Documentation related to MINOTAUR

Nicolas Bolik-Coulon

February 19, 2024

Contents

1 Introduction

MINOTAUR is a program intended to analyze high-resolution relaxometry data. These datasets are recording using a shuttling device moving the sample up and down inside the bore of the NMR magnets, preventing control of the spin cross-relaxation pathways during the transfers and relaxation delays. These cross-relaxation pathways bias the relaxation decays and prevent the quantitative analysis of the relaxometry relaxation rates. MINOTAUR simulates the evolution of the density matrix during the whole sample trajectory to obtain an accurate description of the dynamics from the relaxometry data along with accurate high-field data (i.e. recording using standard pulse sequences).

2 Key improvements compared to version 1

Here are some improvements compared to version 1:

- when simulating the shuttling periods, MINOTAUR devides the sample trajectory into time-increment that are optimized (see original publication for further details). Since the sample speed for shuttling up or down are potentially different, the initial approach was to generate time lists at which the propagator is calculated for both shuttling times (up and down). The current approach no longer optimizes the increment time but a distance increment. The trajectories are divided into similarly to both shuttling periods, and the delay time between each point is calculated independently, depending on the shuttling speed up and down. Thus, only one set of relaxation matrices are needed for both trajectories. The distance increment is optimized similarly to version 1. The starting distance increment is 1 cm. This lead to a time saving close to a factor 2.

- the shuttling points at which the relaxation matrix is computed are identical for each experiments, only the delay lists are adapted. Thus, we reduce the number of relaxation matrices being calculated to its bare minimum, speeding up the calculations even further.

- MINOTAUR calculations can be started from the command line (detailed below).
- better figure qualities are generated.
- a wider range of file format are accepted (columns separated by tabs, comma and spaces, or combination of these are now accepted).

3 Requirements

MINOTAUR has to be run using Python 3. The following libraries need to be installed: corner, emcee, imageio, numpy, matplotlib, multiprocessing, PIL, scipy, tkinter.

4 Preparing for a run

The Demo folder provides examples of C-files containing expressions of relaxation rates relaxation matrices for two common spin systems: a $^{13}C^{1}H^{2}H_2$ methyl group with a vicinal deuterium, and a N-H spin pair with neighboring protons. The spectral density functions are written using the extended model-free formalism, and explicit models of motions for the $^{13}C^{1}H^{2}H_2$ methyl group. We will refer to the given files in the following as a guide for the users to build their own files for their own systems. The readers are invited to check the associated publications for more theoretical details: Nicolas Bolik-Coulon, Milan Zachrdla, Guillaume Bouvignies, Philippe Pelupessy, Fabien Ferrage, Comprehensive analysis of relaxation decays from high-resolution relaxometry, *J. Magn. Reson.* 355 (2023) 107555.

4.1 C-scripts

In this section, we will detail how to implement the expressions for the relaxation rates and relaxation matrix and make it readable by MINOTAUR. When running the run.py script, the user will be prompted to select for a folder containing the C-scripts. This folder should contain the following files:

- setup.py
- Parameters.py
- Rates.h
- Rates.c
- _Rates.c

setup.py This script will be used to compile the C-functions. The part related to the relaxation matrix:

```
setup(
    ext_modules=[Extension(CurrentDirectory + "/_RelaxMat", [CurrentDirectory + "/_Rates.c", CurrentDirectory + "/Rates.c"])],
    include_dirs=numpy.get_include(),
)
```

should not be changed, but the following lines are related to the additional relaxation rates that will be analyzed in MINOTAUR in addition to the intensity decays. The name of the functions can (have to) be changed to follow the synthax: _*RateName*calculation, where *RateName* is a user-defined name for the considered relaxation rate. For example, for a transverse relaxation rate R_2:

```
setup(
    ext_modules=[Extension(CurrentDirectory + "/Rates/_R2calculation", [CurrentDirectory + "/_Rates.c", CurrentDirectory + "/Rates.c"])],
    include_dirs=numpy.get_include(),
)
```

Parameters.py This script is largely user-based and care has to be taken when editing it.
The table RelaxationRates contains the list of *RateName*. Make sure to use the same names as in the setup.py script, and throughout all the subsequent scripts.
The variable name PositionAuto is the position of the operator of interest in the basis where the relaxation matrix is written. The expectation value $\mathcal{E}$ after the shuttling-delay-shuttling periods will be taken in the density matrix σ as $\mathcal{E} = \sigma[\text{PositionAuto}, \text{PositionAuto}]$.
The dictionary Names contains the variable names that will be determined during the analysis. Make sure to keep the same keys (OrderParam, CorrTimes, others). If one key has no associated variable, simply leave an empty table: []. For example, in the case of the NH group analyzed using the extended model free (two order parameters and two internal correlation times), we have:

```
Names = {'OrderParam': ["Sf2", "Ss2"], 'CorrTimes': ["tau_f", "tau_s"], 'others': []}
```

The variables names do not have to match the ones used in the subsequent scripts, but will be used for labelling purposes.
The ImportFunc function contains the list of the C-functions that are defined (see below). The same nomenclature as in the setup.py script has to be used. We changed this function compared to V1 to return a dictionnary where functions are associated to relaxation rates names as defined in the RelaxationRates table.
The Cons function sets constrains on the variables for the MCMC script. The constrains can be edited as wished by the user, but we recommend using the same bounds for the parameter lnf (see the emcee documentations for further informations about the parameter).

Rates.h This file introduces the functions that will be used. It starts with the relaxation matrix which, in C language is defined as a struct with dimensions which has to be set accordingly. In the example of the $^{13}C^{1}H^{2}H_2$ methyl group, the basis contains only 3 operators, and the relaxation matrix has dimension 3×3such that:

```
struct mat2d {

    double m[3][3];

};
```

Then, the functions defined subsequently have to be declared following the same synthax as used in the given example:

```
double R2calculation(double Bo, double *X, double *Tauc, double *OtherInputs );
```

Rates.c This script is user-based only. It does not necessarily contains definitions of the spectral density functions, as these will not be used in MINOTAUR. It must however contain the definition of the relaxation matrix and all the relaxation rates defined in the setup.py, parameters and Rates.h scripts. We find it clearer to separately declare the functions for the spectral density functions and followed this approach for the N-H spin pair and the explicit models of motions in a $^{13}C^{1}H^{2}H_2$ methyl group. In the case of the extended model-free formalism applied to methyl groups, we used RedKite (see Nicolas Bolik-Coulon, Pavel Kadeřávek, Philippe Pelupessy, Jean-Nicolas Dumez, Fabien Ferrage, Samuel F. Cousin, *J. Magn. Reson.* 313 (2020) 106718) and obtained relaxation rate expressions in C-format using the CForm Mathematica function.

_Rates.c This scripts is meant to transform the python inputs into C-readable inputs for the functions, and transform the C-outputs into python-readable objects. For each of the functions defined in the Rates.h script, the following parts must be present in this script (examples are given for transverse relaxation rates but can be applied to any other rates):

- a docstring: an explanation of what the associated functions does

```
static char R2calculation_docstring[] =
    "Calculates the carbon R2 given a set of parameters (field and dynamics as an
    example).";
```

- the line declaring the PyObject for each relaxation rates and the relaxation matrix:

```c
static PyObject *R2calculation_R2calculation(PyObject *self, PyObject *args);
```

- the definition of the methods by this extension module:

```c
static PyMethodDef module_methods[] = {
    {"RelaxMat", RelaxMat_RelaxMat, METH_VARARGS, RelaxMat_docstring},
    {"R2calculation", R2calculation_R2calculation, METH_VARARGS,
    R2calculation_docstring},
/* any additional rates here */
    {NULL, NULL, 0, NULL}
};
```

- the module initialization:

```c
PyMODINIT_FUNC PyInit__R2calculation(void)
{

    PyObject *module;
    static struct PyModuleDef moduledef = {
        PyModuleDef_HEAD_INIT,
        "_R2calculation",
        module_docstring,
        -1,
        module_methods,
        NULL,
        NULL,
        NULL,
        NULL
    };
    module = PyModule_Create(&moduledef);
    if (!module) return NULL;

    /* Load `numpy` functionality. */
    import_array();

    return module;
}
```

- the definition of the PyObjects functions:

```c
static PyObject *R2calculation_R2calculation(PyObject *self, PyObject *args)
{
    double BVal;
    PyObject *X_obj, *Tauc_obj, *OtherInputs_obj;

    /* Parse the input tuple */
    if (!PyArg_ParseTuple(args, "dOOO", &BVal, &X_obj, &Tauc_obj, &OtherInputs_obj)
    )
        return NULL;

    /* Interpret the input objects as numpy arrays. */
    PyObject *X_array = PyArray_FROM_OTF(X_obj, NPY_DOUBLE, NPY_IN_ARRAY);
    PyObject *Tauc_array = PyArray_FROM_OTF(Tauc_obj, NPY_DOUBLE, NPY_IN_ARRAY);
    PyObject *OtherInputs_array = PyArray_FROM_OTF(OtherInputs_obj, NPY_DOUBLE,
    NPY_IN_ARRAY);

    /* If that didn't work, throw an exception. */
    if (X_array == NULL || Tauc_array == NULL || OtherInputs_array == NULL) {
        Py_XDECREF(X_array);
        Py_XDECREF(Tauc_array);
        Py_XDECREF(OtherInputs_array);
        return NULL;
    }

    /* Get pointers to the data as C-types. */
    double *Xarr     = (double*)PyArray_DATA(X_array);
    double *TaucArr    = (double*)PyArray_DATA(Tauc_array);
    double *OtherInputsArr = (double*)PyArray_DATA(OtherInputs_array);

    /* Call the external C function to compute the rate. */
    double R2val;
    R2val = R2calculation( BVal, Xarr, TaucArr, OtherInputsArr );

    /* Clean up. */
    Py_DECREF(X_array);
    Py_DECREF(Tauc_array);
    Py_DECREF(OtherInputs_array);

    PyObject *ret = Py_BuildValue("d", R2val);
    return ret;
}
```

In all the above items, the ones associated to the relaxation matrix do not have to be changed.

Compilation Compilation is done when starting MINOTAUR. MINOTAUR is started by running on the shell:

```
python run.py
```

which prompts a GUI where the user can select the folder containing the above scripts. Upon clicking the *Launch MINOTAUR* button, C-script will be compiled. One can avoid having the GUI opened by writting:

```
python run.py PATH_Cscript
```

where PATH_Cscript is the path to the C-script folder. Alternatively, if the C-scripts are already compiled, you can directly run MINOTAUR by writting in a terminal:

```
python MINOTAUR.py PATH_Cscript
```

Users load their data in a GUI in MINOTAUR. This step is made easier using the *Load previous parameters* button which allows one to load a parameter file from a previous run (or user-generated) to automatically populate the GUI. If the C-scripts have already been compiled, one can populate the GUI from the command line:

```
python MINOTAUR.py PATH_Cscript PATH_Parameters
```

where PATH_Parameters is the path to the parameter file. The user would still have the option to change parameters in the GUI and press the *Start* button to start calculations. Calculations can be started from the command line:

```
python MINOTAUR.py PATH_Cscript PATH_Parameters --start
```

4.2 Inputs for MINOTAUR

The GUI is devided into different parts:

- the fitting parameters part: the user has to set the global tumbling correlation time, the type of shuttling, the number of steps in the MCMC and the number of chains (see emcee documentations for more information). The default value for the number of chains is twice the number of free parameters (add one to the variables added to the Names dictionary in the Parameters.py script), but this can be increased. In addition, the first 500 steps of the MCMC will be discarded when reporting the median and standard deviations or the first 20 % if less than 500 steps are performed. The user can always calculate the distribution afterwards as the whole trajectories are saved in text files.

- the limits for the initial parameters guess. This part will only be used to generate the initial random parameters. Make sure the given ranges also comply with the Cons function from the Parameters.py script.

- PDB: a 4-letter PBD ID can be provided. It will generate scripts to color each residue in PyMOL according to their parameter values.
- files: this is the part where all the files are loaded. We review below how to format them.

A number of files need to be loaded, and follow the logic explained here. In every files, each column can be separated by a tabs, comma or spaces (or a combination of these):

- field calibration: two-column file, the first one with the height from the center of the magnet, the second being the associated magnetic field. The height has to be given in meters.
- experimental setup: contains one line for each relaxometry experiment. All delays are given in milliseconds. Column 1 is the experiment number. Column 2 is the shuttling height. Column 3 is the delay after shuttling to low field. Column 4 is the delay after shuttling to high field. Column 5 is the waiting time before shuttling to low field. Column 6 is the waiting time after shuttling to low field. Column 7 is the delay before shuttling back to high field. Column 8 is the delay after shuttling back to high field. All the subsequent columns are the relaxation delay times.
- the relaxometry intensity folder: it contains one file for each experiments mentioned in the experimental setup file. Each of these files contains one line for each considered amino acid. The lines contain first the amino acid number, and then the intensity associated to the first relaxation delay, the corresponding experimental error, then the intensity for the second relaxation delay and the error, etc... If some data are missing, replace the values by NA. The accepted file extensions are .txt, .dat and .csv.
- other input file: the C-functions are writen so that they can accept as input the magnetic field, the global tumbling correlation time, the different parameters of the model, and other amino acid-dependent variables contained in this file. It contains N+1 columns for N other parameters, the first column being the amino acid number. In the example of the methyl groups analyzed using the extended model free provided here, the other inputs are the position of the vicinal deuterium, which in effect are set to infinity.
- high-field rate files. There should be one file for each type of rate (which has to be specified in the GUI) and each magnetic field (which also has to be specified, in Tesla, in the GUI). Each file contains 3 columns: the amino acid number, the relaxation rate and its experimental error. Upon loading a file, a 'Add high field rates' button will appear, allowing the user to add all its high-fields datasets. If some data are missing for a residue, replace the values by NA or simply remove it from the file.

Loading all the files can be time-consuming. A 'save current parameters' allows the user to generate a file that is readbale in the GUI by clicking the 'load previous parameters' button and which will automatically fill the GUI as previously saved. Such file is also generated by MINOTAUR after starting a run. This file can also be generated manually.

5 Outputs

Multiple output folders are created. First, all input files are saved, in addition to a Parameters.txt file which can be loaded in a MINOTAUR GUI for subsequent runs. The FitAllResidues folder contains the magnetic-field evolution of back-calculated relaxation rates with experimental data and the intensity decays of each residue. The PlotParameters contains the amino-acid evolution of each parameters determined during the MCMC as well as the resulting χ^2. The FittingResults folder contains:

- the magnetic field profile fit;
- the mean and errors of the parameters after the MCMC run;
- the back-calculated low-field R_1;

- a Trajectories folder containing the complete MCMC trajectories in .txt files, along with figures showing the evolution of each parameters during the MCMC (on these figures, the vertical black line shows the cutt-off limit before which points are discarded when calculating the distributions);
- a Correlations folder containing the 2D correlation map after the MCMC run;
- when a PDB ID is provided (and more than one residue is considered), a PDBFiles folder containing scripts that can be run on PyMOL after loading the structure simply by dragging the file over the PyMOL window.

6 Cite MINOTAUR

When publishing results obtained from a MINOTAUR analysis, please cite:
Nicolas Bolik-Coulon, Milan Zachrdla, Guillaume Bouvignies, Philippe Pelupessy, Fabien Ferrage, Comprehensive analysis of relaxation decays from high-resolution relaxometry, *J. Magn. Reson.* 355 (2023) 107555
If using explicit models of motions, we kindly ask you to cite:
Nicolas Bolik-Coulon, Fabien Ferrage, Explicit models of motions to analyze NMR relaxation data in proteins, *J. Chem. Phys.* 157 (2022) 125102
Nicolas Bolik-Coulon, Olivier Languin-Cattoën, Diego Carnevale, Milan Zachrdla, Damien Laage, Fabio Sterpone, Guillaume Stirnemann, Fabien Ferrage, Explicit Models of Motion to Understand Protein Side-Chain Dynamics, *Phys. Rev. Lett.* 129 (2022), 203001