pyDIFRATE: A generalized approach to timescale analysis of dynamics

Albert A. Smith

1 A brief introduction

The detector analysis provides a straightforward methodology for analyzing data sensitive to correlation times. We rely on a basic assumption, that some correlation function describes the motion, and that correlation function can be assumed to be a sum of an arbitrary number of decaying exponential terms,[1] such that it is given by

$$\begin{aligned} C(t) &= \sum_i A_i \exp\left(-t/(10^{z_i} \cdot 1\ \mathrm{s})\right) \\ &= \int_{-\infty}^{\infty} \theta(z) \exp\left(-t/(10^{z} \cdot 1\ \mathrm{s})\right) dz \\ & \quad z = \log_{10}(\tau_c/\ \mathrm{s}) \end{aligned} \tag{1}$$

The correlation function then may have an arbitrary number of terms, each depending on an amplitude, A_i, and correlation time, $z_i = \log_{10}(\tau_i/\mathrm{s})$, which is given on a logarithmic scale. We generalize this to a distribution, where $\theta(z)$ describes the amplitude at all possible correlation times, although we note that this can be reduced to a simple sum of δ-functions, leading to the discrete sum ($\theta(z) = \Sigma_i A_i \delta(z - z_i)$). Note that depending on the type of correlation function, we may denote the distribution slightly differently, and put requirements on the integral of the distribution. For example, for NMR, we replace $\theta(z)$ with $(1 - S^2)\theta(z)$, where S^2 is the generalized order parameter for the total motion, and $\theta(z)$ must integrate to 1.

Then, if a results from a given experiment, which we will denote ζ, depends linearly on the distribution of motion, we can relate the measured parameter, R_ζ^θ, to the distribution of motion (R is used due to detector's origin in NMR, where relaxation rate constants are commonly used).

$$R_\zeta^\theta = \int_{-\infty}^{\infty} \theta(z) R_\zeta(z) dz \tag{2}$$

$R_\zeta(z)$ is the sensitivity of the experiment ζ, where if we have a distribution of motion with only a single correlation time, z_i, and an integral of 1 for that correlation time, then the relaxation rate constant is given by $R_\zeta(z_i)$. Then, any parameter of this form gives some kind of "window" into the dynamics; its timescale specificity selects out only some of the overall motion. However, we may want a more precise description of motion. Then, suppose we have multiple experiments, in which case we can take any linear combination of parameters as follows:

$$
\begin{aligned}
\rho_n^\theta &= \sum_\zeta a_{n,\zeta} R_\zeta^\theta \\
&= \sum_\zeta a_{n,\zeta} \int_{-\infty}^{\infty} \theta(z) R_\zeta(z) dz \\
&= \int_{-\infty}^{\infty} \underbrace{\sum_\zeta a_{n,\zeta} R_\zeta(z)}_{\rho_n(z)} dz \\
\rho_n^\theta &= \int_{-\infty}^{\infty} \theta(z) \rho_n(z) dz
\end{aligned}
\tag{3}
$$

While we can take any linear combination, the goal of the detector analysis is to obtained optimized linear combinations of the experimental sensitivities, $R_\zeta(z)$, to obtain ideal detector sensitivities, $\rho_n(z)$. Usually this implies a set of sensitivities that have been normalized (either to have a maximum of 1, or an integral of 1), and that have been chosen to minimize overlap of the full set of detectors. We may also optimize a set of detectors to match another set of detectors from a different data set. These functions are critical components of the pyDIFRATE package.

The calculation of detector responses from data is performed using a linear least squares fit, where we minimize

$$
\min \left[\sum_\zeta \sum_n \frac{(R_\zeta^\theta - [\mathbf{r}]_{\zeta,n} \rho_n^\theta)^2}{\sigma(R_\zeta)^2} \right] \tag{4}
$$

r is a matrix that has been optimized to first, fit the experimental data as well as possible for the number of detectors used, and second optimized such that

$$
\rho_n(z) = \sum_\zeta [\mathbf{r}^{-1}]_{n,\zeta} R_\zeta(z). \tag{5}
$$

That is, the linear combination of rate constant sensitivities should yield an optimized set of detector sensitivities.

Then, the R_ζ^θ and the ρ_n^θ are, in a mathematical sense, essentially the same thing– a parameter that is linearly related to the distribution of motion. Then, these parameters will be treated equivalently in pyDIFRATE, and will be stored in “data” objects. Similarly, the sensitivities of the experiment, $R_\zeta(z)$, and sensitivities of the detectors, $\rho_n(z)$, are also treated somewhat equivalently as “sens” (sensitivity) objects. Note that we will have a few subclasses of sensitivity objects, depending on the source of data. Currently we have

detector sensitivities, MD-derived sensitivities, and NMR-derived sensitivities. A data object contains a sensitivity object, which describes the sensitivities of all parameters stored in the data object. A data object also usually contains a detector object, which provides instructions for how to analyze the data using detectors. That is, it contains the sensitivities of the detectors as expected, but also contains the matrix **r**, that allows fitting of data. It will also be capable itself of optimizing sensitivities and finding the corresponding **r** matrix that leads to these sensitivities.

2 Dependencies

Installation required

numpy

scipy

pandas

matplotlib

MDAnalysis

Installation not required, functionality lost if missing

Chimera

ChimeraX

Chimera and ChimeraX are used for 3D visualization of detector responses onto molecules. To function, the program must be installed, and currently, we require manual editing of the chimeraX

Standard library

os

copy

multiprocessing

3 Sensitivity Objects

Currently, there are three types of sensitivity objects implemented in pyDIFRATE. These are the “rates” object, responsible for calculating the sensitivity of rate constants for various NMR experiments, “Ct” objects, which provide the sensitivity for time points extracted from a correlation function, and “detect” objects, which optimize detectors and provide their sensitivity, as well as the matrix **r**, which is used to fit data. Each of these classes is a child class of the “model” parent class.

3.1 Model parent class

pyDIFRATE/r_class/mdl_sens.py : contains the model class

pyDIFRATE/r_class/DynamicModels.py : contains functions for applying specific models

The model parent class currently requires further documentation. We summarize critical information here.

Any sensitivity may be modified by assuming that the total correlation is the product of two correlation functions, one which is known and can be defined as a finite sum of decaying exponential terms, and one which is unknown and will be described by detectors.

$$C(t) = \underbrace{\left[S^2 + (1 - S^2)\sum_n A_n \exp(-t/\tau_n)\right]}_{\text{known}} \cdot \underbrace{\int_{-\infty}^{\infty} \theta(z_i) \exp\left(-t/(10_i^z \cdot 1\ \text{s})\right) dz_i}_{\text{analyze w/ detectors}}. \quad (6)$$

This approach is a generalization of the theory presented in Smith et al., although note that as currently implemented, the known correlation function must have $C(0) = 1$, so that the A_n sum to 1.

Adding and editing models is performed with a number of functions, listed here (details to come).

3.1.1 new_mdl

Creates a new model

3.1.2 del_mdl

Deletes a model by index.

3.1.3 MdlPar

3.1.4 _rho_eff

```
_rho_eff(exp_num=None,mdl_num=0,bond=None)
```

Returns the effective sensitivities of experiments. The model class allows multiple models to be simultaneously defined, so one has to select the desired model. Setting mdl_num to None or -1 will return sensitivities without any model applied. Anisotropy in an applied model may result in a different correlation function depending on the orientation of the tensor in the molecule. Therefore, we must specify the bond desired. If it is not defined, then the sensitivity of the first bond will be returned. Note that if the model is not anisotropic, then we will always use the first bond.

3.1.5 _rho_effCSA

```
_rho_effCSA(self,exp_num=None,mdl_num=0,bond=None)
```

Returns the effective sensitivity to the CSA relaxation. Because the CSA and dipole tensors leading to relaxation of a given nucleus may not be co-linear, anisotropic models of motion can have different influences on the dipole and CSA driven relaxation, so

3.1.6 Requirements for children of the "model" class

All children of the model class must have the following attributes

info: Pandas array, with numbered columns. Each row then contains a different parameter describing each experiment (or detector) in the sensitivity object. For example, the "rate" object contains things like "Type"- the name of the experiment, "v0"- the ^{1}H Larmor frequency, etc. Detector objects have the detector center ("z0"), detector width ("Del_z"), etc. It does not matter what is in info, but info does need to be updated so that there is one column for every experiment in the object.

3.1.6.1 tc

```
tc()
```

Returns the correlation time axis.

3.1.6.2 Z

```
z()
```

Returns the log-correlation time.

3.1.6.3 _rho

```
_rho(self,exp_num=None,bond=None)
```

This is a hidden (but accessible) function, that returns the sensitivity of the experiment. Usually, each type of sensitivity object has a non-hidden function that performs the same operation, but this function is what gets called by functions in the model parent class, and also in the detector class.

3.1.6.4 _rhoCSA

```
_rhoCSA(self,exp_num=None,bond=None)
```

This is a hidden (but accessible) function, that returns the sensitivity of the experiment due to CSA. Necessary for properly treating an anisotropic model with non-co-linear CSA and dipole couplings.

3.1.6.5 retSpinSys, retExper

```
retSpinSys(), retExper()
```

These functions return the variables for an NMR experiment's spinsys and for the experimental parameters. These appear in all children of the model class, although return empty lists except for the rates class, and don't appear to be called anyway. Probably should be removed.

3.2 rates class

pyDIFRATE/r_class/sens.py : contains the rates class
pyDIFRATE/r_class/DIFRATE_funs.py: functions to calculate sensitivity of each experiment

The rates class is responsible for calculating and storing the sensitivities of various NMR experiments. One must load experiments in via .new_exp(), or at initiation of the object (arguments provided when the object is initiated are then passed to .new_exp() at the conclusion of the initiation).

3.2.1 __init__

```
__init__(self,tc=None,**kwargs)
```

Call DR.rates(tc=None,z=None,**kwargs)
Initiates the class. tc or z provide the correlation time axis. tc should be correlation times, z should be $\log_{10}$ correlation times. Can be the full correlation time axis (1D), or 3 elements, giving the start, end, and number of points. **kwargs accepts parameters to be inserted into new_exp()

3.2.2 new_exp

4 List of updates to make

4.1 Project hierarchy

Currently, the whole pyDIFRATE project is constructed into a single module via explicit navigation through the various subfolders. I suspect this should not be the case. Furthermore, dependencies between the parts is managed via a similar changing of directories.

Separate the frames package (eval_fr) from the main pyDIFRATE package. Could be a sub-module, but could also be completely separate. Get rid of 'frame2traj' module (replaced by eval_fr).

4.2 Keyword arguments

I used kwargs heavily at the beginning of this project, and would like to minimize its use, as I find it easier to see possible arguments when typing at the command line. I would maintain kwargs use for plotting functions, where we might pass commands such as color and linestyle to matplotlib. Probably also we need to stick with kwargs for things like loading in experiments for NMR sensitivities, loading models (since these could have lots of different possible variables), and load frames (also have lots of possible variables- although perhaps we could move some of the selection options out of kwargs, because these are fairly standard).

4.3 Copies of data

Variables that are returned with a function should always be copies of the internally stored data. I have not fully understood when a copy occurs by default and when the same object is referenced, but one should be aware that if unusual behavior occurs, there is a possibility that I have failed to make a copy.

4.4 All sensitivity objects

Move input arguments “z” from kwargs into the main set of input variables.

4.5 Model objects

In rho_eff and rho_effCSA, why does setting mdl_num to None or -1 both return no model? Same question for bond, where both None and -1 return all bonds? Seems unnecessary.

Let del_mdl take a list of indices.

Correctly implement orientational averaging when calculating model sensitivities. Currently, we calculate the average vector direction, but we ought to use the average tensor.

Implement a calculation of the CSA direction for protein backbone 15N relaxation. Add a generalized method for defining the CSA direction.

Eliminate the retSpinSys and retExper functions from model children, unless they are actually required somewhere.

4.6 Molecule objects

4.7 Chimera

I would like to eliminate all usage of chimera, and replace entirely with chimeraX. Currently, correlation plotting and simple opening of a molecule (no data plotted) relies on chimera.

I would like to find the ChimeraX installation automatically. No idea how this would work….

A “cleanup” function should be implemented, which would delete all chimera scripts that have not been deleted due to program interruption (also remove attribute files? I think this isn’t necessary– their usage has been made obsolete with the updates to ChimeraX).

4.8 eval_fr

Update to include processing for symmetric inner motion.

5 References

(1) Smith, A. A.; Ernst, M.; Meier, B. H. Optimized “Detectors” for Dynamics Analysis in Solid-State NMR. *J. Chem. Phys.*, 2018, *148*, 045104. https://doi.org/10.1063/1.5013316.