

Dynamix

Generated by Doxygen 1.8.18

Contents

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Model Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 error_mode	5
3.1.2.2 max_func_evals	6
3.1.2.3 max_iter	6
3.1.2.4 model	6
3.1.2.5 n_error_iter	6
3.1.2.6 n_iter	6
3.1.2.7 n_residues	6
3.1.2.8 nthreads	6
3.1.2.9 outputdir	6
3.1.2.10 residues	7
3.2 Orient Struct Reference	7
3.2.1 Detailed Description	7
3.2.2 Member Data Documentation	7
3.2.2.1 phi	7

3.2.2.2 theta	7
3.2.2.3 Y2	7
3.2.2.4 Y2c	8
3.3 Relaxation Struct Reference	8
3.3.1 Detailed Description	8
3.3.2 Member Data Documentation	8
3.3.2.1 field	8
3.3.2.2 R	8
3.3.2.3 Rerror	8
3.3.2.4 T	9
3.3.2.5 type	9
3.3.2.6 w1	9
3.3.2.7 wr	9
3.4 Residue Struct Reference	9
3.4.1 Detailed Description	10
3.4.2 Member Data Documentation	10
3.4.2.1 csaC	10
3.4.2.2 csaN	10
3.4.2.3 csisoC	10
3.4.2.4 csisoN	10
3.4.2.5 error_calcs	10
3.4.2.6 error_params	10
3.4.2.7 errors_mean	11
3.4.2.8 errors_std	11
3.4.2.9 ignore	11
3.4.2.10 lim_relaxation	11
3.4.2.11 min_val	11
3.4.2.12 n_relaxation	11
3.4.2.13 orients	11
3.4.2.14 parameters	11
3.4.2.15 relaxation	12
3.4.2.16 S2_dipolar	12
3.4.2.17 temp_relaxation	12
3.5 rrargs Struct Reference	12
3.5.1 Detailed Description	12
3.5.2 Member Data Documentation	12
3.5.2.1 i	12
3.5.2.2 model	13
3.5.2.3 n_iter	13
3.5.2.4 outputdir	13
3.5.2.5 resid	13

4 File Documentation	15
4.1 chisq.c File Reference	15
4.1.1 Function Documentation	15
4.1.1.1 back_calculate()	15
4.1.1.2 optimize_chisq()	16
4.2 crosen.c File Reference	16
4.2.1 Function Documentation	17
4.2.1.1 simplex()	17
4.3 datatypes.c File Reference	17
4.3.1 Function Documentation	19
4.3.1.1 calculate_Y2()	19
4.3.1.2 free_all()	20
4.3.1.3 initialise_dwig()	20
4.4 errors.c File Reference	20
4.4.1 Function Documentation	20
4.4.1.1 calc_errors()	20
4.4.1.2 calc_statistics()	21
4.4.1.3 norm_rand()	21
4.4.1.4 uniform_rand()	21
4.5 main.c File Reference	22
4.5.1 Function Documentation	22
4.5.1.1 main()	22
4.5.1.2 run_residue()	22
4.6 models.c File Reference	23
4.6.1 Function Documentation	24
4.6.1.1 EMF_R1()	24
4.6.1.2 EMF_R2()	25
4.6.1.3 GAF_13CR1()	25
4.6.1.4 GAF_13CR2()	26
4.6.1.5 GAF_15NR1()	27

4.6.1.6 GAF_15NR2()	27
4.6.1.7 GAF_CSA_R2()	28
4.6.1.8 GAF_Dipolar_R1()	28
4.6.1.9 GAF_Dipolar_R2()	29
4.6.1.10 GAF_S2()	29
4.6.1.11 J0_EMF()	29
4.6.1.12 J0_SMF()	30
4.6.1.13 SMF_R1()	30
4.6.1.14 SMF_R2()	30
4.7 read_data.c File Reference	31
4.7.1 Function Documentation	31
4.7.1.1 read_pp()	31
4.7.1.2 read_relaxation_data()	32
4.7.1.3 read_resid_data()	33
4.7.1.4 read_system_file()	33

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Model	5
Orient	7
Relaxation	8
Residue	9
rrargs	12

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

chisq.c	15
crosen.c	16
datatypes.c	17
errors.c	20
main.c	22
models.c	23
read_data.c	31

Chapter 3

Class Documentation

3.1 Model Struct Reference

Public Attributes

- int [max_func_evals](#)
- int [max_iter](#)
- int [n_iter](#)
- int [n_error_iter](#)
- char [outputdir](#) [255]
- int [model](#)
- struct [Residue](#) * [residues](#)
- int [n_residues](#)
- int [nthreads](#)
- int [error_mode](#)

3.1.1 Detailed Description

Struct containing model parameters

3.1.2 Member Data Documentation

3.1.2.1 error_mode

`Model::error_mode`

1 indicates that errors should be calculated, 0 indicates not.

3.1.2.2 max_func_evals

`Model::max_func_evals`

Unused

3.1.2.3 max_iter

`Model::max_iter`

Unused

3.1.2.4 model

`Model::model`

Defined as one of MOD_*

3.1.2.5 n_error_iter

`Model::n_error_iter`

Number of iterations for error calculation

3.1.2.6 n_iter

`Model::n_iter`

Number of iterations in main channel

3.1.2.7 n_residues

`Model::n_residues`

Length of residues; number of residues in protein

3.1.2.8 nthreads

`Model::nthreads`

Number of threads to spawn for calculations

3.1.2.9 outputdir

`Model::outputdir`

Output directory

3.1.2.10 residues

`Model::residues`

Pointer to an array of `n_residues` residues

The documentation for this struct was generated from the following file:

- [datatypes.c](#)

3.2 Orient Struct Reference

Public Attributes

- float [phi](#)
- float [theta](#)
- double complex [Y2](#) [5]
- double complex [Y2c](#) [5]

3.2.1 Detailed Description

Struct defining atomic relative orientations

3.2.2 Member Data Documentation

3.2.2.1 phi

`Orient::phi`

Phi angle (radians)

3.2.2.2 theta

`Orient::theta`

Theta angle (radians)

3.2.2.3 Y2

`Orient::Y2`

Array containing the second order spherical harmonics, Y_{2m} , for $m = -2$ (0) to 2 (5).

3.2.2.4 Y2c

`Orient::Y2c`

Conjugate to Y2.

The documentation for this struct was generated from the following file:

- [datatypes.c](#)

3.3 Relaxation Struct Reference

Public Attributes

- float [field](#)
- float [wr](#)
- float [w1](#)
- int [type](#)
- float [R](#)
- float [Rerror](#)
- float [T](#)

3.3.1 Detailed Description

Struct defining a relaxation measurement

3.3.2 Member Data Documentation

3.3.2.1 field

`Relaxation::field`

Proton nutation frequency for field in which relaxation measurement was made in MHz

3.3.2.2 R

`Relaxation::R`

[Relaxation](#) measurement in s⁻¹

3.3.2.3 Rerror

`Relaxation::Rerror`

Monte-Carlo relaxaxtion measurement error (2 standard deviations) in s⁻¹

3.3.2.4 T

`Relaxation::T`

Temperature of measurement (Kelvin)

3.3.2.5 type

`Relaxation::type`

One of R_15NR1, R_15NR1p, R_13CR1, R_13CR1p depending on which parameter this is a measurement of

3.3.2.6 w1

`Relaxation::w1`

Nutation frequency for spin lock relaxation measurement (in Hz)

3.3.2.7 wr

`Relaxation::wr`

Spinning frequency (MAS) for relaxation measurement (in Hz)

The documentation for this struct was generated from the following file:

- [datatypes.c](#)

3.4 Residue Struct Reference

Public Attributes

- float [S2_dipolar](#)
- float [csisoN](#)
- float [csisoC](#)
- float [csaN](#) [3]
- float [csaC](#) [3]
- struct [Orient](#) [orients](#) [14]
- struct [Relaxation](#) * [relaxation](#)
- struct [Relaxation](#) * [temp_relaxation](#)
- long double ** [error_params](#)
- int [n_relaxation](#)
- int [lim_relaxation](#)
- int [ignore](#)
- double [min_val](#)
- long double * [parameters](#)
- long double * [errors_std](#)
- long double * [errors_mean](#)
- int [error_calcs](#)

3.4.1 Detailed Description

Struct defining a residue

3.4.2 Member Data Documentation

3.4.2.1 csaC

`Residue::csaC`

Anisotropic chemical shift components for carbon

3.4.2.2 csaN

`Residue::csaN`

Anisotropic chemical shift components for nitrogen

3.4.2.3 csisoC

`Residue::csisoC`

Isotropic chemical shift component for carbonyl carbon.

3.4.2.4 csisoN

`Residue::csisoN`

Isotropic chemical shift component for amide nitrogen.

3.4.2.5 error_calcs

`Residue::error_calcs`

Actual number of points used in error calculation

3.4.2.6 error_params

`Residue::error_params`

Pointer to array of pointers, each of which points to set of optimized parameters for each error iteration

3.4.2.7 errors_mean

`Residue::errors_mean`

Pointer to array containing means for each error parameter

3.4.2.8 errors_std

`Residue::errors_std`

Pointer to array containing standard deviations for each error parameter

3.4.2.9 ignore

`Residue::ignore`

Flag to ignore residue from calculations

3.4.2.10 lim_relaxation

`Residue::lim_relaxation`

Length of [Residue::relaxation](#) array

3.4.2.11 min_val

`Residue::min_val`

Optimized minimum chisq value

3.4.2.12 n_relaxation

`Residue::n_relaxation`

Number of relaxation measurements for this residue

3.4.2.13 orients

`Residue::orients`

Orientations for intraresidue atom vectors, defined 0-13 by `OR_*`

3.4.2.14 parameters

`Residue::parameters`

Optimized parameters relating to minimum chisq value

3.4.2.15 relaxation

`Residue::relaxation`

Dynamically allocated pointer to array of structs containing the relaxation data

3.4.2.16 S2_dipolar

`Residue::S2_dipolar`

Dipolar order parameter; used in EMF analysis

3.4.2.17 temp_relaxation

`Residue::temp_relaxation`

Pointer used to keep track of relaxation data during error calculation

The documentation for this struct was generated from the following file:

- [datatypes.c](#)

3.5 rrargs Struct Reference

Public Attributes

- int [i](#)
- struct [Residue](#) * [resid](#)
- int [model](#)
- int [n_iter](#)
- char [outputdir](#) [255]

3.5.1 Detailed Description

Struct containing information relevant to a thread

3.5.2 Member Data Documentation

3.5.2.1 i

`rrargs::i`

Count variable

3.5.2.2 model

`rrargs::model`

[Model](#) type, one of MOD_*

3.5.2.3 n_iter

`rrargs::n_iter`

Number of iterations

3.5.2.4 outputdir

`rrargs::outputdir`

Output directory

3.5.2.5 resid

`rrargs::resid`

Pointer to residue being considered

The documentation for this struct was generated from the following file:

- [datatypes.c](#)

Chapter 4

File Documentation

4.1 chisq.c File Reference

```
#include <stdio.h>
#include <math.h>
```

Functions

- double [optimize_chisq](#) (long double *opts, struct [Residue](#) *resid, int model)
- int [back_calculate](#) (long double *opts, struct [Residue](#) *resid, int model, char *filename)

4.1.1 Function Documentation

4.1.1.1 back_calculate()

```
int back_calculate (
    long double * opts,
    struct Residue * resid,
    int model,
    char * filename )
```

Back calculation function. Loops over all relaxation measurements and predicts using models. Outputs file containing [relaxation id, calculated, real, real error].

Parameters

<i>opts</i>	Pointer to array containing parameters.
<i>resid</i>	Pointer to residue being considered
<i>model</i>	MOD_SMF etc.
<i>filename</i>	File to output calculations into

Returns

Returns 1 if successful, else -1.

4.1.1.2 optimize_chisq()

```
double optimize_chisq (
    long double * opts,
    struct Residue * resid,
    int model )
```

Optimization function. Loops over all relaxation measurements and predicts using models. Calculates $(\text{real} - \text{calc})^2 / (\text{error}^2)$ over all relaxation measurements and returns.

Parameters

<i>opts</i>	Pointer to array containing parameters.
<i>resid</i>	Pointer to residue being considered
<i>model</i>	MOD_SMF etc.

Returns

Returns chisq value.

4.2 crosen.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
```

Macros

- #define **MAX_IT** 1000
maximum number of iterations
- #define **ALPHA** 1.0
reflection coefficient
- #define **BETA** 0.5
contraction coefficient
- #define **GAMMA** 2.0
expansion coefficient

Functions

- double **simplex** (double(*func)(long double[], struct Residue *, int), long double start[], int n, long double EPSILON, long double scale, struct Residue *resid, int model)

4.2.1 Function Documentation

4.2.1.1 simplex()

```
double simplex (
    double(*) (long double[], struct Residue *, int) func,
    long double start[],
    int n,
    long double EPSILON,
    long double scale,
    struct Residue * resid,
    int model )
```

Implementation of Nelder-Mead Simplex method written by Michael F. Hutt.
Has been modified to allow for optimization of the [optimize_chisq\(\)](#) function directly.

Parameters

<i>func</i>	Pointer to function taking arguments (long double[], struct Residue , int) to be optimized (see optimize_chisq())
<i>start[]</i>	Array of starting parameters
<i>n</i>	Length of start [] (eg how many parameters are included)
<i>EPSILON</i>	Convergence requirement - lower means closer convergence but slower operation
<i>scale</i>	Scale factor
<i>resid</i>	Pointer to residue being optimized
<i>model</i>	Model type (MOD_SMF etc)

Returns

Returns minimum of (*func).

4.3 datatypes.c File Reference

```
#include <stdlib.h>
#include <math.h>
#include <complex.h>
```

Classes

- struct [Model](#)
- struct [Orient](#)
- struct [Residue](#)
- struct [Relaxation](#)
- struct [rargs](#)

Macros

- #define `MOD_SMF` 0
Simple `Model` Free.
- #define `MOD_EMF` 1
Extended `Model` Free using S2 dipolar.
- #define `MOD_EMFT` 2
Extended `Model` Free with Temperature Dependence using S2 dipolar.
- #define `MOD_SMFT` 3
Simple `Model` Free with Temperature Dependence.
- #define `MOD_DEMF` 4
Extended `Model` Free.
- #define `MOD_DEMFT` 5
Extended `Model` Free with Temperature Dependence.
- #define `MOD_GAF` 6
Gaussian Axial Fluctuations model.
- #define `MOD_GAFT` 7
Gaussian Axial Fluctuations model with Temperature Dependence.
- #define `DATA_S2` 0
- #define `DATA_CSISON` 1
- #define `DATA_CSISOC` 2
- #define `R_15NR1` 0
- #define `R_15NR1p` 1
- #define `R_13CR1` 2
- #define `R_13CR1p` 3
- #define `OR_NH` 0
- #define `OR_NC` 1
- #define `OR_NCA` 2
- #define `OR_NCSAxx` 3
- #define `OR_NCSAyy` 4
- #define `OR_NCSAzz` 5
- #define `OR_CCAp` 6
- #define `OR_CCAc` 7
- #define `OR_CN` 8
- #define `OR_CNH` 9
- #define `OR_CH` 10
- #define `OR_CCSAxx` 11
- #define `OR_CCSAyy` 12
- #define `OR_CCSAzz` 13
- #define `N_RELAXATION` 50
Approximate number of relaxation measurements; will dynamically allocate if overflows.
- #define `NTHREADS` 40
Unused.
- #define `THREAD_STACK` 32768*2
Bytes per thread. Raise if stack overflows, lower if insufficient stack for number of workers.
- #define `MIN_VAL` 10000000
Maximum value of chisq function for which it will be considered fit.
- #define `RYD` 8.3144621
Rydberg Constant in JK-1mol-1.
- #define `D_NH` 72038.41107
Dipolar couplings taken from NHCO_3GAFSquaredEa for amide N-H.
- #define `D_CC` 13467.66052

- Dipolar couplings taken from NHCO_3GAFSquaredEa for alphatic C-Calpha.*

 - #define `D_CH` 22362.47724
- Dipolar couplings taken from NHCO_3GAFSquaredEa for C-H.*

 - #define `D_CHr` 31491.71046
- Dipolar couplings taken from NHCO_3GAFSquaredEa for C-Hrest.*

 - #define `D_CN` 8175.2
- Dipolar couplings taken from NHCO_3GAFSquaredEa for C-N.*

 - #define `D_CaN` 6180.1
- Dipolar couplings taken from NHCO_3GAFSquaredEa for amide N - aliphatic C.*

 - #define `D_HNr` 13108.32273
- Dipolar couplings taken from NHCO_3GAFSquaredEa for amide N-Hrest.*

 - #define `MODE_15N` 0
 - #define `MODE_13C` 1
 - #define `MODE_REAL` 0
 - #define `MODE_IMAG` 1
 - #define `MODE_COMP` 2
 - #define `DIPOLAR` 1
 - #define `NODIP` 0
 - #define `HALF_PI` $M_PI / 2.$
 - #define `VERBOSE` 1
 - #define `T_DOWN` 0.0000000010000
 - #define `T_UP` 1000000000
 - #define `T_S` 9

Functions

- void `calculate_Y2` (struct `Orient` *or)
- void `initialise_dwlg` (void)
- void `free_all` (struct `Model` *m)

Variables

- long double `Dwig` [5][5]
5x5 array containing Wigner components for pi/2

4.3.1 Function Documentation

4.3.1.1 calculate_Y2()

```
void calculate_Y2 (
    struct Orient * or )
```

Populates the second order spherical harmonics for orientation vector passed as pointer.

Parameters

<i>or</i>	Orientation vector
-----------	--------------------

Returns

NULL

4.3.1.2 free_all()

```
void free_all (
    struct Model * m )
```

Function to free all allocated memory within [Model](#). @params m Pointer to [Model](#) to be freed.

4.3.1.3 initialise_dwig()

```
void initialise_dwig (
    void )
```

Initialises Wigner D matrix in global space.

4.4 errors.c File Reference

```
#include <stdio.h>
#include <math.h>
```

Functions

- long double [uniform_rand](#) (void)
- float [norm_rand](#) (float mean, float std)
- void [calc_statistics](#) (long double *vals, int length, long double *mean, long double *std)
- void * [calc_errors](#) (void *input)

4.4.1 Function Documentation

4.4.1.1 calc_errors()

```
void* calc_errors (
    void * input )
```

Calculates errors for a given residue. Error calculation is done by varying the relaxation values according to a normal distribution with mean (R) and standard deviation (Rerror/2). Then simplex optimization is performed, and the newly optimized parameters stored. Then statistics of these back optimized parameters are taken, with the errors being the standard deviation of these.

Parameters

<i>input</i>	rrang struct containing the residue under consideration.
--------------	--

4.4.1.2 calc_statistics()

```
void calc_statistics (
    long double * vals,
    int length,
    long double * mean,
    long double * std )
```

Calculates mean and standard deviation of values contained in array, then puts these into the given pointers.

Parameters

<i>vals</i>	Pointer to array of values to take statistics of
<i>length</i>	Length of vals
<i>mean</i>	Pointer to long double to contain mean
<i>std</i>	Pointer to long double to contain standard deviation

4.4.1.3 norm_rand()

```
float norm_rand (
    float mean,
    float std )
```

Uses Box-Muller method to generate a normally distributed random number.

Parameters

<i>mean</i>	Mean of normal distribution to select random variable from
<i>std</i>	Standard deviation of normal distribution from which random variable selected

Returns

float Returns normally distributed random number

4.4.1.4 uniform_rand()

```
long double uniform_rand (
    void )
```

Generates uniform random long double from 0 to 1 inclusive.

Returns

long double Long double containing uniform random number.

4.5 main.c File Reference

```
#include <stdio.h>
#include "datatypes.c"
#include "read_data.c"
#include "models.c"
#include "chisq.c"
#include "crosen.c"
#include "errors.c"
#include <time.h>
#include <pthread.h>
```

Functions

- void * [run_residue](#) (void *input)
- int [main](#) (int argc, char *argv[])

4.5.1 Function Documentation**4.5.1.1 main()**

```
int main (
    int argc,
    char * argv[] )
```

Start function. Initialises parameters, loads files, spawns threads and outputs data. @opts filename Takes input as path to file containing System definition (*dx file) @opts -e Enables error mode

4.5.1.2 run_residue()

```
void* run_residue (
    void * input )
```

Operates residue optimization. Generates random parameter guesses and passes these to the simplex function.

Parameters

<i>input</i>	Pointer to rrag containing thread information
--------------	---

SMF parameters are
[0] tau

[1] S2
 SMFT parameters;
 [0] tau
 [1] S2
 [2] Ea
 EMF parameters
 [0] tau slow
 [1] S2 slow
 [2] tau fast
 NOTE: The fast order parameter is calculated as $S2_{dipolar}/S2s$
 EMFT parameters
 [0] tau slow
 [1] S2 slow
 [2] tau fast
 [3] activation energy for slow motion
 [4] activation energy for fast motion
 DEMF parameters
 [0] tau slow
 [1] S2 slow
 [2] tau fast
 [3] S2 fast
 DEMFT parameters
 [0] tau slow
 [1] S2 slow
 [2] tau fast
 [3] S2 fast
 [4] activation energy for slow motion
 [5] activation energy for fast motion
 GAF parameters
 [0] tau slow
 [1] tau fast
 [2-4] alpha, beta, gamma deflections for slow motions
 [5-7] alpha, beta, gamma deflections for fast motions
 GAFT parameters
 [0] tau slow
 [1] tau fast
 [2-4] alpha, beta, gamma deflections for slow motions
 [5-7] alpha, beta, gamma deflections for fast motions
 [8] activation energy for slow motion
 [9] activation energy for fast motion

4.6 models.c File Reference

```

#include <stdio.h>
#include <math.h>
#include <complex.h>

```

Functions

- long double [J0_SMF](#) (long double omega, long double tau, long double S2)
- long double [J0_EMF](#) (long double omega, long double taus, long double S2s, long double tau_f, long double S2f)

- long double **sq** (long double x)
- int **sq_i** (int x)
- double **GAF_Dipolar_R1** (long double omega_obs, long double omega_neigh, long double taus, long double S2s, long double tauf, long double S2f, long double D)
- double **GAF_CSA_R2** (long double omega, float w1, float wr, long double taus, long double S2s, long double tauf, long double S2f, double D2)
- double **GAF_Dipolar_R2** (long double omega_obs, long double omega_neigh, float w1, float wr, long double taus, long double S2s, long double tauf, long double S2f, double D)
- double **SMF_R1** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double tau, long double S2, int mode)
- double **SMF_R2** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double tau, long double S2, int mode)
- double **EMF_R1** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double S2s, long double tauf, long double S2f, int mode)
- double **EMF_R2** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double S2s, long double tauf, long double S2f, int mode)
- int **GAF_S2** (long double sig[3], struct [Orient](#) **A, struct [Orient](#) **B, double *S2[], int length, int mode)
- double **GAF_15NR1** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double tauf, long double *sigs, long double *sigf)
- double **GAF_15NR2** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double tauf, long double *sigs, long double *sigf)
- double **GAF_13CR1** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double tauf, long double *sigs, long double *sigf)
- double **GAF_13CR2** (struct [Residue](#) *res, struct [Relaxation](#) *relax, long double taus, long double tauf, long double *sigs, long double *sigf)

4.6.1 Function Documentation

4.6.1.1 EMF_R1()

```
double EMF_R1 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long double S2s,
    long double tauf,
    long double S2f,
    int mode )
```

Calculates R1 relaxation rate using extended model free analysis Kurbanov2011, 8-10

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>S2s</i>	Order parameter of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>S2f</i>	Order parameter of fast motion
<i>mode</i>	One of MODE_15N or MODE_13C depending on relaxation data type considered.

Returns

R1 Returns R1 as double

4.6.1.2 EMF_R2()

```
double EMF_R2 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long double S2s,
    long double tauf,
    long double S2f,
    int mode )
```

Calculates R2 relaxation rate using extended model free analysis Kurbanov2011, 8-10

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>S2s</i>	Order parameter of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>S2f</i>	Order parameter of fast motion.
<i>mode</i>	One of MODE_15N or MODE_13C depending on relaxation data type considered.

Returns

R2 Returns R2 as double

4.6.1.3 GAF_13CR1()

```
double GAF_13CR1 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long double tauf,
    long double * sigs,
    long double * sigf )
```

Calculates R1 relaxation rate for 13C nucleus under gaussian axial fluctuations.

Warning

Has not been verified against MATLAB model as there isn't really an equivalent to it that I have set up.
Has not been tested on 13C data
I'm unsure of the validity of CSAxy.

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>sigs</i>	Deflection angles [alpha, beta, gamma] for slow motion
<i>sigf</i>	Deflection angles [alpha, beta, gamma] for fast motion

Returns

R1 Returns R1 as double

4.6.1.4 GAF_13CR2()

```
double GAF_13CR2 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long doubletauf,
    long double * sigs,
    long double * sigf )
```

Calculates R2 relaxation rate for 13C nucleus under gaussian axial fluctuations.

Warning

Has not been verified against MATLAB model as there isn't really an equivalent to it that I have set up.

Has not been tested on 13C data

I'm unsure of the validity of CSAxy.

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>sigs</i>	Deflection angles [alpha, beta, gamma] for slow motion
<i>sigf</i>	Deflection angles [alpha, beta, gamma] for fast motion

Returns

R2 Returns R2 as double

4.6.1.5 GAF_15NR1()

```
double GAF_15NR1 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long double tauf,
    long double * sigs,
    long double * sigf )
```

Calculates R1 relaxation rate for 15N nucleus under gaussian axial fluctuations.

Warning

Has not been verified against MATLAB model as there isn't really an equivalent to it that I have set up.
I'm unsure of the validity of NCSAxy.

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>sigs</i>	Deflection angles [alpha, beta, gamma] for slow motion
<i>sigf</i>	Deflection angles [alpha, beta, gamma] for fast motion

Returns

R1 Returns R1 as double

Eq A30, Bremi1997

4.6.1.6 GAF_15NR2()

```
double GAF_15NR2 (
    struct Residue * res,
    struct Relaxation * relax,
    long double taus,
    long double tauf,
    long double * sigs,
    long double * sigf )
```

Calculates R2 relaxation rate for 15N nucleus under gaussian axial fluctuations.

Warning

Has not been verified against MATLAB model as there isn't really an equivalent to it that I have set up.
I'm unsure of the validity of NCSAxy.

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>taus</i>	Correlation time of slow motion
<i>tauf</i>	Correlation time of fast motion
<i>sigs</i>	Deflection angles [alpha, beta, gamma] for slow motion
<i>sigf</i>	Deflection angles [alpha, beta, gamma] for fast motion

Returns

R2 Returns R2 as double

I'm unsure if this should be $R2CSAx + R2CSAy + R2CSAxy$ or $- R2CSAxy$. In the original MATLAB scripts, it is $- R2CSAxy$; in `NHCO_3GAFSquardEa`, the form of $R2CSAxy$ is really odd. In `NH3DEglobalGAF` it is also $- R2CSAxy$ but the form is the same. In Lienin1998's SI, eq 5 and 6 it is $+R2CSAxy$ - same for Bremi1997 eq A30

4.6.1.7 GAF_CSA_R2()

```
double GAF_CSA_R2 (
    long double omega,
    float w1,
    float wr,
    long double taus,
    long double S2s,
    long double tauf,
    long double S2f,
    double D2 ) [inline]
```

Calculates the R2 contribution of CSA of nuclei with frequency omega. D2 is the squared $D22x/D22y/D22xy$ variable (dependent on CSA) Kurbanov2011, eqs 18-20.

4.6.1.8 GAF_Dipolar_R1()

```
double GAF_Dipolar_R1 (
    long double omega_obs,
    long double omega_neigh,
    long double taus,
    long double S2s,
    long double tauf,
    long double S2f,
    long double D ) [inline]
```

Calculates R1 contribution under GAF model for Dipolar interaction between two atoms. `omega_obs` is the frequency of the observed nucleus, `omega_neigh` is the frequency of the other atoms. `D` is the dipolar coupling in rad/s. Equation as 3.28, McConnell1986, only for solid state spins. Kurbanov2011, eq 9 (factor of 1/10 arising from $(2/5)$ in spectral density and 1/4. in dipolar.

4.6.1.9 GAF_Dipolar_R2()

```
double GAF_Dipolar_R2 (
    long double omega_obs,
    long double omega_neigh,
    float w1,
    float wr,
    long double taus,
    long double S2s,
    long double tauf,
    long double S2f,
    double D ) [inline]
```

Calculates R2 contribution of dipolar interaction between two atoms as in GAF_Dipolar_R1 Kurbanov2011, for the case where $\theta_p = 90$ such that $\sin(\theta_p) = 1$ and so $R1p = 0.5 \cdot R1 + R1\delta$

4.6.1.10 GAF_S2()

```
int GAF_S2 (
    long double sig[3],
    struct Orient ** A,
    struct Orient ** B,
    double * S2[],
    int length,
    int mode )
```

Calculates the order parameter relating to two orientations A and B undergoing anisotropic axial fluctuations of magnitude sig. This function parallelises the calculation such that the loop over l, m, mp, k, kp only needs to occur once per motion - also meaning that the calculation of the exponential only needs to be done once. Checks for conjugation are made to reduce the need to calculate complex powers only to the cases where this is necessary. Lienin1998, eq 6

Parameters

<i>sig</i>	Array containing [alpha, beta, gamma] deflection angles
<i>A</i>	Array containing pointers to orientation vectors for 'A' for each S2 being calculated.
<i>B</i>	Array containing pointers to orientation vectors for 'B' for each S2 being calculated.
<i>S2</i>	Array containing pointers to S2 values - this is where output will be put.
<i>length</i>	Number of S2 values being calculated (eg length of S2)
<i>mode</i>	MODE_REAL or MODE_IMAG depending on which result is being returned.

Returns

Returns 1 in all cases.

4.6.1.11 J0_EMF()

```
long double J0_EMF (
    long double omega,
```

```

    long double taus,
    long double S2s,
    long double tauf,
    long double S2f ) [inline]

```

Calculates spectral density function for given frequency according to extended model free analysis. Implements $J(w) = (1 - S2f) \tau_{uf} / (1 + (w \tau_{uf})^2) + S2f (1 - S2s) \tau_{us} / (1 + (w \tau_{us})^2)$, as in eq 2, Clore 1990 (assuming $S2 = S2s * S2f$ and that there is no overall tumbling).

4.6.1.12 JO_SMF()

```

long double JO_SMF (
    long double omega,
    long double tau,
    long double S2 ) [inline]

```

Calculates spectral density function for given frequency (omega) according to simple model free analysis. Implements $J(w) = (1 - S2) \tau / (1 + (w \tau)^2)$ as from Lipari1982 eq 35 (assuming no overall tumbling) and the factor of 2/5 has been taken into the summation (see below).

4.6.1.13 SMF_R1()

```

double SMF_R1 (
    struct Residue * res,
    struct Relaxation * relax,
    long double tau,
    long double S2,
    int mode )

```

Calculates R1 relaxation rate using simple model free analysis. Kurbanov2011, eqs 8-10

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>tau</i>	Correlation time of motion being optimized
<i>S2</i>	Order parameter of motion
<i>mode</i>	One of MODE_15N or MODE_13C depending on relaxation data type considered.

Returns

R1 Returns R1 as double

4.6.1.14 SMF_R2()

```

double SMF_R2 (
    struct Residue * res,

```

```

    struct Relaxation * relax,
    long double tau,
    long double S2,
    int mode )

```

Calculates R2 relaxation rate using simple model free analysis Kurbanov2011, eqs 8-10

Parameters

<i>res</i>	Pointer to residue being considered
<i>relax</i>	Pointer to relaxation measurement being modelled
<i>tau</i>	Correlation time of motion being optimized
<i>S2</i>	Order parameter of motion
<i>mode</i>	One of MODE_15N or MODE_13C depending on relaxation data type considered.

Returns

R2 Returns R2 as double

4.7 read_data.c File Reference

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

```

Functions

- int [read_resid_data](#) (struct [Model](#) *m, char *filename, int dt)
- int [read_pp](#) (struct [Model](#) *m, char *filename, int orient)
- int [read_relaxation_data](#) (struct [Model](#) *m, char *filename)
- int [read_system_file](#) (char *filename, struct [Model](#) *m)
- int [print_system](#) (struct [Model](#) *m, char *filename)

4.7.1 Function Documentation

4.7.1.1 read_pp()

```

int read_pp (
    struct Model * m,
    char * filename,
    int orient )

```

Reads peptide plane orientation data. Should be in a file with 3 spaced columns;
\t Column 1 should contain the residue number indexed from 1
\t Column 2 should contain the theta angle (in radians)
\t Column 3 should contain the phi angle (in radians)

Note

Comments may be inserted by preceeding a line with "

Parameters

<i>m</i>	Pointer to model
<i>filename</i>	Filename
<i>orient</i>	One of OR_* denoting which orientations are being read.

Returns

1 if successful, else -1.

4.7.1.2 read_relaxation_data()

```
int read_relaxation_data (
    struct Model * m,
    char * filename )
```

Reads relaxation data. [Relaxation](#) data should be set out with two components.

The file should begin with the keys 'FIELD', 'WR', 'W1', 'TYPE' denoting the form of the data, eg

\t FIELD = 700

\t WR = 50000

\t W1 = 0

\t TEMP = 300

\t TYPE = 15NR1

Each parameter will default to -1 so if one is not set this may cause errors (though a warning will be given).

TYPE may be one of [15NR1, 15NR1p, 13CR1, 13CR1p] currently, though it should be noted that 13C has not been tested.

Warning

There must be a space on each side of the equals sign (eg, TEMP = 300 is good, TEMP=300 is bad)

Once this head section is complete, it should be followed by "#DATA".

After this, data should be split into three columns;

\t Column 1: residue number (starting from 1)

\t Column 2: [Relaxation](#) rate (in s⁻¹)

\t Column 3: [Relaxation](#) error, two standard deviations (in s⁻¹)

Note

Comments may be inserted by preceeding a line with "

Parameters

<i>m</i>	Pointer to model
<i>filename</i>	Filename

Returns

1 if successful, else -1.

4.7.1.3 read_resid_data()

```
int read_resid_data (
    struct Model * m,
    char * filename,
    int dt )
```

Reads residue specific data from datafile.

Data file being read should be set out in columns, with spaces between them.

\t Column one should contain the residue number as an integer, indexed from 1.

\t Column two should contain the data of note.

\t Column three should contain the error (or 0).

Note

Comments may be inserted by preceeding a line with "

Parameters

<i>m</i>	Pointer to model
<i>filename</i>	File to read
<i>dt</i>	Data type (one of DATA_S2, DATA_CSISOC, DATA_CSISON)

Returns

1 if successful, -1 else.

4.7.1.4 read_system_file()

```
int read_system_file (
    char * filename,
    struct Model * m )
```

Reads system file. System file should begin with key value pairs laid out as

\t KEY = VALUE

With spaces either side of the equals. Keys which may be used are as follows;

\t MODEL (one of SMF, SMFT, EMF, EMFT, GAF, GAFT)

\t S2DIP (file containing dipolar order parameters)

\t CSISON (file containing isotropic chemical shifts for N)

\t CSISOC (file containign isotropic chemical shifts for C)

\t N_RESIDUES (number of residues in protein)

\t OUTPUT (directory for output data)

\t N_ITER (number of iterations for main fitting)
\t IGNORE (may be repeated, once for each residue to ignore.)
\t N_ERROR_ITER (number of iterations for error calculation)
\t OR_* (file containing orientation data. May be one of OR_NH, OR_NC, OR_NCA, OR_NCSAxx, OR_NCSAyy, OR_NCSAzz, OR_CCAp, OR_CCAc, OR_CN, OR_CNH, OR_CCSAxx, OR_CCAyy, OR_CCAzz. Should be set out as in [read_pp\(\)](#).)
\t NTHREADS (number of parallel threads to spawn)
This should then be followed by "#RELAXATION". After this, the files containing relaxation data (as set out in [read_relaxation_data\(\)](#)) should be listed.

Note

Comments may be inserted by preceeding a line with "

Warning

There must be a space on each side of the equals sign (eg, TEMP = 300 is good, TEMP=300 is bad)

Parameters

<i>filename</i>	filename
<i>m</i>	Pointer to model

Returns

1 if successful, else -1.