name cannot be given.

colour_end_name: The name of the ending colour of the linear colour gradient. This can be either one of the X11 or one of the Molmol colour names listed in the description. If this is set, then the ending colour RGB colour array cannot be given.

colour_end_rgb: The ending colour of the linear colour gradient. This is an RGB colour array with values ranging from 0 to 1. If this is set, then the ending colour name cannot be given.

colour_list: The colour list to search for the colour names. This can be either 'molmol' or 'x11'.

file: The optional name of the file.

dir: The optional directory to save the file to.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

This 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 starting and ending colours. These 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, 'white' and [1.0, 1.0, 1.0] both select the same colour. Leaving both colours unset 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 name. To explicitly select these lists, set the colour list to either 'molmol' or 'x11'.

Model-free classic style

Creator: Edward d'Auvergne

Argument string: "classic"

Description: The classic style draws the backbone of a protein in a cylindrical bond 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:

Please see Table 17.22 on page 514.

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.

Please see Table 17.23 on page 515.

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.

Please see Table 17.24 on page 516.

Prompt examples

To create a PyMOL macro mapping the order parameter values, S^2 , onto the structure using the classic style, type:

```
relax> pymol.macro_write('s2')
```

```
relax> pymol.macro_write(data_type='s2')

relax> pymol.macro_write(data_type='s2',
    style="classic", file='s2.pml', dir='
    pymol')
```

17.2.137 pymol.tensor_pdb



Synopsis

Display the diffusion tensor PDB geometric object over the loaded PDB.

Defaults

pymol.tensor_pdb(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 overlaid with the loaded PDB file and displayed within PyMOL. The PDB file containing the geometric object must be created using the complementary structure.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’
```

17.2.138 **pymol.vector_dist**



Synopsis

Display the PDB file representation of the XH vector distribution.

Defaults

```
pymol.vector_dist(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 overlaid with the macromolecule within PyMOL. The PDB file containing the vector distribution must be created using the complementary structure.create_vector_dist user function.

The vector distribution PDB file is read in using the command:

```
load file
```

17.2.139 pymol.view**17.2.140 rdc.back_calc****Synopsis****Synopsis**

View the collection of molecules from the loaded PDB file.

Defaults

Back calculate the residual dipolar couplings.

pymol.view()

Defaults

rdc.back_calc(align_id=None)

Keyword arguments**Description**

align_id: The alignment ID string.

This will simply launch Pymol.

Description**Prompt examples**

This will back calculate the residual dipolar couplings (RDCs) if an alignment tensor is present and inter-dipole vectors have been loaded into the relax data store.

relax> pymol.view()

17.2.141 rdc.calc_q_factors**Synopsis**

Calculate the RDC Q factor for the selected spins.

Defaults

```
rdc.calc_q_factors(spin_id=None, verbosity=1)
```

Keyword arguments

spin_id: The spin ID string for restricting to subset of all selected spins.

verbosity: The amount of information to print out. Set to zero to silence the user function, or one to see all messages.

Description

For this to work, the back-calculated RDC data must first be generated by the analysis specific code. Otherwise a warning will be given.

Prompt 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')
```

17.2.142 rdc.copy**Synopsis**

Copy RDC data from one data pipe to another.

Defaults

```
rdc.copy(pipe_from=None, pipe_to=None, align_id=None,
        back_calc=True)
```

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.

back_calc: A flag which if True will cause any back-calculated RDCs present to also be copied with the real values and errors.

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.

Prompt 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')
```

17.2.143 rdc.corr_plot**Synopsis**

Generate a correlation plot of the measured vs. the back-calculated RDCs.

Defaults

```
rdc.corr_plot(format='grace', title=None, subtitle=None,
file='rdc_corr_plot.agr', dir=None, force=False)
```

Keyword arguments

format: The format of the plot data.

title: The title for the plot, overriding the default.

subtitle: The subtitle for the plot, overriding the default.

file: The name of the Grace file to create.

dir: The directory name.

force: A flag which if True will cause the file to be overwritten.

Description

Two formats are currently supported. If format is set to ‘grace’, then a Grace plot file will be created. If the format is not set then a plain text list of the measured and back-calculated data will be created.

Prompt 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)
```

17.2.144 rdc.delete**Synopsis**

Delete the RDC data corresponding to the alignment ID.

Defaults

```
rdc.delete(align_id=None)
```

Keyword arguments

align_id: The alignment ID string of the data to delete.

Description

This will delete all RDC data associated with the alignment ID in the current data pipe.

Prompt examples

To delete the RDC data corresponding to align_id='PH_gel', type:

```
relax> rdc.delete('PH_gel')
```

17.2.145 rdc.display



Synopsis

Display the RDC data corresponding to the alignment ID.

Defaults

```
rdc.display(align_id=None, bc=False)
```

Keyword arguments

align_id: The alignment ID string.

bc: A flag which if set will display the back-calculated rather than measured RDCs.

Description

This will display all of the RDC data associated with the alignment ID in the current data pipe.

Prompt examples

To display the ‘phage’ RDC data, type:

```
relax> rdc.display('phage')
```

17.2.146 rdc.read



Synopsis

Read the RDC data from file.

Defaults

```
rdc.read(align_id=None, file=None, dir=None, data_type='D', spin_id1_col=1, spin_id2_col=2, data_col=None, error_col=None, sep=None, neg_g_corr=False, absolute=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.

data_type: Specify if the RDC data is in the D or 2D format, or the T = J + D format.

spin_id1_col: The spin ID string column for the first spin.

spin_id2_col: The spin ID string column for the second spin.

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator (the default is white space).

neg_g_corr: A flag which is used to correct for the negative gyromagnetic ratio of 15N. If set to True, all RDC values will be inverted prior to being stored in the relax data store.

absolute: A flag which indicates that the loaded RDCs are signless.

Description

This will read RDC data from a file and associate it with an alignment ID, either a new ID or a preexisting one with no RDC data.

The data type is used to specify how the RDC is defined. It can be set to a number of values:

‘D’ means that the splitting in the aligned sample was taken as J + D.

‘2D’ means that the splitting in the aligned sample was assumed to be $J + 2D$.

‘T’ means that the file contains $T = J + D$ values.

Internally, relax uses the D notation. Therefore if set to ‘2D’, the values will be doubled when read in. If the ‘T’ data type is specified, then J couplings must be present for this data to be of any use.

If the negative gyromagnetic ratio correction flag is set, a sign inversion will be applied to all RDC values to be loaded. This is sometimes needed for 15N if the data is not compensated for the negative gyromagnetic ratio.

The absolute RDCs flag is used for RDCs in which the sign is unknown. All absolute RDCs loaded will be converted to positive values.

Prompt 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, assuming $J + D$ was measured, type one of:

```
relax> rdc.read('phage', 'rdc_err.txt',
    data_type='D', error_col=5)
```

```
relax> rdc.read(align_id='phage', file='
    rdc_err.txt', data_type='D', error_col
    =5)
```

17.2.147 rdc.set_errors



Synopsis

Set the errors for the RDCs.

Defaults

```
rdc.set_errors(align_id=None, spin_id1=None, spin_id2=
None, sd=1.0)
```

Keyword arguments

`align_id`: The optional alignment ID string.

`spin_id1`: The optional spin ID string of the first spin.

`spin_id2`: The optional spin ID string of the second spin.

`sd`: The RDC standard deviation value in Hertz.

Description

If the RDC errors have not already been read from a RDC data file or if they need to be changed, then the errors can be set via this user function.

17.2.148 rdc.weight**Synopsis**

`rdc.weight(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 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.

17.2.149 rdc.write**Synopsis**

`rdc.write(align_id=None, file=None, dir=None, bc=False, force=False)`

Keyword arguments

`align_id`: The alignment ID string.
`file`: The name of the file.
`dir`: The directory name.

`bc`: A flag which if set will write out the back-calculated rather than measured RDCs.

`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 alignment ID is required for selecting which RDC data set will be written to file.

17.2.150 relax_data.back_calc**Synopsis**

Back calculate the relaxation data at the given frequency.

Defaults

```
relax_data.back_calc(ri_id=None, ri_type=None, freq=None)
```

Keyword arguments

ri_id: The relaxation data ID string.

ri_type: The relaxation data type, ie ‘R1’, ‘R2’, or ‘NOE’.

freq: The spectrometer frequency in Hz.

Description

This allows relaxation data of the given type and frequency to be back calculated from the model parameter values. If the relaxation data ID, type and frequency are not given, then relaxation data matching that currently loaded in the relax data store will be back-calculated.

17.2.151 relax_data.copy**Synopsis**

Copy relaxation data from one pipe to another.

Defaults

```
relax_data.copy(pipe_from=None, pipe_to=None, ri_id=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_id: The relaxation data ID string.

Description

This will copy relaxation data from one data pipe to another. If the relaxation ID data string is not given then all relaxation data will be copied, otherwise only a specific data set will be copied.

Prompt 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)
```

```
relax> relax_data.copy(pipe_from='m1',
    pipe_to='m9', ri_id=None)
```

To copy only the NOE relaxation data with the ID string of ‘NOE_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_id='NOE_800')
```



17.2.152 relax_data.delete**Synopsis**

Delete the data corresponding to the relaxation data ID string.

Defaults

```
relax_data.delete(ri_id=None)
```

Keyword arguments

`ri_id`: The relaxation data ID string.

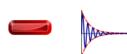
Description

The relaxation data corresponding to the given relaxation data ID string will be removed from the current data pipe.

Prompt examples

To delete the relaxation data corresponding to the ID ‘NOE_600’, type:

```
relax> relax_data.delete('NOE_600')
```

17.2.153 relax_data.display**Synopsis**

Display the data corresponding to the relaxation data ID string.

Defaults

```
relax_data.display(ri_id=None)
```

Keyword arguments

`ri_id`: The relaxation data ID string.

Description

This will display the relaxation data corresponding to the given ID.

Prompt examples

To display the NOE relaxation data at 600 MHz with the ID string ‘NOE_600’, type:

```
relax> relax_data.display('NOE_600')
```

17.2.154 relax_data.peak_intensity_type



Synopsis

Specify if heights or volumes were used to measure the peak intensities.

Defaults

```
relax_data.peak_intensity_type(ri_id=None, type='height')
```

Keyword arguments

ri_id: The relaxation data ID string.

type: The peak intensity type.

Description

This is essential for BMRB data deposition. It is used to specify whether peak heights or peak volumes were measured. The two currently allowed values for the peak intensity type are ‘height’ and ‘volume’.

17.2.155 relax_data.read



Synopsis

Read R₁, R₂, NOE, or R_{2eff} relaxation data from a file.

Defaults

```
relax_data.read(ri_id=None, ri_type=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_id: The relaxation data ID string. This must be a unique identifier.

ri_type: The relaxation data type, i.e. ‘R1’, ‘R2’, ‘NOE’, or ‘R2eff’.

frq: The exact proton frequency of the spectrometer in Hertz. See the ‘sfrq’ parameter in the Varian procpar file or the ‘SF01’ parameter in the Bruker acqus file.

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

This will load the relaxation data into the relax data store. The data is associated with the spectrometer frequency in Hertz. For subsequent analysis, this frequency must be set to the exact field strength. This value is stored in the ‘sfreq’ parameter in the Varian procpar file or the ‘SF01’ parameter in the Bruker acqus file.

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 molecule name, residue number, residue name, spin number and/or spin name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID string can be used to restrict the reading to certain spin types, for example only 15N spins when only residue information is in the file.

Prompt 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', 'NOE', 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_id='NOE_600',
    ri_type='NOE', 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 R₂ 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', 'R2', 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_id='R2_800',
    ri_type='R2', 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 R₁ data out of the file ‘r1.out’ where the columns are separated by the symbol ‘%’

```
relax> relax_data.read('R1_300', 'R1', 300.1
    * 1e6, 'r1.out', sep='%')
```

17.2.156 relax_data.temp_calibration



Synopsis

Specify the per-experiment temperature calibration method used.

Defaults

```
relax_data.temp_calibration(ri_id=None, method=None)
```

Keyword arguments

ri_id: The relaxation data ID string.

method: The per-experiment temperature calibration method.

Description

For the proper measurement of relaxation data, per-experiment temperature calibration is essential. This user function is not for inputting standard MeOH/ethylene glycol/etc. calibration of a spectrometer - this temperature setting is of no use when you are running experiments which pump in large amounts of power into the probe head.

The R₁ experiment should be about the same temperature as a HSQC and hence be close to the standard MeOH/ethylene glycol spectrometer calibration. However the R₂ CPMG or spin lock and, to a lesser extent, the NOE pre-saturation pump a lot more power into the probe head. The power differences can either cause the temperature in the sample to be too high or too low. This is unpredictable as the thermometer used by the VT unit is next to the coils in the probe head and not inside the NMR sample. So the VT unit tries to control the temperature inside the probe head rather than in the NMR sample. However between the thermometer and the sample is the water of the sample, the glass of the NMR tube, the air gap where the VT unit controls air flow and the outside components of the probe head protecting the electronics. If the sample, the probe head or the VT unit is changed, this will have a different affect on the per-experiment temperature. The VT unit responds differently under different conditions and may sometimes over or under compensate by a couple of degrees. Therefore each relaxation data set from each spectrometer requires a per-experiment calibration.

Specifying the per-experiment calibration method is needed for BMRB data deposition. The currently allowed methods are:

```
'methanol',
'monoethylene glycol',
'no calibration applied'.
```

Other methods will be accepted if supplied.

17.2.157 relax_data.temp_control



Synopsis

Specify the temperature control method used.

Defaults

```
relax_data.temp_control(ri_id=None, method=None)
```

Keyword arguments

ri_id: The relaxation data ID string.

method: The control method.

Description

For the proper measurement of relaxation data, explicit temperature control techniques are essential. A number of factors can cause significant temperature fluctuations between individual relaxation experiments. This includes the daily temperature cycle of the room housing the spectrometer, different amounts of power for the individual experiments, etc.

The best methods for eliminating such problems are single scan interleaving and temperature compensation block. Single scan interleaving is the most powerful technique for averaging the temperature fluctuations not only across different experiments, but also across the entire measurement time. The application of off-resonance temperature compensation blocks at the start of the experiment is useful for the R₂ and will normalise the temperature between the individual experiments, but single scan or single fid interleaving is nevertheless required for normalising the temperature across the entire measurement.

Specifying the temperature control method is needed for BMRB data deposition. The currently allowed methods are:

```
'single scan interleaving',
'temperature compensation block',
'single scan interleaving and temperature
compensation block',
'single fid interleaving',
'single experiment interleaving',
'no temperature control applied'.
```

17.2.158 relax_data.type**Synopsis**

Set the type of relaxation data.

Defaults

```
relax_data.type(ri_id=None, ri_type=None)
```

Keyword arguments

ri_id: The relaxation data ID string of the data to set the frequency of.

ri_type: The relaxation data type, *i.e.* ‘R1’, ‘R2’, or ‘NOE’.

Description

This allows the type associated with the relaxation data to be either set or reset. This type must be one of ‘R1’, ‘R2’, or ‘NOE’.

17.2.159 relax_data.write**Synopsis**

Write relaxation data to a file.

Defaults

```
relax_data.write(ri_id=None, file=None, dir=None, bc=False, force=False)
```

Keyword arguments

ri_id: The relaxation data ID string.

file: The name of the file.

dir: The directory name.

bc: A flag which if True will cause the back-calculated data to be written to the file.

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 relaxation data ID string is required for selecting which relaxation data to write to file.

17.2.160 relax_disp.catia_execute**Synopsis**

Perform a relaxation dispersion optimisation using Flemming Hansen's CATIA.

Defaults

```
relax_disp.catia_execute(dir=None, binary='catia')
```

Keyword arguments

dir: The directory containing all of the CATIA input files.

binary: The name of the executable CATIA program file.

Description

CATIA will be executed as

```
$ catia < Fit.catia
```

If you would like to use a different CATIA executable file, change the binary name 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.

17.2.161 relax_disp.catia_input**Synopsis**

Create the input files for Flemming Hansen's CATIA program.

Defaults

```
relax_disp.catia_input(dir='catia', force=False)
```

Keyword arguments

dir: The directory to place the CATIA input files, output directory, etc.

force: A flag which if set to True will cause the files to be overwritten if they already exist.

Description

This will create all of the input file required for CATIA as well as the CATIA results output directory.



17.2.162 relax_disp.cluster



Synopsis

Define clusters of spins for joint optimisation.

Defaults

```
relax_disp.cluster(cluster_id=None, spin_id=None)
```

Keyword arguments

cluster_id: The cluster identification string.

spin_id: The spin identifier string for the spin or group of spins to add to the cluster.

Description

In a relaxation dispersion analysis, the parameters of the model of dispersion can either be optimised for each spin system separately or a number of spins can be grouped or clustered and the dispersion model parameters optimised for all spins in the cluster together. Clusters are identified by unique ID strings. Any spins not within a cluster will be optimised separately and individually.

If the cluster ID string already exists, the spins will be added to that pre-existing cluster. If no spin ID is given, then all spins will be added to the cluster.

The special cluster ID string ‘**free spins**’ is reserved for the pool of non-clustered spins. This can be used to remove a spin system from an already existing cluster by specifying this cluster ID and the desired spin ID.

Prompt examples

To add the spins ‘`:1@N`’ and ‘`:3@N`’ to a new cluster called ‘`cluster`’, type one of:

```
relax> relax_disp.cluster('cluster', ':1,3@N')
      )
relax> relax_disp.cluster(cluster_id='cluster', spin_id=':1,3@N')
```



Synopsis

Set the CPMG pulse sequence information associated with a given spectrum.

Defaults

```
relax_disp.cpmg_setup(spectrum_id=None, cpmg_frq=
None, ncyc_even=True)
```

Keyword arguments

spectrum_id: The spectrum ID string to associate the CPMG pulse sequence information to.

cpmg_frq: The frequency, in Hz, of the CPMG pulse train.

ncyc_even: A flag which if True means that the number of CPMG blocks must be even. This is pulse sequence dependant.

Description

This allows all information about CPMG pulse sequence required for a relaxation dispersion analysis to be specified. This includes:

‘`cpmg_frq`’ allows the CPMG pulse train frequency of a given spectrum to be set. If None is given for frequency, then the spectrum will be treated as a reference spectrum.

‘`ncyc_even`’ specifies if an even number of CPMG blocks are required for the pulse sequence.

Prompt examples

To identify the reference spectrum called ‘`reference_spectrum`’, type:

```
relax> relax_disp.cpmg_setup(spectrum_id='reference_spectrum', cpmg_frq=None)
```

To set a frequency of 200 Hz for the spectrum ‘`200_Hz_spectrum`’, type:

```
relax> relax_disp.cpmg_setup(spectrum_id='200_Hz_spectrum', cpmg_frq=200)
```

17.2.164 relax_disp.cpmgfit_execute



Synopsis

Optimisation of the CPMG data using Art Palmer's CPMGFit program.

Defaults

```
relax_disp.cpmgfit.execute(dir=None, force=False,
binary='cpmgfit')
```

Keyword arguments

dir: The directory containing all of the CPMGFit input files. If not given, this defaults to the model name in lower case.

force: A flag which if set to True will cause the results files to be overwritten if they already exist.

binary: The name of the executable CPMGFit program file.

Description

CPMGFit will be executed once per spin as:

```
$ cpmgfit -grid -xmgr -f dir/spin_x.in | tee
dir/spin_x.out
```

where *x* is replaced by each spin ID string. If you would like to use a different CPMGFit executable file, change the binary name to the appropriate file name. If the file is not located within the environment's path, be sure to include the full path in front of the binary file name so it can be found.

17.2.165 relax_disp.cpmgfit_input



Synopsis

Create the input files for Art Palmer's CPMGFit program.

Defaults

```
relax_disp.cpmgfit.input(dir=None, force=False, binary=
'cpmgfit', spin_id=None)
```

Keyword arguments

dir: The directory to place the files. If not given, this defaults to the model name in lower case.

force: A flag which if set to True will cause the files to be overwritten if they already exist.

binary: The name of the executable CPMGFit program file.

spin_id: The spin identification string.

Description

The following files are created:

```
'dir/spin_x.in',
'dir/run.sh'.
```

One CPMGFit input file is created per spin and named 'dir/spin_x.in', where *x* is the spin ID string. The file 'dir/run.sh' is a batch file for executing CPMGFit for all of the spin input files. If you would like to use a different CPMGFit executable file, change the binary name to the appropriate file name. If the file is not located within the environment's path, be sure to include the full path in front of the binary name so it can be found.

Prompt examples

17.2.166 relax_disp.exp_type



Synopsis

Select the relaxation dispersion experiment type.

Defaults

```
relax_disp.exp_type(spectrum_id=None, exp_type='SQ CPMG')
```

Keyword arguments

`spectrum_id`: The spectrum ID string to associate the spin-lock field strength to.

`exp_type`: The type of relaxation dispersion experiment performed.

Description

For each peak intensity set loaded into relax, the type of experiment it comes from needs to be specified. By specifying this for each spectrum ID, multiple experiment types can be analysed simultaneously. This is assuming that an appropriate dispersion model exists for the experiment combination.

The currently supported experiments include:

‘SQ CPMG’ – The single quantum (SQ) CPMG-type experiments,

‘ZQ CPMG’ – The zero quantum (ZQ) CPMG-type experiments,

‘DQ CPMG’ – The double quantum (DQ) CPMG-type experiments,

‘MQ CPMG’ – The multiple quantum (MQ) CPMG-type experiments,

‘1H SQ CPMG’ – The 1H single quantum (SQ) CPMG-type experiments,

‘1H MQ CPMG’ – The 1H multiple quantum (MQ) CPMG-type experiments,

‘R1rho’ – The R1rho-type experiments.

To set the experiment type to ‘SQ CPMG’ for the spectrum ID ‘nu_4500.0_800MHz’, type one of:

```
relax> relax_disp.exp_type('nu_4500.0_800MHz', 'SQ CPMG')
```

```
relax> relax_disp.exp_type(spectrum_id='nu_4500.0_800MHz', exp_type='SQ CPMG')
```

17.2.167 relax_disp.insignificance**Synopsis**

Deselect all spins with insignificant dispersion.

Defaults

```
relax_disp.insignificance(level=2.0)
```

Keyword arguments

level: The R2eff/R1rho value in rad/s by which to judge insignificance. If the maximum difference between two points on all dispersion curves for a spin is less than this value, that spin will be deselected.

Description

This can be used to deselect all spins which have insignificant dispersion profiles. The insignificance value is the R2eff/R1rho value in rad/s by which to judge the dispersion curves by. If the maximum difference between two points on all dispersion curves for a spin is less than this value, that spin will be deselected.

17.2.168 relax_disp.nessy_input**Synopsis**

Create the input files for Michael Bieri's NESSY program.

Defaults

```
relax_disp.nessy_input(dir=None, force=False, spin_id=None)
```

Keyword arguments

dir: The directory to place the file and to use as the NESSY project directory. If not given, this defaults to the current directory.

force: A flag which if set to True will cause the files to be overwritten if they already exist.

spin_id: The spin identification string.

Description

This will create a single NESSY save file called '`save.NESSY`'. This will contain all of the dispersion data currently loaded in the relax data store. If the directory name is not supplied, this will default to the current directory.

17.2.169 relax_disp.parameter_copy
**Synopsis**

Copy dispersion specific parameters values from one data pipe to another.

Defaults

```
relax_disp.parameter_copy(pipe_from=None, pipe_to=None)
```

Keyword arguments

pipe_from: The name of the pipe to copy from.

pipe_to: The name of the pipe to copy to.

Description

This is a special function for copying relaxation dispersion parameters from one data pipe to another. It is much more advanced than the value.copy user function, in that clustering is taken into account. When the destination data pipe has spin clusters defined, then the new parameter values, when required, will be taken as the median value.

For the cluster specific parameters, *i.e.* the populations of the states and the exchange parameters, a median value will be used as the starting point. For all other parameters, the R20 values for each spin and magnetic field, as well as the parameters related to the chemical shift difference dw, the optimised values of the previous run will be directly copied.

Prompt examples

To copy the CSA values from the data pipe ‘m1’ to ‘m2’, type:

```
relax> value.parameter_copy('m1', 'm2', 'csa')
```

17.2.170 relax_disp.plot_disp_curves
**Synopsis**

Create 2D Grace plots of the dispersion curves for each spin system.

Defaults

```
relax_disp.plot_disp_curves(dir='grace', y_axis='r2_eff', x_axis='disp', num_points=1000, extend_hz=500.0, extend_ppm=500.0, interpolate='disp', force=False)
```

Keyword arguments

dir: The directory name to place all of the spin system files into.

y_axis: Option can be either ‘r2_eff’ which plot ‘r2eff’ for CPMG experiments or ‘r1rho’ for R1rho experiments or option can be ‘r2_r1rho’, which for R1rho experiments plot R₂.

x_axis: Option can be either ‘disp’ which plot ‘CPMG frequency (Hz)’ for CPMG experiments or ‘Spin-lock field strength (Hz)’ for R1rho experiments or option can be either ‘w_eff’ or ‘theta’ for R1rho experiments, which plot ‘Effective field in rotating frame (rad/s)’ or ‘Rotating frame tilt angle θ (rad)’

num_points: The total number of points to generate the interpolated dispersion curves with. This value has no effect for the numeric CPMG-based models.

extend_hz: How far to extend the interpolated dispersion curves beyond the last dispersion point, *i.e.* the nu_CPMG frequency or spin-lock field strength value, in Hertz.

extend_ppm: How far to extend the interpolated dispersion curves beyond the last dispersion point, *i.e.* the spin-lock offset value, in ppm.

interpolate: Either by option ‘disp’ which interpolate CPMG frequency or spin-lock field strength, or by option ‘offset’ which interpole over spin-lock offset.

force: A flag which, if set to True, will cause the files to be overwritten.

Description

This is used to create 2D Grace plots of the dispersion curves of the nu_CPMG frequencies or spin-lock field strength verses the R2eff/R1rho values. One file will be created per spin system with the name ‘`disp_x.agr`’, where x is related to the spin ID string. For each file, one Grace graph will be produced for each experiment.

Four sets of curves of R2eff/R1rho values will be produced per experiment and per magnetic field strength. These are the experimental values, the fitted values, the interpolated dispersion curves for the fitted solution, and the residuals. Different dispersion models result in different interpolated dispersion curves. For the numeric models which use CPMG-type data, the maximum interpolation resolution is constrained by the frequency of a single CPMG block for the entire relaxation period. For all other models, the interpolation resolution is not constrained and can be as fine as desired by setting the total number of interpolation points. Interpolated curves are not produced for the ‘`R2eff`’ model as they are not necessary.

For R1rho models, graphs can be interpolated against Spin-lock offset, but this feature is not available for CPMG experiment types. It is also possible to select values on X-axis of ‘Effective field in rotating frame `w_eff` (rad/s)’ or ‘Rotating frame tilt angle θ (rad)’.

For R1rho models, special Y-value R_2 R1rho can for example be plotted as function of `w_eff`. R_2 is calculated as: $R_2 = (R1rho - R1 \cos^2(\theta)) / \sin^2(\theta)$.

17.2.171 relax_disp.plot_exp-curves



Synopsis

Create 2D Grace plots of the exponential curves.

Defaults

```
relax_disp.plot_exp_curves(file=None, dir='grace', force=False, norm=False)
```

Keyword arguments

`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.

`norm`: A flag which, if set to True, will cause all graphs to be normalised to a starting value of 1. This is for the normalisation of series type data.

Description

This is used to create 2D Grace plots of the individual exponential curves used to find the R2eff or R1rho values. This supplements the `grace.write` user function which is not capable of generating these curves in a reasonable format.

17.2.172 relax_disp.r1_fit
**Synopsis**

Switch between fixed or fitted R₁ values for optimisation.

Defaults

`relax_disp.r1_fit(fit=True)`

Keyword arguments

fit: The flag specifying if R₁ values should be optimised or if loaded R₁ values should be fixed during optimisation.

Description

This user function allows the optimisation of R₁ values to be turned on or off for the relaxation dispersion dispersion models. If turned off, the current values of R₁ will be fixed. Otherwise the R₁ values will be added to the model parameter set. For models which do not support the R₁ parameter for off-resonance effects, this setting will have no effect. Only the models ['No R_{ex}', 'DPL94', 'TP02', 'TAP03', 'MP05', 'NS R1rho 2-site'] support R₁ optimisation.

17.2.173 relax_disp.r20_from_min_r2eff
**Synopsis**

Set the R20 parameter values to that of the minimum R2eff value.

Defaults

`relax_disp.r20_from_min_r2eff(force=True)`

Keyword arguments

force: A flag which if set to True will cause the R20 values to be overwritten if they already exist.

Description

Set the R20 parameter values to that of the minimum R2eff value. This user function will look through all R2eff values per magnetic field strength, find the minimum value, and set the R20, R20A, R20B, and R1rho' parameters of the model to this value. This can serve a number of purposes including using the values for the chi-squared space mapping via the δ_x.map user function, speeding up optimisation by avoiding the grid search for these parameters, and as initial parameter values for other dispersion software.

Instead of finding the initial values for the R20 parameter using the grid search, the minimum for the R2eff points can be used instead. This is often a good initial position for minimisation. For example for a two field CPMG experiment with model CR72, that would drop the number of uniform grid search points from 5D to 3D, i.e. two orders of magnitude faster. When using the standard 21 grid increments per dimension, it would allow the grid search to be 441 times faster. Note that the relaxation dispersion auto-analysis will take all pre-set parameter values into account and will automatically exclude these from the grid search.

Note that for optimisation, that this is an experimental and unpublished feature of the dispersion analysis. If R20 << min(R2eff), the grid search will be performed in a region of the optimisation space quite distant from the true minimum. If unsure, do not activate this option, and let the grid search find a better starting value.

17.2.174 relax_disp.r2eff_err-estimate



Synopsis

Estimate R2eff errors by the Jacobian matrix.

Defaults

```
relax_disp.r2eff_err_estimate(spin_id=None, epsrel=0.0,
                               verbosity=1)
```

Keyword arguments

spin_id: The spin ID string to restrict value setting to.

epsrel: The parameter to remove linear-dependent columns when J is rank deficient.

verbosity: The higher the value, the greater the verbosity.

Description

This is a new experimental feature from version 3.3.

This will estimate R2eff errors by using the exponential decay Jacobian matrix 'J' to compute the covariance matrix of the best-fit parameters.

This can be a huge time saving step, when performing model fitting in R1rho. Errors of R2eff values, are normally estimated by time-consuming Monte-Carlo simulations.

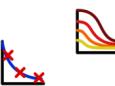
This method is inspired from the GNU Scientific Library (GSL).

The covariance matrix is given by: covar = Qxx = (J^T.W.J)^{-1}, where the weight matrix W is constructed by the multiplication of an Identity matrix I and a weight array w. The weight array is 1/errors^2, which then gives W = I.w = I x 1/errors^2.

Qxx is computed by QR decomposition, J^T.W.J=QR, Qxx=R^{-1}. Q^T. The columns of R which satisfy: —R_{kk}— ≤ epsrel —R_{11}— are considered linearly-dependent and are excluded from the covariance matrix (the corresponding rows and columns of the covariance matrix are set to zero).

The parameter 'epsrel' is used to remove linear-dependent columns when J is rank deficient.

17.2.175 relax_disp.r2eff_read



Synopsis

Read R2eff/R1rho values and errors from a file.

Defaults

```
relax_disp.r2eff_read(id=None, file=None, dir=None,
                      disp_frq=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)
```

Keyword arguments

id: The partial experiment ID string to identify this data with. The full ID string will be constructed as this ID followed by an underscore and then the dispersion point value from the file.

file: The name of the file.

dir: The directory name.

disp_frq: For CPMG-type data, the frequency of the CPMG pulse train. For R1rho-type data, the spin-lock field strength nu1. The units must be Hertz

spin_id_col: The spin ID string column used by the generic 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 file format (alternative to the spin ID column).

res_num_col: The residue number column used by the generic file format (alternative to the spin ID column).

res_name_col: The residue name column used by the generic file format (alternative to the spin ID column).

spin_num_col: The spin number column used by the generic file format (alternative to the spin ID column).

spin_name_col: The spin name column used by the generic file format (alternative to the spin ID column).

data_col: The RDC data column.

error_col: The experimental error column.

sep: The column separator used by the generic format (the default is white space).

Description

This will read R2eff/R1rho data directly from a file. The data will be associated with an experiment ID string. A partial ID is to be supplied and then the full ID string will be constructed as this ID followed by an underscore and then the dispersion point value from the file (as '%s-%s' % (id, disp_point)). The full IDs must already exist and have been used to set the type of dispersion experiment the data is from, spectrometer proton frequency of the data, and if needed the time of the relaxation period.

The format of this text file must be that each row corresponds to a unique spin system and that there is one file per dispersion point (i.e. per CPMG frequency nu_CPMG or per spin-lock field strength nu1). The file must be in columnar format and information to identify the spin must be in columns of the file.

17.2.176 relax_disp.r2eff.read-spin



Synopsis

Read R2eff/R1rho values and errors for a single spin from a file.

Defaults

```
relax_disp.r2eff.read_spin(id=None, spin_id=None, file=
None, dir=None, disp_point_col=None, offset_col=None,
data_col=2, error_col=3, sep=None)
```

Keyword arguments

id: The experiment ID string to identify this data with.

spin_id: The spin identification string.

file: The name of the file.

dir: The directory name.

disp_point_col: The column containing the CPMG frequency or spin-lock field strength (Hz).

offset_col: The column containing the offset information for R1rho-type data.

data_col: The column containing the R2eff or R1rho data.

error_col: The column containing the R2eff or R1rho error.

sep: The column separator (the default is white space).

Description

This will read R2eff/R1rho data for a single spin directly from a file. The data will be associated with an experiment ID string. This ID can be used for setting the type of dispersion experiment the data is from, spectrometer proton frequency of the data, and the time of the relaxation period.

The format of this text file must be that each row corresponds to a dispersion point (i.e. per CPMG frequency nu_CPMG or per spin-lock field strength nu1) and that there is one file per unique spin system. The file must be in columnar format. For R1rho data, the dispersion point column can be substituted for the offset values in Hertz.

17.2.177 relax_disp.relax_time**Synopsis**

Set the relaxation delay time associated with each spectrum.

Defaults

```
relax_disp.relax_time(spectrum_id=None, time=0.0)
```

Keyword arguments

spectrum_id: The spectrum ID string.

time: The time, in seconds, of the relaxation period.

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.

17.2.178 relax_disp.select_model**Synopsis**

Select the relaxation dispersion model.

Defaults

```
relax_disp.select_model(model='R2eff')
```

Keyword arguments

model: The type of relaxation dispersion model to fit.

Description

A number of different dispersion models are supported. This includes both analytic models and numerical models. Models which are independent of the experimental data type are:

'R2eff' – This is the model used to determine the R2eff/R1rho values and errors required as the base data for all other models,

The no chemical exchange models

'No R_{ex}' – This is the model for no chemical exchange being present.

The SQ CPMG-type experiments

The currently supported analytic models are:

'LM63' – The original Luz and Meiboom (1963) 2-site fast exchange equation with parameters {R20, ..., ϕ_ex, kex},

'LM63 3-site' – The original Luz and Meiboom (1963) 3-site fast exchange equation with parameters {R20, ..., ϕ_ex, kex, ϕ_ex2, kex2},

'CR72' – The reduced Carver and Richards (1972) 2-site equation for most time scales whereby the simplification R20A = R20B is assumed. The parameters are {R20, ..., pA, dw, kex},

'CR72 full' – The full Carver and Richards (1972) 2-site equation for most time scales with parameters {R20A, R20B, ..., pA, dw, kex},

'IT99' – The Ishima and Torchia (1999) 2-site model for all time scales with $pA \gg pB$ and with parameters $\{R20, \dots, pA, dw, kex\}$,

'TSMFK01' – The Tollinger, Kay et al. (2001) 2-site very-slow exchange model, range of microsecond to second time scale. Applicable in the limit of slow exchange, when $-R20A-R20B- << k_{AB}, kB << 1/\tau_{CP}$. $R20A$ is the transverse relaxation rate of site A in the absence of exchange. $2*\tau_{CP}$ is the time between successive 180 deg. pulses. The parameters are $\{R20A, \dots, dw, k_{AB}\}$.

'B14' – The Baldwin (2014) 2-site exact solution model for all time scales, whereby the simplification $R20A = R20B$ is assumed. The parameters are $\{R20, \dots, pA, dw, kex\}$,

'B14 full' – The Baldwin (2014) 2-site exact solution model for all time scales with parameters $\{R20A, R20B, \dots, pA, dw, kex\}$,

The currently supported numeric models are:

'NS CPMG 2-site 3D' – The reduced numerical solution for the 2-site Bloch-McConnell equations using 3D magnetisation vectors whereby the simplification $R20A = R20B$ is assumed. Its parameters are $\{R20, \dots, pA, dw, kex\}$,

'NS CPMG 2-site 3D full' – The full numerical solution for the 2-site Bloch-McConnell equations using 3D magnetisation vectors. Its parameters are $\{R20A, R20B, \dots, pA, dw, kex\}$,

'NS CPMG 2-site star' – The reduced numerical solution for the 2-site Bloch-McConnell equations using complex conjugate matrices whereby the simplification $R20A = R20B$ is assumed. It has the parameters $\{R20, \dots, pA, dw, kex\}$,

'NS CPMG 2-site star full' – The full numerical solution for the 2-site Bloch-McConnell equations using complex conjugate matrices with parameters $\{R20A, R20B, \dots, pA, dw, kex\}$,

'NS CPMG 2-site expanded' – The numerical solution for the 2-site Bloch-McConnell equations expanded using Maple by Nikolai Skrynnikov. It has the parameters $\{R20, \dots, pA, dw, kex\}$.

The MMQ CPMG-type experiments

The currently supported models are:

'MMQ CR72' – The the Carver and Richards (1972) 2-site model for most time scales expanded for MMQ CPMG data by Korzhnev et al., 2004, whereby the simplification $R20A = R20B$ is assumed. Its parameters are $\{R20, \dots, pA, dw, dwH, kex\}$.

'NS MMQ 2-site' – The numerical solution for the 2-site Bloch-McConnell equations for combined proton-heteronuclear SQ, ZQ, DQ, and MQ CPMG data whereby the simplification $R20A = R20B$ is assumed. Its parameters are $\{R20, \dots, pA, dw, dwH, kex\}$.

'NS MMQ 3-site linear' – The numerical solution for the 3-site Bloch-McConnell equations linearised with $kAC = kCA = 0$ for combined proton-heteronuclear SQ, ZQ, DQ, and MQ CPMG data whereby the simplification $R20A = R20B = R20C$ is assumed. Its parameters are $\{R20, \dots, pA, dw(AB), dwH(AB), kex(AB), pB, dw(BC), dwH(BC), kex(BC)\}$.

'NS MMQ 3-site' – The numerical solution for the 3-site Bloch-McConnell equations for combined proton-heteronuclear SQ, ZQ, DQ, and MQ CPMG data whereby the simplification $R20A = R20B = R20C$ is assumed. Its parameters are $\{R20, \dots, pA, dw(AB), dwH(AB), kex(AB), pB, dw(BC), dwH(BC), kex(BC), kex(AC)\}$.

The R1rho-type experiments

The currently supported analytic models are:

On-resonance models are:

'M61' – The Meiboom (1961) 2-site fast exchange equation with parameters $\{R1rho', \dots, \phi_{ex}, kex\}$,

'M61 skew' – The Meiboom (1961) 2-site equation for all time scales with $pA \gg pB$ and with parameters $\{R1rho', \dots, pA, dw, kex\}$,

Off-resonance models are:

'DPL94' – The Davis, Perlman and London (1994) 2-site fast exchange equation with parameters $\{R1rho', \dots, \phi_{ex}, kex\}$,

'TP02' – The Trott and Palmer (2002) 2-site equation for all time scales with parameters $\{R1rho', \dots, pA, dw, kex\}$.

'TAP03' – The Trott, Abergel and Palmer (2003) off-resonance 2-site equation for all time scales with parameters $\{R1rho', \dots, pA, dw, kex\}$.

'MP05' – The Miloushev and Palmer (2005) 2-site off-resonance equation for all time scales with parameters $\{R1rho', \dots, pA, dw, kex\}$.

The currently supported numeric models are:

'NS R1rho 2-site' – The numerical solution for the 2-site Bloch-McConnell equations using 3D magnetisation vectors whereby the simplification $R20A = R20B$. Its parameters are $\{R1rho', \dots, pA, dw, kex\}$.

'NS R1rho 3-site linear' – The numerical solution for the 3-site Bloch-McConnell equations using 3D magnetisation vectors whereby the simplification $R20A = R20B = R20C$ is assumed and linearised with $kAC = kCA = 0$. Its parameters are $\{R1rho', \dots, pA, dw(AB), kex(AB), pB, dw(BC), kex(BC)\}$.

'NS R1rho 3-site' – The numerical solution for the 3-site Bloch-McConnell equations using 3D magnetisation vectors. Its parameters are {R1rho', ..., pA, dw(AB), kex(AB), pB, dw(BC), kex(BC), kex(AC)}.

Prompt examples

To pick the 2-site fast exchange model for all selected spins, type one of:

```
relax> relax_disp.select_model('LM63')
```

```
relax> relax_disp.select_model(model='LM63')
```

17.2.179 relax_disp.sherekhan_input



Synopsis

Create the input files for Adam Mazur's ShereKhan program.

Defaults

```
relax_disp.sherekhan_input(force=False, spin_id=None,
dir=None)
```

Keyword arguments

force: A flag which if set to True will cause the files to be overwritten if they already exist.

spin_id: The spin identification string.

dir: The directory name to place ShereKhan cluster folders into.

Description

This creates the files required for the ShereKhan server located at <http://sherekhan.bionmr.org/>. One file per spin cluster per field strength will be created. These will be placed in the directory '`clusterx`' and named '`sherekhan_frqy.in`', where *x* is the cluster index starting from 1 and *y* is the magnetic field strength index starting from 1.

17.2.180 relax_disp.spin_lock_field**Synopsis**

Set the relaxation dispersion spin-lock field strength (nu1).

Defaults

```
relax_disp.spin_lock_field(spectrum_id=None, field=None)
```

Keyword arguments

spectrum_id: The spectrum ID string to associate the spin-lock field strength to.

field: The spin-lock field strength, nu1, in Hz.

Description

This sets the spin-lock field strength, nu1, for the specified R1rho spectrum in Hertz.

Prompt examples

To set a spin-lock field strength of 2.1 kHz for the spectrum ‘nu1_2.1kHz_relaxT_0.010’, type one of:

```
relax> relax_disp.spin_lock_field(2100, 'nu1_2.1kHz_relaxT_0.010')

relax> relax_disp.spin_lock_field(field=2100, spectrum_id='nu1_2.1kHz_relaxT_0.010')
```

17.2.181 relax_disp.spin_lock_offset**Synopsis**

Set the relaxation dispersion spin-lock offset (omega_rf).

Defaults

```
relax_disp.spin_lock_offset(spectrum_id=None, offset=None)
```

Keyword arguments

spectrum_id: The spectrum ID string to associate the spin-lock offset to.

offset: The spin-lock offset, omega_rf, in ppm.

Description

This sets the spin-lock offset, omega_rf, for the specified R1rho spectrum in ppm.

Prompt examples

To set a spin-lock offset of 110.0 ppm for the spectrum ‘nu1_2.1kHz_relaxT_0.010’, type one of:

```
relax> relax_disp.spin_lock_offset('nu1_2.1kHz_relaxT_0.010', 110.0)

relax> relax_disp.spin_lock_offset(spectrum_id='nu1_2.1kHz_relaxT_0.010', offset=110.0)
```

17.2.182 relax_disp.write_disp_curves

Synopsis

Create text files of the dispersion curves for each spin system.

Defaults

```
relax_disp.write_disp_curves(dir=None, force=False)
```

Keyword arguments

dir: The directory name to place all of the spin system files into.

force: A flag which, if set to True, will cause the files to be overwritten.

Description

This is used to created text files of the dispersion curves of R_{2eff}/R_{1rho} values, both measured and back calculated from the optimised dispersion model. The columns of the text file will be the experiment name, the magnetic field strength (as the proton frequency in MHz), dispersion point (nu_CPMG or the spin-lock field strength), the experimental R_{2eff} value, the back-calculated R_{2eff} value, and the experimental R_{2eff} error. One file will be created per spin system with the name ‘`disp_x.out`’, where *x* is the spin ID string.

17.2.183 relax_fit.relax_time

Synopsis

Set the relaxation delay time associated with each spectrum.

Defaults

```
relax_fit.relax_time(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.

17.2.184 relax_fit.select_model**Synopsis**

Select the relaxation curve type.

Defaults

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

Keyword arguments

model: The type of relaxation curve to fit.

Description

A number of relaxation experiments are supported and include:

The ‘**exp**’ model. This is the default two parameter exponential fit. The magnetisation starts at I_0 and decays to zero. The parameters are [Rx, I_0] and the equation is $I(t) = I_0 \cdot \exp(-Rx \cdot t)$.

The ‘**inv**’ model. This is the inversion recovery experiment (IR). The magnetisation starts at a negative value at $-I_0$ and relaxes to a positive I_{∞} value. The parameters are [Rx, I_0 , I_{∞}] and the equation is $I(t) = I_{\infty} - I_0 \cdot \exp(-Rx \cdot t)$.

The ‘**sat**’ model. This is the saturation recovery experiment (SR). The magnetisation starts at zero and relaxes to a positive I_{∞} value. The parameters are [Rx, I_{∞}] and the equation is $I(t) = I_{\infty} \cdot (1 - \exp(-Rx \cdot t))$.

17.2.185 reset**Synopsis**

Reinitialise the relax data storage object.

Defaults

```
reset()
```

Description

All of the data of the relax data storage object will be erased and hence relax will return to its initial state.

17.2.186 residue.copy**Synopsis**

Copy all data associated with a residue.

Defaults

```
residue.copy(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 ID 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 ID string of the residue to copy the data to. If left blank, the new residue will have the same name as the old.

Description

This will copy all the data associated with the identified residue to the new, non-existent residue. The new residue cannot currently exist.

Prompt 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')
```

17.2.187 residue.create**Synopsis**

Create a new residue.

Defaults

```
residue.create(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, 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.

Prompt 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')
```

17.2.188 residue.delete

**Synopsis**

Delete residues from the current data pipe.

Defaults

`residue.delete(res_id=None)`

Keyword arguments

`res_id`: The residue ID string.

Description

This can be used to delete a single or sets of residues. See the ID string documentation for more information. If spin system/atom ids are included a RelaxError will be raised.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

17.2.189 residue.display

**Synopsis**

Display information about the residue(s).

Defaults

`residue.display(res_id=None)`

Keyword arguments

`res_id`: The residue ID string.

Description

This will display the residue data loaded into the current data pipe.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.



Prompt examples

17.2.190 residue.name



Synopsis

Name the residues.

Defaults

```
residue.name(res_id=None, name=None, force=False)
```

Keyword arguments

res_id: The residue ID 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 simply allows residues to be named (or renamed).

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

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)
```

Prompt examples

17.2.191 residue.number



Synopsis

Number the residues.

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)
```

Defaults

```
residue.number(res_id=None, number=None, force=False)
```

Keyword arguments

res_id: The residue ID 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 simply allows residues to be numbered. The new number cannot correspond to an existing residue.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

17.2.192 results.display**Synopsis**

Display the results.

Defaults

`results.display()`

Description

This will print to screen (STDOUT) the results contained within the current data pipe.

17.2.193 results.read**Synopsis**

Read the contents of a relax results file into the relax data store.

Defaults

`results.read(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

This 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 the file with '.bz2' appended followed by the file name with '.gz' appended will be searched for.

17.2.194 results.write**Synopsis**

Write the results to a file.

Defaults

```
results.write(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

This will write the entire contents of the current data pipe into an XML formatted file. This results file can then be read back into relax at a later point in time, or transferred to another machine. This is in contrast to the `state.save` user function whereby the entire data store, including all data pipes, are saved into a similarly XML formatted file.

To place the results file in the current working directory in the prompt and scripting modes, leave the directory unset. If the directory 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 compression type 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.

17.2.195 script**Synopsis**

Execute a relax script.

Defaults

```
script(file=None, dir=None)
```

Keyword arguments

file: The name of the file containing the relaxation data.

dir: The directory where the file is located.

Description

This will execute a relax or any ordinary Python script.

17.2.196 select.all**Synopsis**

Select all spins in the current data pipe.

Defaults

`select.all()`

Description

This will select all spins, irregardless of their current state.

Prompt examples

To select all spins, simply type:

`relax> select.all()`

17.2.197 select.display**Synopsis**

Display the current spin selection status.

Defaults

`select.display()`

Description

This simply prints out the current spin selections.

Prompt examples

To show the current selections, type:

`relax> select.display()`

17.2.198 select.domain



To select all spins of the domain ‘N-dom’, preserving the current selections, simply type one of:

```
relax> select.domain('N-dom', 'AND', True)
```

```
relax> select.domain(domain_id='N-dom',
                      boolean='AND', change_all=True)
```

Synopsis

Select all spins and interatomic data containers of a domain.

Defaults

```
select.domain(domain_id=None, boolean='AND',
              change_all=True)
```

Keyword arguments

domain_id: The domain ID string of the domain to select.

boolean: The boolean operator specifying how interatomic data containers should be selected.

change_all: A flag specifying if all non-matching spin and interatomic data containers should be deselected.

Description

This will select all spins and interatomic data containers of a given domain. This is defined by the domain ID string as specified by the previously executed domain-related user functions.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Prompt examples

To select all spins of the domain ‘N-dom’, simply type one of:

```
relax> select.domain('N-dom', change_all=
                      True)
```

```
relax> select.domain(domain_id='N-dom',
                      change_all=True)
```

Prompt examples

17.2.199 select.interatom



Synopsis

Select specific interatomic data containers.

Defaults

```
select.interatom(spin_id1=None, spin_id2=None,
boolean='OR', change_all=False)
```

Keyword arguments

spin_id1: The spin ID string of the first spin of the interatomic data container.

spin_id2: The spin ID string of the second spin of the interatomic data container.

boolean: The boolean operator specifying how interatomic data containers should be selected.

change_all: A flag specifying if all other interatomic data containers should be changed.

Description

This is used to select specific interatomic data containers which store information about spin pairs such as RDCs, NOEs, dipole-dipole pairs involved in relaxation, etc. The ‘`change_all`’ flag default is False meaning that all interatomic data containers currently either selected or deselected will remain that way. Setting this to True will cause all interatomic data containers not specified by the spin ID strings to be selected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

To select all N-H backbone bond vectors of a protein, assuming these interatomic data containers have been already set up, type one of:

```
relax> select.interatom('ON', 'OH')
```

```
relax> select.interatom(spin_id1='ON',
                           spin_id2='OH')
```

To select all H-H interatomic vectors of a small organic molecule, type one of:

```
relax> select.interatom('OH*', 'OH*')
```

```
relax> select.interatom(spin_id1='OH*',
                           spin_id2='OH*')
```

17.2.200 select.read



Synopsis

Select the spins contained in a file.

Defaults

```
select.read(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).

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 molecule name, residue number, residue name,

spin number and/or spin name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID string 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 default is False meaning that all spins currently either selected or deselected will remain that way. Setting this to True will cause all spins not specified in the file to be deselected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Prompt 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)
```

17.2.201 select.reverse**Synopsis**

Reversal of the spin selection for the given spins.

Defaults

```
select.reverse(spin_id=None)
```

Keyword arguments

spin_id: The spin ID string.

Description

By supplying the spin ID string, a subset of spins can have their selection status reversed.

Prompt examples

To select all currently deselected spins and deselect those which are selected type:

```
relax> select.reverse()
```

**17.2.202 select.sn_ratio****Synopsis**

Select spins with signal to noise ratio higher or lower than the given ratio.

Defaults

```
select.sn_ratio(ratio=10.0, operation='>', all_sn=True)
```

Keyword arguments

ratio: The signal to noise ratio to compare to.

operation: The comparison operation by which to select the spins.

all_sn: A flag specifying if all the signal to noise ratios per spin should match the comparison operator, or if just a single comparison match is enough.

Description

The comparison operation is the method which to select spins according to: `operation(sn_ratio, ratio)`.

The possible operations are: '`<`':strictly less than, '`<=`':less than or equal, '`>`':strictly greater than, '`>=`':greater than or equal, '`==`':equal, '`!=`':not equal.

The '`all_sn`' flag default is True, meaning that if all of the spin's signal to noise levels evaluates to True in the comparison, the spin is selected.

Prompt examples

To select all spins with a signal to noise ratio higher than 10.0:

```
relax> select.sn_ratio(ratio=10.0, operation = '>')
```

```
relax> select.sn_ratio(ratio=10.0, operation = '>', all_sn=False)
```

17.2.203 select.spin**Synopsis**

Select specific spins.

Defaults

```
select.spin(spin_id=None, boolean='OR', change_all=False)
```

Keyword arguments

spin_id: The spin ID 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 default is False meaning that all spins currently either selected or deselected will remain that way. Setting this to True will cause all spins not specified by the spin ID string to be selected.

Boolean operators

The boolean operator can be used to change how spin systems or interatomic data containers 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.

Please see Table 17.1 on page 463.

Prompt 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')
```

17.2.204 sequence.attach_protons**Synopsis**

Attach protons to all heteronuclei.

Defaults

```
sequence.attach_protons()
```

Description

This can be used to attach protons to all the heteronuclei in the current data pipe. For each proton, a spin container will be created. This should be used when the sequence information is not being extracted from a 3D structure. Note that the proton spin containers will not possess any positional information, so for analyses which require this position or vectors from one atom to this proton, it should not be used.

Prompt examples

To attach protons, simply type:

```
relax> sequence.attach_protons()
```

17.2.205 sequence.copy**Synopsis**

Copy the molecule, residue, and spin sequence data from one data pipe to another.

Defaults

```
sequence.copy(pipe_from=None, pipe_to=None, empty=True)
```

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.

empty: A flag which if True will create a molecule, residue, and spin sequence in the target pipe lacking all of the spin data of the source pipe. If False, then the spin data will also be copied.

Description

This will copy the sequence data between data pipes. The destination data pipe must not contain any sequence data. If the source and destination pipes are not specified, then both will default to the current data pipe (hence providing one is essential).

Prompt 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')
```

17.2.206 sequence.display**Synopsis**

Display sequences of molecules, residues, and/or spins.

Defaults

```
sequence.display(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 which if True will cause the molecule name column to be shown.

res_num_flag: A flag which if True will cause the residue number column to be shown.

res_name_flag: A flag which if True will cause the residue name column to be shown.

spin_num_flag: A flag which if True will cause the spin number column to be shown.

spin_name_flag: A flag which if True will cause the spin name column to be shown.

Description

This will print out the sequence information of all loaded spins in the current data pipe.

Prompt examples

17.2.207 sequence.read



Synopsis

Read the molecule, residue, and spin sequence from a file.

Defaults

```
sequence.read(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 molecule name, residue number, residue name, spin number and/or spin name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID string 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 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)
```

17.2.208 sequence.write**Synopsis**

Write the molecule, residue, and spin sequence to a file.

Defaults

```
sequence.write(file=None, dir=None, sep=None,
mol_name_flag=True, res_num_flag=True, res_name_flag=
True, spin_num_flag=True, spin_name_flag=True, 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 which if True will cause the molecule name column to be shown.

res_num_flag: A flag which if True will cause the residue number column to be shown.

res_name_flag: A flag which if True will cause the residue name column to be shown.

spin_num_flag: A flag which if True will cause the spin number column to be shown.

spin_name_flag: A flag which 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

Write the sequence data to file. If no directory name is given, the file will be placed in the current working directory.

17.2.209 spectrometer.frequency
 ω
Synopsis

Set the spectrometer proton frequency of the experiment.

Defaults

```
spectrometer.frequency(id=None, frq=None, units='Hz')
```

Keyword arguments

id: The experiment identification string to set the frequency of.

frq: The spectrometer frequency. See the 'sfrq' parameter in the Varian procpar file or the 'SF01' parameter in the Bruker acqus file.

units: The units of frequency.

Description

This allows the spectrometer frequency of a given experiment to be set. The expected units are that of the proton resonance frequency in Hertz. See the 'sfrq' parameter in the Varian procpar file or the 'SF01' parameter in the Bruker acqus file for the exact value.

17.2.210 spectrometer-.temperature
**Synopsis**

Specify the temperature of an experiment.

Defaults

spectrometer.temperature(id=None, temp=None)

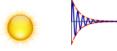
Keyword arguments

id: The experiment identification string.

temp: The temperature of the experiment in Kelvin.

Description

This allows the temperature of an experiment to be set. This value should be in Kelvin. In certain analyses, for example those which use pseudocontact shift data, knowledge of the temperature is essential. For the pseudocontact shift, the experiment ID string should match one of the alignment IDs.

17.2.211 spectrum.baseplane-rmsd
**Synopsis**

Set the baseplane RMSD of a given spin in a spectrum for error analysis.

Defaults

spectrum.baseplane_rmsd(error=0.0, spectrum_id=None, spin_id=None)

Keyword arguments

error: The baseplane RMSD error value.

spectrum_id: The spectrum ID string.

spin_id: The spin ID string.

Description

The spectrum ID identifies the spectrum associated with the error and must correspond to a previously loaded set of intensities. If the spin ID is unset, then the error value for all spins will be set to the supplied value.

17.2.212 spectrum.delete**Synopsis**

Delete the spectral data corresponding to the spectrum ID string.

Defaults

```
spectrum.delete(spectrum_id=None)
```

Keyword arguments

`spectrum_id`: The unique spectrum ID string.

Description

The spectral data corresponding to the given spectrum ID string will be removed from the current data pipe.

Prompt examples

To delete the peak height data corresponding to the ID '`R1 ncyc5`', type:

```
relax> spectrum.delete('R1 ncyc5')
```

17.2.213 spectrum.error_analysis**Synopsis**

Perform an error analysis for peak intensities.

Defaults

```
spectrum.error_analysis(subset=None)
```

Keyword arguments

`subset`: The list of spectrum ID strings to restrict the error analysis to.

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.

The error analysis can be restricted to a subset of the loaded spectral data. This is useful, for example, if half the spectra have been collected on one spectrometer and the other half on a different spectrometer.

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:

Please see Table 17.25 on page 609.

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.

Table 17.25: The six peak intensity error analysis types.

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

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

$$\text{sigma}^2 = \text{sum}(\{\text{I}_i - \text{I}_{\text{av}}\}^2) / (n - 1),$$

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, I_i is the peak intensity for spectrum i , and I_{av} 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 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:

$$\text{sigma_vol}^2 = \text{sigma_i}^2 * N,$$

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:

$$\text{sigma_vol} = \text{sigma_i} * \sqrt{n * m},$$

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.



17.2.214 spectrum.error_analysis_per_field

Synopsis

Use spectrum IDs per each field strength for an error analysis for peak intensities.

Defaults

`spectrum.error_analysis_per_field()`

Description

Please see the `spectrum.error_analysis` user function documentation.

This user function will collect all spectrum IDs for each field strength separately, and call the `spectrum.error_analysis` with these.

This function is meant as a short-cut for the `spectrum.error_analysis` function.

Prompt examples

To collect all spectrum IDs per field strength, and perform peak intensity error analysis:

`relax> spectrum.error_analysis_per_field()`

17.2.215 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(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 ID string.

spin_id: Restrict setting the number to certain spins.

Description

For a complete description of which integration methods and how many points N are used for different integration techniques, please see the spectrum.error_analysis user function documentation.

The spectrum ID identifies the spectrum associated with the value of N and must correspond to a previously loaded set of intensities. If the spin ID is unset, then the number of summed points for all spins will be set to the supplied value.

17.2.216 spectrum.read-intensities



Synopsis

Read peak intensities from a file.

Defaults

```
spectrum.read_intensities(file=None, dir=None, spectrum_id=None, dim=1, int_method='height', int_col=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, ncproc=None)
```

Keyword arguments

file: The name of the file or the list of files containing the intensity data.

dir: The directory where the file is located.

spectrum_id: The unique spectrum ID string or list of strings to associate with the peak intensity values. If multiple files are given, then each file should have a corresponding spectrum ID string. If ‘auto’ is provided for a NMRPipe seriesTab formatted file, the IDs are auto generated in form of Z_A{i}.

dim: Associate the data with the spins of any dimension in the peak list. This defaults to w1, the heteronucleus in HSQC type experiments.

int_method: The method by which peaks were integrated.

int_col: The optional column containing the peak intensity data (used by the generic intensity file format, or if the intensities are in a non-standard column).

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 column).

res_num_col: The residue number column used by the generic intensity file format (alternative to the spin ID column).

res_name_col: The residue name column used by the generic intensity file format (alternative to the spin ID column).

spin_num_col: The spin number column used by the generic intensity file format (alternative to the spin ID column).

spin_name_col: The spin name column used by the generic intensity file format (alternative to the spin ID column).

sep: The column separator used by the generic intensity format (the default is white space).

spin_id: The spin ID string used 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 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 spectral dimension is used to specify if the intensity data should be loaded into the spins identified by the first dimension w1, second dimension w2, etc.

The integration method is required for the subsequent error analysis. When peak heights are measured, this 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 value should be set to ‘point sum’. All other volume integration methods, e.g. line shape fitting, the value 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 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 ‘1t’. The default is to assume that columns 0, 1, 2, and 3 (1st, 2nd, 3rd, and 4th) 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 integration column to the appropriate number (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 11th column) which contains either the peak height or peak volume data. Because the columns are fixed, the integration column number 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 peak volumes (or evolumes), int_col must be set to 15.

NMRPipe seriesTab: The file should be a NMRPipe-format Spectral Series list. If the spectrum_id=‘auto’, the IDs are auto generated in form of Z_A{i}.

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. Alternatively a spin ID string column can be used. The peak intensities can be placed in another column specified by the integration column number. Intensities from multiple spectra can be placed into different columns, and these can then be specified simultaneously by setting the integration column value to a list of columns. This list must be matched by setting the spectrum ID to a list of the same length. If columns are delimited by a character other than whitespace, this can be specified with the column separator. The spin ID can be used to restrict the loading to specific spin subsets.

Multiple files

The data from multiple files can be loaded simultaneously if a list of files is supplied. In this case, a list of spectrum ID strings of equal length must be supplied.

Prompt 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')
```

17.2.217 spectrum.read_spins



Synopsis

Read peak assignments from a file and create spins.

Defaults

```
spectrum.read_spins(file=None, dir=None, dim=1,
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 intensity data.

dir: The directory where the file is located.

dim: Associate the data with the spins of any dimension in the peak list. This defaults to w1, the heteronucleus in HSQC type experiments.

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 column).

res_num_col: The residue number column used by the generic intensity file format (alternative to the spin ID column).

res_name_col: The residue name column used by the generic intensity file format (alternative to the spin ID column).

spin_num_col: The spin number column used by the generic intensity file format (alternative to the spin ID column).

spin_name_col: The spin name column used by the generic intensity file format (alternative to the spin ID column).

sep: The column separator used by the generic intensity format (the default is white space).

spin_id: The spin ID string used to restrict the loading of data to certain spin subsets.

Description

The spectral dimension is used to specify if the intensity data should be loaded into the spins identified by the first dimension w1, second dimension w2, etc.

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 '1t'. The default is to assume that columns 0, 1, 2, and 3 (1st, 2nd, 3rd, and 4th) 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 integration column to the appropriate number (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 '1e'. As the columns are fixed, the peak intensity column is hardwired to number 10 (the 11th column) which contains either the peak height or peak volume data. Because the columns are fixed, the integration column number 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 peak volumes (or evolumes), int_col must be set to 15.

NMRPipe seriesTab: The file should be a NMRPipe-format Spectral Series list. If the spectrum_id='auto', the IDs are auto generated in form of Z_A{i}.

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. Alternatively a spin ID string column can be used. The peak intensities can be placed in another column specified by the integration column number. Intensities from multiple spectra can be placed into different columns, and these can then be specified simultaneously by setting the integration column value to a list of columns. This list must be matched by setting the spectrum ID to a list of the same length. If columns are delimited by a character other than whitespace, this can be specified with the column separator. The spin ID can be used to restrict the loading to specific spin subsets.

Prompt examples

To read the spin assignments from the Sparky formatted files 'ref.list' and 'sat.list', type:

```
relax> spectrum.read_spins(file='ref.list')
```

```
relax> spectrum.read_spins(file='sat.list')
```

To read the spin assignments from the XEasy formatted files 'ref.text' and 'sat.text', type:

```
relax> spectrum.read_spins(file='ref.text')
```

```
relax> spectrum.read_spins(file='sat.text')
```

17.2.218 spectrum.replicated**Synopsis**

Specify which spectra are replicates of each other.

Defaults

spectrum.replicated(spectrum_ids=None)

Keyword arguments

spectrum_ids: The list of replicated spectra ID strings.

Description

This is used to identify which of the loaded spectra are replicates of each other. Specifying the replicates is essential for error analysis if the baseplane RMSD has not been supplied.

Prompt 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 R₂ spectra ‘ncyc2’ and ‘ncyc2b’ are the same time point, type:

```
relax> spectrum.replicated(['ncyc2', 'ncyc2b
    '])
```

17.2.219 spectrum.sn_ratio**Synopsis**

Calculate the signal to noise ratio for all selected spins.

Defaults

spectrum.sn_ratio()

Description

This user function will per spin calculate the signal to noise ratio: S/N.

Prompt examples

To calculate the Signal to Noise ratio per spin.

```
relax> spectrum.sn_ratio()
```

17.2.220 spin.copy



Synopsis

Copy all data associated with a spin.

Defaults

```
spin.copy(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. If left blank, the new spin will have the same name as the old.

Description

This will copy all the data associated with the identified spin to the new, non-existent spin. The new spin must not already exist.

Prompt 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')
```

17.2.221 spin.create



Synopsis

Create a new spin.

Defaults

```
spin.create(spin_name=None, spin_num=None,
res_name=None, res_num=None, mol_name=None)
```

Keyword arguments

spin_name: The name of the spin.

spin_num: The spin number.

res_name: The name of the residue to add the spin to.

res_num: The number of the residue to add the spin to.

mol_name: The name of the molecule to add the spin to.

Description

This will add a new spin data container to the relax data storage object. The same spin number cannot be used more than once.

Prompt examples

The following sequence of commands will add the spins 1 C4, 2 C9, 3 C15 to residue number 10:

```
relax> spin.create('C4', 1, res_num=10)
```

```
relax> spin.create('C9', 2, res_num=10)
```

```
relax> spin.create('C15', 3, res_num=10)
```

17.2.222 spin.create_pseudo



Synopsis

Create a spin system representing a pseudo-atom.

Defaults

```
spin.create_pseudo(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.

members: A list of the atoms (as spin ID strings) that the pseudo-atom is composed of.

averaging: The positional averaging technique.

Description

This 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.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Prompt 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'])
```

17.2.223 spin.delete**Synopsis**

Delete spins.

Defaults

spin.delete(spin_id=None)

Keyword arguments

spin_id: The spin identifier string.

Description

This can be used to delete a single or sets of spins. See the identification string documentation below for more information.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

17.2.224 spin.display**Synopsis**

Display information about the spin(s).

Defaults

spin.display(spin_id=None)

Keyword arguments

spin_id: The spin identification string.

Description

This will display the spin data loaded into the current data pipe.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Prompt examples

17.2.225 spin.element



The set all spins of residue 1 to be carbons, type one of:

```
relax> spin.element('@1', 'C', force=True)
relax> spin.element(spin_id='@1', element='C',
                     , force=True)
```

Synopsis

Set the element type of the spin.

Defaults

```
spin.element(element=None, spin_id=None, force=False)
```

Keyword arguments

element: The IUPAC element name.

spin_id: The spin identification string corresponding to one or more spins.

force: A flag which if True will cause the element to be changed.

Description

This allows the element type of the spins to be set.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Prompt examples

17.2.226 spin.isotope



The set all spins of residue 1 to the ‘`13C`’ nuclear isotope,
type one of:

```
relax> spin.isotope('01', '13C', force=True)
```

```
relax> spin.isotope(spin_id='01', isotope='
    13C', force=True)
```

Synopsis

Set the spins’ nuclear isotope type.

Defaults

```
spin.isotope(isotope=None, spin_id=None, force=False)
```

Keyword arguments

`isotope`: The nuclear isotope name in the AE notation - the atomic mass number followed by the element symbol.

`spin_id`: The spin identification string corresponding to one or more spins.

`force`: A flag which if True will cause the nuclear isotope to be changed.

Description

This allows the nuclear isotope type of the spins to be set.

Spin ID 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 - one per spin - 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>, ...] @<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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string ‘`@H*`’ will select the protons ‘H’, ‘H2’, ‘H98’.

17.2.227 spin.name



Synopsis

Name the spins.

Defaults

```
spin.name(name=None, spin_id=None, force=False)
```

Keyword arguments

name: The new name.

spin_id: The spin identification string corresponding to one or more spins.

force: A flag which if True will cause the spin to be renamed.

Description

This simply allows spins to be named (or renamed). Spin naming often essential. For example when reading Sparky peak list files, then the spin name must match that in the file.

Spin ID 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 - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

Prompt 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('01', 'C11', force=True)
```

```
relax> spin.name('02', 'C12', force=True)
```

```
relax> spin.name('03', 'C13', force=True)
```

Prompt examples

17.2.228 spin.number



Synopsis

Number the spins.

Defaults

```
spin.number(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 simply allows spins to be numbered. The new number cannot correspond to an existing spin number.

Spin ID 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 'Q' character. Each token can be composed of multiple elements - one per spin - 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. If only the molecule ID component is specified, then all spins of the molecule will match.

Regular expression can be used to select spins. For example the string '@H*' will select the protons 'H', 'H2', 'H98'.

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)
```

17.2.229 state.load**Synopsis**

Load a saved program state.

Defaults

```
state.load(state='state.bz2', 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 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.

For more advanced users, file descriptor objects are 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.

Prompt 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)
```

17.2.230 state.save**Synopsis**

Save the program state.

Defaults

```
state.save(state='state.bz2', dir=None, compress_type=1,
force=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.

Description

This 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.

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 compression type which can be set to

- 0** – No compression (no file extension).
- 1** – bzip2 compression ('.bz2' file extension).
- 2** – gzip compression ('.gz' file extension).

Prompt 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)
```

17.2.231 statistics.aic



Synopsis

Calculate and store Akaike’s Information Criterion (AIC) for each model.

Defaults

```
statistics.aic()
```

Description

This will perform a calculation to obtain the chi-squared statistic for the current parameter values for each model, count the number of parameters per model and calculate Akaike’s Information Criterion (AIC) using the formula $AIC = \chi^2 + 2k$. The AIC values, chi-squared values, and number of parameters will be stored in the appropriate location for the model in the relax data store.

17.2.232 statistics.model**Synopsis**

Calculate and store the model statistics.

Defaults

statistics.model()

Description

This will perform a back-calculation to obtain the chi-squared statistic for the current parameter values, count the number of parameters and data points per model, and place all the values in the relax data store.

17.2.233 structure.add_atom**Synopsis**

Add an atom.

Defaults

```
structure.add_atom(mol_name=None, atom_name=None,
res_name=None, res_num=None, pos=None, element=
None, atom_num=None, chain_id=None, segment_id=
None, pdb_record=None)
```

Keyword arguments

mol_name: The name of molecule container to create or add the atom to.

atom_name: The atom name.

res_name: The residue name.

res_num: The residue number.

pos: The atomic coordinates. For specifying different coordinates for each model of the ensemble, a list of lists can be supplied.

element: The element name.

atom_num: The optional atom number.

chain_id: The optional chain ID string.

segment_id: The optional segment ID string.

pdb_record: The optional PDB record name, e.g. ‘ATOM’ or ‘HETATM’.

Description

This allows atoms to be added to the internal structural object. To use the same atomic coordinates for all models, the atomic position can be an array of 3 values. Alternatively different coordinates can be used for each model if the atomic position is a rank-2 array where the first dimension matches the number of models currently present.

17.2.234 structure.add_helix**Synopsis**

Define an α helix.

Defaults

```
structure.add_helix(start=None, end=None, mol_name=None)
```

Keyword arguments

start: The residue number for the start of the helix.

end: The residue number for the end of the helix.

mol_name: Define the secondary structure for a specific molecule.

Description

This allows α helical secondary structure to be defined for the internal structural object.

17.2.235 structure.add_model**Synopsis**

Add a new model.

Defaults

```
structure.add_model(model_num=None)
```

Keyword arguments

model_num: The number of the new model.

Description

This allows new models to be added to the internal structural object. Note that no structural information is allowed to be present.

17.2.236 structure.add_sheet



17.2.237 structure.atomic_fluctuations



Synopsis

Define a β sheet.

Defaults

```
structure.add_sheet(strand=None, sheet_id='A',
strand_count=2, strand_sense=0, start=None, end=None,
mol_name=None, current_atom=None, prev_atom=None)
```

Keyword arguments

strand: Strand number which starts at 1 for each strand within a sheet and increases by one.

sheet_id: The sheet identifier. To match the PDB standard, sheet IDs should range from 'A' to 'Z'.

strand_count: The number of strands in the sheet.

strand_sense: Sense of strand with respect to previous strand in the sheet. 0 if first strand, 1 if parallel, -1 if anti-parallel.

start: The residue number for the start of the sheet.

end: The residue number for the end of the sheet.

mol_name: Define the secondary structure for a specific molecule.

current_atom: The name of the first atom in the current strand, to link the current back to the previous strand.

prev_atom: The name of the last atom in the previous strand, to link the current back to the previous strand.

Description

This allows β sheet secondary structure to be defined for the internal structural object.

Synopsis

Create an interatomic distance fluctuation correlation matrix.

Defaults

```
structure.atomic_fluctuations(pipes=None, models=
None, molecules=None, atom_id=None, measure=
'distance', file=None, format='text', dir=None, force=
False)
```

Keyword arguments

pipes: The data pipes to generate the interatomic distance fluctuation correlation matrix for.

models: The list of models for each data pipe to generate the interatomic distance fluctuation correlation matrix for. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to generate the interatomic distance fluctuation correlation matrix for. This allows differently named molecules in the same or different data pipes to be superimposed. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest. This can be used to restrict the correlation matrix to one atom per residue, for example.

measure: The type of fluctuation to investigate. This allows for both interatomic distance and vector angle fluctuations to be calculated.

file: The name of the text file to create.

format: The output format. For all formats other than the text file, a second file will be created with the same name as the text file but with the appropriate file extension added.

dir: The directory to save the file to.

force: A flag which if set to True will cause any pre-existing files to be overwritten.

Description

This is used to visualise the interatomic fluctuations between different structures. By setting the measure argument, different categories of fluctuations can seen:

'distance' – The interatomic distance fluctuations is the default option. The corrected sample standard deviation (SD) is calculated for the distances between all atom pairs, resulting in a pairwise matrix of SD values. This is frame independent and hence is superimposition independent.

'angle' – The interatomic vector angle fluctuations. The corrected sample standard deviation (SD) is calculated for the angles between the inter atom vectors all atom pairs to an average vector. This also produces a pairwise matrix of SD values.

'parallax shift' – The interatomic parallax shift fluctuations. The corrected sample standard deviation (SD) is calculated for the parallax shift between the inter atom vectors all atom pairs to an average vector. This also produces a pairwise matrix of SD values. The parallax shift is calculated as the dot product of the interatomic vector and the unit average vector, times the unit average vector. It is a frame and superimposition dependent measure close to orthogonal to the interatomic distance fluctuations. It is similar to the angle measure however, importantly, it is independent of the distance between the two atoms.

For the output file, the currently supported formats are:

'text' – This is the default value and will result in a single text file being created.

'gnuplot' – This will create both a text file with the data and a script for visualising the correlation matrix using gnuplot. The script will have the same name as the text file, however the file extension will be changed to *.gnu.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to '**'@N,C,CA,O'**', assuming those are the names of the atoms in the 3D structural file.

Prompt examples

To create the interatomic distance fluctuation correlation matrix for the models 1, 3, and 5, type:

```
relax> structure.atomic_fluctuations(models
= [[1, 3, 5]], file=
'atomic_fluctuation_matrix')
```

To create the interatomic distance fluctuation correlation matrix for the molecules 'A', 'B', 'C', and 'D', type:

```
relax> structure.atomic_fluctuations(
molecules=[['A', 'B', 'C', 'D']], file=
'atomic_fluctuation_matrix')
```

17.2.238 structure.com**Synopsis**

Calculate the centre of mass (CoM) for all structures.

Defaults

structure.com(model=None, atom_id=None)

Keyword arguments

model: The optional structural model number to restrict the calculation of the centre of mass to.

atom_id: The atom identification string to restrict the CoM calculation to.

Description

This user function will calculate the centre of mass (CoM) for all loaded structures, printing out the position and storing it in the current data pipe.

Prompt examples

To determine the centre of mass of all structure, simply type:

```
relax> structure.com()
```

17.2.239 structure.connect_atom**Synopsis**

Connect two atoms.

Defaults

structure.connect_atom(index1=None, index2=None)

Keyword arguments

index1: The global index of the first atom.

index2: The global index of the second atom.

Description

This allows atoms to be connected in the internal structural object. The global index is normally equal to the PDB atom number minus 1.

17.2.240 structure.create_diff_tensor_pdb



Synopsis

Create a PDB file to represent the diffusion tensor.

Defaults

```
structure.create_diff_tensor_pdb(scale=1.8e-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 to place the file into.

force: A flag which, if set to True, will overwrite the any pre-existing file.

Description

This 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 $\tau_{\text{-e}}$, 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 Å and has the default value of 1.8e-6 s.Angstrom). Therefore the rotational diffusion rate per Å is equal the inverse of the scale value (which defaults to 5.56e5 s^-1.Angstrom^-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:

Please see Table 17.26 on page 630.

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 Å within 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 , $\mathfrak{D}_{||}$, \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 Å 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.

Table 17.26: Diffusion tensor PDB representation sizes using the default scaling for different diffusion tensors

τ_m (ns)	\mathfrak{D}_{iso} (s^{-1})	Radius (\AA)
1	1.67e8	300
3	5.56e7	100
10	1.67e7	30
30	5.56e6	10

Description
17.2.241 <code>structure.create_rotor_pdb</code>
 +
Synopsis
Create a PDB file representation of a rotor.
Defaults
<code>structure.create_rotor_pdb(file='rotor.pdb', dir=None, rotor_angle=0.0, axis=None, axis_pt=None, centre=None, span=2e-09, blade_length=5e-10, force=False, staggered=False)</code>
Keyword arguments
file: The name of the PDB file.
dir: The directory to place the file into.
rotor_angle: The angle of the rotor motion in degrees.
axis: The vector defining the rotor axis.
axis_pt: A point lying anywhere on the rotor axis. This is used to define the position of the axis in 3D space.
centre: The central point of the representation. If this point is not on the rotor axis, then the closest point on the axis will be used for the centre.
span: The distance from the central point to the rotor blades (meters).
blade_length: The length of the representative rotor blades.
force: A flag which if True will overwrite the file if it already exists.
staggered: A flag which if True will cause the rotor blades to be staggered. This is used to avoid blade overlap.

17.2.242 structure.create_vector_dist



Synopsis

Create a PDB file representation of the distribution of XH bond vectors.

Defaults

```
structure.create_vector_dist(length=2e-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 creates a PDB file containing an artificial vectors, the length of which default to 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.

17.2.243 structure.delete



Synopsis

Delete structural information.

Defaults

```
structure.delete(atom_id=None, model=None, verbosity=1, spin_info=True)
```

Keyword arguments

- atom_id:** The atom identification string.
- model:** Individual structural models from a loaded ensemble can be deleted by specifying the model number.
- verbosity:** The amount of information to print out. Set to zero to silence the user function, or one to see all messages.
- spin_info:** A flag which if True will cause all structural information in the spin containers and interatomic data containers to be deleted as well. If False, then only the 3D structural data will be deleted.

Description

This will delete structural information from the current data pipe. All spin and sequence information loaded from these structures will be preserved - this only affects the structural data. The atom ID argument can be used to restrict deletion to parts of the loaded molecules, or the model argument can be used to delete individual structural models from an ensemble.

Prompt examples

To delete everything, simply type:

```
relax> structure.delete()
```

To delete residues 50 to 100 of the molecule called ‘Ap4Aase’, type one of:

```
relax> structure.delete(':50-100')
```

```
relax> structure.delete(atom_id=':50-100')
```

17.2.244 structure.delete_ss**Synopsis**

Delete secondary structure information.

Defaults

structure.delete_ss()

Description

This will delete all secondary structure information from the current data pipe.

Prompt examples

To delete all secondary structure, simply type:

```
relax> structure.delete_ss()
```

17.2.245 structure.displacement**Synopsis**

Determine the rotational and translational displacement between a set of models or molecules.

Defaults

structure.displacement(pipes=None, models=None, molecules=None, atom_id=None, centroid=None)

Keyword arguments

pipes: The data pipes to determine the displacements for.

models: The list of models for each data pipe to determine the displacements for. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to determine the displacements for. This allows differently named molecules in the same or different data pipes to be superimposed. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest.

centroid: The alternative position of the centroid.

Description

This user function allows the rotational and translational displacement between different models or molecules to be calculated. The information will be printed out in various formats and held in the relax data store. This is directional, so there is a starting and ending position for each displacement. Therefore the displacements in all directions between all models and molecules will be calculated.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to ‘@N,C,CA,O’, assuming those are the names of the atoms in the 3D structural file.

By supplying the position of the centroid, an alternative position than the standard rigid body centre is used as the focal point of the motion. This allows, for example, a pivot of a rotational domain motion to be specified. This is not a formally correct algorithm, all translations will be zero, but does give an indication to the amplitude of the pivoting angle.

Prompt examples

To determine the rotational and translational displacements between all sets of models, type:

```
relax> structure.displacement()
```

17.2.246 structure.find_pivot



Synopsis

Find the pivot point of the motion of a set of structures.

Defaults

```
structure.find_pivot(pipes=None, models=None,
                     molecules=None, atom_id=None, init_pos=None,
                     func_tol=1e-05, box_limit=200)
```

Keyword arguments

pipes: The data pipes to use in the motional pivot algorithm.

models: The list of models for each data pipe to use in the motional pivot algorithm. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to use in the motional pivot algorithm. This allows differently named molecules in the same or different data pipes to be used. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest.

init_pos: The initial position of the pivot.

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-5.

box_limit: The pivot point is constrained within a box of +/- x Å using the logarithmic barrier function together with simplex optimisation. This argument is the value of x .

Description

This is used to find pivot point of motion between a set of structural models. If the list of models is not supplied, then all models will be used.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different

atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to ‘`ON,C,CA,O`’, assuming those are the names of the atoms in the 3D structural file.

By supplying the position of the centroid, an alternative position than the standard rigid body centre is used as the focal point of the superimposition. This allows, for example, the superimposition about a pivot point.

17.2.247 `structure.get_pos`



Synopsis

Extract the atomic positions from the loaded structures for the given spins.

Defaults

```
structure.get_pos(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 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 averaging the atomic positions, then average position of all models will be loaded into the spin container. Otherwise the positions from all models will be loaded separately.

Prompt examples

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='ON')
```

Prompt examples

17.2.248 structure.load_spins



Synopsis

Load spins from the structure into the relax data store.

Defaults

```
structure.load_spins(spin_id=None, from_mols=None,
mol_name_target=None, ave_pos=True, spin_num=True)
```

Keyword arguments

spin_id: The spin identification string for the selective loading of certain spins into the relax data store.

from_mols: The list of similar, but not necessarily identical molecules to load spin information from.

mol_name_target: The name of target molecule container, overriding the name of the loaded structures.

ave_pos: A flag specifying if the position of the atom is to be averaged across models.

spin_num: A flag specifying if the spin number should be loaded.

Description

This 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 the spin ID is left unspecified, then all molecules, residues, and atoms will be placed within the data store (and all atoms will be treated as spins).

As an alternative to using structural models, by specifying the list of molecules to load spins from similar though not necessarily identical molecules will be combined. In this case, the target molecule name must be supplied to create a single combined molecule. And only a single model can be loaded in the current data pipe. The spin numbering will be dropped to allow for sequential atom numbering in the PDB and other formats. Therefore only the residue number and name and atom name will be preserved for creating the spin containers. If the spin is only present in a subset of the structures, then the positional information will only be taken from that subset and hence the number of positions might be different for different spins.

If averaging the atomic positions, then average position of all models or molecules will be loaded into the spin container. Otherwise the positions from all models or molecules will be loaded separately.

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)
```

17.2.249 structure.mean**Synopsis**

Calculate the mean structure from all loaded models.

Defaults

```
structure.mean(pipes=None, models=None, molecules=None, atom_id=None, set_mol_name=None, set_model_num=None)
```

Keyword arguments

pipes: The data pipes containing structures to average.

models: The list of models for each data pipe containing structures to average. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to average. This allows differently named molecules in the same or different data pipes to be averaged. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest. This can be used to restrict the averaged structure to one atom per residue, for example.

set_mol_name: Set the optional name of the averaged molecule.

set_model_num: Set the optional model number of the averaged molecule.

Description

This will calculate and store the mean structure from a collection of related molecules. If a new molecule name or model number is not supplied, the mean structure will replace all the models in the internal structural object. This is provided as a structural aid, specifically for superimposition purposes.

17.2.250 structure.pca**Synopsis**

Principle component analysis (PCA) of the motions in an ensemble of structures.

Defaults

```
structure.pca(pipes=None, models=None, molecules=None, obs_pipes=None, obs_models=None, obs_molecules=None, atom_id=None, algorithm='eigen', num_modes=4, format='grace', dir=None)
```

Keyword arguments

pipes: The data pipes to perform the PC analysis on.

models: The list of models for each data pipe to perform the PC analysis on. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to perform the PC analysis on. The PCA will only be calculated for atoms with identical residue name and number and atom name. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

obs_pipes: The data pipes in the PC analysis which will have zero weight. These structures are for comparison.

obs_models: The list of models for each data pipe in the PC analysis which will have zero weight. These structures are for comparison. The number of elements must match the pipes argument. If no models are given, then all will be used.

obs_molecules: The list of molecules for each data pipe in the PC analysis which will have zero weight. These structures are for comparison. The PCA will only be calculated for atoms with identical residue name and number and atom name. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest.

algorithm: The PCA algorithm used to find the principle components of. This can be either 'eigen' for an eigenvalue/eigenvector decomposition, or 'svd' for a singular value decomposition.

num_modes: The number of PCA modes to calculate.

format: The format of the plot data.

dir: The directory to save the graphs into.

Description

Perform a principle component analysis (PCA) for all the chosen structures. 2D graphs of the PC projections will be generated and placed in the specified directory.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

A subset of the structures can be set as ‘observing’. This means that they will have a weight of zero when constructing the covariance matrix and determining its eigenvectors. Therefore the structures will not contribute to the principle components, but will be present and compared to structures used in the analysis.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to ‘@N,C,CA,O’, assuming those are the names of the atoms in the 3D structural file.

Prompt examples

To determine the PCA modes of all models in the current data pipe, simply type:

```
relax> structure.pca()
```

17.2.251 structure.read_gaussian



Synopsis

Reading structures from Gaussian log files.

Defaults

```
structure.read_gaussian(file=None, dir=None,
set_mol_name=None, set_model_num=None, verbosity=1)
```

Keyword arguments

file: The name of the Gaussian log file.

dir: The directory where the file is located.

set_mol_name: Set the names of the read molecules. If unset, then the molecules will be automatically labelled based on the file name or other information. This can either be a single name or a list of names.

set_model_num: Set the model numbers of the loaded molecules. This can be a single number or list of numbers.

verbosity: The amount of information to print out. Set to zero to silence the user function, or one to see all messages.

Description

The atomic positions from a Gaussian log file can be read into relax. If optimisation has been preformed, the last set of atomic coordinates from the log will be read to obtain the final structure. The log file can be Gzip or Bzip2 compressed.

The setting of molecule names is used to name the molecules within the Gaussian file. If not set, then the molecules will be named after the file name, with the molecule number appended if more than one exists. By setting the molecule name or setting the model number, the loaded structure can be stored as a specific model or as a different molecule.

Prompt examples

To load all structures from the Gaussian file ‘taxol.log’ in the directory ‘~/logs’, including all models and all molecules, type one of:

```
relax> structure.read_gaussian('taxol.log',
'~/logs')
```

```
relax> structure.read_gaussian(file='taxol.log', dir=logs')
```

17.2.252 structure.read_pdb



Synopsis

Reading structures from PDB files.

Defaults

```
structure.read_pdb(file=None, dir=None, read_mol=None, set_mol_name=None, read_model=None, set_model_num=None, alt_loc=None, verbosity=1, merge=False)
```

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. The molecules are numbered consecutively from 1. If unset, then all molecules will be loaded. By providing a list of numbers such as [1, 2], multiple molecules will be read.

set_mol_name: Set the names of the read molecules. If unset, then the molecules will be automatically labelled based on the file name or other information. This can either be a single name or a list of names.

read_model: If set, only the given model number(s) from the PDB file will be read. Otherwise all models will be read. This can be a single number or list of numbers.

set_model_num: Set the model numbers of the loaded molecules. If unset, then the PDB model numbers will be preserved if they exist. This can be a single number or list of numbers.

alt_loc: The PDB ATOM record ‘Alternate location indicator’ field value.

verbosity: The amount of information to print out. Set to zero to silence the user function, or one to see all messages.

merge: A flag which if set to True will try to merge the PDB structure into the currently loaded structures.

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 or 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.

In a PDB file, the models are specified by the MODEL PDB record. All the supported PDB readers in relax recognise this. The internal reader defines molecules using the TER PDB record. In both cases, the molecules will be numbered consecutively from 1.

Setting the molecule name allows the molecule within the PDB (within one model) to have a custom name. 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.

This 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.

If a PDB file contains alternative atomic locations, then the alternate location indicator must be specified to allow one of the multiple coordinate sets to be selected.

Prompt 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 10th model from the file ‘test.pdb’ and naming it ‘CaM’, use one of:

```
relax> structure.read_pdb('test.pdb',
                           read_model=10, set_mol_name='CaM')
relax> structure.read_pdb(file='test.pdb',
                           read_model=10, set_mol_name='CaM')
```

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)
```

17.2.253 structure.read_xyz



Synopsis

Reading structures from XYZ files.

Defaults

```
structure.read_xyz(file=None, dir=None, read_mol=None,
set_mol_name=None, read_model=None, set_model_num=
None, verbosity=1)
```

Keyword arguments

file: The name of the XYZ file.

dir: The directory where the file is located.

read_mol: If set, only the given molecule(s) will be read. The molecules are numbered consecutively from 1. If unset, then all molecules will be loaded. By providing a list of numbers such as [1, 2], multiple molecules will be read.

set_mol_name: Set the names of the read molecules. If unset, then the molecules will be automatically labelled based on the file name or other information. This can either be a single name or a list of names.

read_model: If set, only the given model number(s) from the PDB file will be read. Otherwise all models will be read. This can be a single number or list of numbers.

set_model_num: Set the model numbers of the loaded molecules. If unset, then the PDB model numbers will be preserved if they exist. This can be a single number or list of numbers.

verbosity: The amount of information to print out. Set to zero to silence the user function, or one to see all messages.

Description

The XYZ files with different models, which defined as an ensemble of the same molecule but with different atomic positions, can be read into relax. If there are several molecules in one xyz file, please separate them into different files and then load them individually. Loading different models and different molecules is controlled by specifying the molecule number read, setting the molecule names, specifying which model to read, and setting the model numbers.

The setting of molecule names is used to name the molecules within the XYZ (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.

Prompt examples

To load all structures from the XYZ file ‘test.xyz’ in the directory ‘~/xyz’, including all models and all molecules, type one of:

```
relax> structure.read_xyz('test.xyz', '~/xyz')
      )
relax> structure.read_xyz(file='test.xyz',
      dir='xyz')
```

To load the 10th model from the file ‘test.xyz’ and naming it ‘CaM’, use one of:

```
relax> structure.read_xyz('test.xyz',
      read_model=10, set_mol_name='CaM')
relax> structure.read_xyz(file='test.xyz',
      read_model=10, set_mol_name='CaM')
```

To load models 1 and 5 from the file ‘test.xyz’ as two different structures of the same model, type one of:

```
relax> structure.read_xyz('test.xyz',
      read_model=[1, 5], set_model_num=[1,
      1])
relax> structure.read_xyz('test.xyz',
      set_mol_name=['CaM_1', 'CaM_2'],
      read_model=[1, 5], set_model_num=[1,
      1])
```

To load the files ‘test_1.xyz’, ‘test_2.xyz’, ‘test_3.xyz’ and ‘test_4.xyz’ as models, type the following sequence of commands:

```
relax> structure.read_xyz('test_1.xyz',
      set_mol_name='test_1', set_model_num=1)
relax> structure.read_xyz('test_2.xyz',
      set_mol_name='test_2', set_model_num=2)
relax> structure.read_xyz('test_3.xyz',
      set_mol_name='test_3', set_model_num=3)
relax> structure.read_xyz('test_4.xyz',
      set_mol_name='test_4', set_model_num=4)
```

17.2.254 structure.rmsd



Synopsis

Determine the RMSD between structures.

Defaults

```
structure.rmsd(pipes=None, models=None, molecules=None, atom_id=None, atomic=False)
```

Keyword arguments

pipes: The data pipes to determine the RMSD for.

models: The list of models for each data pipe to determine the RMSD for. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to determine the RMSD for. The RMSD will only be calculated for atoms with identical residue name and number and atom name. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest.

atomic: A flag which if True will allow for per-atom RMSDs to be additionally calculated.

Description

This allows the root mean squared deviation (RMSD) between all structures to be calculated. The RMSDs for individual structures to the mean structure will be calculated and reported. These values averaged to produce a global RMSD stored in the structural object of the current data pipe. If the ‘atomic’ argument is set, per-atom RMSDs will additionally be calculated stored in spin containers.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to ‘@N,C,CA,O’, assuming those are the names of the atoms in the 3D structural file.

Prompt examples

To determine the RMSD of all models in the current data pipe, simply type:

```
relax> structure.rmsd()
```

For the backbone heavy atom RMSD of all models in the current data pipe, simply type:

```
relax> structure.rmsd(atom_id='@N,C,CA,O')
```

To calculate the C-alpha backbone RMSDs of all models in the current data pipe, type:

```
relax> structure.rmsd(atom_id='CA', atomic=True)
```

17.2.255 structure.rotate**Synopsis**

Rotate the internal structural object about the given origin by the rotation matrix.

Defaults

```
structure.rotate(R=array([[ 1., 0., 0.], [ 0., 1., 0.], [ 0., 0., 1.]]), origin=None, model=None, atom_id=None)
```

Keyword arguments

R: The rotation matrix in forwards rotation notation.

origin: The origin or pivot of the rotation.

model: The model to rotate (which if not set will cause all models to be rotated).

atom_id: The atom identification string.

Description

This is used to rotate the internal structural data by the given rotation matrix. If the origin is supplied, then this will act as the pivot of the rotation. Otherwise, all structural data will be rotated about the point [0, 0, 0]. The rotation can be restricted to one specific model.

17.2.256 structure.sequence_alignment**Synopsis**

Multiple sequence alignment (MSA) of structural data.

Defaults

```
structure.sequence_alignment(pipes=None, models=None, molecules=None, msa_algorithm='Central Star', pairwise_algorithm=None, matrix=None, gap_open_penalty=10.0, gap_extend_penalty=1.0, end_gap_open_penalty=0.0, end_gap_extend_penalty=0.0)
```

Keyword arguments

pipes: The data pipes to use in the sequence alignment.

models: The list of models for each data pipe to use in the sequence alignment. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to use in the sequence alignment. This allows differently named molecules in the same or different data pipes to be superimposed. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

msa_algorithm: The multiple sequence alignment (MSA) algorithm used to align all the primary sequence of all structures of interest.

pairwise_algorithm: The pairwise alignment algorithm to align each pair of sequences.

matrix: The substitution matrix to use in the pairwise sequence alignment algorithm.

gap_open_penalty: The penalty for introducing gaps, as a positive number.

gap_extend_penalty: The penalty for extending a gap, as a positive number.

end_gap_open_penalty: The optional penalty for opening a gap at the end of a sequence.

end_gap_extend_penalty: The optional penalty for extending a gap at the end of a sequence.

Description

To find the atoms in common between different molecules, a MSA of the primary sequence of the molecules is required. This sequence alignment will then subsequently be used by any other user function which operates on multiple molecules. The following MSA algorithms can be selected:

'Central Star' – This is a heuristic, progressive alignment method using pairwise alignments to construct a MSA. It consists of four major steps – pairwise alignment between all sequence pairs, finding the central sequence, iteratively aligning the sequences to the gapped central sequence, and introducing gaps in previous alignments during the iterative alignment.

'residue number' – This will simply align the molecules based on residue number.

For the MSA algorithms which require pairwise alignments, the following subalgorithms can be used:

'NW70' – The Needleman-Wunsch alignment algorithm. This has been modified to use the logic of the EMBOSS software for handling gap opening and extension penalties, as well as end penalties.

For the MSAs or pairwise alignments which require a substitution matrix, one of the following can be used:

'BLOSUM62' – The BLOcks SUbstitution Matrix for proteins with a cluster percentage $\geq 62\%$.

'PAM250' – The point accepted mutation matrix for proteins with $n = 250$ evolutionary distance.

'NUC 4.4' – The nucleotide 4.4 matrix for DNA/RNA.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

Prompt examples

To superimpose the structures in the '**A**' data pipe onto the structures of the '**B**' data pipe using backbone heavy atoms, type:

```
relax> structure.sequence_alignment(pipes=['B', 'A'], atom_id='@N,C,CA,O')
```

17.2.257 structure.superimpose



Synopsis

Superimpose a set of structures.

Defaults

```
structure.superimpose(pipes=None, models=None, molecules=None, atom_id=None, displace_id=None, method='fit to mean', centre_type='centroid', centroid=None)
```

Keyword arguments

pipes: The data pipes to use in the superimposition.

models: The list of lists of models for each data pipe to use in the superimposition. The number of elements in the first dimension must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to use in the superimposition. This allows differently named molecules in the same or different data pipes to be superimposed. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest. This allows a subset of all residues or atoms to be used in the superimposition. For example for protein backbone heavy atoms, this can be set to '@N,C,CA,O'.

displace_id: The atom identification string for restricting the displacement to a subset of all atoms. If not set, then all atoms will be translated and rotated. If supplied as a list of IDs, then the number of items must match the number of structures.

method: The superimposition method.

centre_type: The type of centre to use for the superimposition, i.e. either the standard centroid superimposition or a superimposition using the centre of mass (CoM).

centroid: The alternative position of the centroid.

Description

This allows a set of related structures to be superimposed to each other. If a multiple sequence alignment (MSA) of the molecules has already been performed with the structure.sequence_alignment user function, this will

allow residues with different numbering to be superimposed. Otherwise only residues with the same numbering will be used in the superimposition. Two superimposition methods are currently supported:

'fit to mean' – All models are fit to the mean structure. This is the default and most accurate method for an ensemble description. It is an iterative method which first calculates a mean structure and then fits each model to the mean structure using the Kabsch algorithm. This is repeated until convergence.

'fit to first' – This is quicker but is not as accurate for an ensemble description. The Kabsch algorithm is used to rotate and translate each model to be superimposed onto the first model of the first data pipe.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to '`@N,C,CA,O`', assuming those are the names of the atoms in the 3D structural file.

The displacement ID string, which is similar to the atom ID, gives finer control over which atoms are translated and rotated by the algorithm. When not set this allows, for example, to align structures based on a set of backbone heavy atoms and the backbone protons and side-chains are displaced by default. Or if set to the same as the atom ID, if a single domain is aligned, then just that domain will be displaced.

By supplying the position of the centroid, an alternative position than the standard rigid body centre is used as the focal point of the superimposition. This allows, for example, the superimposition about a pivot point.

Prompt examples

To superimpose all sets of models, type one of:

```
relax> structure.superimpose()
```

```
relax> structure.superimpose(method='fit to mean')
```

To superimpose the models 1, 2, 3, 5 onto model 4, type:

```
relax> structure.superimpose(models=[[4, 1, 2, 3, 5]], method='fit to first')
```

To superimpose an ensemble of protein structures using only the backbone heavy atoms, type one of:

```
relax> structure.superimpose(atom_id='@N,C,CA,O')
```

```
relax> structure.superimpose(method='fit to mean', atom_id='@N,C,CA,O')
```

To superimpose the structures in the 'A' data pipe onto the structures of the 'B' data pipe using backbone heavy atoms, type one of:

```
relax> structure.superimpose(['B', 'A'], None, 'fit to first', '@N,C,CA,O')
```

```
relax> structure.superimpose(pipes=['B', 'A'], method='fit to first', atom_id='@N,C,CA,O')
```

17.2.258 structure.translate**Synopsis**

Laterally displace the internal structural object by the translation vector.

Defaults

```
structure.translate(T=None, model=None, atom_id=None)
```

Keyword arguments

T: The translation vector.

model: The model to translate (which if not set will cause all models to be translated).

atom_id: The atom identification string.

Description

This is used to translate the internal structural data by the given translation vector. The translation can be restricted to one specific model.

17.2.259 structure.web_of_motion**Synopsis**

Create a PDB representation of motion between structures using a web of interconnecting lines.

Defaults

```
structure.web_of_motion(pipes=None, models=None, molecules=None, atom_id=None, file=None, dir=None, force=False)
```

Keyword arguments

pipes: The data pipes to generate the web between.

models: The list of models for each data pipe to generate the web between. The number of elements must match the pipes argument. If no models are given, then all will be used.

molecules: The list of molecules for each data pipe to generate the web between. This allows differently named molecules in the same or different data pipes to be superimposed. The number of elements must match the pipes argument. If no molecules are given, then all will be used.

atom_id: The atom identification string of the coordinates of interest.

file: The name of the PDB file.

dir: The directory to save the file to.

force: A flag which if set to True will cause any pre-existing files to be overwritten.

Description

This will create a PDB representation of the motion between the atoms of a given set of structures. Identical atoms of the structures are concatenated into one model, within a temporary internal structural object, linked together using PDB CONECT records, and then written to the PDB file.

Support for multiple structures is provided by the data pipes, model numbers and molecule names arguments. Each data pipe, model and molecule combination will be treated as a separate structure. As only atomic coordinates with the same residue name and number and atom name will be assembled, structures with slightly different atomic structures can be compared. If the list of models is not supplied, then all models of all data pipes will be

used. If the optional molecules list is supplied, each molecule in the list will be considered as a separate structure for comparison between each other.

The atom ID string, which uses the same notation as the spin ID, can be used to restrict the coordinates compared to a subset of molecules, residues, or atoms. For example to only use backbone heavy atoms in a protein, set the atom ID to ‘ON,C,CA,O’, assuming those are the names of the atoms in the 3D structural file.

Prompt examples

To create a web of motion for the models 1, 3, and 5, type:

```
relax> structure.web_of_motion(models=[[1, 3, 5]], file='web.pdb')
```

To create a web of motion for the molecules ‘A’, ‘B’, ‘C’, and ‘D’, type:

```
relax> structure.web_of_motion(molecules=[['A', 'B', 'C', 'D']], file='web.pdb')
```

17.2.260 structure.write_pdb



Synopsis

Writing structures to a PDB file.

Defaults

```
structure.write_pdb(file=None, dir=None, model_num=None, compress_type=0, force=False)
```

Keyword arguments

file: The name of the PDB file.

dir: The directory where the file is located.

model_num: Restrict the writing of structural data to a single model in the PDB file.

compress_type: The type of compression to use when creating the file.

force: A flag which if set to True will cause any pre-existing files to be overwritten.

Description

This will write all of the structural data loaded in the current data pipe to be converted to the PDB format and written to file. Specifying the model number allows single models to be output.

The default behaviour of this function is to not compress the file. The compression can, however, be changed to either bzip2 or gzip compression. If the ‘.bz2’ or ‘.gz’ extension is not included in the file name, it will be added. This behaviour is controlled by the compression type which can be set to

0 – No compression (no file extension).

1 – bzip2 compression (‘.bz2’ file extension).

2 – gzip compression (‘.gz’ file extension).

Prompt examples

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)
```

17.2.261 system.cd



Synopsis

Change the current working directory to the specified path.

Defaults

```
system.cd(path=None)
```

Keyword arguments

path: The path to the new current working directory.

Description

The equivalent of python module os.chdir(path). Change the current working directory to the specified path.

To change the current working directory, type:

```
relax> system.cd("/path/to/dir")
```

17.2.262 system.pwd**Synopsis**

Display the current working directory.

Defaults

system.pwd()

Description

This will display the current working directory.

The directory can be changed with the system.cd(path) user function.

relax> system.pwd()

relax> system.cd("/path/to/dir")

17.2.263 system.sys_info**Synopsis**

Display all system information relating to this version of relax.

Defaults

system.sys_info()

Description

This will display all of the relax, Python, python package and hardware information currently being used by relax. This is useful for seeing if all packages are up to date and if the correct software versions are being used. It is also very useful information for reporting relax bugs.

17.2.264 system.time**Synopsis**

Display the current time.

Defaults

`system.time()`

Description

This user function will display the current time which can be useful for timing long calculations by having time information in any saved log files.

17.2.265 value.copy**Synopsis**

Copy parameters from one data pipe to another.

Defaults

`value.copy(pipe_from=None, pipe_to=None, param=None, force=False)`

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. Only one parameter may be selected.

`force`: A flag which, if set to True, will cause the destination parameter to be overwritten.

Description

If this 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.

Relaxation curve fitting parameters

Please see Table 17.27 on page 650.

Model-free parameters

Please see Table 17.28 on page 650.

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 data pipe corresponds to the 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 obtain the correct value:

Table 17.27: Relaxation curve fitting parameters.

Name	Description
rx	Either the R ₁ or R ₂ relaxation rate
i0	The initial intensity
iinf	The intensity at infinity

Table 17.28: Model-free parameters.

Name	Description
s2	S^2 , the model-free generalised order parameter ($S^2 = S_f^2 \cdot S_{2s}$)
s2f	S_f^2 , the faster motion model-free generalised order parameter
s2s	S_s^2 , the slower motion model-free generalised order parameter
local_tm	The spin specific global correlation time (seconds)
te	Single motion effective internal correlation time (seconds)
tf	Faster motion effective internal correlation time (seconds)
ts	Slower motion effective internal correlation time (seconds)
rex	Chemical exchange relaxation ($\sigma_{\text{ex}} = R_{\text{ex}} / \omega^{\star 2}$)
csa	Chemical shift anisotropy (unitless)

```
value = rex / (2.0 * pi * frequency) ** 2
```

where:

rex is the chemical exchange value for the current frequency.

frequency is the proton frequency corresponding to the data.

Reduced spectral density mapping parameters

Please see Table 17.29 on page 651.

In reduced spectral density mapping, the CSA value must be set prior to the calculation of spectral density values.

Consistency testing parameters

Please see Table 17.30 on page 651.

In consistency testing, the CSA value, angle Theta ('orientation') and global correlation time must be set prior to the calculation of consistency functions.

N-state model parameters

Please see Table 17.31 on page 651.

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 given as the index argument. So the Euler angle γ of the third state is specified using the parameter name 'gamma' and index of 2.

Relaxation dispersion parameters

Please see Table 17.32 on page 652.

Any of the relaxation dispersion parameters which are of the 'float' type can be set. Note that setting values for parameters which are not part of the model will have no effect.

Frame order parameters

Please see Table 17.6 on page 474.

Prompt examples

To copy the CSA values from the data pipe 'm1' to 'm2', type:

```
relax> value.copy('m1', 'm2', 'csa')
```

Table 17.29: Reduced spectral density mapping parameters.

Name	Description
j0	Spectral density value at 0 MHz - $J(0)$
jwx	Spectral density value at the frequency of the heteronucleus - $J(\omega_X)$
jwh	Spectral density value at the frequency of the proton - $J(\omega_H)$
csa	Chemical shift anisotropy (unitless)

Table 17.30: Consistency testing parameters.

Name	Description
j0	Spectral density value at 0 MHz (from Farrow et al. (1995) JBNMR, 6: 153-162)
f_eta	Eta-test (from Fushman et al. (1998) JACS, 120: 10947-10952)
f_r2	R2-test (from Fushman et al. (1998) JACS, 120: 10947-10952)
csa	Chemical shift anisotropy (unitless)
orientation	Angle between the $^{15}\text{N}-\text{H}$ vector and the principal axis of the ^{15}N chemical shift tensor
tc	The single global correlation time estimate/approximation

Table 17.31: N-state model parameters.

Name	Description	Type
Axx	The Axx component of the alignment tensor	float
Ayy	The Ayy component of the alignment tensor	float
Axy	The Axy component of the alignment tensor	float
Axz	The Axz component of the alignment tensor	float
Ayz	The Ayz component of the alignment tensor	float
probs	The probabilities of each state	list
alpha	The α Euler angles (for the rotation of each state)	list
beta	The β Euler angles (for the rotation of each state)	list
gamma	The γ Euler angles (for the rotation of each state)	list
paramagnetic_centre	The paramagnetic centre	list

Table 17.32: Relaxation dispersion parameters.

Name	Description	Type
r2eff	The effective transversal relaxation rate	dict
i0	The initial intensity	dict
r1	The longitudinal relaxation rate	dict
r2	The transversal relaxation rate	dict
r2a	The transversal relaxation rate for state A in the absence of exchange	dict
r2b	The transversal relaxation rate for state B in the absence of exchange	dict
pA	The population for state A	float
pB	The population for state B	float
pC	The population for state C	float
phi_ex	The $\phi_{ex} = pA.pB.dw^{**2}$ value (ppm^2)	float
phi_ex_B	The fast exchange factor between sites A and B (ppm^2)	float
phi_ex_C	The fast exchange factor between sites A and C (ppm^2)	float
padw2	The $pA.dw^{**2}$ value (ppm^2)	float
dw	The chemical shift difference between states A and B (in ppm)	float
dw_AB	The chemical shift difference between states A and B for 3-site exchange (in ppm)	float
dw_AC	The chemical shift difference between states A and C for 3-site exchange (in ppm)	float
dw_BC	The chemical shift difference between states B and C for 3-site exchange (in ppm)	float
dwH	The proton chemical shift difference between states A and B (in ppm)	float
dwH_AB	The proton chemical shift difference between states A and B for 3-site exchange (in ppm)	float
dwH_AC	The proton chemical shift difference between states A and C for 3-site exchange (in ppm)	float
dwH_BC	The proton chemical shift difference between states B and C for 3-site exchange (in ppm)	float
kex	The exchange rate	float
kex_AB	The exchange rate between sites A and B for 3-site exchange with $kex_AB = k_AB + k_BA$ (rad.s^-1)	float
kex_AC	The exchange rate between sites A and C for 3-site exchange with $kex_AC = k_AC + k_CA$ (rad.s^-1)	float
kex_BC	The exchange rate between sites B and C for 3-site exchange with $kex_BC = k_BC + k_CB$ (rad.s^-1)	float
kB	Approximate chemical exchange rate constant between sites A and B (rad.s^-1)	float
kC	Approximate chemical exchange rate constant between sites A and C (rad.s^-1)	float
tex	The time of exchange ($tex = 1/kex$)	float
k_AB	The exchange rate from state A to state B	float
k_BA	The exchange rate from state B to state A	float

17.2.266 value.display \AA **Reduced spectral density mapping parameters**

Please see Table 17.11 on page 487.

Synopsis

Display spin specific parameter values.

Defaults`value.display(param=None, scaling=1.0)`**Keyword arguments**

param: The parameter to display. Only one parameter may be selected.

scaling: The factor to scale parameters by.

Description

The values corresponding to the given parameter will be displayed. The scaling argument can be used to scale the parameter values. This can be useful for example in the case of the model-free R_{ex} parameter to obtain the spectrometer dependent value from the omega-ex field strength independent internal value. Or to scale correlation times from seconds down to nanosecond or picosecond timescales.

Relaxation curve fitting parameters

Please see Table 17.33 on page 654.

Steady-state NOE parameters

Please see Table 17.9 on page 487.

Model-free parameters

Please see Table 17.34 on page 654.

For model-free theory it is assumed that R_{ex} values are scaled quadratically with field strength. The values will be very small as they will be written out as a field strength independent value. Hence use the following formula to convert the value to that expected for a given magnetic field strength:

```
Rex = value * (2.0 * pi * frequency) ** 2
```

The frequency is that of the proton in Hertz.

Consistency testing parameters

Please see Table 17.12 on page 488.

Relaxation dispersion parameters

Please see Table 17.5 on page 474.

Prompt examples

To show all CSA values, type:

```
relax> value.display('csa')
```

To display the model-free R_{ex} values scaled to 600 MHz, type one of:

```
relax> value.display('rex', scaling=(2.0*pi*600e6)**2)
```

```
relax> value.display(param='rex', scaling=(2.0*pi*600e6)**2)
```

Table 17.33: Relaxation curve fitting parameters.

Name	Description
rx	Either the R ₁ or R ₂ relaxation rate
i0	The initial intensity
iinf	The intensity at infinity

Table 17.34: Model-free parameters.

Name	Description
s2	S^2 , the model-free generalised order parameter ($S^2 = S_f^2 \cdot S_{2s}$)
s2f	S_f^2 , the faster motion model-free generalised order parameter
s2s	S_s^2 , the slower motion model-free generalised order parameter
local_tm	The spin specific global correlation time (seconds)
te	Single motion effective internal correlation time (seconds)
tf	Faster motion effective internal correlation time (seconds)
ts	Slower motion effective internal correlation time (seconds)
rex	Chemical exchange relaxation ($\sigma_{\text{ex}} = R_{\text{ex}} / \omega^{\star 2}$)
csa	Chemical shift anisotropy (unitless)

17.2.267 value.read Å**Synopsis**

Read spin specific parameter values from a file.

Defaults

```
value.read(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. Only one parameter may be selected.

scaling: The factor to scale parameters by.

file: The name of the file containing the values.

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.

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 molecule name, residue number, residue name, spin number and/or spin name columns can be supplied allowing this information to be in separate columns. Note that the numbering of columns starts at one. The spin ID string 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 this 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.

Relaxation curve fitting parameters

Please see Table 17.27 on page 650.

Model-free parameters

Please see Table 17.28 on page 650.

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 data pipe corresponds to the 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 obtain the correct value:

```
value = rex / (2.0 * pi * frequency) ** 2
```

where:

rex is the chemical exchange value for the current frequency.

frequency is the proton frequency corresponding to the data.

Reduced spectral density mapping parameters

Please see Table 17.29 on page 651.

In reduced spectral density mapping, the CSA value must be set prior to the calculation of spectral density values.

Consistency testing parameters

Please see Table 17.30 on page 651.

In consistency testing, the CSA value, angle Theta (‘orientation’) and global correlation time must be set prior to the calculation of consistency functions.

Relaxation dispersion parameters

Please see Table 17.32 on page 652.

Any of the relaxation dispersion parameters which are of the ‘float’ type can be set. Note that setting values for parameters which are not part of the model will have no effect.

Prompt 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')
```

17.2.268 value.set

\AA

Synopsis

Set parameter values.

Defaults

value.set(val=None, param=None, index=0, spin_id=None, error=False, force=True)

Keyword arguments

val: The value(s).

param: The parameter(s).

index: The list index for when the parameter is a list of values. This is ignored in all other cases.

spin_id: The spin ID string to restrict value setting to.

error: A flag which if True will cause the error rather than parameter to be set.

force: A flag which, if set to True, will cause the destination parameter to be overwritten.

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.

The value can be None, a single value, or an array of values while the parameter can be None, a string, or array of strings. The choice of which combination determines the behaviour of this function. The following table describes what occurs in each instance. 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.

Please see Table 17.35 on page 657.

Spin ID string

For spin-specific parameters, the spin ID string can be used to restrict the value setting to a specific spin system or group of spins. It has no effect for global parameters such as in the N-state model and frame order analyses.

Relaxation curve fitting parameters

Please see Table 17.36 on page 657.

Model-free parameters

Please see Table 17.37 on page 657.

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 data pipe corresponds to the 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 obtain the correct value:

$$\text{value} = \text{rex} / (2.0 * \pi * \text{frequency}) ^\star 2$$

where:

rex is the chemical exchange value for the current frequency.

frequency is the proton frequency corresponding to the data.

Reduced spectral density mapping parameters

Please see Table 17.38 on page 657.

In reduced spectral density mapping, the CSA value must be set prior to the calculation of spectral density values.

Consistency testing parameters

Please see Table 17.39 on page 658.

In consistency testing, the CSA value, angle Theta (‘orientation’) and global correlation time must be set prior to the calculation of consistency functions.

N-state model parameters

Please see Table 17.40 on page 658.

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 given as the index argument. So the Euler angle γ of the third state is specified using the parameter name ‘gamma’ and index of 2.

Table 17.35: The value and parameter combination options 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.

Table 17.36: Relaxation curve fitting parameter value setting.

Name	Description	Default
rx	Either the R ₁ or R ₂ relaxation rate	8.0
i0	The initial intensity	10000.0
iinf	The intensity at infinity	0.0

Table 17.37: Model-free parameter value setting.

Name	Description	Default
s2	S^2 , the model-free generalised order parameter ($S^2 = S_f^2.S_{2s}$)	0.8
s2f	S_f^2 , the faster motion model-free generalised order parameter	0.8
s2s	S_s^2 , the slower motion model-free generalised order parameter	0.8
local_tm	The spin specific global correlation time (seconds)	1e-08
te	Single motion effective internal correlation time (seconds)	1e-10
tf	Faster motion effective internal correlation time (seconds)	1e-11
ts	Slower motion effective internal correlation time (seconds)	1e-09
rex	Chemical exchange relaxation ($\sigma_{ex} = R_{ex} / \omega^{**2}$)	0.0
csa	Chemical shift anisotropy (unitless)	-0.000172

Table 17.38: Reduced spectral density mapping parameter value setting.

Name	Description	Default
j0	Spectral density value at 0 MHz - $J(0)$	None
jwx	Spectral density value at the frequency of the heteronucleus - $J(\omega_X)$	None
jwh	Spectral density value at the frequency of the proton - $J(\omega_H)$	None
csa	Chemical shift anisotropy (unitless)	-0.000172

Table 17.39: Consistency testing parameter value setting.

Name	Description	Default
j0	Spectral density value at 0 MHz (from Farrow et al. (1995) JBNMR, 6: 153-162)	None
f_eta	Eta-test (from Fushman et al. (1998) JACS, 120: 10947-10952)	None
f_r2	R2-test (from Fushman et al. (1998) JACS, 120: 10947-10952)	None
cса	Chemical shift anisotropy (unitless)	-0.000172
orientation	Angle between the 15N-1H vector and the principal axis of the 15N chemical shift tensor	15.7
tc	The single global correlation time estimate/approximation	1.3e-08

Table 17.40: N-state model parameter value setting.

Name	Description	Default	Type
Axx	The Axx component of the alignment tensor	None	float
Ayy	The Ayy component of the alignment tensor	None	float
Axy	The Axy component of the alignment tensor	None	float
Axz	The Axz component of the alignment tensor	None	float
Ayz	The Ayz component of the alignment tensor	None	float
probs	The probabilities of each state	0.0	list
alpha	The α Euler angles (for the rotation of each state)	0.0	list
beta	The β Euler angles (for the rotation of each state)	0.0	list
gamma	The γ Euler angles (for the rotation of each state)	0.0	list
paramagnetic_centre	The paramagnetic centre	None	list

Relaxation dispersion parameters

Please see Table 17.41 on page 659.

Any of the relaxation dispersion parameters which are of the ‘float’ type can be set. Note that setting values for parameters which are not part of the model will have no effect.

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')
```

Frame order parameters

Please see Table 17.6 on page 474.

To set the NH bond length of all spins to 1.02 Å, type:

```
relax> value.set(1.02 * 1e-10, 'r')
```

```
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=['r', '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'])
```

Prompt 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')
```

Table 17.41: Relaxation dispersion parameter value setting.

Name	Description	Default	Type
r2eff	The effective transversal relaxation rate	10.0	dict
i0	The initial intensity	10000.0	dict
r1	The longitudinal relaxation rate	2.0	dict
r2	The transversal relaxation rate	10.0	dict
r2a	The transversal relaxation rate for state A in the absence of exchange	10.0	dict
r2b	The transversal relaxation rate for state B in the absence of exchange	10.0	dict
pA	The population for state A	0.9	float
pB	The population for state B	0.5	float
pC	The population for state C	0.5	float
phi_ex	The $\phi_{ex} = pA.pB.dw^{**2}$ value (ppm^2)	5.0	float
phi_ex_B	The fast exchange factor between sites A and B (ppm^2)	5.0	float
phi_ex_C	The fast exchange factor between sites A and C (ppm^2)	5.0	float
padw2	The $pA.dw^{**2}$ value (ppm^2)	1.0	float
dw	The chemical shift difference between states A and B (in ppm)	1.0	float
dw_AB	The chemical shift difference between states A and B for 3-site exchange (in ppm)	1.0	float
dw_AC	The chemical shift difference between states A and C for 3-site exchange (in ppm)	1.0	float
dw_BC	The chemical shift difference between states B and C for 3-site exchange (in ppm)	1.0	float
dwH	The proton chemical shift difference between states A and B (in ppm)	1.0	float
dwH_AB	The proton chemical shift difference between states A and B for 3-site exchange (in ppm)	1.0	float
dwH_AC	The proton chemical shift difference between states A and C for 3-site exchange (in ppm)	1.0	float
dwH_BC	The proton chemical shift difference between states B and C for 3-site exchange (in ppm)	1.0	float
kex	The exchange rate	1000.0	float
kex_AB	The exchange rate between sites A and B for 3-site exchange with $kex_AB = k_AB + k_BA$ (rad.s^-1)	1000.0	float
kex_AC	The exchange rate between sites A and C for 3-site exchange with $kex_AC = k_AC + k_CA$ (rad.s^-1)	1000.0	float
kex_BC	The exchange rate between sites B and C for 3-site exchange with $kex_BC = k_BC + k_CB$ (rad.s^-1)	1000.0	float
kB	Approximate chemical exchange rate constant between sites A and B (rad.s^-1)	1000.0	float
kC	Approximate chemical exchange rate constant between sites A and C (rad.s^-1)	1000.0	float
tex	The time of exchange (tex = 1/kex)	0.001	float
k_AB	The exchange rate from state A to state B	2.0	float
k_BA	The exchange rate from state B to state A	1000.0	float

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')
```

17.2.269 value.write



Synopsis

Write spin specific parameter values to a file.

Defaults

```
value.write(param=None, file=None, dir=None, scaling=1.0, comment=None, bc=False, force=False)
```

Keyword arguments

param: The parameter.

file: The name of the file.

dir: The directory name.

scaling: The factor to scale parameters by.

comment: Text which will be added to the start of the file as comments. All lines will be prefixed by '# '.

bc: A flag which if True will cause the back calculated values to be written to file rather than the actual data.

force: A flag which, if set to True, will cause the file to be overwritten.

Description

The values corresponding to the given parameter will be written to file. The scaling argument can be used to scale the parameter values. This can be useful for example in the case of the model-free R_{ex} parameter to obtain the spectrometer dependent value from the omega_ex field strength independent internal value. Or to scale correlation times from seconds down to nanosecond or picosecond timescales.

Relaxation curve fitting parameters

Please see Table 17.33 on page 654.

Steady-state NOE parameters

Please see Table 17.9 on page 487.

Model-free parameters

Please see Table 17.34 on page 654.

For model-free theory it is assumed that R_{ex} values are scaled quadratically with field strength. The values will be very small as they will be written out as a field strength independent value. Hence use the following formula to convert the value to that expected for a given magnetic field strength:

```
Rex = value * (2.0 * pi * frequency) ** 2
```

View the structures loaded into the relax data store using VMD.

The frequency is that of the proton in Hertz.

17.2.270 vmd.view

Synopsis

View the structures loaded into the relax data store using VMD.

Defaults

Reduced spectral density mapping parameters

vmd.view()

Please see Table 17.11 on page 487.

Description

This will launch VMD with all of the structures loaded into the relax data store.

Consistency testing parameters

Prompt examples

Relaxation dispersion parameters

relax> vmd.view()

Please see Table 17.5 on page 474.

Prompt 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)
```

To write the model-free R_{ex} values scaled to 600 MHz to the file ‘rex_600’, type one of:

```
relax> value.write('rex', 'rex_600', scaling
    =(2.0*pi*600e6)**2)

relax> value.write(param='rex', file='
    rex_600', scaling=(2.0*pi*600e6)**2)
```


Chapter 18

Licence

18.1 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.

18.2 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 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. 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 them 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 prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. 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.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially

in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular

programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty

adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- (c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- (d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- (e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified ver-

sions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- (a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- (b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- (c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

- (d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- (e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- (f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate

or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or

other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS 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.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

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 state 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) <textyear> <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 3 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, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
```

```
This program 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, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU 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 Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Bibliography

- Abragam, A. (1961). *The Principles of Nuclear Magnetism*. Clarendon Press, Oxford.
- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In: Petrov, B. N. and Csaki, F. (eds.): *Proceedings of the Second International Symposium on Information Theory*. Budapest, pages 267–281, Akademia Kiado.
- Baldwin, A. J. (2014). An exact solution for R_{2,eff} in CPMG experiments in the case of two site chemical exchange. *J. Magn. Reson.*, **244**, 114–124. ([10.1016/j.jmr.2014.02.023](https://doi.org/10.1016/j.jmr.2014.02.023)).
- Baldwin, A. J. and Kay, L. E. (2013). An R1rho expression for a spin in chemical exchange between two sites with unequal transverse relaxation rates. *J. Biomol. NMR*, **55**(2), 211–218. ([10.1007/s10858-012-9694-6](https://doi.org/10.1007/s10858-012-9694-6)).
- Barbato, G., Ikura, M., Kay, L. E., Pastor, R. W., and Bax, A. (1992). Backbone dynamics of calmodulin studied by ¹⁵N relaxation using inverse detected two-dimensional NMR spectroscopy: the central helix is flexible. *Biochemistry*, **31**(23), 5269–5278. ([10.1021/bi00138a005](https://doi.org/10.1021/bi00138a005)).
- Bieri, M., d'Auvergne, E., and Gooley, P. (2011). relaxGUI: a new software for fast and simple NMR relaxation data analysis and calculation of ps-ns and μ s motion of proteins. *J. Biomol. NMR*, **50**, 147–155. ([10.1007/s10858-011-9509-1](https://doi.org/10.1007/s10858-011-9509-1)).
- Bieri, M. and Gooley, P. R. (2011). Automated NMR relaxation dispersion data analysis using NESSY. *BMC Bioinformatics*, **12**, 421. ([10.1186/1471-2105-12-421](https://doi.org/10.1186/1471-2105-12-421)).
- Bloch, F. (1946). Nuclear induction. *Phys. Rev.*, **70**(7-8), 460–474. ([10.1103/PhysRev.70.460](https://doi.org/10.1103/PhysRev.70.460)).
- Bloembergen, N., Purcell, E. M., and Pound, R. V. (1948). Relaxation effects in nuclear magnetic resonance absorption. *Phys. Rev.*, **73**(7), 679–712. ([10.1103/PhysRev.73.679](https://doi.org/10.1103/PhysRev.73.679)).
- Bonev, I. A. and Gosselin, C. M. (2006). Analytical determination of the workspace of symmetrical spherical parallel mechanisms. *Comput. Appl. Mech. Eng.*, **22**(5), 1011–1017. ([10.1109/TRO.2006.878983](https://doi.org/10.1109/TRO.2006.878983)).
- Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. General considerations. *J. Inst. Maths. Applies.*, **6**(1), 76–90. ([10.1093/imamat/6.1.76](https://doi.org/10.1093/imamat/6.1.76)).
- Brüschweiler, R., Liao, X., and Wright, P. E. (1995). Long-range motional restrictions in a multidomain zinc-finger protein from anisotropic tumbling. *Science*, **268**(5212), 886–889. ([10.1126/science.7754375](https://doi.org/10.1126/science.7754375)).

- Butterwick, J. A., Loria, P. J., Astrof, N. S., Kroenke, C. D., Cole, R., Rance, M., and Palmer, 3rd, A. G. (2004). Multiple time scale backbone dynamics of homologous thermophilic and mesophilic ribonuclease HI enzymes. *J. Mol. Biol.*, **339**(4), 855–871. ([10.1016/j.jmb.2004.03.055](https://doi.org/10.1016/j.jmb.2004.03.055)).
- Carver, J. P. and Richards, R. E. (1972). General 2-site solution for chemical exchange produced dependence of T2 upon Carr-Purcell pulse separation. *J. Magn. Reson.*, **6**(1), 89–105. ([10.1016/0022-2364\(72\)90090-X](https://doi.org/10.1016/0022-2364(72)90090-X)).
- 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. ([10.1023/b:jnmr.0000032504.70912.58](https://doi.org/10.1023/b:jnmr.0000032504.70912.58)).
- 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. ([10.1021/ja00168a070](https://doi.org/10.1021/ja00168a070)).
- Crawford, N. R., Yamaguchi, G. T., and Dickman, C. A. (1999). A new technique for determining 3-D joint angles: the tilt/twist method. *Clin. Biomech. (Bristol, Avon)*, **14**(3), 153–165. ([10.1016/S0268-0033\(98\)00080-1](https://doi.org/10.1016/S0268-0033(98)00080-1)).
- 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/>. ([10.187/2281](https://doi.org/10.187/2281)).
- 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. ([10.1023/a:1021902006114](https://doi.org/10.1023/a:1021902006114)).
- 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. ([10.1007/s10858-006-9007-z](https://doi.org/10.1007/s10858-006-9007-z)).
- d'Auvergne, E. J. and Gooley, P. R. (2007). Set theory formulation of the model-free problem and the diffusion seeded model-free paradigm. *Mol. BioSyst.*, **3**(7), 483–494. ([10.1039/b702202f](https://doi.org/10.1039/b702202f)).
- d'Auvergne, E. J. and Gooley, P. R. (2008a). Optimisation of NMR dynamic models. *J. Biomol. NMR*, **40**(2), 107–133.
- d'Auvergne, E. J. and Gooley, P. R. (2008b). 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. ([10.1007/s10858-007-9214-2](https://doi.org/10.1007/s10858-007-9214-2)).
- d'Auvergne, E. J. and Gooley, P. R. (2008c). 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. ([10.1007/s10858-007-9213-3](https://doi.org/10.1007/s10858-007-9213-3)).
- Davis, D. G., Perlman, M. E., and London, R. E. (1994). Direct measurements of the dissociation-rate constant for inhibitor-enzyme complexes via the T1rho and T2 (CPMG) methods. *J. Magn. Reson.*, **104**(3), 266–275. ([10.1006/jmrb.1994.1084](https://doi.org/10.1006/jmrb.1994.1084)).

- Einstein, A. (1905). Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen (The motion of elements suspended in static liquids as claimed in the molecular kinetic theory of heat). *Ann. Physik*, **17**(8), 549–560. ([10.1002/andp.19053220806](#)).
- Erdelyi, M., d'Auvergne, E., Navarro-Vazquez, A., Leonov, A., and Griesinger, C. (2011). Dynamics of the glycosidic bond: Conformational space of lactose. *Chem. Eur. J.*, **17**(34), 9368–9376. ([10.1002/chem.201100854](#)).
- 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. ([10.1007/bf00211779](#)).
- Favro, L. D. (1960). Theory of the rotational brownian motion of a free rigid body. *Phys. Rev.*, **119**(1), 53–62. ([10.1103/PhysRev.119.53](#)).
- Fletcher, R. (1970). A new approach to variable metric algorithms. *Comp. J.*, **13**(3), 317–322. ([10.1093/comjnl/13.3.317](#)).
- Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients. *Comp. J.*, **7**(2), 149–154. ([10.1093/comjnl/7.2.149](#)).
- 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. ([10.1006/jmbi.1996.0771](#)).
- Fushman, D., Tjandra, N., and Cowburn, D. (1998). Direct measurement of 15N chemical shift anisotropy in solution. *J. Am. Chem. Soc.*, **120**(42), 10947–10952. ([10.1021/ja981686m](#)).
- Fushman, D., Tjandra, N., and Cowburn, D. (1999). An approach to direct determination of protein dynamics from 15N NMR relaxation at multiple fields, independent of variable 15N chemical shift anisotropy and chemical exchange contributions. *J. Am. Chem. Soc.*, **121**(37), 8577–8582. ([10.1021/ja9904991](#)).
- 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. ([10.1090/s0025-5718-1970-0258249-6](#)).
- Hansen, D. F., Vallurupalli, P., Lundstrom, P., Neudecker, P., and Kay, L. E. (2008). Probing chemical shifts of invisible states of proteins with relaxation dispersion NMR spectroscopy: how well can we do? *J. Am. Chem. Soc.*, **130**(8), 2667–2675. ([10.1021/ja078337p](#)).
- Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Res. Natn. Bur. Stand.*, **49**(6), 409–436. ([10.6028/jres.049.044](#)).
- Horne, J., d'Auvergne, E. J., Coles, M., Velkov, T., Chin, Y., Charman, W. N., Prankerd, R., Gooley, P. R., and Scanlon, M. J. (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. ([10.1016/j.jmb.2007.05.067](#)).

- Huang, T., Wang, J., and Whitehouse, D. J. (1999). Closed form solution to workspace of hexapod-based virtual axis machine tools. *J. Mech. Des.*, **121**(1), 26–31. ([10.1115/1.2829424](https://doi.org/10.1115/1.2829424)).
- Hurvich, C. M. and Tsai, C. L. (1989). Regression and time-series model selection in small samples. *Biometrika*, **76**(2), 297–307. ([10.1093/biomet/76.2.297](https://doi.org/10.1093/biomet/76.2.297)).
- Ishima, R. and Torchia, D. A. (1999). Estimating the time scale of chemical exchange of proteins from measurements of transverse relaxation rates in solution. *J. Biomol. NMR*, **14**(4), 369–372. ([10.1023/A:1008324025406](https://doi.org/10.1023/A:1008324025406)).
- Ishima, R. and Torchia, D. A. (2005). Error estimation and global fitting in transverse-relaxation dispersion experiments to determine chemical-exchange parameters. *J. Biomol. NMR*, **32**(1), 41–54. ([10.1007/s10858-005-3593-z](https://doi.org/10.1007/s10858-005-3593-z)).
- Kay, L. E., Torchia, D. A., and Bax, A. (1989). Backbone dynamics of proteins as studied by ¹⁵N inverse detected heteronuclear NMR spectroscopy: application to staphylococcal nuclelease. *Biochemistry*, **28**(23), 8972–8979. ([10.1021/bi00449a003](https://doi.org/10.1021/bi00449a003)).
- Kleckner, I. R. and Foster, M. P. (2012). GUARDD: user-friendly MATLAB software for rigorous analysis of CPMG RD NMR data. *J. Biomol. NMR*, **52**(1), 11–22. ([10.1007/s10858-011-9589-y](https://doi.org/10.1007/s10858-011-9589-y)).
- Korein, J. U. (1985). *A geometric investigation of reach*. MIT Press, Cambridge, MA, USA.
- 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. ([10.1016/s0079-6565\(00\)00028-5](https://doi.org/10.1016/s0079-6565(00)00028-5)).
- Korzhnev, D. M., Kloiber, K., Kanelis, V., Tugarinov, V., and Kay, L. E. (2004a). Probing slow dynamics in high molecular weight proteins by methyl-TROSY NMR spectroscopy: application to a 723-residue enzyme. *J. Am. Chem. Soc.*, **126**(12), 3964–3973. ([10.1021/ja039587i](https://doi.org/10.1021/ja039587i)).
- Korzhnev, D. M., Kloiber, K., and Kay, L. E. (2004b). Multiple-quantum relaxation dispersion NMR spectroscopy probing millisecond time-scale dynamics in proteins: theory and application. *J. Am. Chem. Soc.*, **126**(23), 7320–7329. ([10.1021/ja049968b](https://doi.org/10.1021/ja049968b)).
- Korzhnev, D. M., Neudecker, P., Mittermaier, A., Orekhov, V. Y., and Kay, L. E. (2005a). Multiple-site exchange in proteins studied with a suite of six NMR relaxation dispersion experiments: an application to the folding of a Fyn SH3 domain mutant. *J. Am. Chem. Soc.*, **127**(44), 15602–15611. ([10.1021/ja054550e](https://doi.org/10.1021/ja054550e)).
- Korzhnev, D. M., Orekhov, V. Y., and Kay, L. E. (2005b). Off-resonance R(1rho) NMR studies of exchange dynamics in proteins with low spin-lock fields: an application to a Fyn SH3 domain. *J. Am. Chem. Soc.*, **127**(2), 713–721. ([10.1021/ja0446855](https://doi.org/10.1021/ja0446855)).
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Stat.*, **22**(1), 79–86. ([10.1214/aoms/1177729694](https://doi.org/10.1214/aoms/1177729694)).
- Lee, L. K., Rance, M., Chazin, W. J., and Palmer, A. G. (1997). Rotational diffusion anisotropy of proteins from simultaneous analysis of N-15 and C-13(alpha) nuclear spin relaxation. *J. Biomol. NMR*, **9**(3), 287–298. ([10.1023/a:1018631009583](https://doi.org/10.1023/a:1018631009583)).

- Lefevre, J. F., Dayie, K. T., Peng, J. W., 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. ([10.1021/bi9526802](#)).
- Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, **2**, 164–168.
- Linhart, H. and Zucchini, W. (1986). *Model Selection*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., New York, NY, USA.
- 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. ([10.1021/ja00381a009](#)).
- 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. ([10.1021/ja00381a010](#)).
- Luz, Z. and Meiboom, S. (1963). Nuclear magnetic resonance study of protolysis of trimethylammonium ion in aqueous solution - order of reaction with respect to solvent. *J. Chem. Phys.*, **39**(2), 366–370. ([10.1063/1.1734254](#)).
- 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. ([10.1006/jmbi.1994.0073](#)).
- Marquardt, D. W. (1963). An algorithm for least squares estimation of non-linear parameters. *SIAM J.*, **11**, 431–441. ([10.1137/0111030](#)).
- Mazur, A., Hammesfahr, B., Griesinger, C., Lee, D., and Kollmar, M. (2013). ShereKhan—calculating exchange parameters in relaxation dispersion data from CPMG experiments. *Bioinformatics*, **29**(14), 1819–1820. ([10.1093/bioinformatics/btt286](#)).
- McConnell, H. M. (1958). Reaction rates by nuclear magnetic resonance. *J. Chem. Phys.*, **28**(3), 430–431. ([10.1063/1.1744152](#)).
- Meiboom, S. (1961). Nuclear magnetic resonance study of proton transfer in water. *J. Chem. Phys.*, **34**(2), 375–388. ([10.1063/1.1700960](#)).
- Millet, O., Loria, J. P., Kroenke, C. D., Pons, M., and Palmer, A. G. (2000). The static magnetic field dependence of chemical exchange linebroadening defines the NMR chemical shift time scale. *J. Am. Chem. Soc.*, **122**(12), 2867–2877.
- Miloushev, V. Z. and Palmer, 3rd, A. G. (2005). R(1rho) relaxation for two-site chemical exchange: general approximations and some exact solutions. *J. Magn. Reson.*, **177**(2), 221–227. ([10.1016/j.jmr.2005.07.023](#)).
- Moré, J. J. and Thuente, D. J. (1994). Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Maths. Softw.*, **20**(3), 286–307. ([10.1145/192115.192132](#)).
- Morin, S. (2011). A practical guide to protein dynamics from ^{15}N spin relaxation in solution. *Prog. NMR Spectrosc.*, **59**(3), 245–262. ([10.1016/j.pnmrs.2010.12.003](#)).
- Morin, S. and Gagné, S. (2009a). Simple tests for the validation of multiple field spin relaxation data. *J. Biomol. NMR*, **45**, 361–372. ([10.1007/s10858-009-9381-4](#)).

- Morin, S. and Gagné, S. M. (2009b). NMR dynamics of PSE-4 β -lactamase: An interplay of ps-ns order and μ s-ms motions in the active site. *Biophys. J.*, **96**(11), 4681–4691. ([10.1016/j.bpj.2009.02.068](https://doi.org/10.1016/j.bpj.2009.02.068)).
- Morin, S., Linnet, T. E., Lescanne, M., Schanda, P., Thompson, G. S., Tollinger, M., Teilmann, K., Gagne, S., Marion, D., Griesinger, C., Blackledge, M., and d'Auvergne, E. J. (2014). relax: the analysis of biomolecular kinetics and thermodynamics using NMR relaxation dispersion data. *Bioinformatics*, **30**(15), 2219–2220. ([10.1093/bioinformatics/btu166](https://doi.org/10.1093/bioinformatics/btu166)).
- Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York.
- O'Connell, N. E., Grey, M. J., Tang, Y., Kosuri, P., Miloushev, V. Z., Raleigh, D. P., and Palmer, 3rd, A. G. (2009). Partially folded equilibrium intermediate of the villin headpiece HP67 defined by ^{13}C relaxation dispersion. *J. Biomol. NMR*, **45**(1-2), 85–98. ([10.1007/s10858-009-9340-0](https://doi.org/10.1007/s10858-009-9340-0)).
- Orekhov, V. Y., Korzhnev, D. M., Dierckx, T., Kessler, H., and Arseniev, A. S. (1999a). 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. ([10.1023/a:1008356809071](https://doi.org/10.1023/a:1008356809071)).
- Orekhov, V. Y., Korzhnev, D. M., Pervushin, K. V., Hoffmann, E., and Arseniev, A. S. (1999b). Sampling of protein dynamics in nanosecond time scale by ^{15}N NMR relaxation and self-diffusion measurements. *J. Biomol. Struct. Dyn.*, **17**(1), 157–174. ([10.1080/07391102.1999.10508348](https://doi.org/10.1080/07391102.1999.10508348)).
- Orekhov, V. Y., Pervushin, K. V., Korzhnev, D. M., and Arseniev, A. S. (1995). Backbone dynamics of (1-71)bacterioopsin and (1-36)bacterioopsin studied by 2-dimensional H-1-N-15 NMR-spectroscopy. *J. Biomol. NMR*, **6**(2), 113–122. ([10.1007/BF00211774](https://doi.org/10.1007/BF00211774)).
- Palmer, 3rd, A. G. and Massi, F. (2006). Characterization of the dynamics of biomacromolecules using rotating-frame spin relaxation NMR spectroscopy. *Chem. Rev.*, **106**(5), 1700–1719. ([10.1021/cr0404287](https://doi.org/10.1021/cr0404287)).
- Perrin, F. (1934). Mouvement Brownien d'un ellipsoïde (I). Dispersion diélectrique pour des molécules ellipsoïdales. *J. Phys. Radium*, **5**, 497–511. ([10.1051/jphysrad:01934005010049700](https://doi.org/10.1051/jphysrad:01934005010049700)).
- Perrin, F. (1936). Mouvement Brownien d'un ellipsoïde (II). Rotation libre et dépolarisation des fluorescences. Translation et diffusion de molécules ellipsoïdales. *J. Phys. Radium*, **7**, 1–11. ([10.1051/jphysrad:01936007010100](https://doi.org/10.1051/jphysrad:01936007010100)).
- Polak, E. and Ribièvre, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, **16**, 35–43.
- Schurr, J. M., Babcock, H. P., and Fujimoto, B. S. (1994). A test of the model-free formulas. Effects of anisotropic rotational diffusion and dimerization. *J. Magn. Reson. B*, **105**(3), 211–224. ([10.1006/jmrb.1994.1127](https://doi.org/10.1006/jmrb.1994.1127)).
- Schwarz, G. (1978). Estimating dimension of a model. *Ann. Stat.*, **6**(2), 461–464. ([10.1214/aos/1176344136](https://doi.org/10.1214/aos/1176344136)).

- Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, **24**(111), 647–656. ([10.1090/s0025-5718-1970-0274029-x](https://doi.org/10.1090/s0025-5718-1970-0274029-x)).
- Sobol', I. (1967). Point distribution in a cube and approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, **7**, 784–802. ([10.1016/0041-5553\(67\)90144-9](https://doi.org/10.1016/0041-5553(67)90144-9)).
- Spencer, A. J. M. (1980). *Continuum Mechanics*. Longman Group UK Limited, Essex, England.
- Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, **20**(3), 626–637. ([10.1137/0720042](https://doi.org/10.1137/0720042)).
- Sugase, K., Konuma, T., Lansing, J. C., and Wright, P. E. (2013). Fast and accurate fitting of relaxation dispersion data using the flexible software package GLOVE. *J. Biomol. NMR*, **56**(3), 275–283. ([10.1007/s10858-013-9747-5](https://doi.org/10.1007/s10858-013-9747-5)).
- Sun, H., d'Auvergne, E. J., Reinscheid, U. M., Dias, L. C., Andrade, C. K. Z., Rocha, R. O., and Griesinger, C. (2011). Bijvoet in solution reveals unexpected stereoselectivity in a michael addition. *Chem. Eur. J.*, **17**(6), 1811–1817. ([10.1002/chem.201002520](https://doi.org/10.1002/chem.201002520)).
- Tjandra, N., Wingfield, P., Stahl, S., and Bax, A. (1996). Anisotropic rotational diffusion of perdeuterated HIV protease from N-15 NMR relaxation measurements at two magnetic. *J. Biomol. NMR*, **8**(3), 273–284. ([10.1007/bf00410326](https://doi.org/10.1007/bf00410326)).
- Tollinger, M., Skrynnikov, N. R., Mulder, F. A. A., Forman-Kay, J. D., and Kay, L. E. (2001). Slow dynamics in folded and unfolded states of an sh3 domain. *J. Am. Chem. Soc.*, **123**(46), 11341–11352. ([10.1021/ja011300z](https://doi.org/10.1021/ja011300z)).
- Trott, O., Abergel, D., and Palmer, A. (2003). An average-magnetization analysis of R-1 rho relaxation outside of the fast exchange. *Mol. Phys.*, **101**(6), 753–763. ([10.1080/0026897021000054826](https://doi.org/10.1080/0026897021000054826)).
- Trott, O. and Palmer, 3rd, A. G. (2002). R1rho relaxation outside of the fast-exchange limit. *J. Magn. Reson.*, **154**(1), 157–160. ([10.1006/jmre.2001.2466](https://doi.org/10.1006/jmre.2001.2466)).
- Trott, O. and Palmer, 3rd, A. G. (2004). Theoretical study of R(1rho) rotating-frame and R2 free-precession relaxation in the presence of n-site chemical exchange. *J. Magn. Reson.*, **170**(1), 104–112. ([10.1016/j.jmr.2004.06.005](https://doi.org/10.1016/j.jmr.2004.06.005)).
- Viles, J., Duggan, B., Zaborowski, E., Schwarzsinger, S., Huntley, J., Kroon, G., Dyson, H. J., and Wright, P. (2001). Potential bias in NMR relaxation data introduced by peak intensity analysis and curve fitting methods. *J. Biomol. NMR*, **21**, 1–9. ([10.1023/A:1011966718826](https://doi.org/10.1023/A:1011966718826)).
- Woessner, D. E. (1962). Nuclear spin relaxation in ellipsoids undergoing rotational brownian motion. *J. Chem. Phys.*, **37**(3), 647–654. ([10.1063/1.1701390](https://doi.org/10.1063/1.1701390)).
- 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. ([10.1016/j.jmb.2004.08.001](https://doi.org/10.1016/j.jmb.2004.08.001)).
- Zucchini, W. (2000). An introduction to model selection. *J. Math. Psychol.*, **44**(1), 41–61. ([10.1006/jmps.1999.1276](https://doi.org/10.1006/jmps.1999.1276)).

Index

- SourceForge, 285
- AIC, *see* model selection, AIC
- AICc, *see* model selection, AICc
- angles, 447–450, 452, 452, 467–469, 477–480, 482–484, 538, 540, 549, 581, 582, 626, 627, 629, 630, 633, 650, 655, 656
- anisotropic diffusion, *see* diffusion, anisotropic
- ANOVA model selection, *see* model selection, ANOVA
- API documentation, 281, 291
- argument, 8
- keyword, 8
- asymmetric diffusion, *see* diffusion, ellipsoid (asymmetric)
- Augmented Lagrangian, *see* optimisation, constraint algorithm, Method of Multipliers
- axially symmetric diffusion, *see* diffusion, spheroid (axially symmetric)
- Bayesian model selection, *see* model selection, Bayesian
- BFGS, *see* optimisation, algorithm, BFGS
- BIC, *see* model selection, BIC
- bond length, 502, 505, 506, 658
- Bootstrap model selection, *see* model selection, Bootstrap
- branches, 287
- Brownian diffusion, *see* diffusion, Brownian
- bug, 31
- design, 31
- report, 198
- search, 32
- tracker, 23, 24, 31, 32, 286, 291, 296
- bzip2, *see* compression, bzip2
- C module compilation, 24, 290, *see* SCons, C module compilation
- camel case, 281
- Cauchy point, *see* optimisation, algorithm, Cauchy point
- CG-Steihaug, *see* optimisation, algorithm, CG-Steihaug
- chemical exchange, 502, 586, 650, 655, 656
- chi-squared, 85, 90, 91, 105, 190, 303, 304, 305, 310, 313, 496, 550, 569, 583, 623, 624
- chi-squared gradient, 313
- chi-squared Hessian, 313
- clean up, 291
- commit access, 284
- commit log, 285
- compression, 478, 479, 483, 597, 622, 646
- bzip2, 596, 597, 622, 623, 637, 639, 646
- gzip, 596, 597, 622, 637, 639, 646
- uncompressed, 596, 622, 639
- consistency testing, 133
- constraint, 468, 475, 480, 497, 498, 500, 541
- copy, 445, 466, 489, 493, 506–508, 545, 552, 565, 570, 581, 592, 604, 615, 649, 650
- correlation time, 467–470, 475, 502, 629, 650, 653, 655, 656, 660
- cross rate, *see* relaxation rate, cross rate
- cross-relaxation, *see* relaxation rate, cross-relaxation
- cross-validation model selection, *see* model selection, cross-validation
- ctypes, 24

Dasha, *see* software, Dasha
 data pipe, **10**
 delete, **446**, **466**, **493**, **503**, **509**, **546**, **554**,
566, **571**, **593**, **608**, **617**, **631**, **632**
 diff, **32**
 diffusion, **87**
 anisotropic, **468**, **469**
 Brownian, **87**, **483**, **528**, **558**, **562**,
 629
 ellipsoid (asymmetric), **88**, **94**, **102**,
 350, **452**, **468**, **470**, **492**, **629**
 sphere (isotropic), **89**, **102**, **367**,
 452, **467**, **468**, **470**, **471**, **479–482**,
 537, **538**, **629**
 spheroid (axially symmetric), **89**,
 93, **94**, **102**, **363**, **452**, **467**, **468**,
 470, **492**, **629**
 tensor, **452**, **466–470**, **476**, **492**, **528**,
 555, **562**, **629**, **631**
 direction cosine, **350**, **363**
 discrepancy, **105**
 Kullback-Leibler, **105**
 display, **446**, **454**, **467**, **471**, **494**, **509**,
 528, **547**, **557**, **558**, **562**, **563**,
 567, **571**, **593**, **596**, **598**, **604**,
 612, **613**, **617**, **648**, **649**, **653**
 distribution archive, **24**, **32**, **277**, **291**
 doc string, **280**
 dogleg, *see* optimisation, algorithm,
 dogleg
 eigenvalues, **447**, **467–469**, **629**, **636**
 ellipsoidal diffusion, *see* diffusion,
 ellipsoid (asymmetric)
 Ensemble analysis, **141**
 epydoc, **281**
 Euler angles, **102**, **350**, **467–469**, **482**,
 540, **650**, **656**
 exact trust region, *see* optimisation,
 algorithm, exact trust region
 exponential curve fitting, **4**
 Fletcher-Reeves, *see* optimisation,
 algorithm, Fletcher-Reeves
 floating point number, **7**, **447**, **468**, **469**,
 472, **536**
 forks, **289**
 Frame order
 axis permutations, **254**
 linear constraints, **257**
 matrix, **385**, **387**, **391**, **399**, **403**, **407**,
 412, **420**, **427**, **439**
 model
 double rotor, **433**
 free rotor, **391**
 isotropic cone, **394**
 isotropic cone, free rotor, **405**
 isotropic cone, torsionless, **400**
 pseudo-ellipse, **408**
 pseudo-ellipse, free rotor, **427**
 pseudo-ellipse, torsionless, **419**
 rigid, **385**
 rotor, **386**
 model nesting, **260**
 simulation, **375**
 frame order, **237**
 frame order analysis
 rigid model, **264**
 Frequentist model selection, *see* model
 selection, Frequentist
 function class, **9**, **10**
 git, **32**, **277**, **284**
 check out, **291**
 checkout, **32**
 clone, **32**
 conflict, **288**
 manual, **32**
 merge, **290**
 GNU/Linux, *see* Operating system,
 GNU/ Linux
 Google, **30**
 GPG
 key, **32**
 signature, **32**, **33**
 GPL, **4**, **663**
 Grace, *see* software, Grace
 gradient, **304**
 GUI, *see* User interface, GUI
 gzip, *see* compression, gzip
 help system, **9**, **9**, **443**
 Hessian, **304**
 Hestenes-Stiefel, *see* optimisation,
 algorithm, Hestenes-Stiefel
 Hypothesis testing model selection, *see*
 model selection, Hypothesis
 testing
 indentation, **280**

- installation, 23
integer, 7, 293
interatomic data container, 11
isotropic diffusion, *see* diffusion, sphere (isotropic)
- keyword argument, *see* argument, keyword
Kullback-Leibler discrepancy, *see* discrepancy, Kullback-Leibler
- Levenberg-Marquardt, *see* optimisation, algorithm, Levenberg-Marquardt
licence, 663
linking, 297
Linux, *see* Operating system, GNU/Linux
list, 7, 293
Logarithmic barrier, *see* optimisation, constraint algorithm, Logarithmic barrier
longitudinal relaxation, *see* relaxation rate, spin-lattice
- Mac OS X, *see* Operating system, Mac OS X
mailing list, 30, 30, 277, 286, 295
archive, 30, 31
relax-announce, 30, 277
relax-commits, 30, 31, 277
relax-devel, 30–32, 56, 59, 74, 107, 197, 277, 286, 287, 291
relax-users, xxx, 30–32, 146, 198, 277
make, 290
manual
 HTML, 30
map, 448–450, 455, 460, 471, 472, 486, 489, 496, 513, 525, 526, 530–532, 534–536, 551, 552, 559, 561, 583, 587, 650, 655, 656
Method of Multipliers, *see* optimisation, constraint algorithm, Method of Multipliers
minfx, *see* optimisation, minfx
model elimination, 5, 455, 475, 530–532, 534–536
model selection, 5
- AIC, 5, 105, 158, 205, 455, 506, 507, 623
AICc, 5, 205, 506
ANOVA, 5
Bayesian, 158
BIC, 5, 205, 506
bootstrap, 5, 455, 506, 529
cross-validation, 5, 455, 506
frequentist, 205
hypothesis testing, 5
model-free analysis, 85
Modelfree, *see* software, Modelfree
modelling, 475
molecular dynamics simulation, 143
molecule, 447, 459, 463, 464, 467, 468, 479, 507, 508, 508, 509, 509, 510, 510, 511, 511, 529, 547, 548, 555, 564, 572, 573, 584, 592–595, 600, 601, 604–606, 611–613, 615–621, 624–627, 629, 631–647, 654
MOLMOL, *see* software, MOLMOL
Monte Carlo simulation, 5, 51, 61, 204
Monte Carlo simulations, 264
MPI, 19
 mpi4py, 19, 23
 OpenMPI, 19, 210, 211, 263
MS Windows, *see* Operating system, MS Windows
multi-processor framework, 17
N-state model, 141
Nelder-Mead simplex, *see* optimisation, algorithm, Nelder-Mead simplex
Newton, *see* optimisation, algorithm, Newton
Newton-CG, *see* optimisation, algorithm, Newton-CG
Newton-Raphson, *see* optimisation, algorithm, Newton
NMR, 453–455, 457, 458, 460, 479, 490, 491, 496, 549, 573, 611–613
NMRView, *see* software, NMRView
NOE, 4, 71, 141, 143
NumPy, 23
- OpenDX, *see* software, OpenDX
OpenMPI, *see* MPI, OpenMPI
Operating system
 GNU/Linux, 24, 290

- Mac OS X, **25**, 290
 MS Windows, **24**, 290
 Unix, **290**
- optimisation, **8**, **85**, **301**, 460, 468, 475, **496**, **497**, 497, 498, **500**, 500, **501**, 507, 529–536, 583, 633, 649, 655, 656
- algorithm, **5**, **305**
 BFGS, **304**, **306**, 307, 497
 Cauchy point, **307**
 CG-Steihaug, **308**
 conjugate gradient, **497**, 498
 coordinate descent, **306**
 dogleg, **307**, 498
 exact trust region, **308**, 498
 Fletcher-Reeves, **308**
 Hestenes-Stiefel, **308**
 Levenberg-Marquardt, **310**, 460
 Nelder-Mead simplex, **204**, 263, **303**, **309**, 500, 633
 Newton, **305**, **306**, 310, 460, 497, **498**
 Newton-CG, **306**, 309
 Polak-Ribi  re, **308**
 Polak-Ribi  re +, **308**
 steepest descent, **305**, **307**, 309
- constraint algorithm
 Logarithmic barrier, **204**, **312**
 Method of Multipliers, **311**
- minfx, **263**, **301**
 space, **303**
 step-length selection algorithm, **307**
 topology, **303**
- optimise, **467**, 480, **496**, 506, 530–535, 538, 540, 555, 577, 581, 583, 590
- order parameter, **482**, 502, 525, 526, **536**, **559**, 561
- OS X, *see* Operating system, Mac OS X
- parameter
 bounds, **458**, 468, **471**, **478**, **483**, 500, **501**, 540, 552
 limit, **475**, 485, 486, 498, 587
- parameter convolution, **343**
- patches, **288**
- PCS, *see* Pseudo-contact shift
- PDB, **58**, **73**, **113**, 468, 469, 477–480, 482, 483, 501, 527–529, 536–538, 541, 542, 544, 557, 558, 562–564,
- 624, 626, 628–631, **635**, 638–640, 645–647
- peak
 height, **73**
 intensity, **51**, **59**, **74**, 206
 volume, **73**
- plot, **471**, 485, 486, 546, 549, 566, **581**, **582**, 636
- Polak-Ribi  re, *see* optimisation, algorithm, Polak-Ribi  re
- Polak-Ribi  re +, *see* optimisation, algorithm, Polak-Ribi  re +
- Pseudo-contact shifts, **141**, **142**
- PyMOL, *see* software, PyMOL
- pyreadline, **24**
- Python, **3**, **6**, **7**, **8**, **14**, **16**, **23**, **293**, **456**, **497**, 597, **635**, 647, 648
- R₁, *see* relaxation rate, spin-lattice
- R₂, *see* relaxation rate, spin-spin
- RDC, *see* Residual dipolar couplings
- read, **454**, **459**, 464, 490, 491, 494, **528**, 537, 540, 547, 548, 557, 562, 563, 567, 568, 572, 573, 584, 585, 596, 597, 601, 605, 611–613, 620, 629, **631**, **635**, 637–640, 654
- reduced spectral density mapping, **4**, **129**
- regular expression, **485**, 497, 508–511, 593–595, 616–621
- relax, *see* software, relax
- relaxation, **455**, **459**, 460, **462**, **464**, **467**, 486, 489–491, **496**, 502, 505, **551**, 552, 570–577, 579, 581–583, 585–587, 589–591, **597**, 600, 601, 650, 655, 658
- relaxation curve-fitting, **51**
- relaxation dispersion, **4**, **147**
- R_{1ρ}-type experiment, **197**
- Analytic CPMG model, **159**
- Analytic MMQ CPMG model, **170**
- Analytic R1rho model, **176**
- Analytical model, **148**
- B14 full model, **149**, **164**
- B14 model, **149**, **166**
- Base model, **156**
- BK13 full model, **197**
- BK13 model, **197**
- CPMG-type experiment, **197**
- CR72 full model, **149**, **161**

- CR72 model, 149, 162
DPL94 model, 150, 178
IT99 model, 149, 162
LM63 3-site model, 148, 160
LM63 model, 148, 159
M61 model, 150, 177
M61 skew model, 150, 177
MMQ CR72 model, 149, 170
MP05 model, 150, 180
No Rex model, 148, 158
NS CPMG 2-site 3D full model, 149, 169
NS CPMG 2-site 3D model, 149, 169
NS CPMG 2-site expanded model, 149, 166
NS CPMG 2-site star full model, 149, 169
NS CPMG 2-site star model, 149, 170
NS MMQ 2-site model, 150, 172
NS MMQ 3-site linear model, 174
NS MMQ 3-site model, 150, 175
NS MMQ 3-site model linear, 150
NS R1rho 2-site model, 150, 181
NS R1rho 3-site linear model, 150, 183
NS R1rho 3-site model, 150, 181
Numeric CPMG model, 166
Numeric MMQ CPMG model, 172
Numeric R1rho model, 180
Numerical model, 148
R2eff model, 148, 156
TAP03 model, 150, 179
TP02 model, 150, 178
TP04 model, 197
TSMFK01 model, 149, 163
- relaxation rate
 cross rate, 86
 cross-relaxation, 86
 spin-lattice, 86
 spin-spin, 86
- repository, 32, 277, 286
 branch versioning, 287
 branches, 287
 forks, 289
 keeping up to date, 287
 merging branch back, 289
 patches, 288
- rebasing, 288
Residual dipolar coupling, 141, 142
RMSD, 74
rotation, 467, 469, 478, 482, 483, 528, 536, 539, 562, 629, 632, 633, 642
- SciPy, 23
SCons, 24, 32, 290
 API documentation, 291
 binary distribution, 32, 291
 C module compilation, 290
 clean up, 291
 help, 290
 install, 24
 source distribution, 291
 user manual (HTML version), 291
 user manual (PDF version), 290
scripting, 472, 475, 512, 557, 597, 612, 613
 sample scripts, 14
 script file, 455, 456, 460, 472, 475, 512, 557, 597, 597, 612, 613, 627
sequence, 481, 484, 497, 530–535, 555, 577, 592, 594, 595, 603, 603, 604, 604, 605, 605, 606, 606, 615, 620, 621, 631, 634, 635, 639, 640, 642, 643
simplex algorithm, *see* optimisation, algorithm, Nelder-Mead simplex
simulated annealing, 143
software
 Dasha, 6, 27, 310, 455, 460, 460, 461, 461
 Grace, 3, 5, 26, 70, 75, 76, 485, 485, 486, 546, 549, 566, 581, 582
 Modelfree, 6, 27, 310, 455, 541–543
 MOLMOL, 3, 5, 27, 512, 512, 513, 513, 525, 525, 526, 526, 527, 527, 528, 528, 529, 529, 559, 561
 NMRView, 59, 73
 OpenDX, 3, 5, 26, 471, 472
 PyMOL, 3, 5, 27
 relax, 312
 Sparky, 59, 73, 74, 215, 457, 459, 609, 612, 613, 620
 Tensor, 310
 XEasy, 59, 73, 612, 613
 Sparky, *see* software, Sparky
 spherical angles, 363

spherical diffusion, *see* diffusion, sphere (isotropic)
spheroidal diffusion, *see* diffusion, spheroid (axially symmetric)
spin container, 11
spin-lattice relaxation, *see* relaxation rate, spin-lattice
spin-spin relaxation, *see* relaxation rate, spin-spin
standard deviation, 51
steepest descent, *see* optimisation, algorithm, steepest descent
step-length, *see* optimisation, step-length selection algorithm
string, 7, 293
support request
 tracker, 296
SVN, *see* Subversion
symbolic link, 24
tab completion, 9
tar, 24, 477, 481, 484, 496, 497, 552, 604, 635
task
 tracker, 296
Tensor, *see* software, Tensor
terminal, 6
test suite, 14, 284
tracker
 bug, 296
 support request, 296
 task, 296
transverse relaxation, *see* relaxation rate, spin-spin
under-fitting, 106
Unix, *see* Operating system, Unix
user functions, 8, 9–11, 443
User interface
 GUI, 3, 14, 23, 293
 prompt, 6, 293
 scripting, 12, 293
user manual
 HTML compilation, 291
 PDF compilation, 290
web site, 30
Windows, *see* Operating system, MS Windows
write, 458, 486, 495, 550, 569, 575, 597, 606, 646, 647, 660, 661
wxPython, 23
XEasy, *see* software, XEasy