pyDIFRATE:

Dynamics detectors analysis of NMR relaxation and MD simulation

Albert A. Smith

Table of Contents

1. Introduction 5

1.1 What is pyDIFRATE 5

1.2 Typical workflow 6

1.3 Some notes on object-oriented programming 6

What’s an object and a class? 6

Why use classes in pyDIFRATE? 6

2. data class 6

Variables 6

a. R 6

b. Rstd 7

Functions 7

a. load\_ 7

b. plot\_rho 7

c. plot\_fit 7

d. plot\_cc 7

e. draw\_rho3D 7

f. draw\_cc3D 7

3. Sensitivity classes 7

3.1 Experimental sensitivities (‘rates’ class) 7

Variables 7

a. 7

Functions 7

a. new\_exp 7

b. set\_par 9

3.2 MD sensitivities (‘Ctsens’) 9

4. Detector class 9

5. Models (the model superclass) 9

6. Funtions 9

6.1 NucInfo 9

7. References 10

Glossary of terms

Table 1. Detector terminology and symbols

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Symbol** | **Units** | **Description** |
| *Correlation time* |  | s | Correlation time of some motion in the system. |
| *Log-correlation time* | *z* | unitless (vs. 1 s) | Base-10 logarithm of the correlation time, given by . |
| *Rotational diffusion correlation time* |  | s | Correlation time of isotropic rotational of a molecule in solution (tumbling). |
| *Log-* | *zr* | unitless (vs. 1 s) | Base-10 logarithm of the rotational correlation time, given by |
| *Effective correlation time* |  | s | Effective correlation of an internal motion, where the molecule is undergoing tumbling with correlation time,. Given by . |
| *Log-effective correlation time* |  | unitless (vs. 1 s) | Base-10 logarithm of the effective correlation time, given by . |
| *Distribution of motion* |  | unitless | Describes how motion is distributed as a function of correlation time, where *z* = log10(*τ*c /1 s). (1–*S*2) gives the total amplitude of motion, so that *θ*(*z*) always integrates to one. |
| *Distribution of internal motion* |  | unitless | This is the same as the distribution of motion for solid-state analysis. In solution-state analysis, this distribution only accounts for internal motion of the molecule- in other words, tumbling of the molecule is factored out, and the log-correlation times are *not* effective correlation times. |
| *Distribution of total motion* |  | unitless | This is the distribution of all motions for a molecule tumbling in solution, including the tumbling itself. Motion resulting from internal motion is modified to have an effective correlation time, *zeff*, which results from the internal correlation time and the tumbling correlation time. |
| *Relaxation rate constant* |  | s-1 | The relaxation-rate constant obtained under experimental conditions denoted by *ζ*, for a distribution of motion (1–*S*2)*θ*(*z*). May be obtained by integrating the product of the sensitivity of that rate constant, *Rζ*(*z*), times the distribution of motion, (1–*S*2)*θ*(*z*). |
| *Sensitivity* |  | s-1 | The relaxation rate constant obtained under experimental conditions denoted by *ζ*, for a mono-exponential correlation function, having correlation time *τ*c = 10z s, and amplitude 1–*S*2 = 1. |
| *Solution-state sensitivity* |  | s-1 | Sensitivity of an experiment to the internal motion of a molecule, with , when the molecule is tumbling in solution. This function has one term to account for attenuation of relaxation due to rotational diffusion, and a second term to account for relaxation induced by the internal motion, given as *.* |
| *Detector* | – | – | A mathematical tool used to quantify the amount of motion for a range of correlation times. |
| *Detector sensitivity* |  | unitless | Defines how a detector responds to a particular correlation time, *τ*c = 10z s. Its value as a function of *z* is obtained by taking a linear combination of rate constant sensitivities (using the same linear combination as is used to obtain the detector responses). |
| *Detector response* |  | unitless | A quantity, describing the amount of motion for a particular range of correlation times, rigorously defined as the integral of the product of the detector sensitivity, *ρn*(*z*), and the distribution of motion, (1–*S*2)*θ*(*z*). Obtained by taking an appropriate linear combination of experimental rate constants (strictly speaking, by fitting a vector of the rate constants to the detection vectors, ). |
| *Normalized rate constant* |  | unitless | The relaxation rate constant divided by some normalization constant, *cζ*, to yield a dimensionless relaxation rate constant. For solution state relaxation, we first subtract away the relaxation rate constant obtained for an internally rigid motion, , such that . |
| *Allowed region* | – | – | For a given set of experiments, the allowed region is all sets of rate constants () that can be obtained for any arbitrary distribution of motion, given by (1–*S*2)*θ*(*z*). Usually this space is represented in terms of the . |
| *Detection vector* |  | s-1 | A vector containing carefully chosen values of the , so that a vector containing the full set of experimentally determined relaxation rate constants is assumed to be a linear combination of all detection vectors, given by . |
| *Sum of normalized rate constants* |  | unitless | Sum of all normalized rate constants for an experimental data set, used for calculating the ratio of rates. Note that for the reduced space for internal motion (solution-state), this term is replaced, often by , where the corresponding sensitivity, , remains negative for all correlation times (see main text, Eq. (22)). |
| *Ratio of rates* |  | unitless | For experimental conditions denoted by *ζ*, this is the ratio of the normalized rate constants, , divided by the sum of normalized rate constants, , which is used for defining positions in the reduced space. |
| *Reduced space* | – | – | For a set of experiments, the reduced space is defined by the ratios of rates, *κζ*, for that set of experiments. The dimensionality of this space is one less than the number of experiments- achieved by omitting one of the experiments when calculating the *κζ*. |
| *Reduced vector* |  | unitless | Vector of ratios of rates, *κζ*, defining a position in the reduced space. These positions can be used to define detection vectors, although note that the reduced vector only defines the direction of the detection vector, but not the length. |
| *Effective width* |  | unitless  (vs. 1 s) | The effective width of a detector is defined as the detector integral divided by its maximum, given on a base-10 log scale. |
| *Detector center* |  | unitless  (vs. 1 s) | This gives the center of the detector sensitivity, on a logarithmic scale (unitless, with reference to 1 s using a base-10 log). Defined as follows: |

Table 2. Manual terminology

|  |  |
| --- | --- |
| **Name** | **Description** |
| *Vector* | DIFRATE characterizes reorientational motion of vectors. Usually, these are bonds, but may also be other vectors (such as the direction of 15N chemical shift anisotropy, which is not exactly co-linear with the H–N bond). |

# Introduction

## What is pyDIFRATE

pyDIFRATE is the Python implementation of the “detectors” method of dynamics data analysis. DIFRATE stands for Disstortion Free Relaxation Analysis Technique, and was introduced to analyze NMR relaxation data with minimal assumptions. In their most basic form, detectors do not introduce any bias to the data analysis, by applying models to the data. pyDIFRATE goes well beyond the basic detectors approach, and re-introduces explicit models of motion, to be used in conjunction with detectors. In this case, it is unlikely to be distortion free, but we anyway continue to use the original name.

## What are detectors

Detectors describe the amount of motion on a given bond (more generally, a vector), for a particular range of correlation times. This relies on the assumption that a correlation function of motion for a particular vector in the molecule is given by a sum of decaying exponential terms. For example, for a unit vector, with time dependence , the correlation function for a rank-2 tensor is

|  |  |
| --- | --- |
|  | (1) |

Contributions to the overall correlation function are then characterized by a correlation time, , and an amplitude, . If we know the distribution of motion, , then we know exactly how much motion can be found at any given correlation time. Knowing the correlation time and amplitude is useful for a number of reasons: first, correlation time is connected to the free energy of a motion. Faster motions are usually more local (less is moving, so costs less), or slower motions may result from the motion being hindered by some large energy barrier. Additionally, motions happening simultaneously need to have a correlation time in common. Finally, every individual bond may experience many motions, making it difficult to separate motion based on the overall amplitude alone, whereas additional separation is possible by timescale.

Unfortunately, it is rarely possible to obtain the full distribution of motion. Therefore, we use detectors, which sample a part of the correlation function. A detector response is defined as:

|  |  |
| --- | --- |
|  | (2) |

 is the detector sensitivity of the *n*th detector (usually we obtain several detectors). Ideally, the sensitivity is relatively narrow, non-negative function, so that we may interpret the detector response as a measure of the motion around the detector center (). While less precise than knowing the full distribution, the a set of detectors is often the maximum information we may obtain about the overall correlation function, without making assumptions about the specific model of motion.””

Detectors have a number of advantages for analyzing dynamics data, over explicit models. First, much of the dynamics data that we are able to obtain can be recast in the same mathematical form as a detector. For example, an NMR relaxation rate constant may be calculated from the distribution of motion as

|  |  |
| --- | --- |
| . | (3) |

Here,  is a relaxation rate constant, for some experiment, *ζ*, where  is the sensitivity of that experiment (obtained by calculating the relaxation rate constant for a motion with  and ). Similarly, the amplitude of a correlation function is given by

|  |  |
| --- | --- |
|  | (4) |

Here,  is the sensitivity of the time point in the correlation function corresponding to *t*.

Then, experimental rate constants, points in a correlation function (MD derived), and detector responses may all be handled the same way. pyDIFRATE recognizes this and treats all simply as “data”, and keeps track of the sensitivity of each experiment/time point/simulation. Given data, pyDIFRATE takes the set of sensitivities provided, and finds an optimized linear combination of those sensitivities to yield detector sensitivities. Note how this works: suppose we have two experiments, *ζ* and *ξ*. We may take a linear combination of the experimental rate constants:

|  |  |
| --- | --- |
|  | (5) |

Part of the functionality of pyDIFRATE is to find values of *a* and *b*, which give ideal detector sensitivities. This works for very large data sets as well, and linear combinations are first optimized to efficiently use the data to yield high signal-to-noise for the detector response, and second to be narrow (or to satisfy some target functions provided by the user). Note the user must request the number of detectors to be extracted from the data (more reduces signal-to-noise but increases resolution).

While detectors may be used alone for analysis and comparison of dynamics between methods, we may also apply specific models of motion. These models separate that particular model from the rest of the distribution of motion, such that motion that is not explained by the model is still described by a detector response (or rate constant, etc.). In this case, one assumes

|  |  |
| --- | --- |
|  | (6) |

where the overall correlation function is the product of a “model” correlation function and some other correlation function, the former of which we know the form of, and the latter of which we still characterize with detectors (*pyDIFRATE does not currently fit the model correlation function, but this is a functionality that I want to add later)*. Then, the model correlation function can always be described by a sum of decaying exponentials (here discretized, note that the *Ai* and  should always add to 1), and the other correlation function is still described by a distribution.

|  |  |
| --- | --- |
|  | (7) |

Multiplying this through results in

|  |  |
| --- | --- |
|  | (8) |

Then, for a given rate constant (or detector response, or correlation function time point), the measured rate constant is given by

|  |  |
| --- | --- |
|  | (9) |

Then, the rate constant also has a sensitivity to the distribution of motion, even in the presence of the model, here referred to as  (note we must first subtract away  from  to have exactly the same relationship). This also follows for detectors and correlation functions.

Then, we see that each data type (correlation function, rate constant, detector) are mathematically the same, with the only different the shape of the sensitivity functions. Furthermore, we may apply models to each data type, and we obtain a new data type, which maintains the relationship between data, sensitivity, and distribution of motion.

It follows that pyDIFRATE is constructed to take advantage of these similarities. Sensitivity “classes” (see below) exist for NMR relaxation rate constants, MD-derived correlation functions, and detectors. A set of detectors may be generated from any of these classes (including the detector class itself). Furthermore, each of these class is a child of a “model” superclass, which allows calculation of effective sensitivities for a given model (and subsequently generate detectors from the effective sensitivities).

## Typical workflow

## Some notes on object-oriented programming

pyDIFRATE (unlike its MATLAB-based predecessor, DIFRATE) uses objects for data storage and processing. This has considerable advantages, but can be confusing for users unfamiliar with the concepts of objects and classes. We briefly discuss objects/classes, and why they are useful for pyDIFRATE.

* + - 1. What’s an object and a class?

An object is a container for both variables and code. One is more often familiar with having variables and code separately, for example, we might have experimental data in data, an optimized set of detectors defined in r, and a function for fitting data as fit\_data. Then, the following operation creates a new variable, fit, by fitting data.

fit=DR.fit(data,r)

On the other hand, if data is an object, then it may contain the experimental data, the optimized set of detectors, r, and an internal function (that is, a “method”) for fitting. Then, a fit is obtained by calling the method ‘fit’.

fit=data.fit()

No arguments are needed for the call to data.fit(), because all the required information to perform fitting is already contained in data. Another seemingly unusual usage of objects is the call to a function for which no result is returned. For example, before we fit the data contained in data, we have to optimize a set of detectors to fit that data. This is done by running

data.detect.r\_auto(4)

This is a function that automatically generates *N* detectors (here *N*=4) out of the given data set. No variable is returned, because this function edits information stored inside data.detect. We can inspect the result of this optimization in a number of ways, however.

r=data.detect.r()

will return the optimized ‘r’ matrix (the detection vectors, ),

data.detect.plot\_rho\_z()

will create a plot of the detector sensitivities, , and

rho\_z=data.detect.rhoz()

will return a matrix containing the same detector sensitivities.

That’s the basic idea of the difference of a normal variable and an object. Note that one may have many “objects” that function the same way, but have different data in them. These objects are different “instances” of the same class. We’ll talk about the different classes in the manual (data, sensitivity, and detector classes, in addition to the model superclass). Then, class is just referring to the piece of code that creates the objects.

* + - 1. Why use classes in pyDIFRATE?

Although powerful, the behavior of objects may be rather confusing, so why add this complexity to DIFRATE? The short answer is that some usages of detectors require very careful bookkeeping of many factors. For example, suppose we want to analyze dynamics of a molecule that tumbles with a large anisotropy in solution. A file, ‘data\_info.txt’ contains both the experimental results for backbone 15N relaxation and a description of those experiments. We first load the data and experimental information:

data=DR.io.load\_nmr(‘data\_info.txt’)

We can import a structure (‘struct.pdb’)

data.sens.molecule.load\_struct(‘struct.pdb’)

data.sens.molecule.select\_atoms(Nuc=‘15N’)

data.sens.molecule.set\_selection()

This loads the pdb file, tells it that we have data for dynamics of protein backbone 15N dynamics, and sets the selection (more complex bond selections may require several commands, so we finalize the selection with a separate function, set\_selection()). Next we introduce the model of tumbling:

data.sens.new\_mdl(Model=’AnisoDif’,tM=2e-9,xi=3,euler=[0.773,1.576,0])

data.new\_detect(mdl\_num=0)

We introduce the model of tumbling (anisotropic diffusion, in this case with mean correlation time, anisotropy, and euler angles). The resulting experimental sensitivities for this model are transferred to the detector object in the second line (multiple models may be used, so we have to specify which one with mdl\_num=0). Note that under most cases, the internal detector object (data.detect) should be generated from the internal sensitivity object (data.sens), to maintain consistency of the data, as done here. Finally, we optimize a set of detectors, and fit the resulting data.

data.detect.r\_auto(4,bond=-1)

fit=data.fit()

While objects are not completely necessary to achieve this, by keeping data and sensitivities together, we have to worry less about correct processing of the data even in complex cases such as anisotropic tumbling.

* + - 1. Other peculiarities of objects

An object has “attributes”. These are either functions or variables, accessed usually by typing something like:

R=rates.R()

This returns a variable, R, which contains the sensitivities of the experiments given in the NMR rate sensitivity object (rates). However, the () indicate that we access a function, not a variable in rates. If we were to try to access the data directly, an error would be returned

R=rates.R[:]

There are a couple reasons for doing this. First, the program does not calculate

# data class

*File: data/data\_class.py*

*@ DR.data*

The data class’s primary purpose is to store and present experimental (and simulated) data. As a class, it can also contain “sens” (section 3) and “detect” (section 4) objects, which should contain information on the sensitivity of the experiments stored in the data, and detectors that may be used to analyze that data. The results of data analysis are also stored in the data class (in a new instance), and has all the same functionality of the original data (plotting, further modeling, analysis, etc.).

* + - 1. Variables
         1. R

‘data.R’ stores the data. Each “experiment” is a new column, and each vector a new row.

* + - * 1. Rstd
      1. Functions
         1. load\_
         2. plot\_rho
         3. plot\_fit
         4. plot\_cc
         5. draw\_rho3D
         6. draw\_cc3D

Draws data onto a molecule (stored in data.molecule

# Sensitivity classes

## Experimental sensitivities (‘rates’ class)

*File: r\_class/sens.py*

*@ DR.rates*

* + - 1. Variables
      2. Functions
         1. new\_exp

This function creates a new NMR experiment or experiments for the sensitivity. One must provide parameters required to calculate the sensitivity of the experiment. This includes experimental parameters and parameters describing the relative spin system (parameters may also later be edited with the set\_par function). When calling new\_exp, we may define multiple experiments by inputting multiple experimental parameters. However, all experiments defined in a single call to new\_exp must have the same spin system. Experiments with different spin systems may be defined in a single ‘rates’ object, but these can only be produced with multiple ‘new\_exp’ calls.

For example, let’s suppose we have measured *R*1, *R*2, and NOE type relaxation experiments at 2 fields (400, 700 MHz). We can define all these experiments simultaneously as:

rates.new\_exp(Nuc=’15N’,Type=[‘R1’,’R2’,’NOE’],v0=[400,400,400,700,700,700])

The number of experiments is defined by the parameter with the most entries, in this case, v0, with 6 entries. The other entries are then repeated to match the number of experiments. Here, this means Type is repeated once, so we get *R*1, *R*2, and NOE at both 400 and 700. Note, one can load the same experiments, albeit in a different order, with

rates.new\_exp(Nuc=’15N’,Type=[‘R1’,’R1’,’R2’,’R2’,’NOE’,’NOE’],v0=[400,700])

Note that here, the spin system is being defined with several defaults (see Table 4). In some cases, we must manually specify the full spin system. For example, suppose we have relaxation of a lipid, and are interested in dynamics of H–C bonds in the middle of a chain. Then, we use *R*1 relaxation at a number of fields:

rates.new\_exp(Nuc=’13C’,Nuc1=[‘1H’,’1H’],dXY=[43E3,43E3],Type=’R1’,v0=[400,700])

The 13C is bonded to two protons, so we have to include both of these in the spin system. This is why there are two entries for Nuc1 and dXY. To allow this functionality, we cannot allow defining different spin systems with a single call to new\_exp, because it would not be possible to distinguish if Nuc1 and dXY have two entries because the 13C is bonded to two protons, or because there are two different *R*1 experiments.

We may still have multiple spin systems in a ‘rates’ object, we simply make multiple calls to new\_exp. For example, we may also have 2H relaxation data for the same bond of the lipid from which we had 13C data. After the above call to new\_exp, we could also call

rates.new\_exp(Nuc=’2H’,Type=’R1’,v0=700)

Table 3. Spin system parameters in pyDIFRATE

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Unit (required?)** | **Description** | **Notes** |
| v0 | MHz (yes) | 1H Larmor frequency for the given magnetic field | – |
| Nuc | String (yes) | Nucleus undergoing relaxation | Some strings will set default values for other parameters, see Table 4. |

Table 4. Spin system defaults

|  |  |
| --- | --- |
| **\*Nuc** | **†Defaults** |
| ‘15N’,’N15’,’N’ | Nuc1=’1H’  dXY=-22954.8  CSA=113.0 |
| ‘13Ca’,’Ca’ | Nuc1=’1H’  dXY= 44782.0  CSA=20.0 |
| ‘13CO’,’CO’ | CSA=155 |
| ‘CD2H’,’CHD2’,’13CD2H’,’13CHD2’ | Nuc1=[’1H’,’2H’,’2H’]  dXY= [43587.9, 6781.8, 6781.8]  CSA=16.67 |
| ‘2H’,’D’ | QC=60104 |
| ‘H217O’ | QC=8.2E6 |

\*Nuclear strings are not case sensitive

**†**Unlisted parameters are zero or empty

Table 5. Parameters for NMR experiments in pyDIFRATE

|  |  |  |
| --- | --- | --- |
| **Experiment** | **Exp. Pars** | **Description** |
| All experiments | v0 / MHz: | 1H Larmor frequency for the given magnetic field |
| R­1 (Type=R1) | ­­– | Average (over all orientations) of the *R*1 relaxation rate constant. |
| R1ρ  (Type=’R1p’) | v1 / kHz | Relaxation under a spin-lock |

* + - * 1. set\_par

Allows editing of an experiment

## MD sensitivities (‘Ctsens’)

*File: r\_class/Ctsens.py*

*@ DR.Ct*

# Detector class

*File: r\_class/detectors.py*

*@ DR.detect*

# Models (the model superclass)

*File: r\_class/mdl\_sens.py*

*@ DR.detect, DR.Ct, DR.sens*

# Molecule class (structures)

*File: Struct/structure.py*

*@ DR.molecule*

The molecule class is usually a subclass of the various sensitivity classes and the detector class, that is, all child classes of the models class (although it may be initialized as its own object, and later inserted into the “molecule” attribute of another object).

*Functions*

* + - * 1. select\_atoms

Options for selecting atoms to define bonds in the molecule. One needs to define pairs of atoms (that will be stored in molecule.sel1 and molecule.sel2. Note that in case one needs to define multiple bonds to the same atom, one must also define molecule.sel1in and molecule.sel2in

sel1/sel2: use [MDAnalysis selection strings](https://www.mdanalysis.org/docs/documentation_pages/selections.html) to define selection 1 or selection 2

sel1in/sel2in: Use indices

index1/index2: Filter sel1/sel2 by index

resi: Select only residues stored in this list

select: Apply additional selection criteria (this criteria *and* the other criteria must be satisfied. For sake of indexing, this criteria is applied first)

Nuc: Select atom pairs by keyword (index1/index2 not applied to this selection method). Example is ‘N’, which will select the backbone N and HN (currently, CO for backbone C’ is implemented, as well as

* + - * 1. Variables

# Functions

## NucInfo

*File: r\_class/DIFRATE\_funs.py (NucInfo)*

*@ DR.funs.NucInfo*

Returns information about a given nucleus. Call as:

value=DR.funs.NucInfo(Nuc\_str,info\_str)

Nuc\_str is a string specifying which nucleus we should return information about. Usually, one should provide the mass number and chemical symbol (for example, ‘15N’ or ‘N15’ for a Nitrogen-15 nucleus). However, if only the symbol is provided, this function will preferentially yield data for spin-1/2 nuclei. If no spin-1/2 nuclei exist, then it will yield non-zero spin nuclei. After selecting for spin-1/2 nuclei, or non-zero spin nuclei, priority will be given based on natural abundance. For example, requesting ‘C’ returns data for 13C, since this is the only spin-1/2 nucleus for carbon. ‘O’ will return 17O, since this is the only non-zero spin for oxygen. ‘K’ will return 19K, because all potassium isotopes have non-zero spin, but none are spin-1/2. Then, 19K is the most abundant nucleus. Note that ‘D’ will return information for 2H.

info\_str is a string specifying what information should be provided. Options are ‘gyro’, the gyromagnetic ratio in Hz/T (if info\_str is not provided, this is the default), ‘spin’, which provides the spin, ‘abund’, which provides the natural abundance in %, ‘mass’, which provides the mass number, and ‘allinfo’ finally which provides a Pandas array with all information for the requested nucleus.

NucInfo may also be called without any arguments, which will return a Pandas array with all information on all nuclei.

Table 6. Behavior of sensitivity functions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | Class | bond=  -1 | bond  =None | exp\_num  =None | mdl\_num  =None | Notes |
| .\_rho | sens | 2 | 2 |  |  |  |

Options:

We need setting bond = -1 to return all bonds for the .\_rho functions, because setting bond = None returns the average in the detectors class. This creates the problem that for non-bond specific functions, we don’t return the same dimension of matrix (for example, as is always return by .\_rho in the rates class.

# References