# **Chapter 1 – Introduction.**

**1. What is RAT.**

For any calculation problem, the process of interacting with our data …… blah blah data analysis really is basically about designing cats

**1.2 RAT basic modelling principle**

Even when we are at the calculations point our data analysis – that the data has been properly reduced and we have some idea of a modelling strategy, we can still then split the problem specification into two parts.

* The **Model Definition** describes our data, our model, the various parameters we may have along with their limits or priors, and other quantities such as bulk SLD’s or backgrounds.
* The **Controls Block** summarises the actions which we want to do with our model. So it specifies which algorithm we might want to apply to improve our model fit, along with algorithm parameters such as maximum function evaluations or gradients etc.

Keeping these two separate gives a high degree of flexibility as to how we can approach a data analysis problem. For example, suppose we have defined a Model Definition block which contains the data and model details for a given fitting problem, we then decide that we would like to optimise this first using a genetic algorithm, and then run a Bayesian analysis to obtain the parameter posterior distributions. To do all this we only need to define our model once, and run it twice whilst simply modifying the Controls Block to tweak the algorithm that is run in each case. Also, once we are satisfied with our model, we can save it or export it, and re-use this basic pattern for subsequent analyses with different, related data sets. Keeping the Model and Controls separate gives a high degree of flexibility as to how tasks can be formulated and run.

So, an input into RAT always conforms to this picture: a model definition class to specify the problem, and a controls definition class that tells RAT what analysis task you would like to do:

problemDef Class

controlsDef Class

RAT Toolbox

Output problemDef Class

Results

Block

The outputs are always another problemDef class, and a results block. The new problemDef class is identical to the inputted one, except with updated values of the fitted parameters (e.g. after running a fit). The results block as a set of arrays containing the results of the calculations, such as simulated reflectivities, SLD profiles or parameter distributions. We will discuss more about these in the next chapter where we look at the outputs in more detail. Similarly, we will look in more detail at the input classes in chapter 2.

In the next section, we will look at an example calculation, in order to introduce the basics of the RAT toolbox. Before we proceed, it’s useful to keep a couple of things in mind:

* *Coding language: It’s in Matlab* – Well, yes and no! The RAT toolbox is written completely in C++ (including calling of custom model scripts), and so in principle can be called from any language. So for each case, we have the input classes and scripts (Matlab in our example), which are then parsed into a form that C++ understands (’mex’ interface for Matlab). The C++ carries out the calculation, and then the reverse happens, where the output from the C++ calculation is converted back to a Matlab format. This collection of input classes and routines for communicating between Matlab and the underlying C++ are called the ‘Matlab API’. This manual details the use of the Matlab API to use RAT. However, the scheme in figure 1.2 could just as easily be for any language. So, some Python versions of the problemDef classes, along with some interface routines to communicate between Python and C++ would constitute a Python API for this toolbox. We are planning to provide four choices of API for RAT: Matlab, Octave, Python and Java. For the first release, only the Matlab API is currently available.
* *What about different model types?* - There are many possible types of model, and this is done by having more than one version of the problemDef class. These are (those in red not initial release):
* *Standard Problem:* Problem types which are well described by a non-polarised beam, with no absorption (i.e., real refractive index only).
* *Standard Problem with Absorption* –
* *Domains Problem Type* –
* *Oil/Water problem type* –
* *Polarised problem type -*

Within each problem class, there is the option of ‘Custom Layer’ or ‘Custom XY’ model definitions. These will be discussed in more detail in chapter 3.

**1.3. An example – A simple model of a lipid layer**

*(a) Specifying the Model.*

In the next chapter, we'll look in detail how to set up the problem definition for a given situation. Initially though, it's useful to take a pre-prepared problem definition, and to see how this is then used in RAT. As an example, we'll use some neutron reflectivity data for a lipid monolayer, collected at various deuterations, which we want to analyse simultaneously.

In terms of reflectivity, the interface we want to model (i.e. a monolayer at an air-water interface) is usually well modeled by two layers: the hydrophobic tail regions of the lipids, which locate outside the bulk water interface, and the hydrophilic heads wich are adjacent (or embedded) in the bulk aqueous phase.

In our example, the layers can be either deuterated of hydrogenated, and the bulk water can either be D2O or ACMW.

Shape

Description automatically generated with low confidence

We are going to analyze our monolayer data using a RasCAL type ‘standard layers’ model, in that we identify which parameters we need to describe the model, group these into layers (which are defined as a thickness, roughness, SLD and hydration), and then group the layers along with data into contrasts. The advantage of this approach is that it's simple to share parameters between layers, so a layer representing deuterated headgroups should share the same thickness and roughness parameters as a layer representing hydrogenated heads, but they should differ from each other in their SLD.

The problem definition in RAT is done by making an instance of a ‘projectClass’ object, and then using the class methods to set up the parameters, layers and so on on this instance. The details of setting up a projectClass object is discussed in chapter 2, but for now, we'll look at a pre-prepared example.

% Load in a pre-made problem Definition class

problem = load('twoContrastExample.mat');

problem = problem.problem;

% Have a look at what we have

disp(problem)

ModelType: 'Standard Layers'

experimentName: 'DSPC monolayers'

Geometry: 'air/substrate'

Parameters: ----------------------------------------------------------------------------------------------

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_** **\_\_\_\_\_**

1 "Substrate Roughness" 2 3 13 true

2 "Tails Thickness" 10 20 30 true

3 "Heads Thickness" 3 11 16 true

4 "Tails Roughness" 2 5 9 true

5 "Heads Roughness" 2 5 9 true

6 "Deuterated Tails SLD" 4e-06 6e-06 2e-05 true

7 "Hydrogenated Tails SLD" -6e-07 -4e-07 0 true

8 "Deuterated Heads SLD" 1e-06 3e-06 8e-06 true

9 "Hydrogenated Heads SLD" 1e-07 1.4e-06 3e-06 true

10 "Heads Hydration" 0 0.3 0.5 true

Layers: --------------------------------------------------------------------------------------------------

**p** **Name** **Thickness** **SLD** **Roughness** **Hydration** **Hydrate with**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_**

1 "Hydrogenated Heads" "Heads Thickness" "Hydrogenated Heads SLD" "Heads Roughness" "Heads Hydration" "bulk out"

2 "Deuterated Heads" "Heads Thickness" "Deuterated Heads SLD" "Heads Roughness" "Heads Hydration" "bulk out"

3 "Hydrogenated Tails" "Tails Thickness" "Hydrogenated Tails SLD" "Tails Roughness" <missing> "bulk in"

4 "Deuterated Tails" "Tails Thickness" "Deuterated Tails SLD" "Tails Roughness" <missing> "bulk in"

Bulk In: --------------------------------------------------------------------------------------------------

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_** **\_\_\_\_\_\_\_\_\_** **\_\_\_** **\_\_\_\_\_** **\_\_\_** **\_\_\_\_\_**

1 "SLD Air" 0 0 0 false

Bulk Out: -------------------------------------------------------------------------------------------------

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_** **\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_** **\_\_\_\_\_**

1 "SLD D2O" 6.2e-06 6.35e-06 6.35e-06 false

2 "SLD ACMW" -1e-06 0 1e-06 true

Scalefactors: -------------------------------------------------------------------------------------------------

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_** **\_\_\_\_\_** **\_\_\_\_** **\_\_\_\_\_**

1 "Scalefactor 1" 0.02 0.23 0.25 true

Backgrounds: -----------------------------------------------------------------------------------------------

(a) Background Parameters:

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_** **\_\_\_\_\_**

1 "Backs value ACMW" 1e-07 5.5e-06 1e-05 true

2 "Backs Value D2O" 1e-08 2.8e-06 1e-05 true

(b) Backgrounds:

**p** **Name** **Type** **Value 1** **Value 2** **Value 3** **Value 4** **Value 5**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_**

1 "Background ACMW" "constant" "Backs Value ACMW" "" "" "" ""

2 "Background D2O" "constant" "Backs Value D2O" "" "" "" ""

Resolutions: ---------------------------------------------------------------------------------------------

(a) Resolutions Parameters:

**p** **Name** **Min** **Value** **Max** **Fit?**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_** **\_\_\_\_\_** **\_\_\_\_** **\_\_\_\_\_**

1 "Resolution par 1" 0.01 0.03 0.05 false

(b) Resolutions:

**p** **Name** **Type** **Value 1** **Value 2** **Value 3** **Value 4** **Value 5**

**\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_**

1 "Resolution 1" "gaussian" "Resolution par 1" "" "" "" ""

Data: ------------------------------------------------------------------------------------------------------

**Name** **Data** **Data Range** **Simulation Range**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

"Simulation" "No Data" "-" "[ 0.0050 , 0.7000 ]"

"H-tail / D-head / ACMW" "Data array: [51 x 3]" "[ 0.0518 , 0.5888 ]" "[ 0.0050 , 0.7000 ]"

"D-tail / H-head / D2O" "Data array: [51 x 3]" "[ 0.0518 , 0.5888 ]" "[ 0.0050 , 0.7000 ]"

Custom Files: ------------------------------------------------------------------------------------------------------

**Name** **Filename** **Language** **Path**

**\_\_\_\_** **\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_** **\_\_\_\_**

"" "" "" ""

Constrasts: ----------------------------------------------------------------------------------------------

**p** **1** **2**

**\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

"name" "D-tail/H-Head/D2O" "H-tail/D-Head/ACMW"

"Data" "D-tail / H-head / D2O" "H-tail / D-head / ACMW"

"Background" "Background D2O" "Background ACMW"

"Bulk in" "SLD air" "SLD air"

"Bulk out" "SLD D2O" "SLD ACMW"

"Scalefactor" "Scalefactor 1" "Scalefactor 1"

"Resolution" "Resolution 1" "Resolution 1"

"Model" "Deuterated tails" "hydrogenated tails"

"" "Hydrogenated heads" "deuterated heads"

This may initially look quite complicated, but it is fairly self explanatory in terms of representing a typical RasCAL model (and should make sense to anyone familiar to the RasCAL gui, with some thought). The various aspects of the model definition are grouped together, then eventually combined to make our contrasts. These groups are:

*The Parameters Group* – This block defines all the parameters that we need to specify our model. In our layers case, we need 10 parameters to define our system: A bulk interface roughness, thickness’ and roughness for the headgroups and tails, and SLD values for the layers, depending on whether they are deuterated or not. In this block we also define the parameter values and their allowed ranges, and specify if they are included in the fit:-

Table

Description automatically generated with medium confidence

*The Layers Group: -* Once we have our parameters, we then need to group these into layers, in traditional RasCAL style. For our model, we always have two layers – a headgroup and the associated tails. Each of which can be deuterated, so we set up 4 layers in total, sharing the parameters between the layers as necessary:

Text

Description automatically generated with medium confidence

*‘Instrument’ Parameters: (Backgrounds, scalefactors and resolutions) –* These are necessary to specify our model, and are specified in much the same way as the parameters. The backround and resolutions blocks have a more complicated format to allow flexibility in how these are specified, which will be discussed further in chapter 2. These are the parameters that appear in the ‘experimental Parameters’ tab of the RasCAl1 model builder, and are subsequently included in the definitions of the contrasts at the end of the worksheet.

*Data* – Each contrast has to have a dataset associated with it, whether or not this is contains data or not. An empty data object (i.e. containing no data and just simulation ranges), means RAT will calculate the reflectivity only. When data is present, chi-squared will also be calculated. For our problem, we have two datasets and these are coded in to the data block ready to be incorporated into contrasts:

A picture containing diagram

Description automatically generated

*Contrasts* – Once we have defined all the components of our model, we need to group them together into contrasts. We have two datasets we want to consider, so two contrasts. We have the relevant instrument parameters, and also we specify which layers are included in each contrast (‘model’).

Text

Description automatically generated with medium confidence

*(b) Running our Model.*

As implied from figure (1), running RAT requires not only our input model specification, but also a controls block telling RAT what to do. We will discuss the controls block in more detail in Chapter 4, but for this demo we will just make an instance of the controls block and modify a few parameters to run the demo:

Graphical user interface, text, application

Description automatically generated

This makes an instance of the controlsDef class we have called ‘controls’. The various properties of the class allow the type of calculation to be specified, in terms of parallelisation, choice of algorithm to be applied and so on. Here we are specifying a single threaded calculation of our reflectivities only (the default) – in other words we are not asking RAT to do any kind of fit with our parameters. We can now send or problem definition and controls classes to the RAT toolbox:

Graphical user interface, application

Description automatically generated with medium confidence

It is worth noticing here that this is always the general format for calling RAT. There are two inputs – a problem definition and a controls block, and the result is two outputs – another copy of the controlsDef class, and a new, results block.

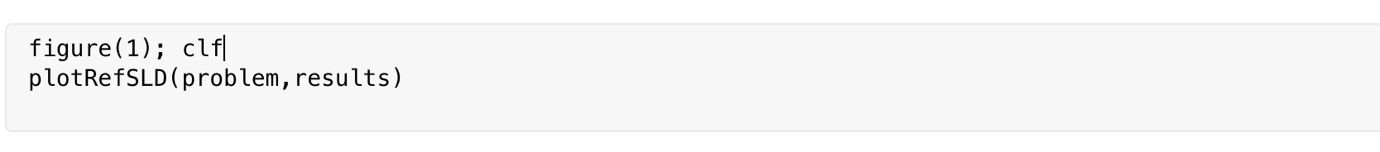
The controlsDef that returns is a copy of our input, except that the parameter values will be changed by any procedure done. So, if we run a simple fit, then the output problemDef will have the values of the best-fit parameters. Here, we are not doing any fitting yet, and so the output is an exact copy. Also, we overwrite out original input class with the output by using the same name (‘problem’) as an input and an output, but you don’t have to do it this way.

Once we’ve run our model through RAT, then the second output (we call ‘results’ here) is an array which contains the output of the calculation (we will discuss this in more detail in chapter 4) :

Graphical user interface, application

Description automatically generated

This contains the results of our calculations, so for us including the SLD profiles and reflectivities calculated from our problemDef class. We can now plot the output, either manually (by taking the relevant parts from the ‘results’ array), or using one of the supplied plotting utilities:



Chart

Description automatically generated

We can see that our model is looking fairly sensible, but that our guess values for the parameters are pretty wide of the mark.

To do something more than just simulating the reflectivity, we change the procedure we specify in the controlsDef class. At the moment, controls.procedure is set to ‘calculate’, which does a one time calculation of the reflectivity given the current parameters only. To do a fit, we can just change the ‘procedure’ attribute of our controlsDef from ‘calculate’ to ‘simplex’, which reveals a new set of parameters in the controlsDef block..