Chapter 1

Program functions

The following is an alphabetical list of the functions availible 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)'.

1.1 Function name: 'intro_on'

Function for turning the function introductions on.

1.2 Function name: 'init_data'

Function for reinitialising self.relax.data

1.3 Function name: 'fix'

Function for either fixing or allowing parameter values to change.

```
Keyword Arguments
~~~~~~~~~~~~~~~~~

run:  The name of the run.

element:  Which element to fix.

fixed:  A flag specifying if the parameters should be fixed or allowed to change.


Description
~~~~~~~~~~~

The keyword argument 'element' can be any of the following:

'diff' - the diffusion tensor parameters.  This will allow all diffusion tensor parameters
to be toggled.

an integer - if an integer number is given, then all parameters for the residue
corresponding to that number will be toggled.
```

```
'all_res' - using this keyword, all parameters from all residues will be toggled.

'all' - all parameter will be toggled.  This is equivalent to combining both 'diff' and
'all_res'.


The flag 'fixed', if set to 1, will fix parameters, while a value of 0 will allow parameters
to vary.


Only parameters corresponding to the given run will be affected.
```

1.4 Function name: 'system'

Function which executes the user supplied shell command.

1.5 Function name: 'grid_search'

The grid search function.

```
Keyword Arguments
~~~~~~~~~~~~~~~~~


run:  The name of the run to apply the grid search to.

lower:  An array of the lower bound parameter values for the grid search.  The length of the
array should be equal to the number of parameters in the model.

upper:  An array of the upper bound parameter values for the grid search.  The length of the
array should be equal to the number of parameters in the model.

inc:  The number of increments to search over.  If a single integer is given then the number
of increments will be equal in all dimensions.  Different numbers of increments in each
direction can be set if 'inc' is set to an array of integers of length equal to the number
of parameters.

constraints:  A flag specifying whether the parameters should be constrained.  The default
is to turn constraints on (constraints=1).

print_flag:  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.
```

1.6 Function name: 'eliminate'

Function for model elimination.

```
Keyword arguments
~~~~~~~~~~~~~~~~~


run:  The name of the run(s).  By supplying a single string, array of strings, or None, a
single run, multiple runs, or all runs will be selected respectively.

function:  A user supplied function for model elimination.
```

```
args:  A tuple of arguments for model elimination.


Description
~~~~~~~~~~~

This function is used for model validation to eliminate or reject models prior to model
selection.  Model validation is a part of mathematical modelling whereby models are either
accepted or rejected.

Empirical rules are used for model rejection and are listed below.  However these can be
overridden by supplying a function.  The function should accept five arguments, a string
defining a certain parameter, the value of the parameter, the run name, 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 1, otherwise it should
return 0.  The function will be executed multiple times, once for each parameter of the
model.

The 'args' keyword argument should be a tuple, a list enclosed in round brackets, and will
be passed to the user supplied function or the inbuilt function.  For a description of the
arguments accepted by the inbuilt functions, see below.

Once a model is rejected, the select flag corresponding to that model will be set to 0 so
that model selection, or any other function, will then skip the model.



Model-free model elimination rules
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Local tm.

The local tm, in some cases, may exceed the value expected for a global correlation time.
Generally the tm 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 overriden by supplying the value (in
seconds) as the first element of the args tuple.


Internal correlation times {te, tf, ts}.

These parameters may experience the same problem as the local tm in that the model fails and
the parameter value is stuck at the upper limit.  These parameters are constrained using the
formula 'te, tf, ts <= 2.tm'.  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.tm.  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 tm value greater than 25 ns and models with
internal correlation times greater than 1.5 times tm, set 'args' to (25 * 1e-9, 1.5).
```

1.7 Function name: 'intro_off'

Function for turning the function introductions off.

1.8 Function name: ‘calc’

Function for calculating the function value.

```
Keyword Arguments
~~~~~~~~~~~~~~~~~


run:  The name of the run.
```

1.9 Function name: ‘angles’

Function for calculating the angles between the XH bond vector and the diffusion tensor.

```
Keyword Arguments
~~~~~~~~~~~~~~~~~


run:  The name of the run.


Description
~~~~~~~~~~~


If the diffusion tensor is isotropic for the run, then nothing will be done.

If the diffusion tensor is axially symmetric, then the angle alpha will be calculated for
each XH bond vector.

If the diffusion tensor is fully anisotropic, then the three angles will be calculated.
```

1.10 Function name: ‘minimise’

Minimisation function.

```
Arguments
~~~~~~~~~


The arguments, which should all be strings, specify the minimiser as well as its options.  A
minimum of one argument is required.  As this calls the function 'generic_minimise' the full
list of allowed arguments is shown below in the reproduced 'generic_minimise' docstring.
Ignore all sections except those labelled as minimisation algorithms and minimisation
options.  Also do not select the Method of Multipliers constraint algorithm as this is used
in combination with the given minimisation algorithm if the keyword argument 'constraints'
is set to 1.  The grid search algorithm should also not be selected as this is accessed
using the 'grid' function instead.  The first argument passed will be set to the
minimisation algorithm while all other arguments will be set to the minimisation options.

Keyword arguments differ from normal arguments having the form "keyword = value".  All
arguments must precede keyword arguments in python.  For more information see the examples
section below or the python tutorial.
```

```
Keyword Arguments
~~~~~~~~~~~~~~~~~


run:  The name of the run.

func_tol:  The function tolerance.  This is used to terminate minimisation once the function
value between iterations is less than the tolerance.  The default value is 1e-25.

grad_tol:  The gradient tolerance.  Minimisation is terminated if the current gradient value
is less than the tolerance.  The default value is None.

max_iterations:  The maximum number of iterations.  The default value is 1e7.

constraints:  A flag specifying whether the parameters should be constrained.  The default
is to turn constraints on (constraints=1).

scaling:  The diagonal scaling flag.  The default that scaling is on (scaling=1).


print_flag:  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
~~~~~~~~~~~


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 extemely 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 S2 = 0.5, te = 200 ps, and Rex = 15 1/s at 600 MHz,
the unscaled parameter vector would be [0.5, 2.0e-10, 1.055e-18].  Rex 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*6*1e8)**2].  By dividing the unscaled
parameter vector by the scaling vector the scaled parameter vector is [0.5, 0.2, 15.0].  To
revert to the original unscaled parameter vector, the scaled parameter vector and scaling
vector are multiplied.


Examples
~~~~~~~~


To minimise the model-free run 'm4' using Newton minimisation together with the GMW81
Hessian modification algorithm, the More and Thuente line search algorithm, a function
tolerance of 1e-25, no gradient tolerance, a maximum of 10,000,000 iterations, constraints
turned on to limit parameter values, and have normal printout, type any combination of:

relax> minimise('newton', run='m4')
relax> minimise('Newton', run='m4')
relax> minimise('newton', 'gmw', run='m4')
relax> minimise('newton', 'mt', run='m4')
relax> minimise('newton', 'gmw', 'mt', run='m4')
relax> minimise('newton', 'mt', 'gmw', run='m4')
relax> minimise('newton', run='m4', func_tol=1e-25)
relax> minimise('newton', run='m4', func_tol=1e-25, grad_tol=None)
relax> minimise('newton', run='m4', max_iter=1e7)
relax> minimise('newton', run=name, constraints=1, max_iter=1e7)
relax> minimise('newton', run='m4', print_flag=1)

To minimise the model-free run 'm5' using constrained Simplex minimisation with a maximum of
5000 iterations, type:

relax> minimise('simplex', run='m5', constraints=1, max_iter=5000)
```

```
-----------------------------------------------------------------------------------------

Reproduction of the docstring of the generic_minimise function.  Only take note of the
minimisation algorithms and minimisation options sections, the other sections are not
relevant for this function.  The Grid search and Method of Multipliers algorithms cannot be
selected as minimisation algorithms for this function.
-----------------------------------------------------------------------------------------


Generic minimisation function.

This is a generic function which can be used to access all minimisers using the same set of
function arguments.  These are the function tolerance value for convergence tests, the maximum
number of iterations, a flag specifying which data structures should be returned, and a flag
specifying the amount of detail to print to screen.


Keyword Arguments
~~~~~~~~~~~~~~~~~

func:  The function which returns the value.

dfunc:  The function which returns the gradient.

d2func:  The function which returns the Hessian.

args:  The tuple of arguments to supply to the functions func, dfunc, and d2func.

x0:  The vector of initial parameter value estimates (as an array).

min_algor:  A string specifying which minimisation technique to use.

min_options:  A tuple to pass to the minimisation function as the min_options keyword.

func_tol:  The function tolerance value.  Once the function value between iterations decreases
below this value, minimisation is terminated.

grad_tol:  The gradient tolerance value.

maxiter:  The maximum number of iterations.

A:  Linear constraint matrix m*n (A.x >= b).

b:  Linear constraint scalar vector (A.x >= b).

l:  Lower bound constraint vector (l <= x <= u).

u:  Upper bound constraint vector (l <= x <= u).

c:  User supplied constraint function.

dc:  User supplied constraint gradient function.

d2c:  User supplied constraint Hessian function.

full_output:  A flag specifying which data structures should be returned.  The following values
will return, in tuple form, the following data:
    0 - xk
    1 - (xk, fk, k, f_count, g_count, h_count, warning)
where the data names correspond to:
    xk:      The array of minimised parameter values.
    fk:      The minimised function value.
    k:       The number of iterations.
    f_count: The number of function calls.
    g_count: The number of gradient calls.
    h_count: The number of Hessian calls.
    warning: The warning string.

print_flag:  A flag specifying how much information should be printed to standard output during
minimisation.  0 means no output, 1 means minimal output, and values above 1 increase the amount
```

of output printed.

Minimisation algorithms

A minimisation function is selected if the minimisation algorithm argument, which should be a string, matches a certain pattern. Because the python regular expression 'match' statement is used, various strings can be supplied to select the same minimisation algorithm. Below is a list of the minimisation algorithms available together with the corresponding patterns.

This is a short description of python regular expression, for more information, see the regular expression syntax section of the Python Library Reference. Some of the regular expression syntax used in this function is:

    [] - A sequence or set of characters to match to a single character. For example,
    '[Nn]ewton' will match both 'Newton' and 'newton'.

    ^ - Match the start of the string.

    $ - Match the end of the string. For example, '^[Ll][Mm]$' will match 'lm' and 'LM' but
    will not match if characters are placed either before or after these strings.

To select a minimisation algorithm, set the argument to a string which matches the given pattern.

Parameter initialisation methods:

Minimisation algorithm	Patterns
Grid search	'^[Gg]rid'

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]$' or '^[Nn][Cc][Gg]$'

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:

Minimisation algorithm	Patterns

```
| Fletcher-Reeves                   | '^[Ff][Rr]$' or '^[Ff]letcher[-_ ][Rr]eeves$'      |
| Polak-Ribiere                     | '^[Pp][Rr]$' or '^[Pp]olak[-_ ][Rr]ibiere$'        |
| Polak-Ribiere +                   | '^[Pp][Rr]\+$' or '^[Pp]olak[-_ ][Rr]ibiere\+$'    |
| Hestenes-Stiefel                  | '^[Hh][Ss]$' or '^[Hh]estenes[-_ ][Ss]tiefel$'     |
|___________________________________|____________________________________________________|


Miscellaneous unconstrained methods:
__________________________________________________________________________________________
|                                   |                                                    |
| Minimisation algorithm            | Patterns                                           |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Simplex                           | '^[Ss]implex$'                                     |
| Levenberg-Marquardt               | '^[Ll][Mm]$' or '^[Ll]evenburg-[Mm]arquardt$'      |
|___________________________________|____________________________________________________|


Constrained methods:
__________________________________________________________________________________________
|                                   |                                                    |
| Minimisation algorithm            | Patterns                                           |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Method of Multipliers             | '^[Mm][Oo][Mm]$' or '[Mm]ethod of [Mm]ultipliers$' |
|___________________________________|____________________________________________________|



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.
__________________________________________________________________________________________
|                                   |                                                    |
| Line search algorithm             | Patterns                                           |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Backtracking line search          | '^[Bb]ack'                                         |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Nocedal and Wright interpolation  | '^[Nn][Ww][Ii]' or                                 |
| based line search                 | '^[Nn]ocedal[ _][Ww]right[ _][Ii]nt'               |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Nocedal and Wright line search    | '^[Nn][Ww][Ww]' or                                 |
| for the Wolfe conditions          | '^[Nn]ocedal[ _][Ww]right[ _][Ww]olfe'             |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| More and Thuente line search      | '^[Mm][Tt]' or '^[Mm]ore[ _][Tt]huente$'           |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| No line search                    | '^[Nn]one$'                                        |
|___________________________________|____________________________________________________|


Hessian modifications.  These are used in the Newton, Dogleg, and Exact trust region algorithms.
__________________________________________________________________________________________
|                                   |                                                    |
| Hessian modification              | Patterns                                           |
|___________________________________|____________________________________________________|
|                                   |                                                    |
| Unmodified Hessian                | '[Nn]one'                                          |
|___________________________________|____________________________________________________|
|                                   |                                                    |
```

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'

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.

Hessian type	Patterns
Quasi-Newton BFGS	'^[Bb][Ff][Gg][Ss]$'
Newton	'^[Nn]ewton$'

For Newton minimisation, the default line search algorithm is the More and Thuente line search, while the default Hessian modification is the GMW algorithm.

1.11 Function name: ‘nuclei’

Function for setting the gyromagnetic ratio of the heteronucleus.

```
Keyword arguments
~~~~~~~~~~~~~~~~~

heteronuc:  The type of heteronucleus.


Description
~~~~~~~~~~~

The heteronuc argument can be set to the following strings:

    N - Nitrogen, -2.7126e7
    C - Carbon, 2.2e7
```

1.12 Function name: ‘pdb’

The pdb loading function.

```
Keyword Arguments
~~~~~~~~~~~~~~~~~
```

```
run:  The run to assign the structure to.

file:  The name of the PDB file.

dir:  The directory where the file is located.

model:  The PDB model number.

heteronuc:  The name of the heteronucleus as specified in the PDB file.

proton:  The name of the proton as specified in the PDB file.

load_seq:  A flag specifying whether the sequence should be loaded from the PDB file.


Description
~~~~~~~~~~~

To load a specific model from the PDB file, set the model flag to an integer i.  The
structure beginning with the line 'MODEL i' in the PDB file will be loaded.  Otherwise all
structures will be loaded starting from the model number 1.

To load the sequence from the PDB file, set the 'load_seq' flag to 1.  If the sequence has
previously been loaded, then this flag will be ignored.

Once the PDB structures are loaded, unit XH bond vectors will be calculated.  The vectors
are calculated using the atomic coordinates of the atoms specified by the arguments
heteronuc and proton.  If more than one model structure is loaded, the unit XH vectors for
each model will be calculated and the final unit XH vector will be taken as the average.


Example
~~~~~~~

To load all structures from the PDB file 'test.pdb' in the directory '~/pdb' for use in the
model-free analysis run 'm8' where the heteronucleus in the PDB file is 'N' and the proton
is 'H', type:

relax> pdb('m8', 'test.pdb', '~/pdb', 1, 'N', 'H')
relax> pdb(run='m8', file='test.pdb', dir='pdb', model=1, heteronuc='N', proton='H')


To load the 10th model from the file 'test.pdb', use:

relax> pdb('m1', 'test.pdb', model=10)
relax> pdb(run='m1', file='test.pdb', model=10)
```

1.13 Function name: ‘model_selection’

Function for model selection.

```
Keyword arguments
~~~~~~~~~~~~~~~~~

method:  The model selection technique (see below).

modsel_run:  The run name to assign to the results of model selection.

runs:  An array containing the names of all runs to include in model selection.


Description
~~~~~~~~~~~
```

```
The following model selection methods are supported:

AIC:  Akaike's Information Criteria.

AICc:  Small sample size corrected AIC.

BIC:  Bayesian or Schwarz Information Criteria.

Bootstrap:  Bootstrap model selection.

CV:  Single-item-out cross-validation.

Expect:  The expected overall discrepancy (the true values of the parameters are required).

Farrow:  Old model-free method by Farrow et al., 1994.

Palmer:  Old model-free method by Mandel et al., 1995.

Overall:  The realised overall discrepancy (the true values of the parameters are required).

For the methods 'Bootstrap', 'Expect', and 'Overall', the function 'monte_carlo' should have
previously been run with the type argument set to the appropriate value to modify its
behaviour.

If the runs argument is not supplied then all runs currently set or loaded will be used for
model selection, although this could cause problems.


Example
~~~~~~~

For model-free analysis, if the preset models 1 to 5 are minimised and loaded into the
program, the following commands will carry out AIC model selection and assign the results
to the run name 'mixed':

relax> model_selection('AIC', 'mixed')
relax> model_selection(method='AIC', modsel_run='mixed')
relax> model_selection('AIC', 'mixed', ['m1', 'm2', 'm3', 'm4', 'm5'])
relax> model_selection(method='AIC', modsel_run='mixed', runs=['m1', 'm2', 'm3', 'm4', 'm5'])
```