

OpenModel

Example Models and Applications

Neil Crout
School of Biosciences
University of Nottingham
UK

www.openmodel.info

2.4.1; 3 February 2016

© OpenModel Team, University of Nottingham, 2005-
FreePascal Compiler © The FreePascal Team 1993-

Table of Contents

1.	What is OpenModel	3
2.	About this example model guide	3
3.	Degradation Kinetics Examples	4
3.1	Single first order kinetics (SFO)	4
3.1	Two-compartment model	16
3.1.1	Create the Model Script	17
3.1.2	Attach the Observations	21
3.1.3	Fit the Model	23
3.1.4	Apply the model	26
3.2	Parallel Kinetics	28
3.3	Temperature and Moisture Dependent Degradation	31
4.	Radiocaesium in the tissues of sheep	34
4.1	Background	34
4.2	Model Overview	34
4.3	Model Implementation	35
4.4	Using the Observations to Measure the Fit	36
4.5	Fitting the Model	37
4.6	Classic Confidence Intervals	40
4.7	Metropolis-Hastings Confidence Intervals	41
4.8	More things to try...	42
5.	Uptake of Radiocaesium from Soil to Plants	44
5.1	Iterating over Samples	44
5.2	Model Reduction	45
5.3	Method Overview	45
5.4	Practical Implementation	46
5.4.1	Identifying Candidate Variables	46
5.4.2	Screening variable replacement 1 by 1	47
5.4.3	Factorial reduction	48
6.	References	51

1. What is OpenModel

OpenModel is an integrated package designed to allow the specification, solution, visualisation, parameterisation and numerical analysis of certain classes of mathematical models. It is ideally suited to the solution of models which can be specified as a combination of ordinary differential equations and simple assignments.

Its key features are:

- Modular structure
- Standard text specification of equations (paradigm is the solution of ordinary differential equations)
- Tabular, graphical and spatial outputs
- Model parameterisation (variety of methods: Marquardt, Metropolis-Hastings, Great Deluge)
- Uncertainty estimation
- Variable replacement

2. About this example model guide

The guide presents some example models implemented using the various features of OpenModel. Inevitably the guide is incomplete and does not illustrate all the possible types of model that can be implemented. We aim to improve it as we discover defects or omissions and respond to suggestions where possible.

All the example models presented are included in the OpenModel installation; they will be included in the OpenModel Files folder. You may find these examples more informative than the guide itself 😊.

Some examples are presented in a detailed step by step format in order to illustrate the use of OpenModel to implement and apply the model. This serves mainly as a tutorial in the use of the software. Other examples are intended to illustrate the overall approach to model implementation for different types of examples presented. These examples are easier to read but probably require some basic knowledge of OpenModel.

This guide should be used together with the main User Guide which details the features and use of OpenModel.

The degradation kinetics examples were specifically suggested by Stefan Meinecke (UBA, Germany's Federal Environmental Protection Agency). Whether or not you are interested in models of chemical degradation in the environment these are excellent model case studies as they illustrate many OpenModel features. Two detailed step by step examples of simple models are presented. These are supplemented by an overview of some more sophisticated models.

3. Degradation Kinetics Examples

The fate of potentially harmful chemicals when they are introduced into the environment is often of concern, especially from an environmental regulation perspective. Models can be usefully applied to predict the subsequent transfer between different components of the environment and the time dependent (kinetic) changes in chemical degradation.

A feature of this sort of work is the widespread use of observational data to fit model parameters. In this guide 2 detailed step by step examples of simple models are presented. These are supplemented by an overview of some more sophisticated models.

3.1 Single first order kinetics (SFO)

We imagine the degradation of a chemical substance after its addition to water. The simplest case is to assume that the substance degrades by first order kinetics (i.e. the relative rate of degradation remains constant). This is equivalent to radioactive decay.

In the practical case the concentration of the substance is measured (as a time series of concentration values in Table 1).

Table 1: Dataset A (Focus 2011)

time	Parent
0	101.24
3	99.27
7	90.11
14	72.19
30	29.71
62	5.98
90	1.54
118	0.39

The equations of the model are

$$\frac{dParent}{dt} = -k_1 \times Parent$$
$$Parent = M_0 \text{ at } t = 0$$

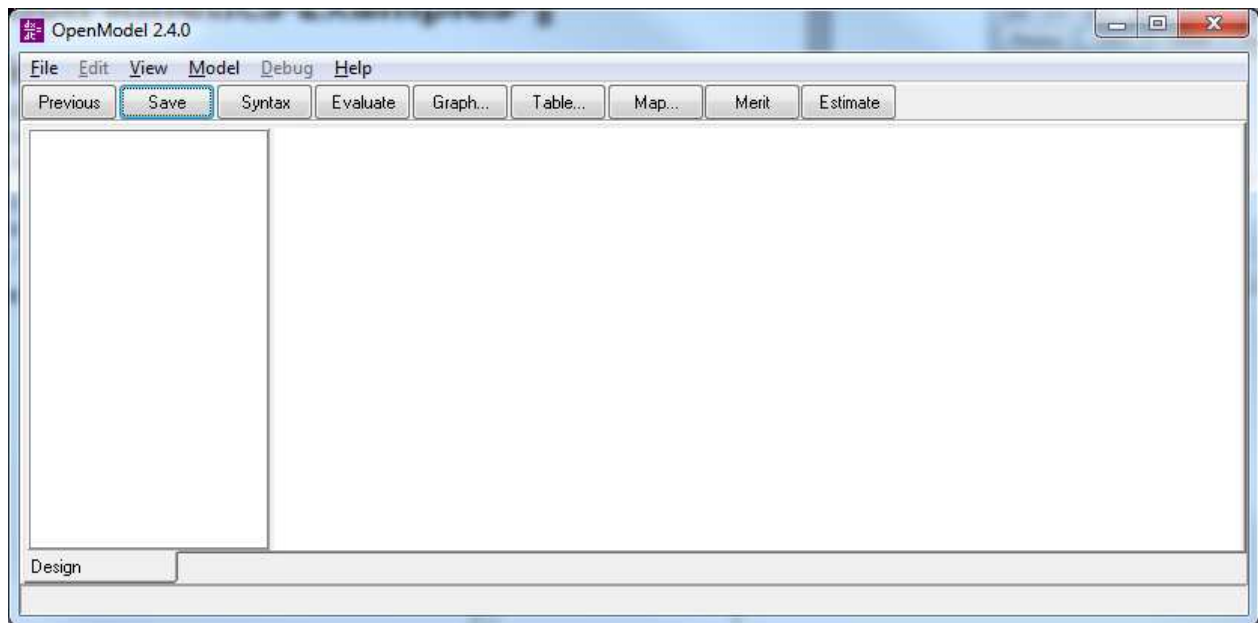
where **Parent** is the concentration of the added chemical at any point in time, **k₁** is the degradation constant and **M₀** is the concentration of the parent added to the water at t=0.

The purpose of our work is to estimate the degradation constant so the model can be used to make environmental assessments of the chemical measured. To accomplish this in OpenModel it is useful to think of the implementation in 3 stages:

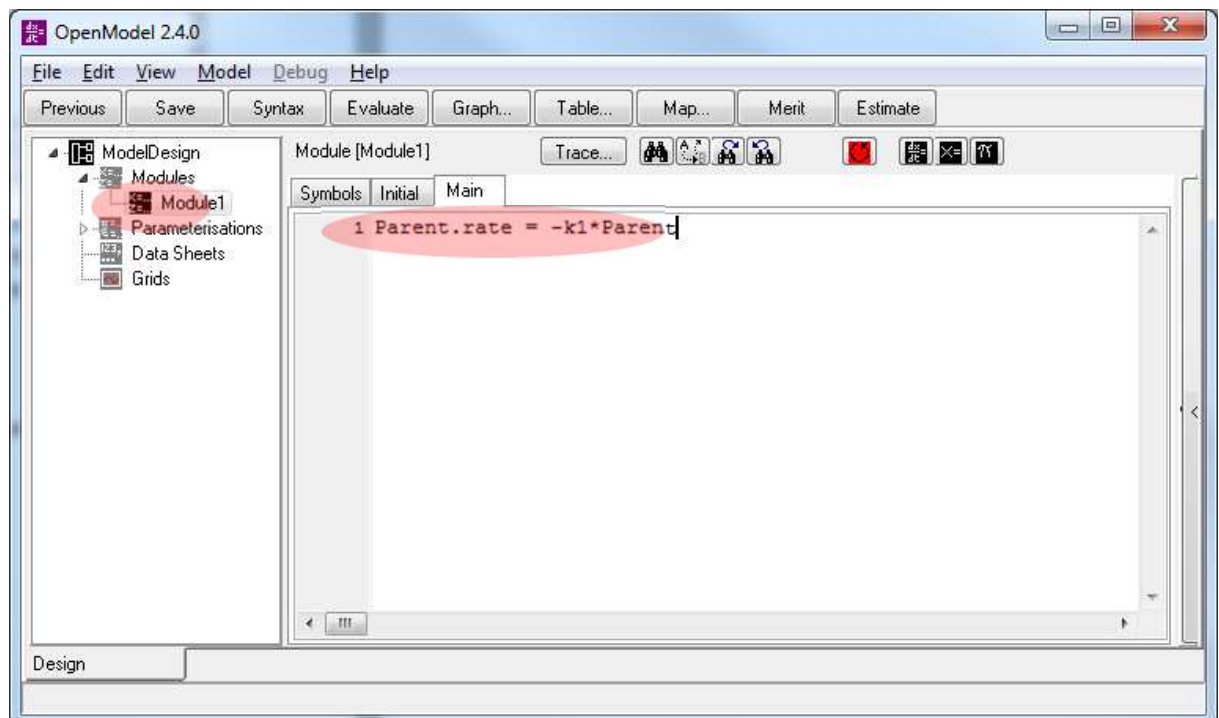
- Creating the model script so the model equations can be solved using arbitrary values of the model parameters (**k₁**).
- Attaching the observational data so the model predictions can be compared to reality and the agreement quantified using what OpenModel called a 'merit' object.
- 'Fit' the model to the observed data to estimate the best fitting model parameters (i.e. the parameter value which give the best agreement to the observations).

We will now implement this model in OpenModel and estimate **k₁**; step by step. There are a lot of steps, but they are all quite small!

1. Start OpenModel (it will be blank). Select File|New from the menu.

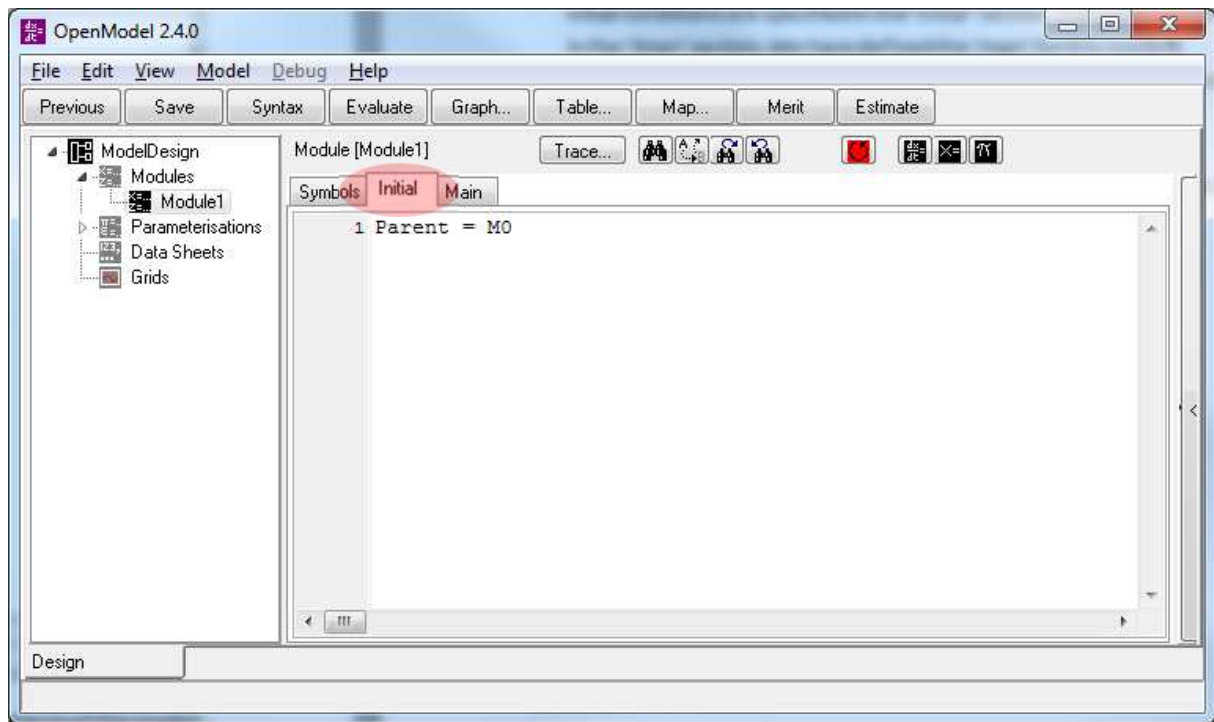



2. Click on **Module1** in the left hand tree view (this view organises the main elements of the model). The empty scripts for the model will be shown. Type in the equation. The syntax in OpenModel for the differential equation is **Parent.rate = -k1*Parent**. You can see this entered into the 'main' script below.

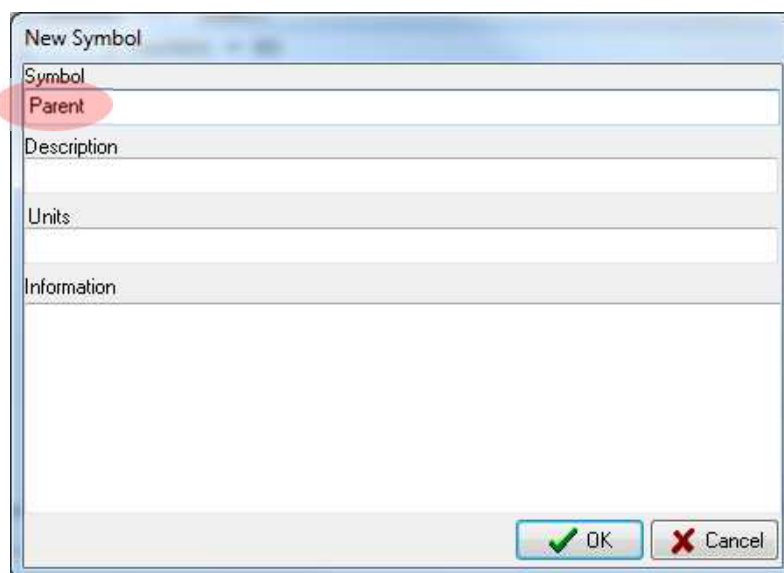


OpenModel scripts have two sections: 'main' and 'initial'. This is because the standard methods for solving differential equations rely on the definition of initial conditions. In OpenModel the initial conditions are specified in the 'initial' section. The subsequent rates of change are defined in the 'Main' section. So far we have defined the 'main' for this model

3. To define the 'initial' section select the initial tab and enter the initial equation as shown below.

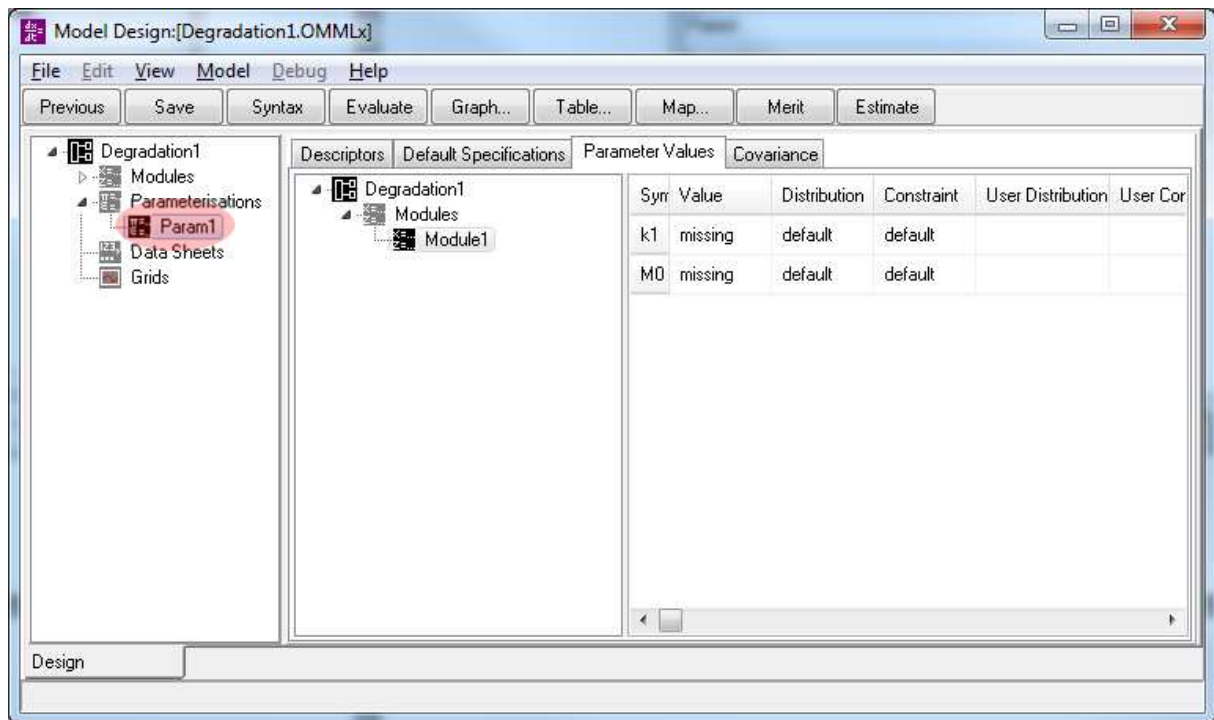


4. OpenModel is very strict about the names of the symbols it uses. Until we define them it will not 'know' what we mean by **Parent**, **k1** and **m0**. We have to define these. **Parent** is a differential equation. **k1** and **m0** are parameters. In OpenModel parameters are constants in the model evaluation which we can estimate by fitting the model to the observations. The quickest method to define these 'symbols' is to click the symbol buttons above the script views. Start by clicking the differential equation symbol button (). The dialog shown below will appear. Specify the symbol as **Parent**.

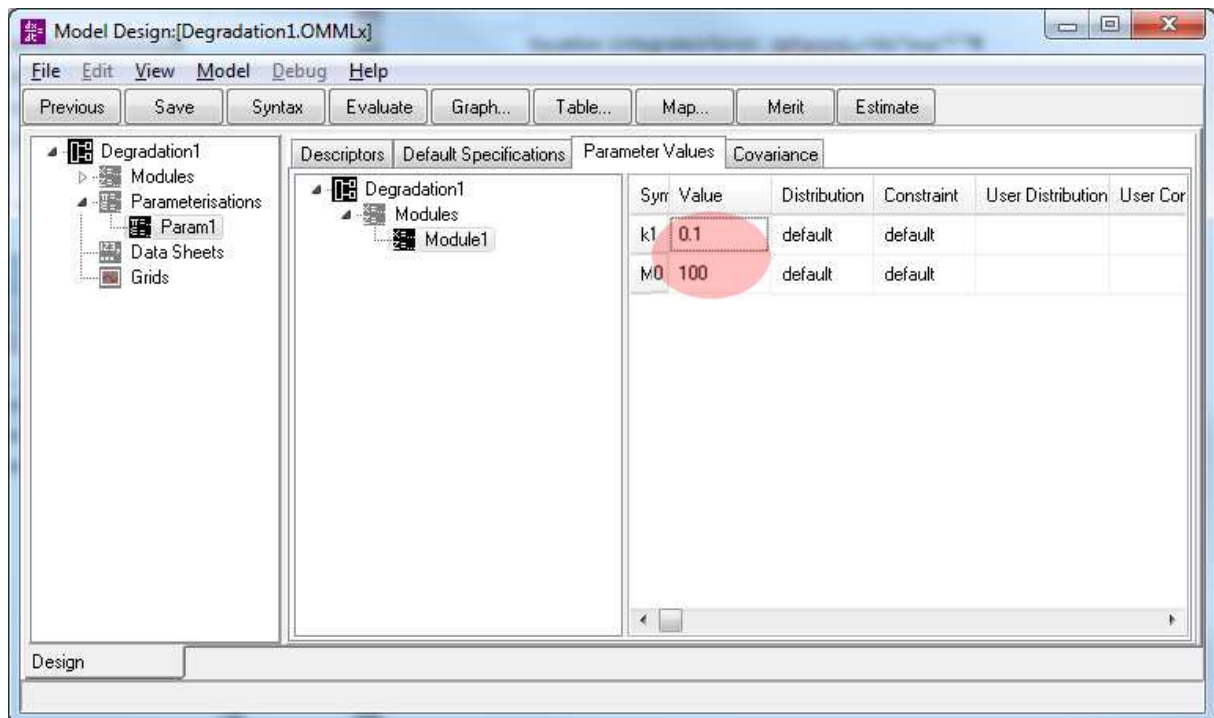


5. Repeat the procedure for the two parameters, **k1** and **m0**, by clicking the parameter button.
6. In a moment we will be able to evaluate the model (i.e. solve the differential equation) but to do this we need to supply some initial estimates for **k1** and **m0**. We propose **k1=0.1** and **m0=100** (these are guessed simply by looking at the observed data) but be reassured, it is not important to make good guesses, they can easily be adjusted later.

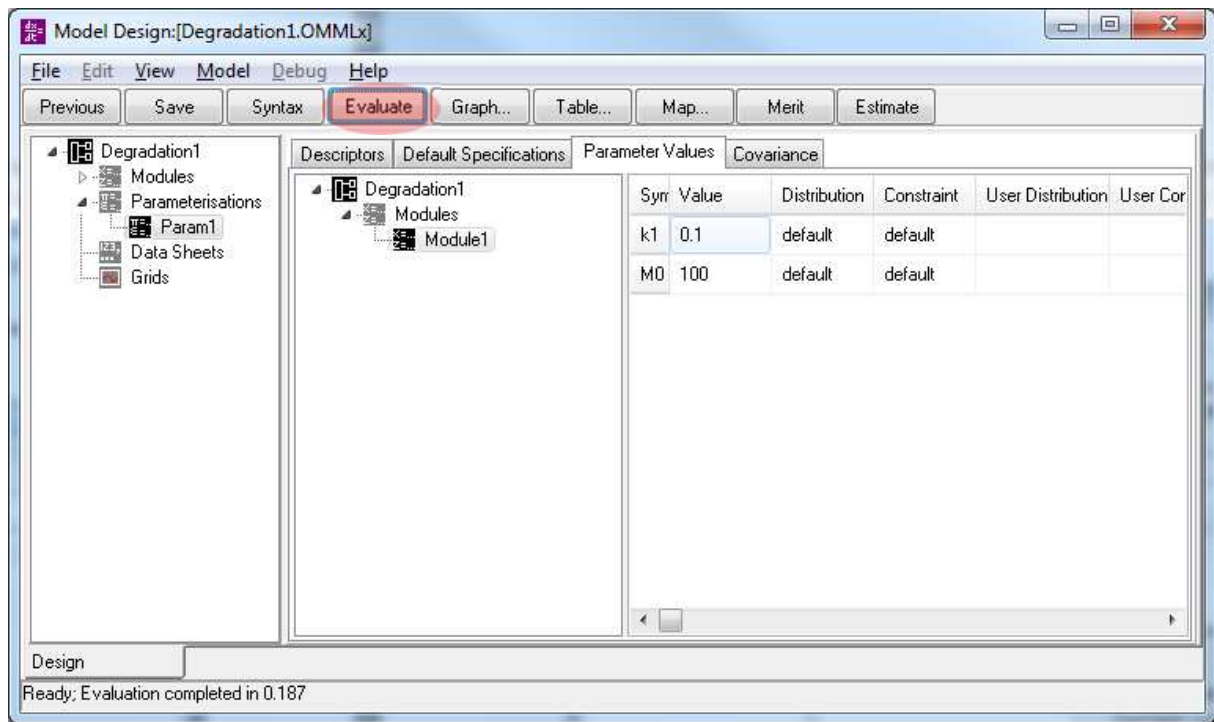
7. To set these values expand the parameterisation node of the ModelDesign tree on the left hand side of the OpenModel display



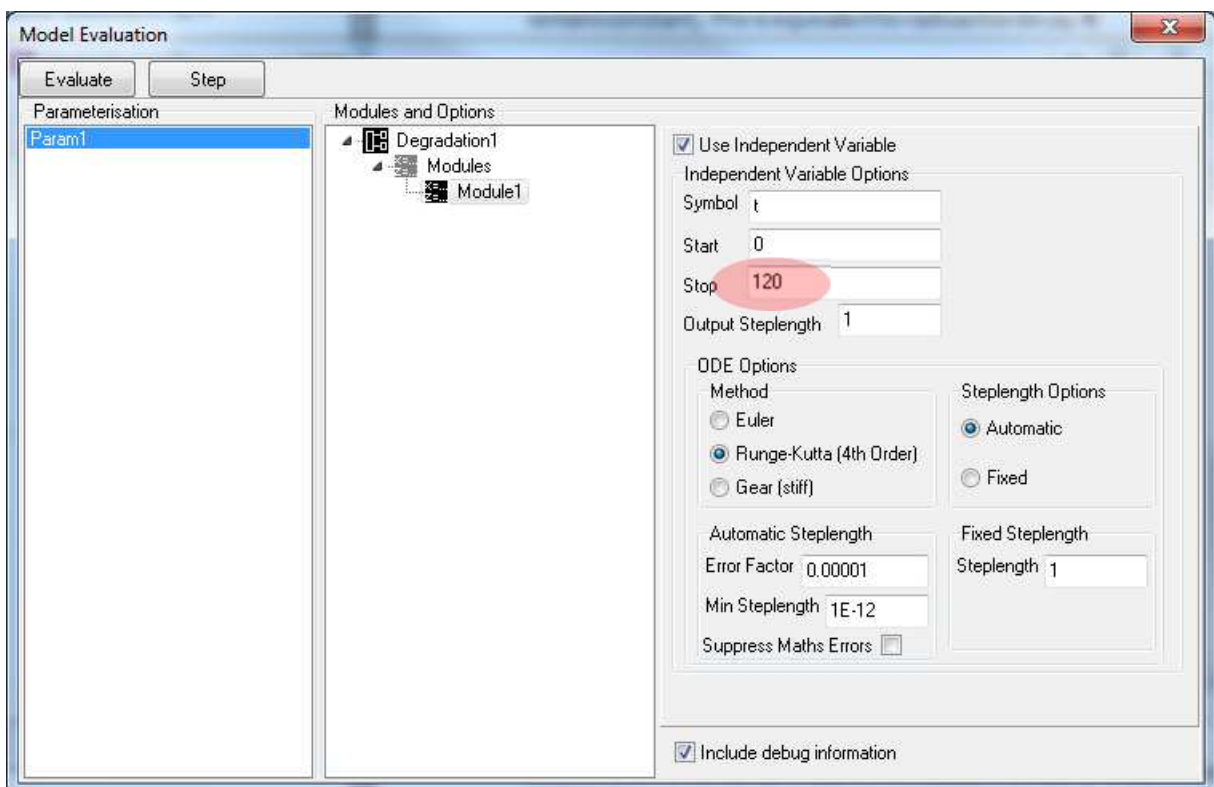
8. Enter the values into the parameterisation view.



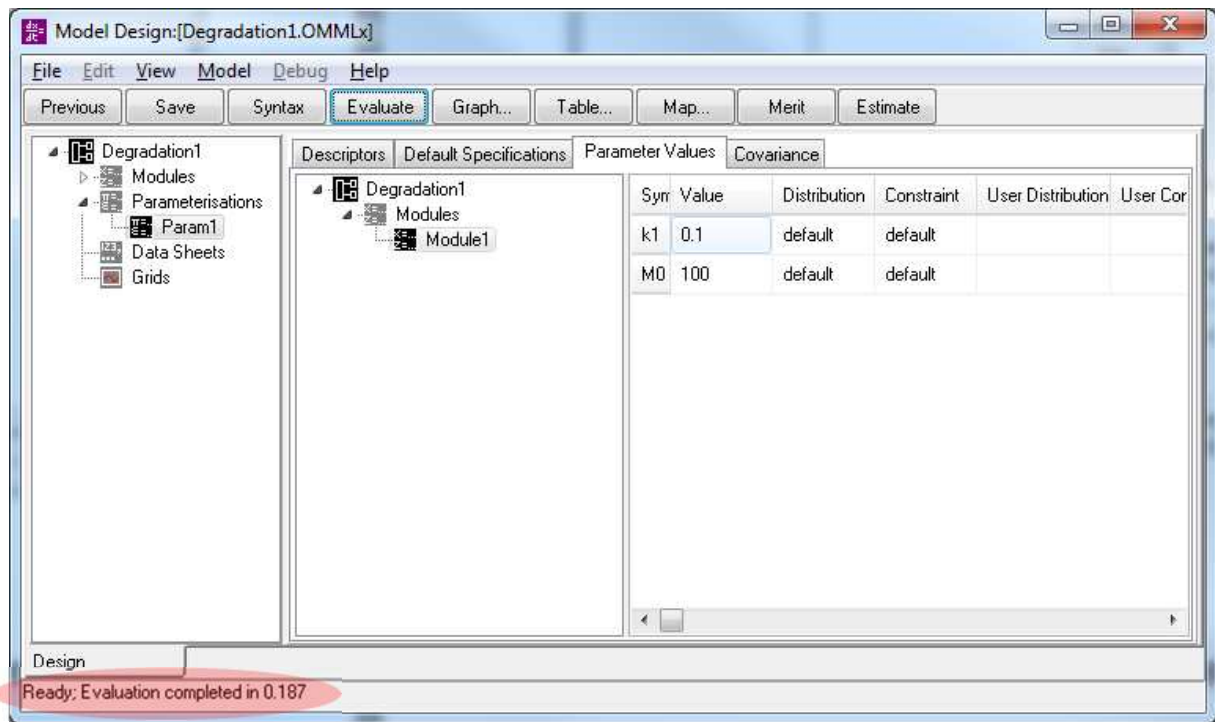
9. Before going any further we should save the model (**File|save**). To check we have not made any mistakes in syntax or definitions run an evaluation of the model. Click on the **Evaluate** button on the OpenModel tool bar



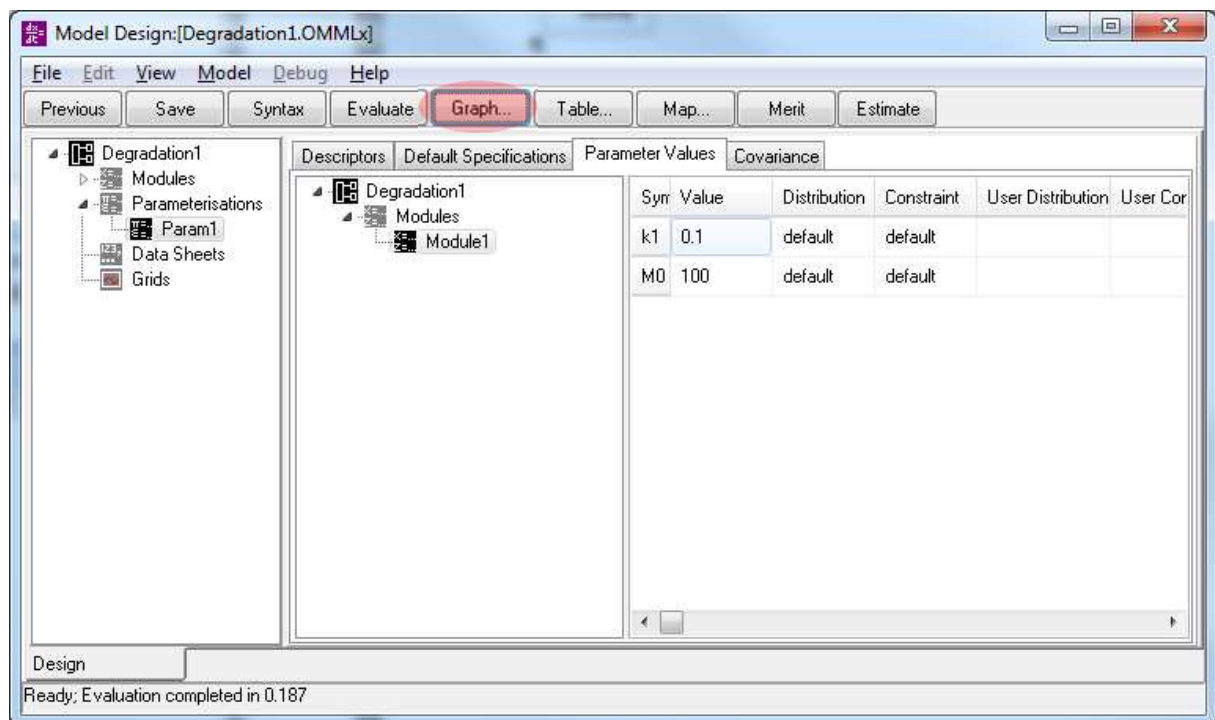
10. This will reveal the Model Evaluation dialog box as shown below. This defines the initial and final value of the model independent variable (in this example the default independent variable t is ideal). The default end time is 100, however our observations extend to $t=118$, therefore in the dialog below the stop time is changed to 120. Once these changes are made click Evaluate.



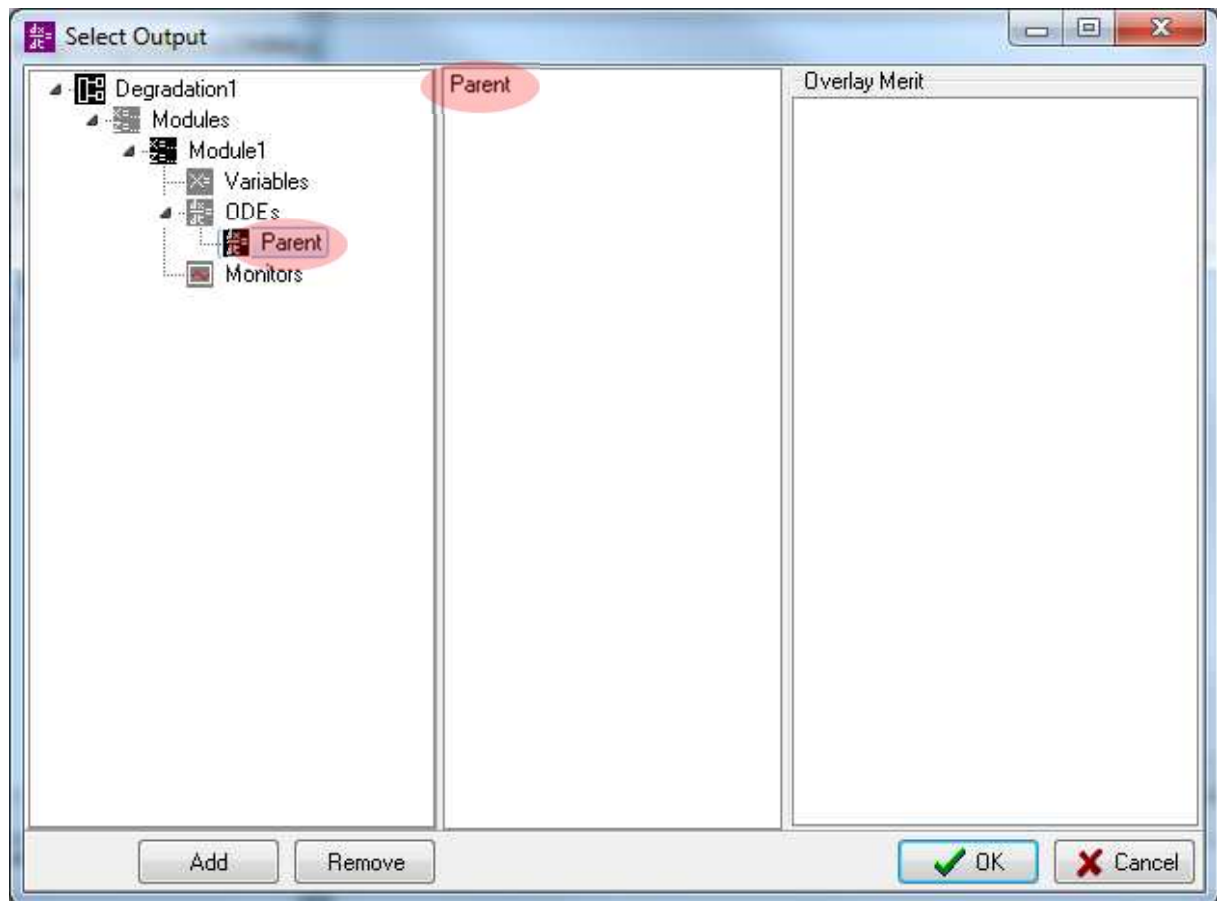
11. If everything is correctly defined the model will compile and run. It will give several messages on the status bar and tell you when it is complete.



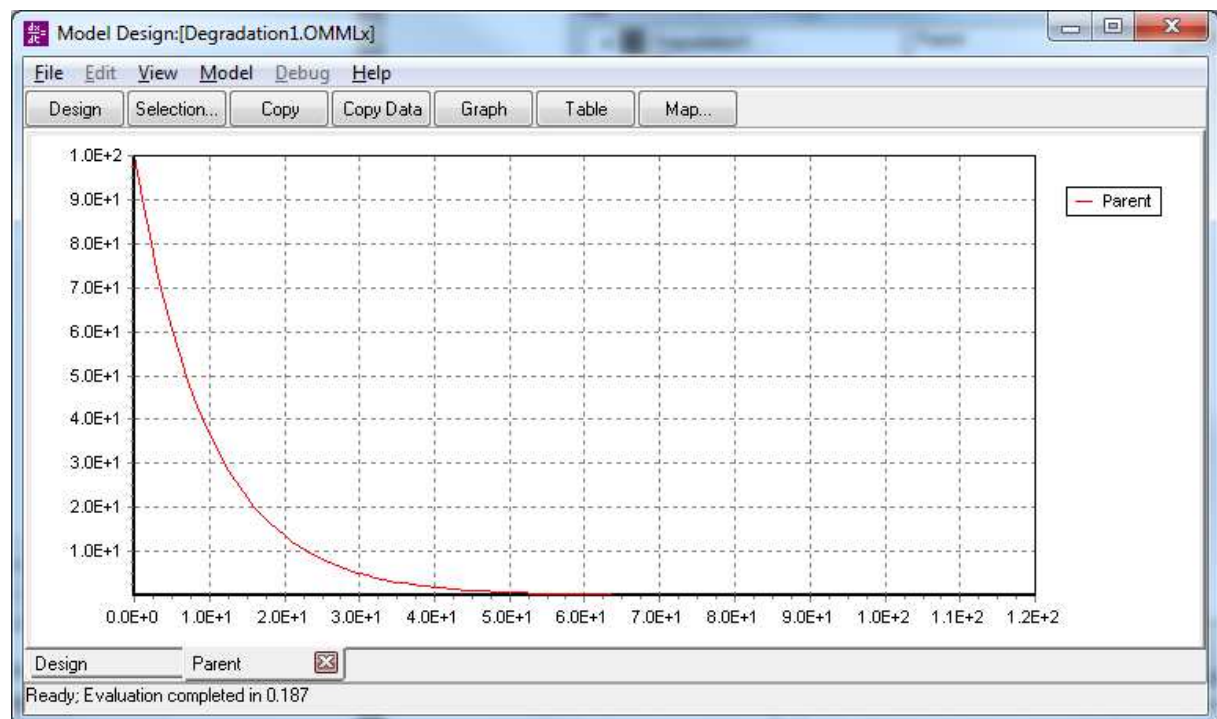
12. Hopefully your model has also run and completed. Now what? Lets make a graph of the results, just to check it is giving the sort of result we expect. Click graph on the OpenModel toolbar.



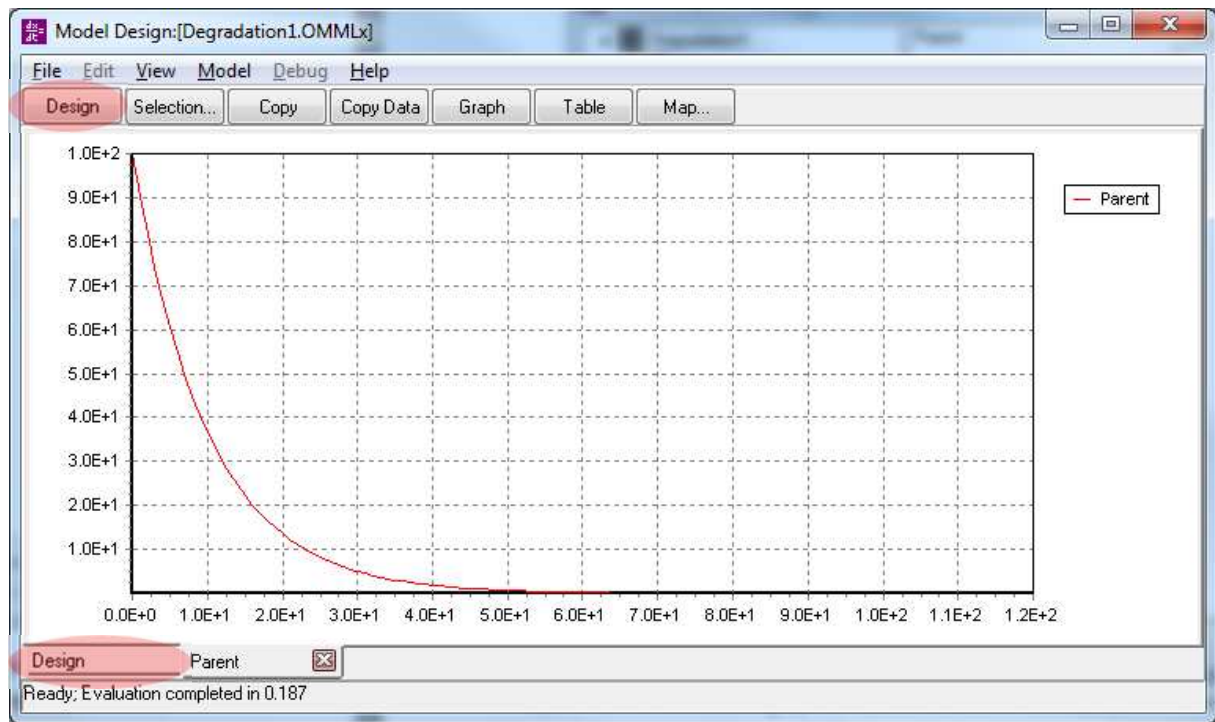
13. This will display the graph dialog. Expand the model tree to reveal the differential equation symbol **parent**. Double click it so it moves to the middle list. All the symbols listed in the centre will be shown on the graph when you click ok. In our case the model only has a single symbol so the setup is simple!



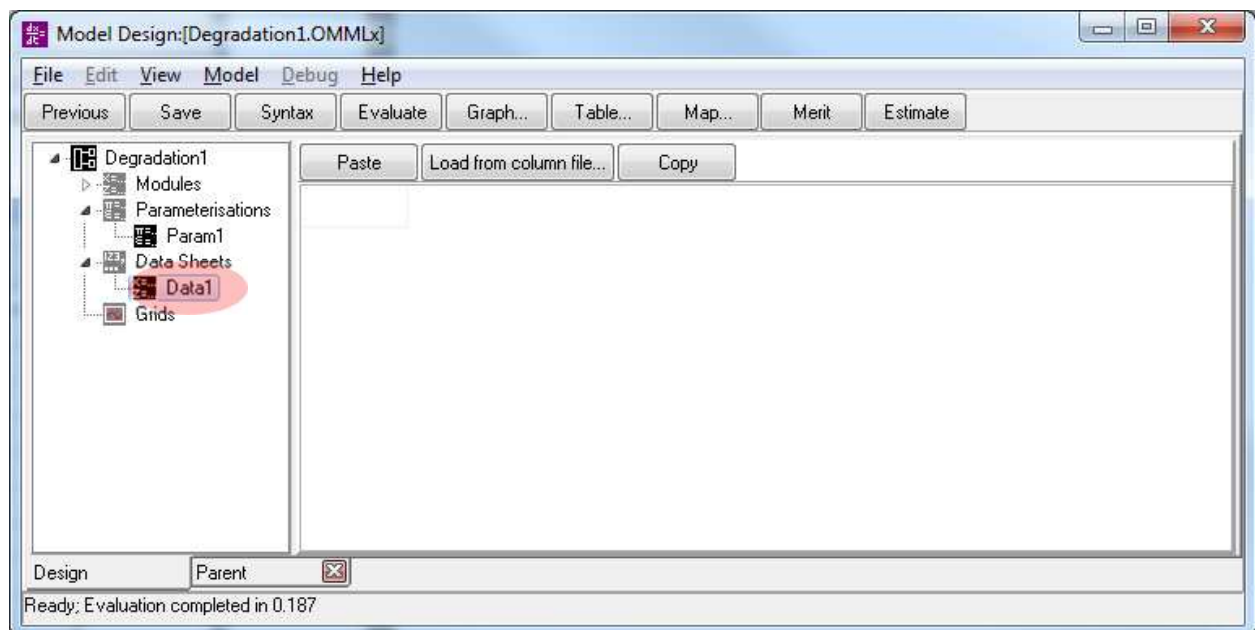
14. When OK is clicked a graph of **Parent** vs **time** will be displayed within the main OpenModel display as an additional tab as shown below.



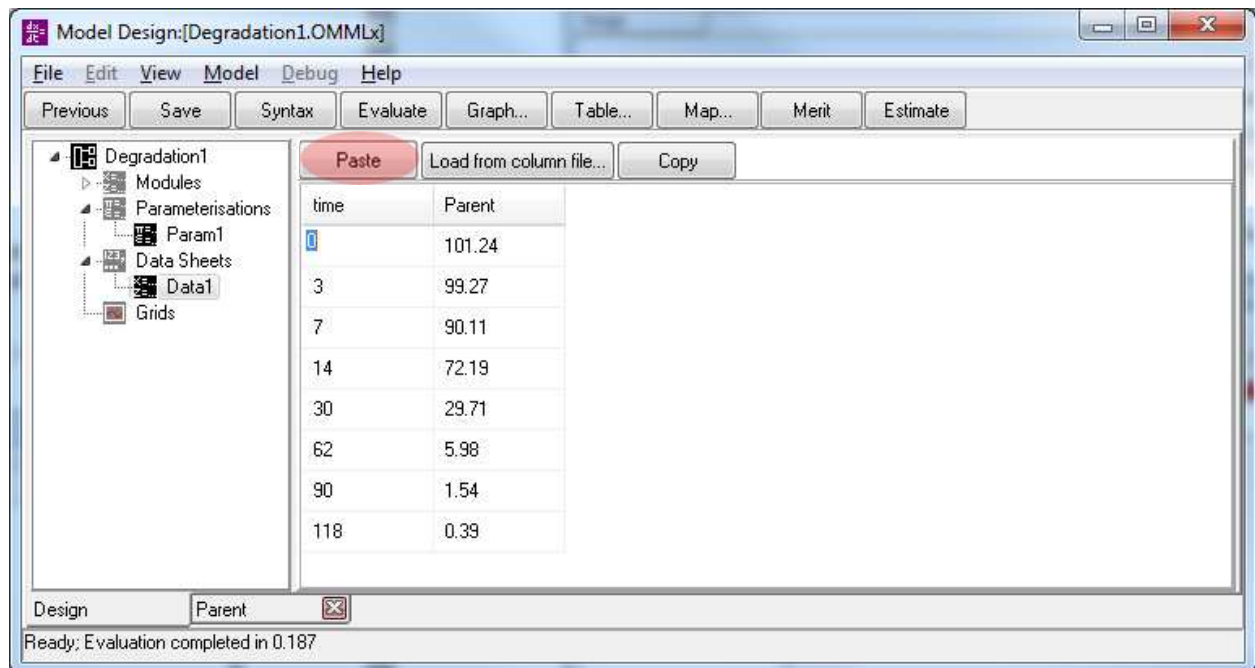
15. We will now add the model data and compare this to the predicted values of **Parent**. Return to the Design tab (either click the Design button on the tool bar or click the Design tab itself).



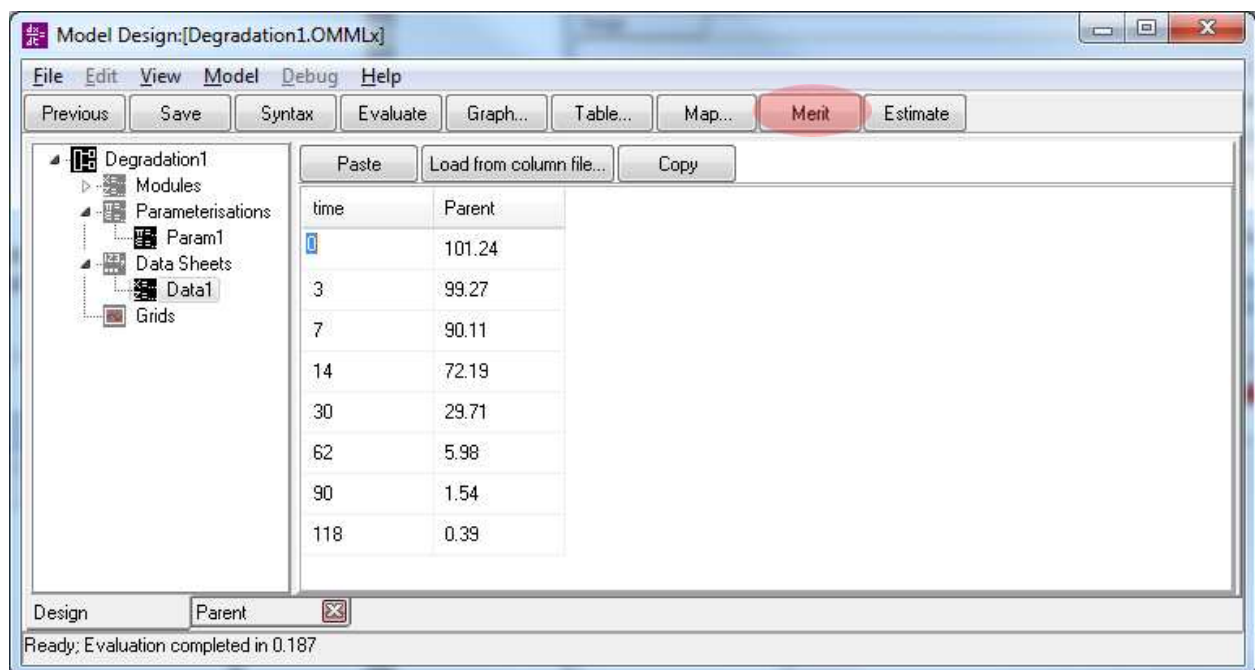
16. Select the 'Data Sheets' node in the model design tree. Right click the node to reveal a pop up menu. Select Add Data Sheet



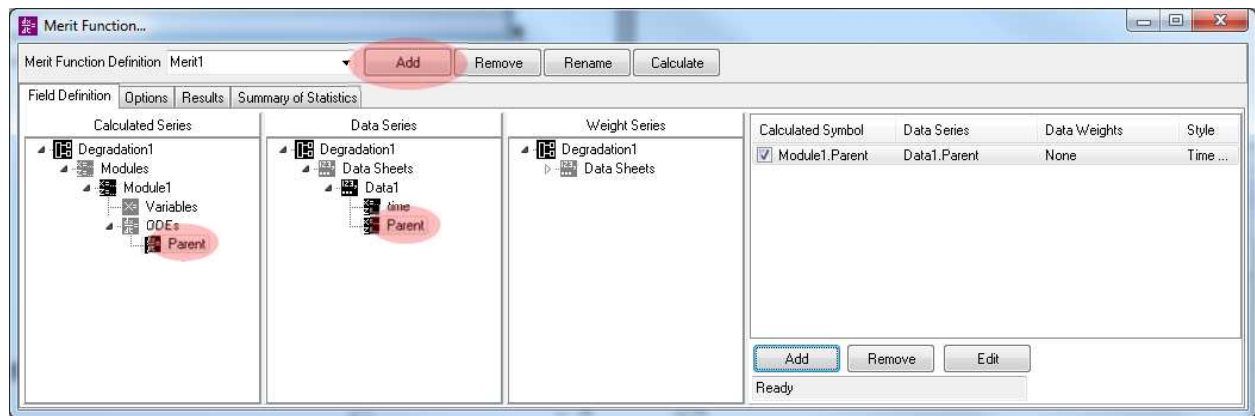
17. This will open up a blank data sheet. Copy and select the data from the excel work book provided (Degradation Examples – sheet example 1). Click Paste on the Data Sheet Toolbar and the data will be pasted in as shown below.



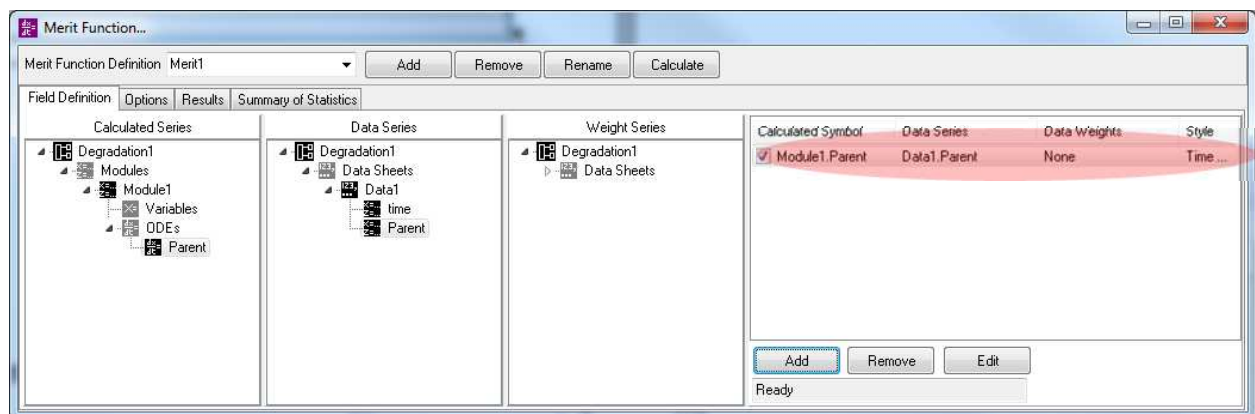
18. Save the model! This is always a good idea. Next we want to compare the observed data to our model predictions. This requires a merit object to be defined. Click merit on the main tool bar



19. That will show the merit dialog box. The main purpose of this is to allow you to associate model calculated symbols with observed values. Expand the **calculated series** and **Data series** trees to show the available symbols (in this simple model there is just one series for **calculated** and one for **data**). Click **parent** in both trees and then click the add button.

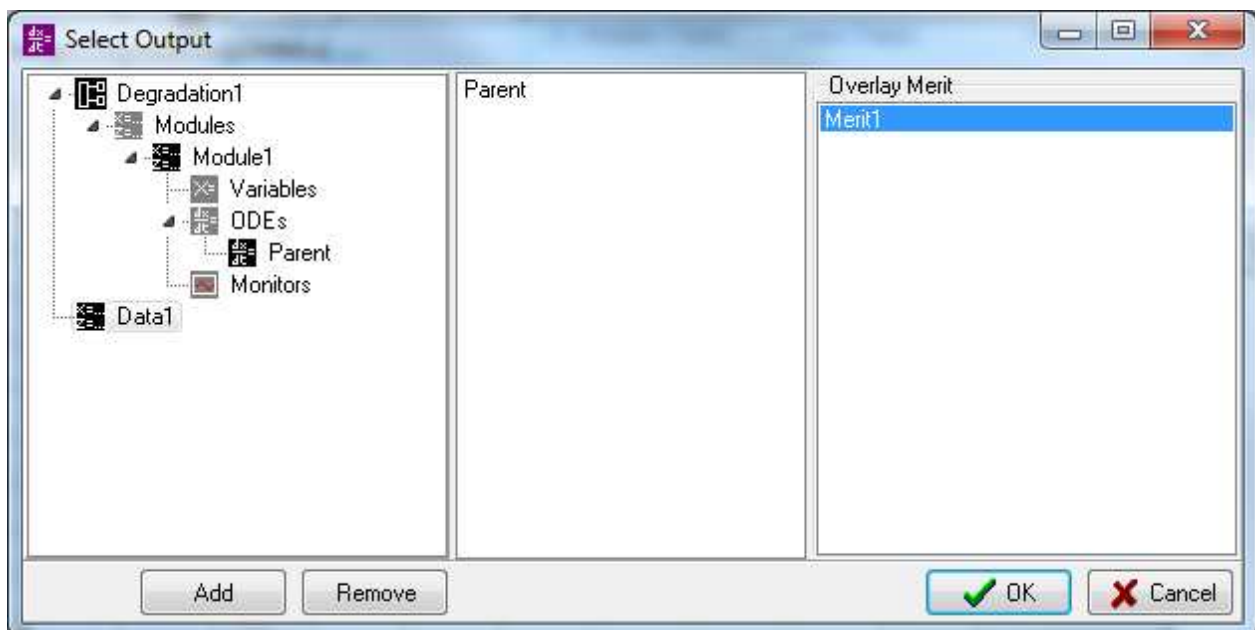


20. This creates a merit series which is listed in the right hand view.

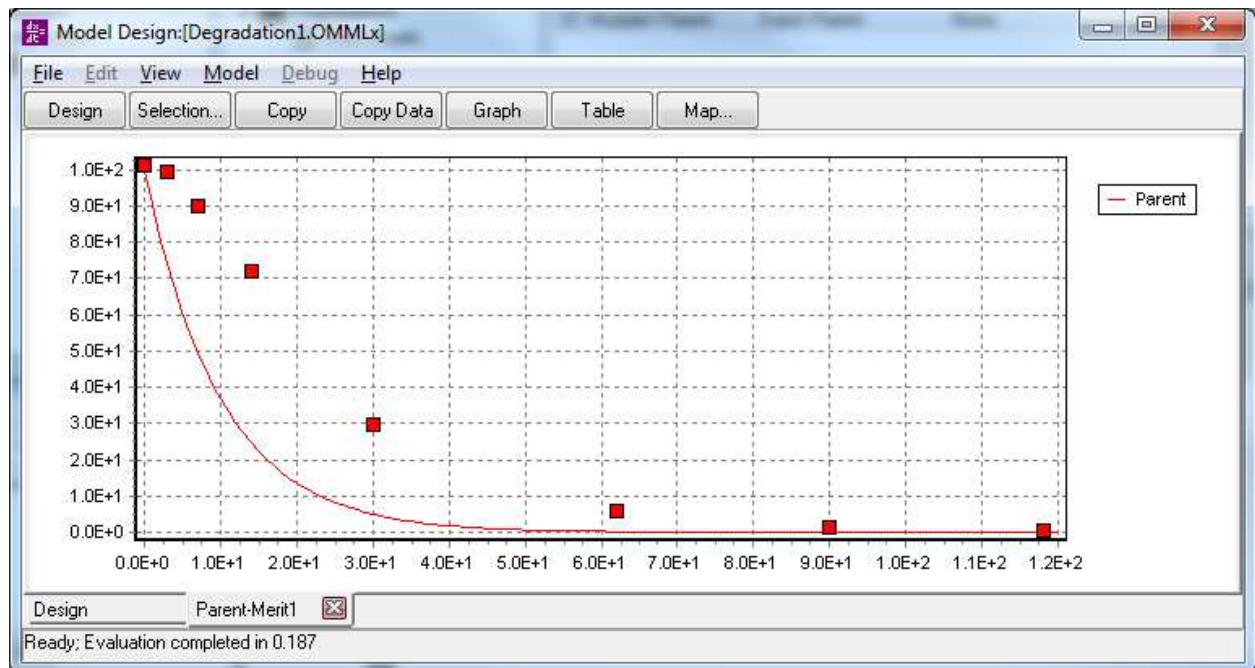


21. That's it, the merit object has been defined. It associates the calculated and observed data. We can then use this to create overlays on the graphs (in just a moment) or to specify what we want to fit when we estimate parameters.

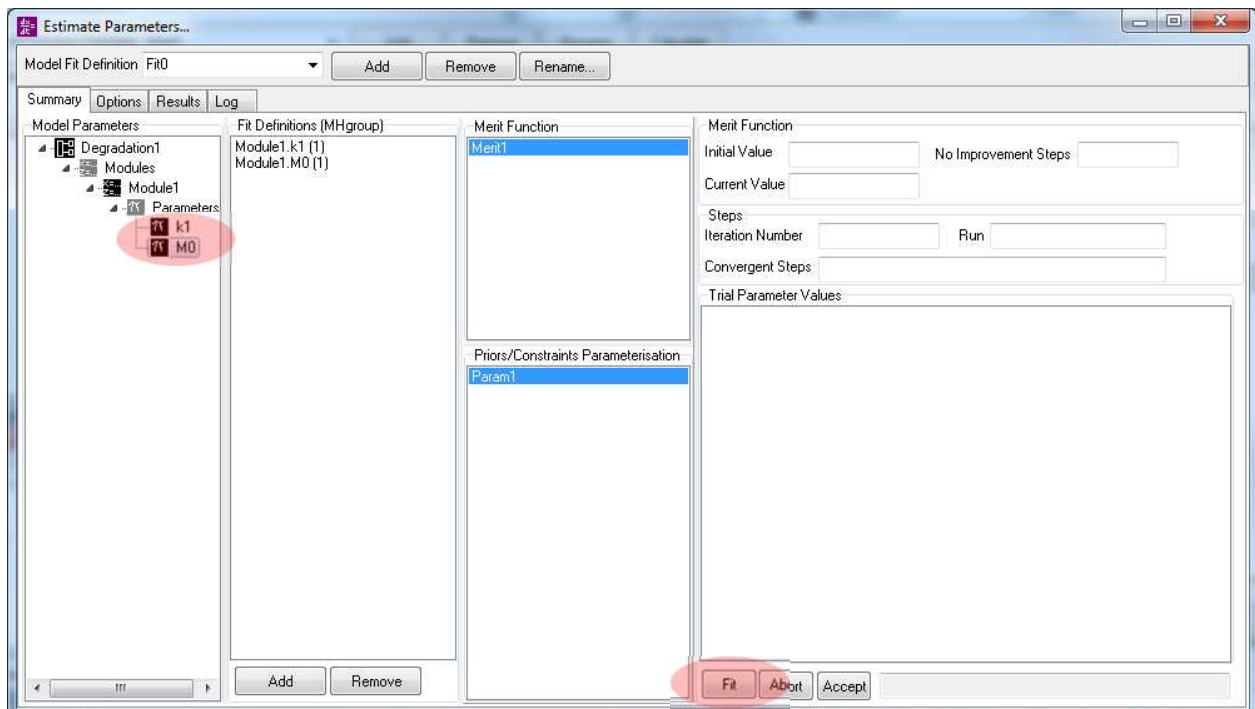
22. Return to the main OpenModel window. Select the graph tab and double click anywhere in the graph window. This will re-open the graph selection dialog. This will now show that there is a merit object available as an Overlay Merit. Select the new merit object (**merit1**) in the example below. Click OK



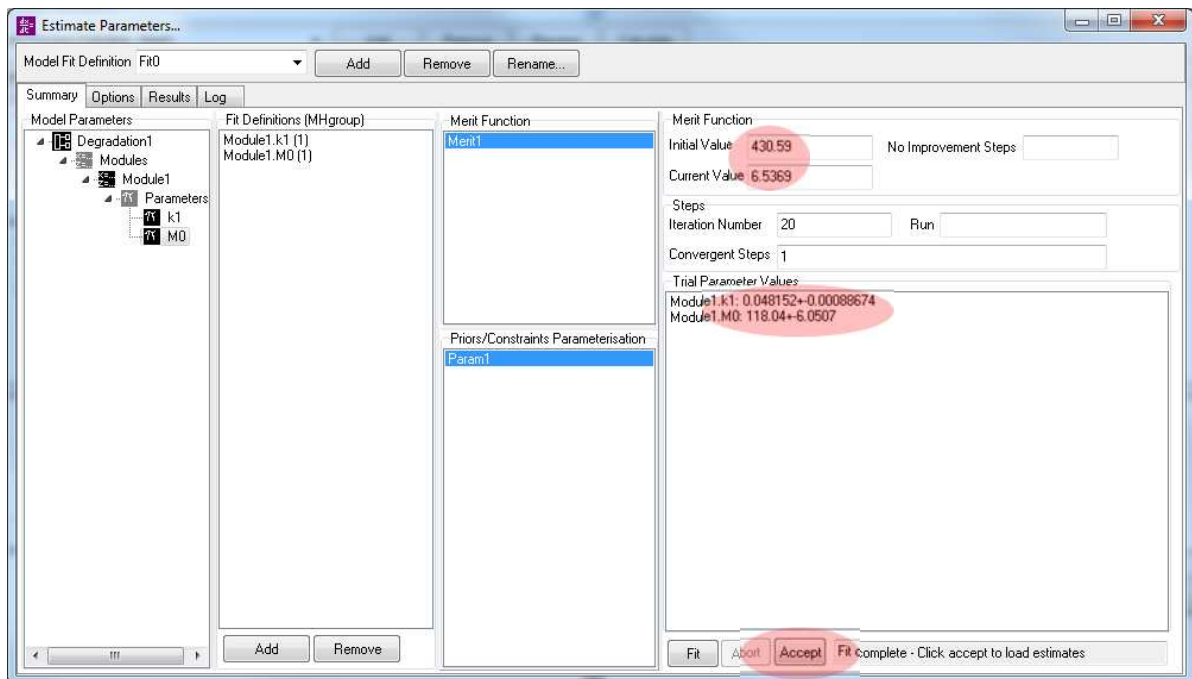
23. The graph selection will be updated to show both the calculated and observed values as shown below. The fit is not perfect, but as we guessed the parameter values that is not surprising!



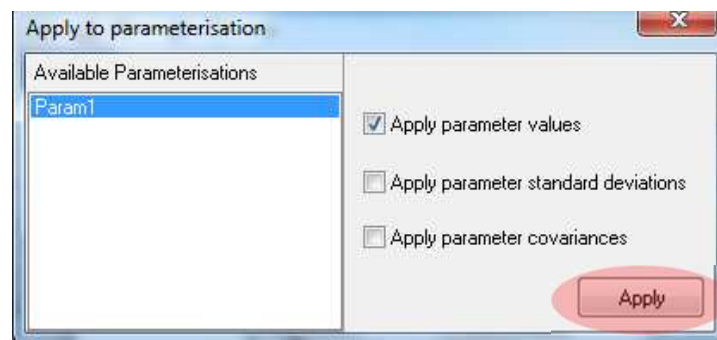
24. Return the model design tab and select estimate from the tool bar (or **Model | Estimate** from the menu). This will open the estimate parameters dialog. Expand the model design tree and double click the parameters we wish to fit (**k0** and **k1**). To fit the model click fit and the default fitting procedure will run.



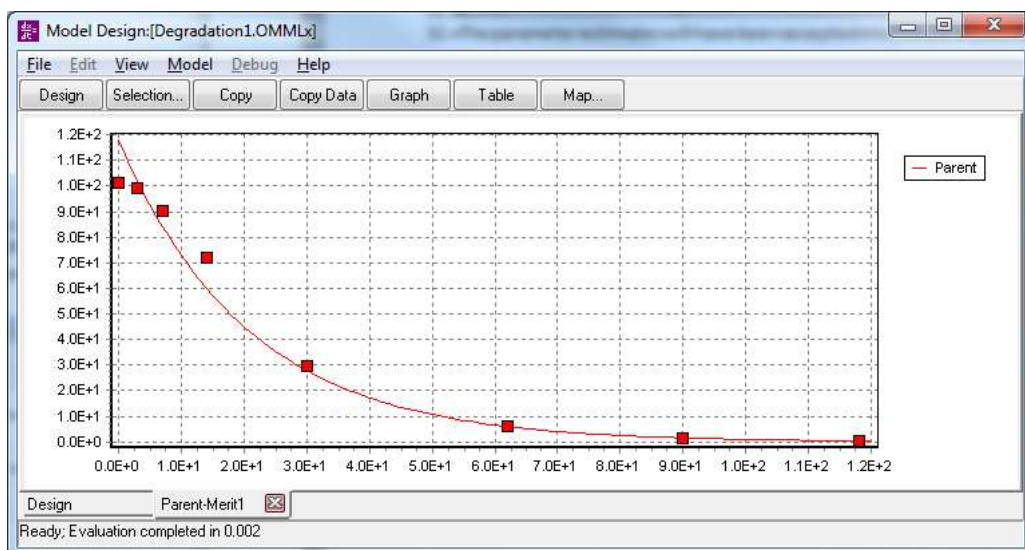
25. Once the fit is complete. This shows that the residual sums of squares (the merit object) have been reduced from 430.6 to 6.5 (big improvement). The parameter estimates are also shown. To accept these so they can be used in the model click the **Accept** button.



26. This reveals the dialog box shown below. To accept these so they can be used in the model click the **Apply** button.



27. The parameter estimates will have been accepted into the parameterisation **Param1**.
 28. Re-evaluate the model using the revised parameters. A much improved fit, although the model is not following the data so well at the start of the model simulation.



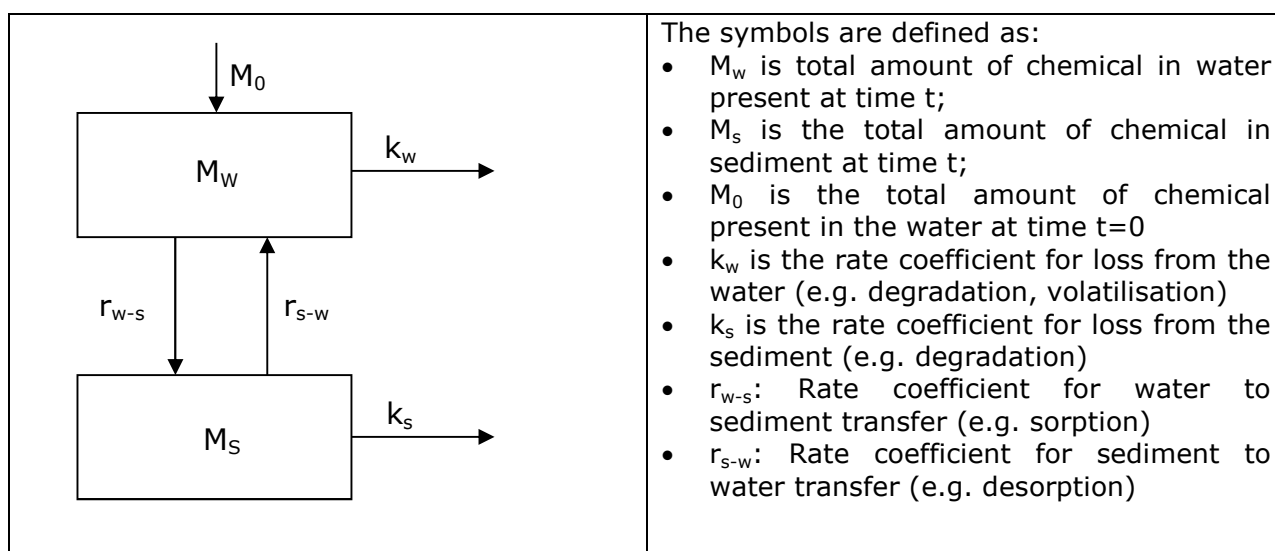
29. Assuming the model is appropriate we have estimated the value of M_0 (the amount initially added to the water system), 118 and the rate coefficient k_1 , 0.048. Good work.

3.2 Two-compartment model

This example extends the single first order kinetic model in 3.1 to a two-compartment model. This is conceptualised as the transfer of a contaminant between a water column (for example in a lake) and the sediment below the water.

The contaminant is added to the water compartment. From here it can be either (a) removed from the system due to degradation to a harmless form or volatilisation to the atmosphere or (b) transfers to the sediment. Once in the sediment the contaminant can be either (a) transferred back to the water column or (b) be removed from the system by degradation to a harmless form.

It is always useful to have a conceptual diagram for these types of models.



The definition of the model as differential equations is

$$\frac{dM_w}{dt} = -k_w M_w - r_{w-s} M_w + r_{s-w} M_s$$

$$\frac{dM_s}{dt} = -k_s M_s + r_{w-s} M_w - r_{s-w} M_s$$

The aim of our work is to use observations of M_w and M_s for a known M_0 to estimate the values of the rate coefficient k_w , k_s , r_{w-s} and r_{s-w} so the model can be used to make environmental assessments of the chemical measured. This is readily accomplished in OpenModel, although less experienced users usually find it helpful to think of the implementation in 3 (or maybe 4) stages:

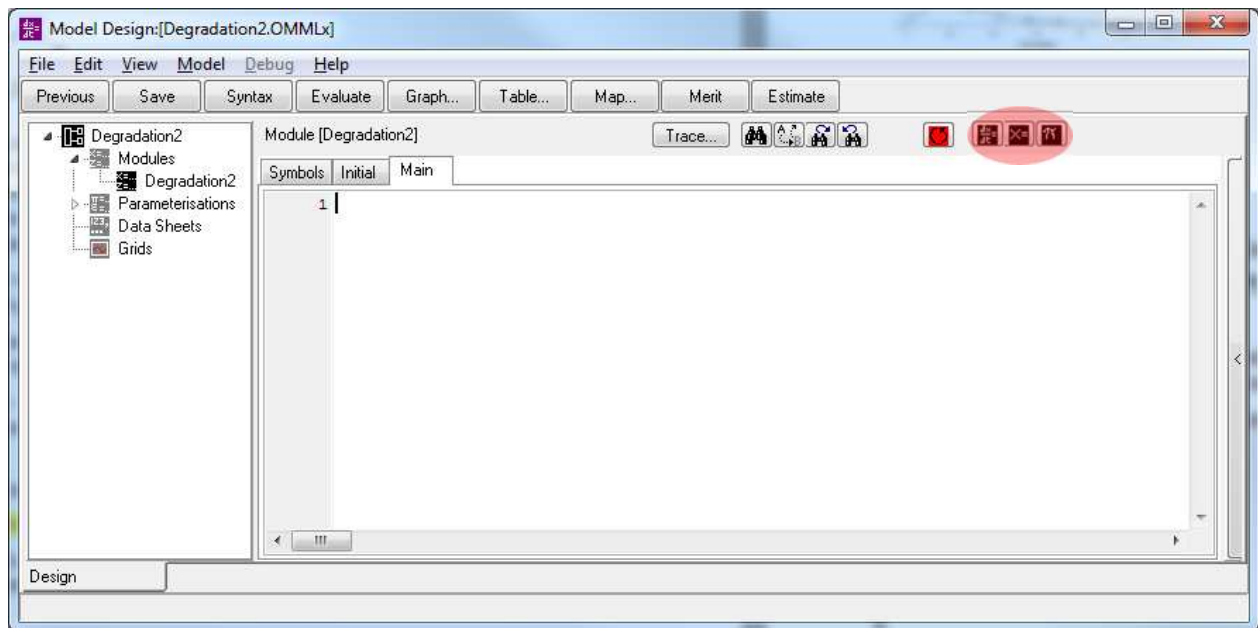
- Creating the model script so the model equations can be solved using arbitrary values of the model parameters (k_s , k_w , r_{s-w} , r_{w-s}).
- Attaching the observational data so the model predictions can be compared to reality and the agreement quantified using what OpenModel calls a 'merit' object.
- 'Fit' the model to the observed data thereby estimating the best model parameters (i.e. the parameter values which give the best agreement to the observations).
- Use the model for your intended purpose!

We will now work through the OpenModel steps, the overall procedure is exactly the same as for the single compartment model in 3.1, albeit requiring more details to be specified.

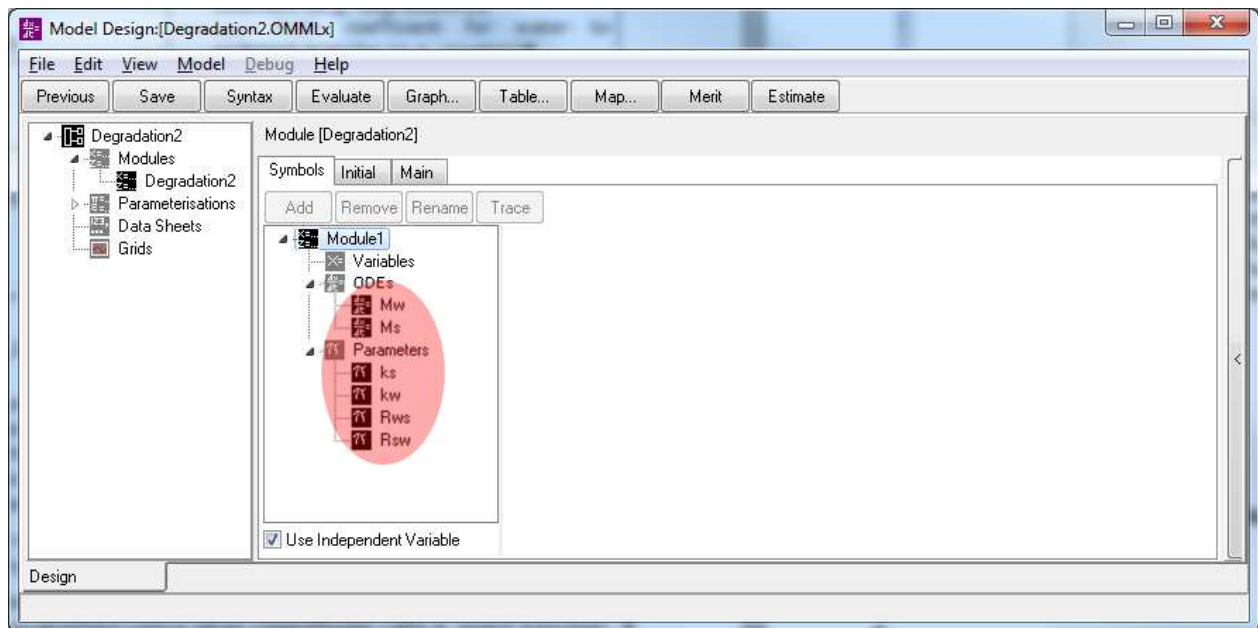
3.2.1 Create the Model Script

Working in steps:

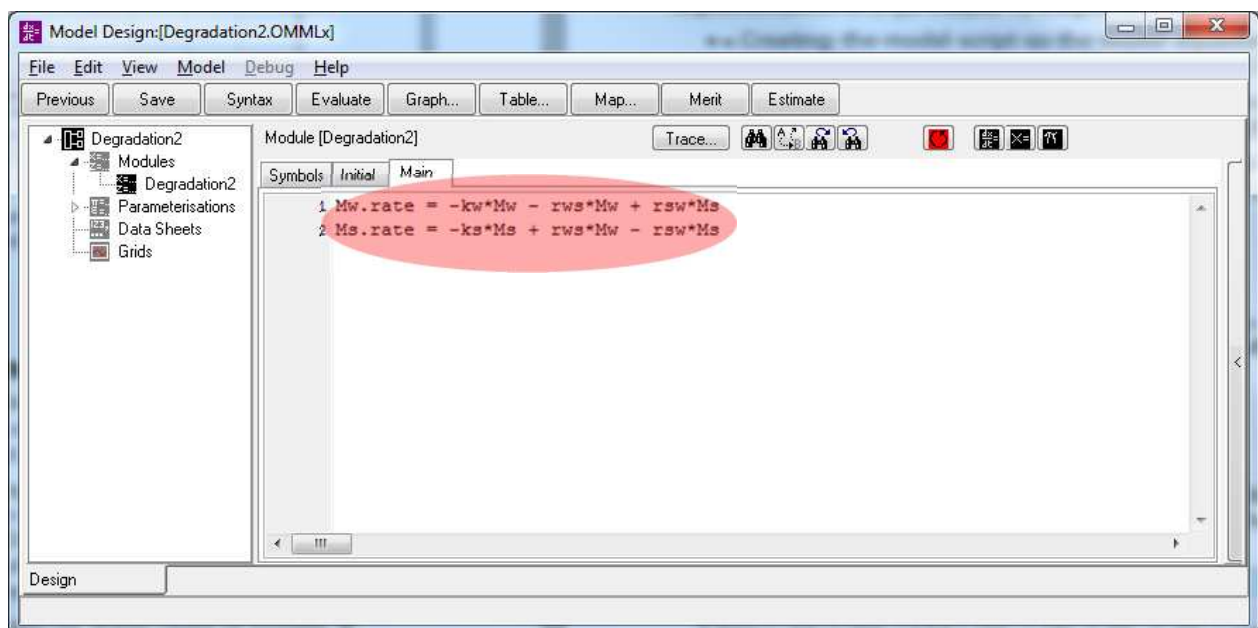
1. The very first step is to create a new model (**File|New**) and save it as Degradation2.OMMLx (**File|saveAs**). OpenModel now displays an empty model script window as shown below.



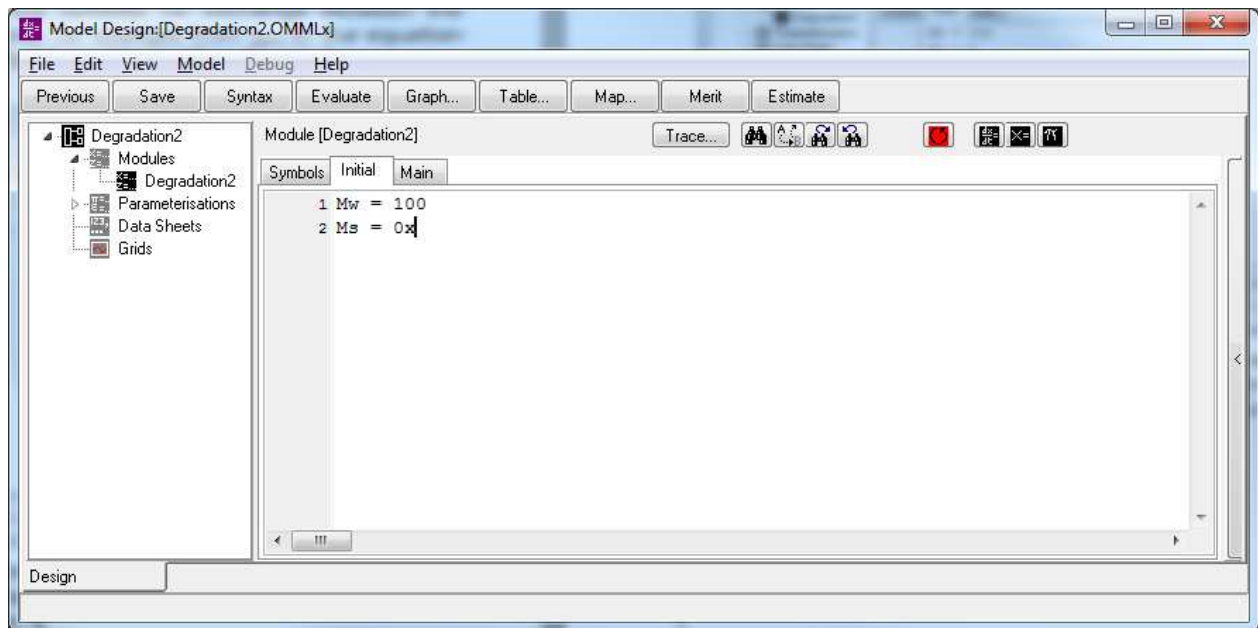
2. The model has 2 differential equations (M_w and M_s) and 4 parameters (k_s , k_w , r_{s-w} , r_{w-s}). We now introduce these differential equations and parameters into the model so you can use them in the model script in due course. To do this use the **Add Differential Symbol** and **Add Parameter Symbol** buttons on the script tool bar as indicated above. These are used once for each symbol, so twice for the differential equations and 4 times for the parameters. The new symbols are shown below in the symbol tree view of the model design. Note that the symbols for exchange between the compartments have been slightly changed, for example from r_{w-s} in the equation definition, to **rws**. This is because OpenModel would not allow the '-' character in a symbol name.



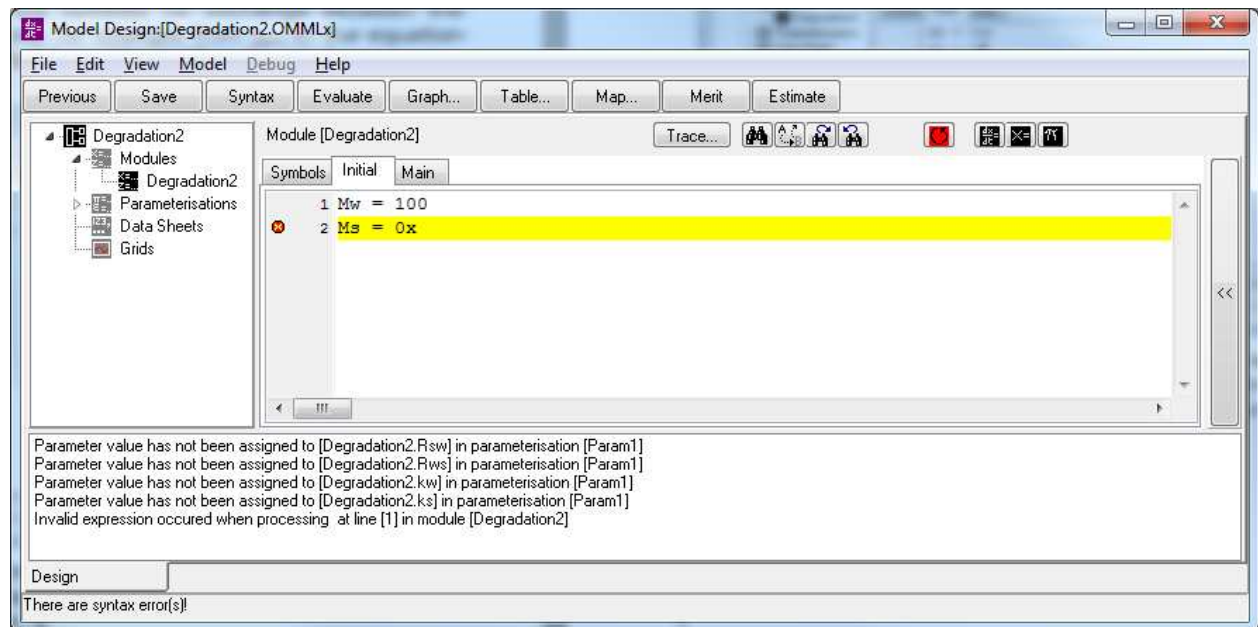
- Now enter the equations for the 2 differential equations, as specified earlier. Note that the OpenModel syntax for dMw/dt is **Mw.rate**. The differential equations are entered into the 'Main' view of the module.



- The script specified in the Main view is used to solve the equations over the required time period (we will specify this in a moment). The model script is evaluated iteratively to solve the differential equations using numerical methods called 'initial value' methods. As the name implies these methods require an initial value to be specified for each differential equation. These initial values are specified in the 'Initial' view as shown below. The script in the initial view is evaluated just once, before the main script. In this example the initial (i.e. $t=0$) situation is that 100 units of chemical is present in the water with nothing in the sediment. The model will calculate progress from this condition.

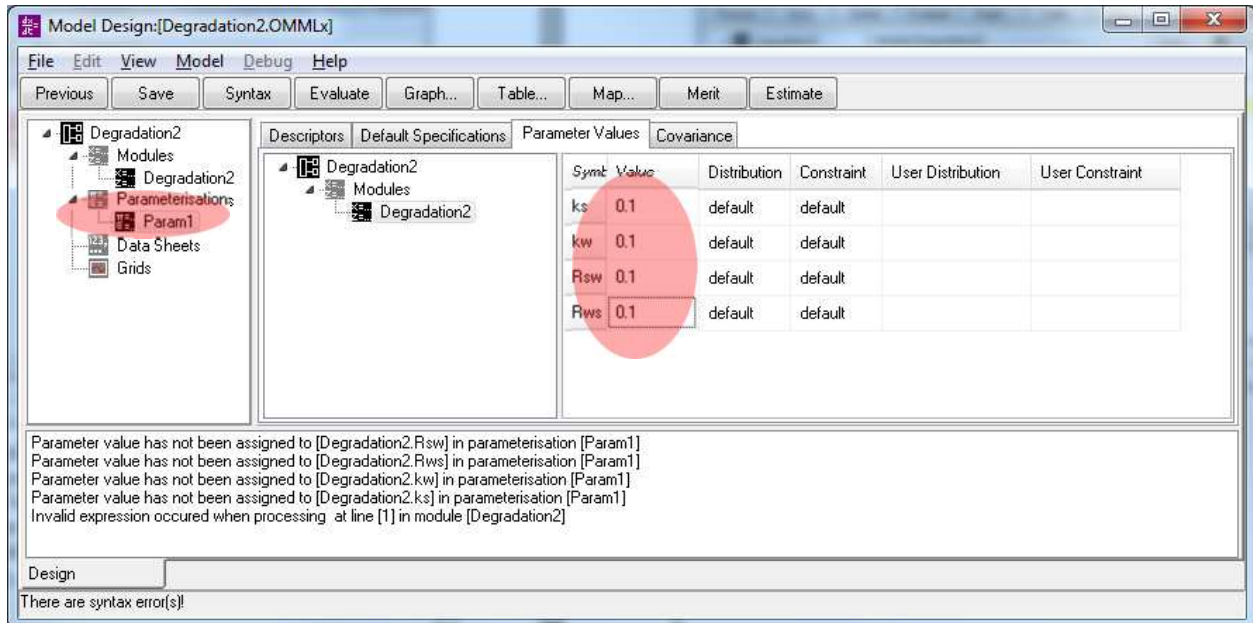


5. We are now ready to check the syntax of the model and see if we have made any mistakes (little mistakes are perfectly normal). Click the **syntax** button on the tool bar. The model will be checked and compiled. Problems are reported as a list below the script view and if they are syntax errors in the script they are also highlighted in the displayed script. Double clicking on an error in the error list will take you to the position in the script where the error has been found. In this case there is a syntax error in the script, a value of **0x** has been entered as the initial value of **ms**, rather than 0. This is easily corrected in the script. OpenModel has also told you that you have not defined the values of the four parameters. This is not an error as such but OpenModel can't evaluate the model unless you supply values for these parameters.

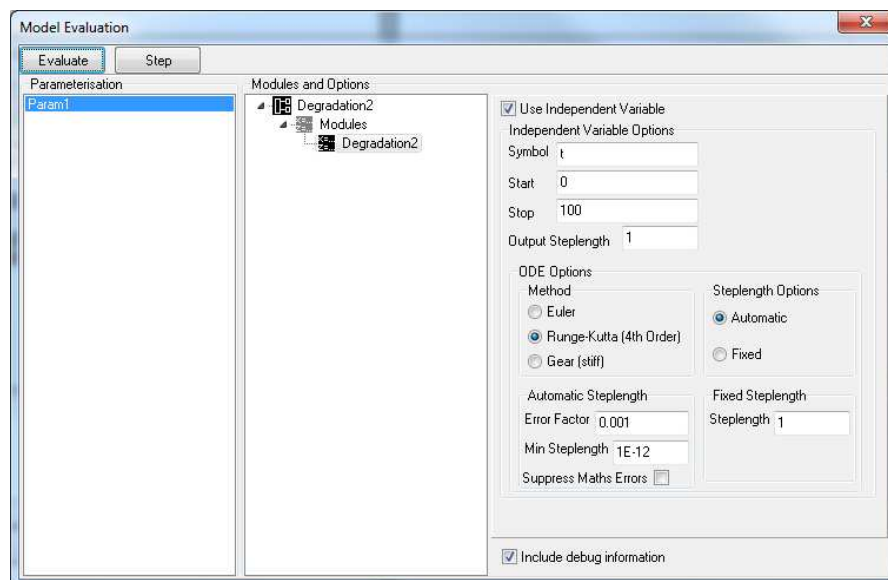


6. Users undertaking this type of work for the first time are often perplexed at this point, why do you have to give the values of the parameters? After all the aim of the model was stated to be estimating the parameter values! We don't know what they are! We have to make a guess so the model can be evaluated; quite likely this will give an awful agreement between model and observation. However OpenModel's fitting procedures can then adjust the parameter values to improve the fit. In principle any guess will do, however a sensible choice is something that gives

changes over roughly the right timescale. Our data runs over a period of 100 days or so, therefore rate coefficients of between 0.1 and 0.01 might be reasonable first guess. Expand the Parameterisation node in the main model tree view and select the default parameterisation **Param1**. The current values of parameters will be displayed. These will be shown as missing, but you can simply enter a value for each one (0.1 in the screenshot below).

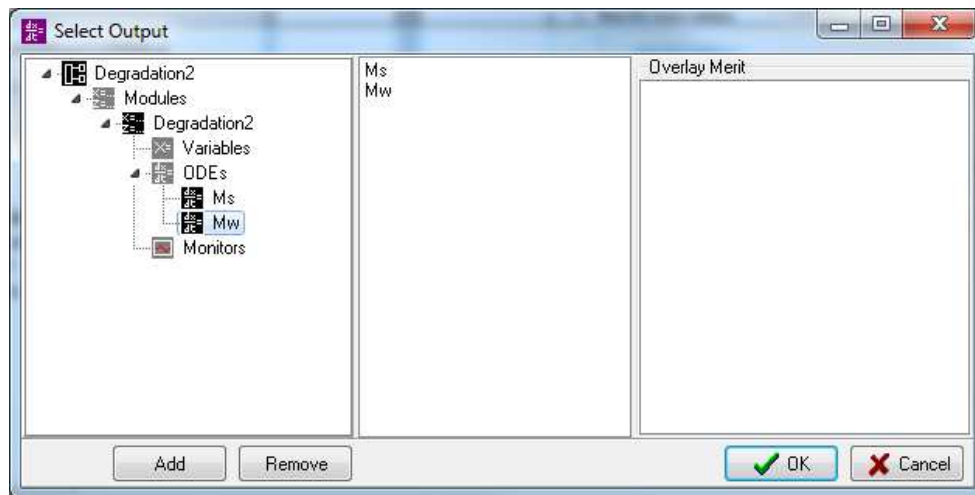


7. We can now evaluate the model. Select **Model|Evaluate** from the menu or click the **Evaluate** button on the tool bar. This will display the Model Evaluation dialog as shown below. There are various options here, for now the key ones are the **start** and **stop** times (default values are 0 and 100 respectively). The model will run from t=0 to t=100, let's leave those values for now and click **Evaluate**.

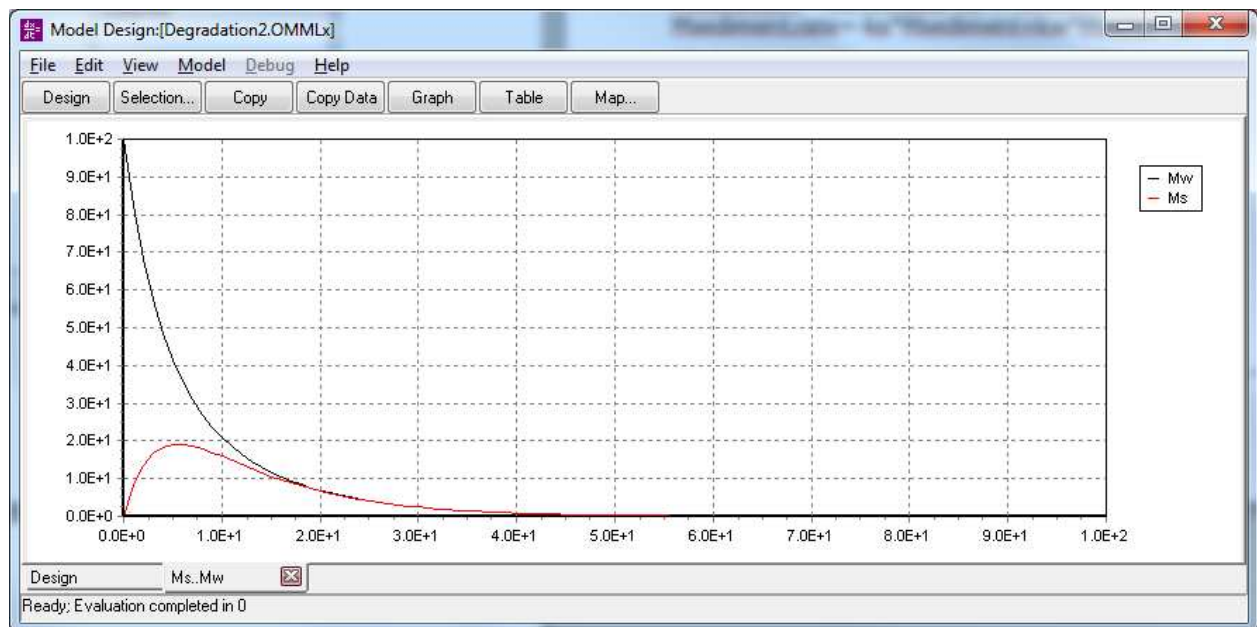


8. When the model runs various messages are displayed on the status line, and after a few moments (hopefully) OpenModel reports that it has completed the model evaluation. You can now display graphs and tables of the calculated values. To create a graph click the **Graph** button on the tool bar or select **view|Graph** from the main menu. This will display a dialog box for selecting what you would like to 'graph'. The full model is shown in the left hand tree, select symbols and add them to the middle

list (double click the symbol in the tree, or click the **Add** button). In the example below both **Mw** and **Ms** have been selected.



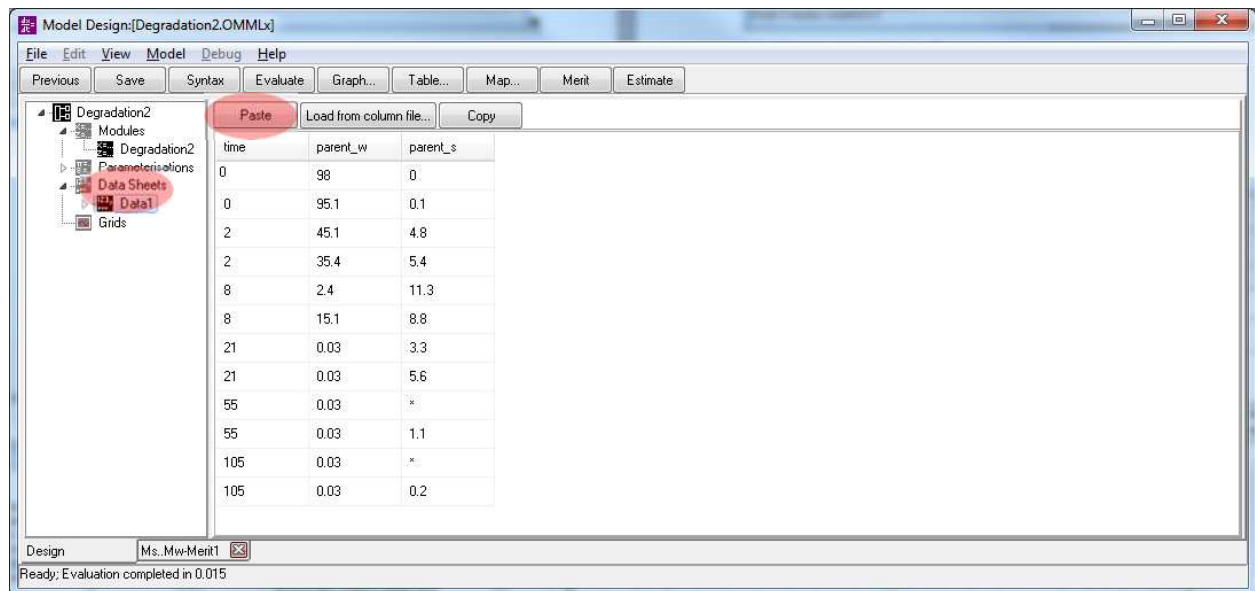
9. Clicking **OK** in the selection dialog will create the graph, as shown below.



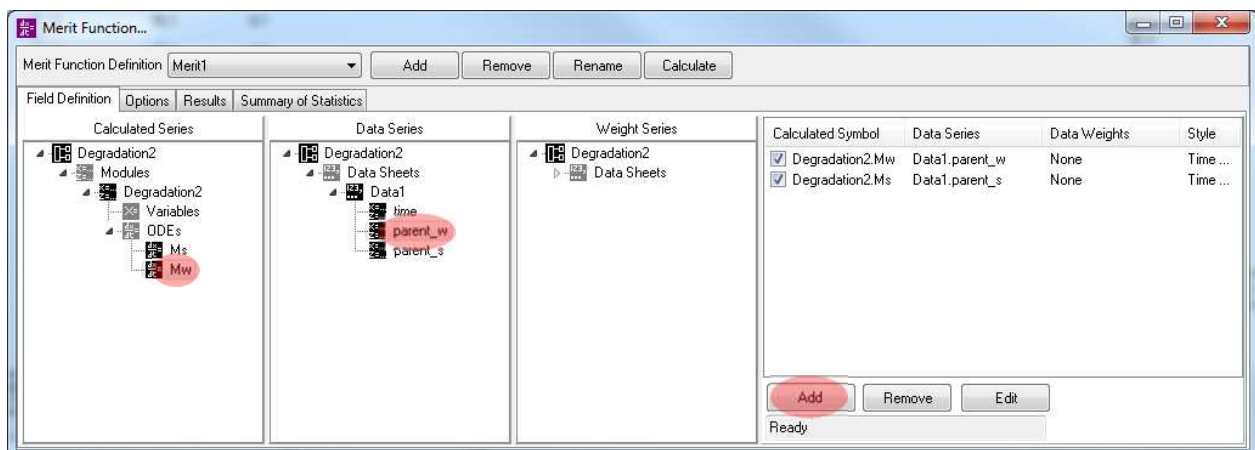
10. We have created the model, run it, and seen the results. However this is with arbitrary parameter values, next we need to connect the observational data and compare the predictions of our model with reality.

3.2.2 *Attach the Observations*

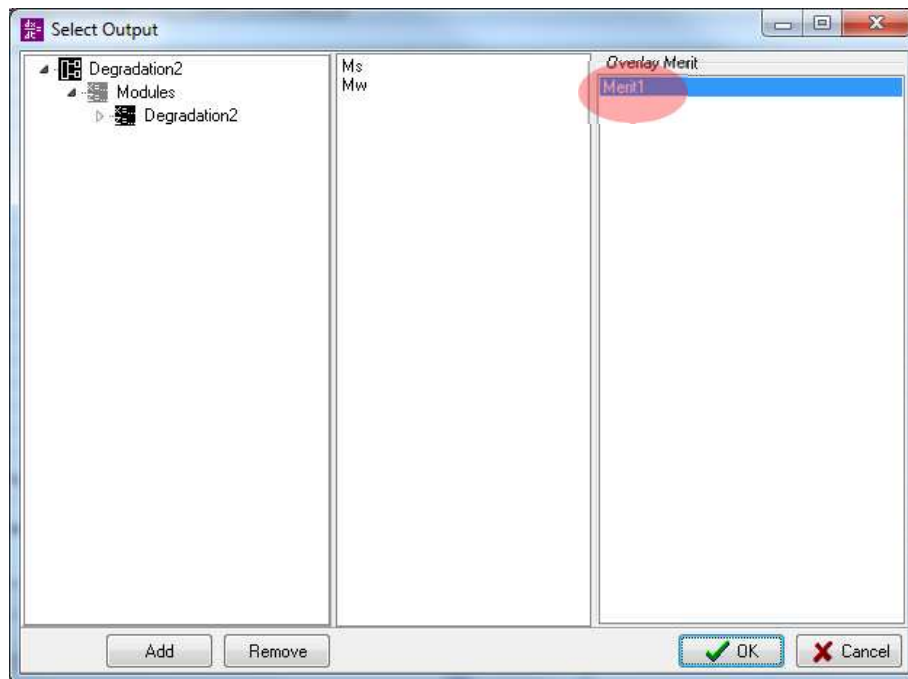
11. The observed values are included in the model using data sheets. Right click on the Data Sheets node in the main model tree view and select **Add Data Sheet**. A blank data sheet will be created and displayed. The simplest method for adding the data to the sheet is to copy it from excel and paste using the **Paste** button on the data sheet tool bar.



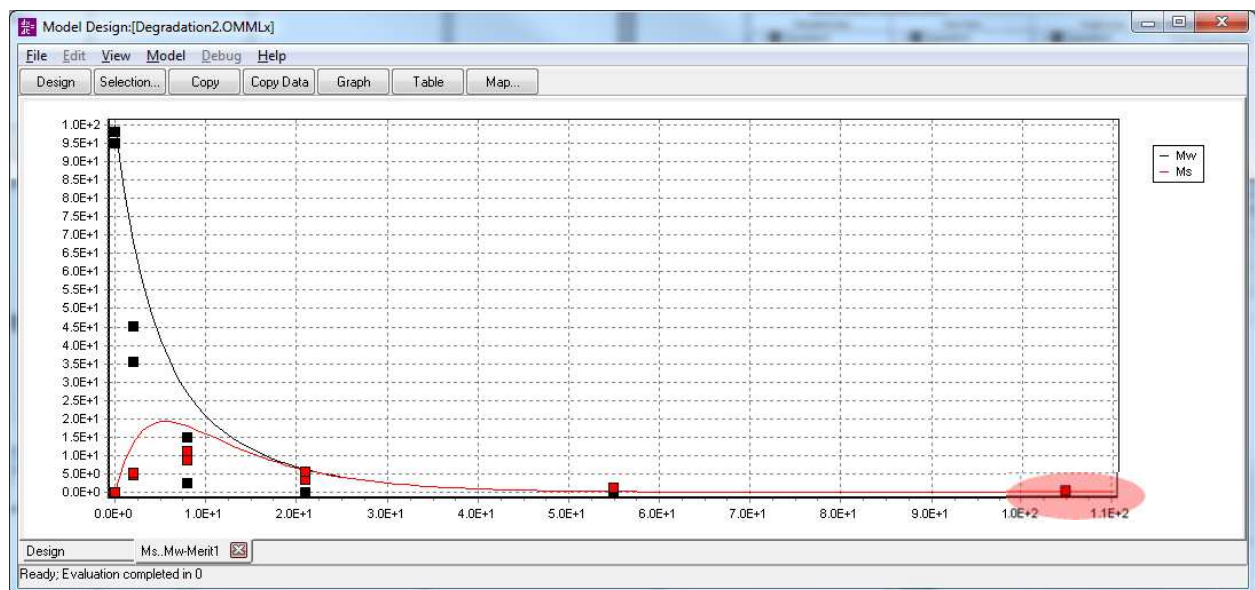
12. OpenModel does not 'know' what this newly added data is for, we have to connect the data columns with the symbols in the model. This is accomplished using a Merit object. Merit objects serve two purposes. They define which data column relates to which model symbol. They also can be used to specify how the agreement between model and observation is to be calculated (e.g. residual sums of squares, weighted sums of squares). For now we will just associate the observations and predicted symbols. Click Merit on the tool bar, or select **view|Merit** from the main menu. This will display the merit dialog box as shown below. The calculated series view shows the model calculated symbols, the data series the available data series. To associate **Mw** with **parent_w** select one, then the other, and click **Add** (or equivalently double click one of the selected nodes). The associated series are listed in the right hand view.



13. We can now overlay the observed values on the model predictions in the previously created graph. Double click the graph, this will display the selection dialog. Select **merit1** as the overlay merit.

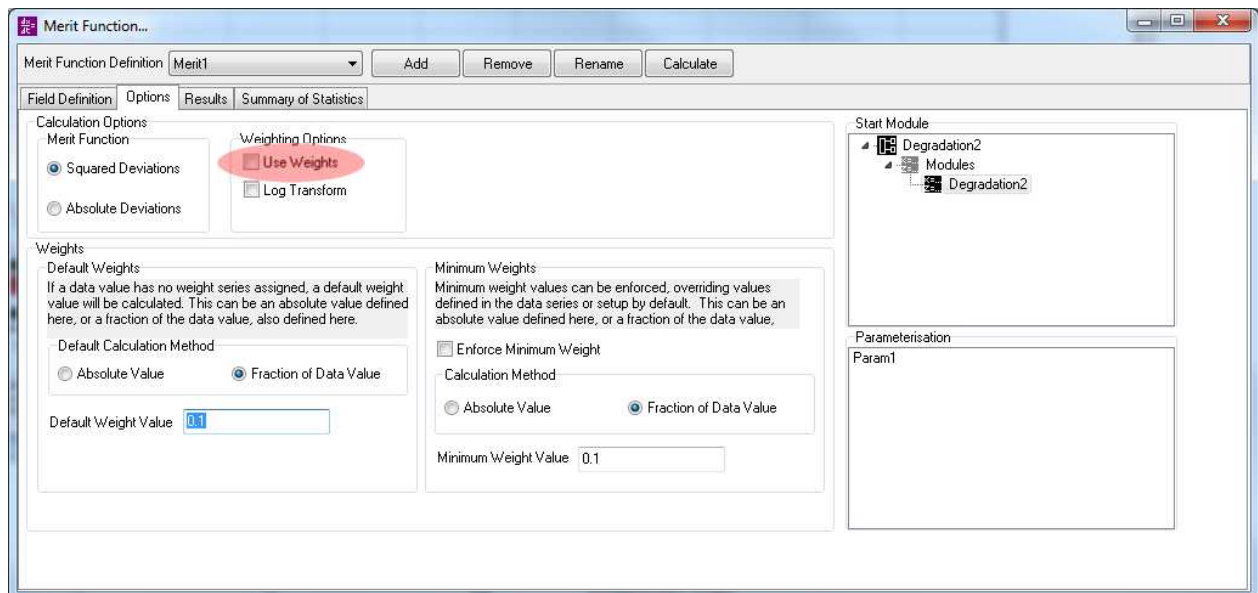


14. The observations will be overlaid on the model predictions as shown below. Note in the graph below the model has been reevaluated with a stop time of 110 to fully encompass the time range of the data.

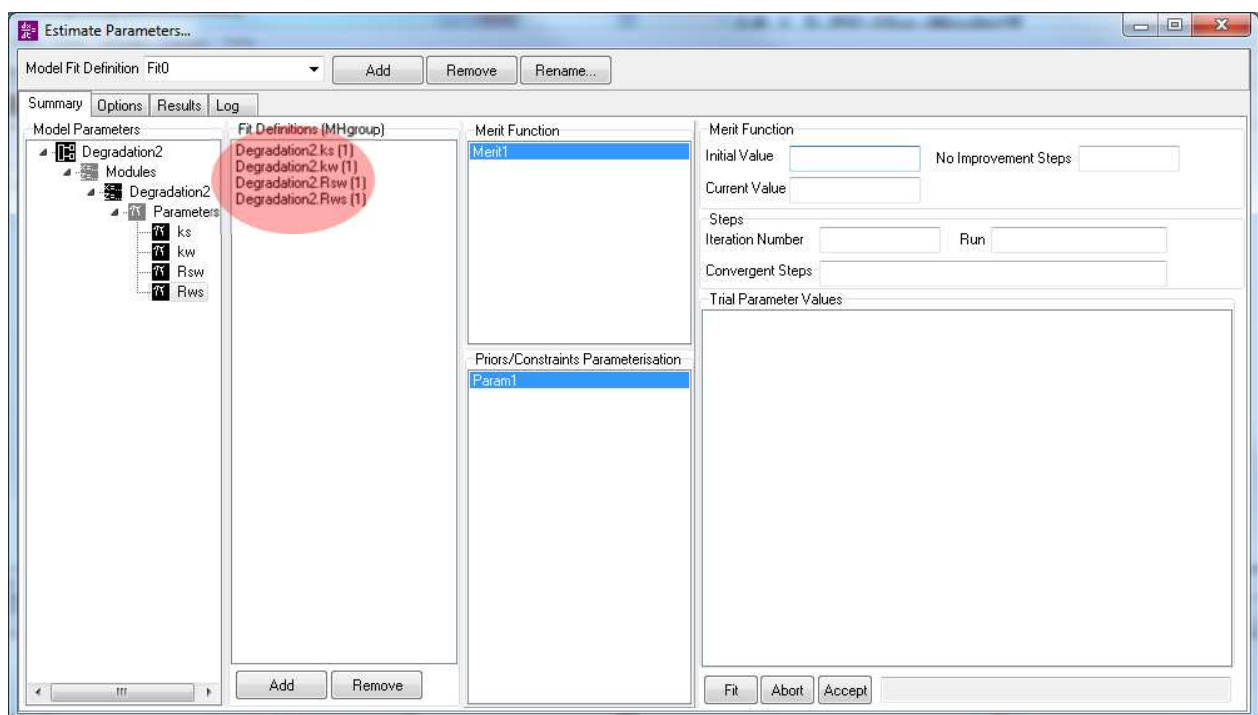


3.2.3 Fit the Model

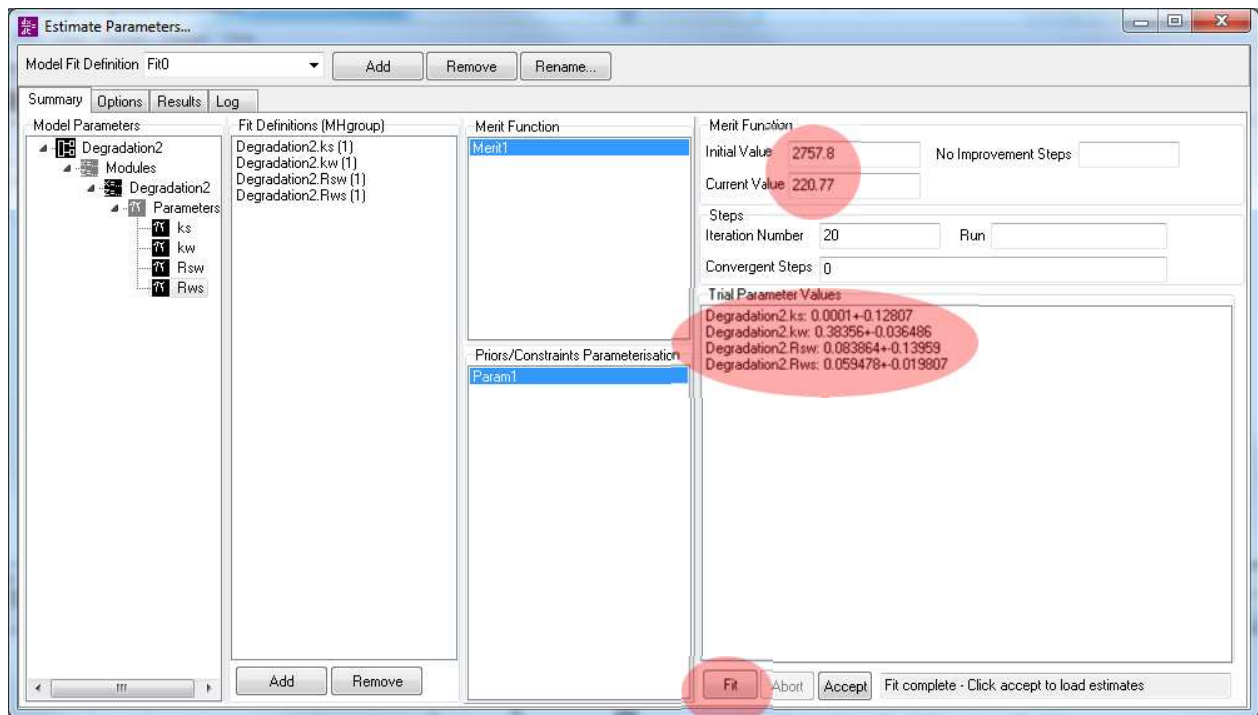
15. Now we will estimate the parameter values by fitting the model. To do this we create a fit object which defines which parameters we wish to estimate (along with various fitting method options). The fitting procedure tries to minimise the deviation between the model predictions and the observed values. The deviation is calculated by the merit object; fit and merit objects are used in combination. A feature of OpenModel is that you can define any number of fit and merit objects and use them to readily explore the effect of different fitting arrangements on your findings.
16. For this example we will fit the model by minimising the residual sums of squares (very conventional). By default OpenModel merit objects assume a weighted residual sums of squares. So we should revisit the merit function specification and switch off **Use Weights** as shown below.



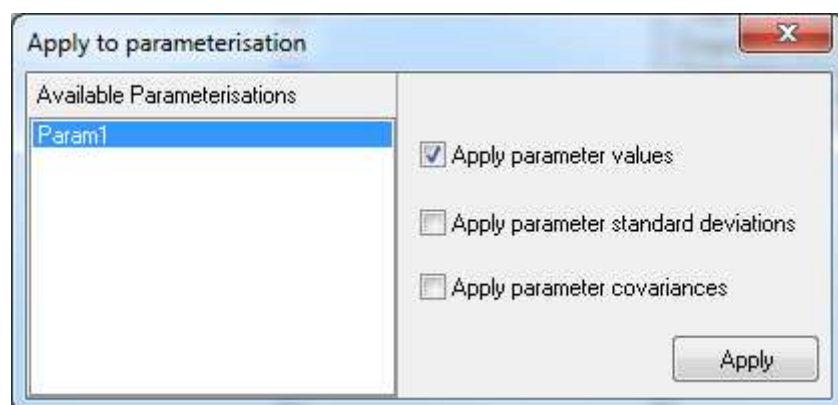
17. Next we set up the fit object. Select **Model|Estimate Parameters** from the main menu or click the **Estimate** button on the toolbar. The available model parameters are shown in the left hand tree view. Select the node for parameters you wish to estimate (either click the **Add** button or double click the node). The selected parameters will be added to the fit definitions list. This shows which parameters will be adjusted when the fit procedure runs.



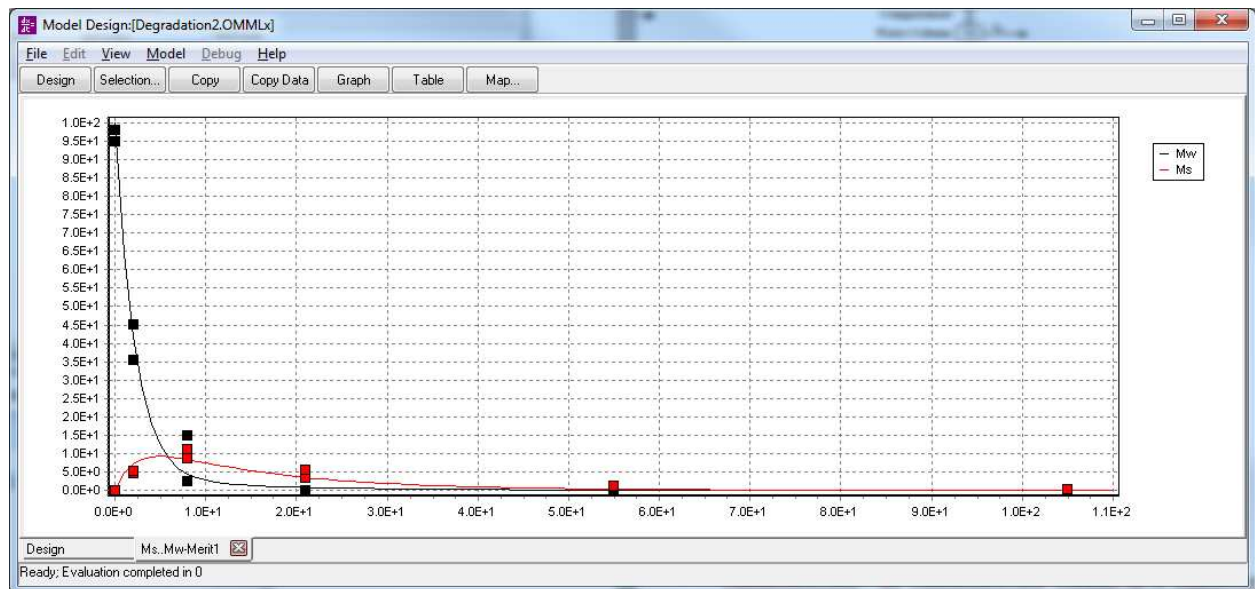
18. The fit object has several different methods and related options. For now we will simply use the default set up. Click the **Fit** button and the fitting procedure will commence. All the fitting methods are iterative and you will see progress reported within the dialog. Hopefully you will see the values of the merit function (residual sums of squares in this case) go down and when the fit is complete the final parameter estimates are shown in the trial parameter list.



19. If you are content with the estimated parameters and wish to use them to run the model you will need to accept them into a model parameterisation object. OpenModel can have any number of 'parameterisations', each of these can specify a different set of parameter values (and their distributions). It can be very convenient to have alternative parameterisations readily available, for example, you may wish to try these new parameters while also retaining the initial estimates. However for the purposes of this example we will keep things simple and simply update the existing model parameters with the new estimates. This requires us to click the **Accept** button in the Estimate Parameters dialog to display the dialog shown below. This lists available parameterisations (we have just one, but there can be any number) and when we click the **Apply** button the newly estimated values will be written into the selected parameterisation.



20. We can now re-evaluate the model and see whether the revised parameterisation gives us more satisfying results. The previously created graph will be automatically updated when the model is run enabling you to make a visual assessment of the model's goodness of fit.



21. In addition to the visual assessment of the model's performance in reproducing the observations you are likely to want to use some more quantitative measures such as Nash-Sutcliffe, AIC etc. A range of statistics are available in both the merit function and fit object results views (more details in the user guide).

3.2.4 Apply the model

After all that effort to make the model and fit it to the observational data you probably have some purpose in mind? What can we now use it for?

Continuous Input

One possible application is to use the parameterised model to assess the environmental levels of the chemical which will occur if there is a continuous and constant rate of release into the water column.

The equations of the model are modified to

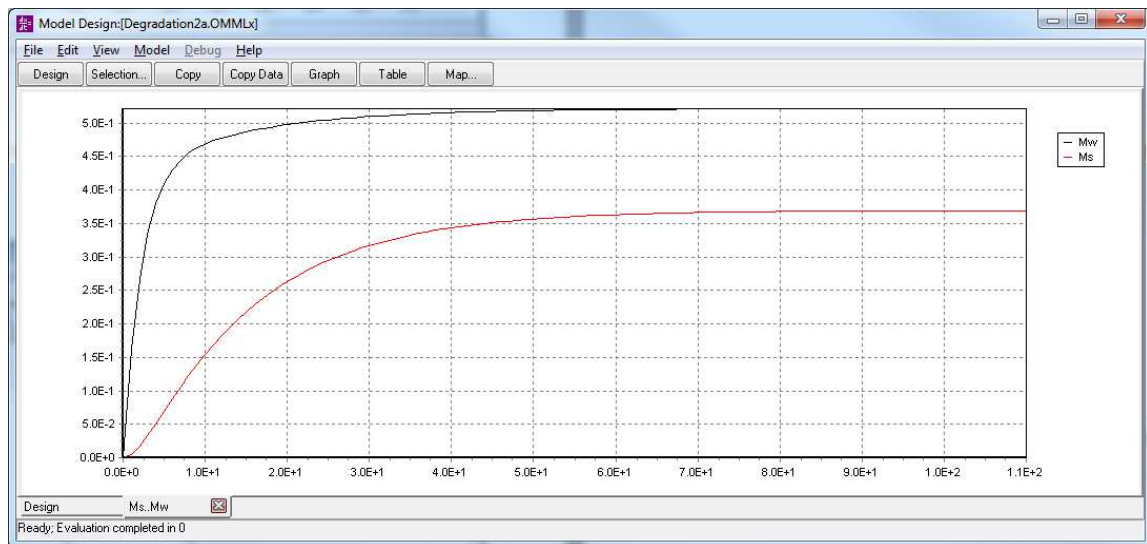
$$\frac{dM_w}{dt} = -k_w M_w - r_{w-s} M_w + r_{s-w} M_s + \text{Input}$$

$$\frac{dM_s}{dt} = -k_s M_s + r_{w-s} M_w - r_{s-w} M_s$$

Where **Input** is a specified constant value.

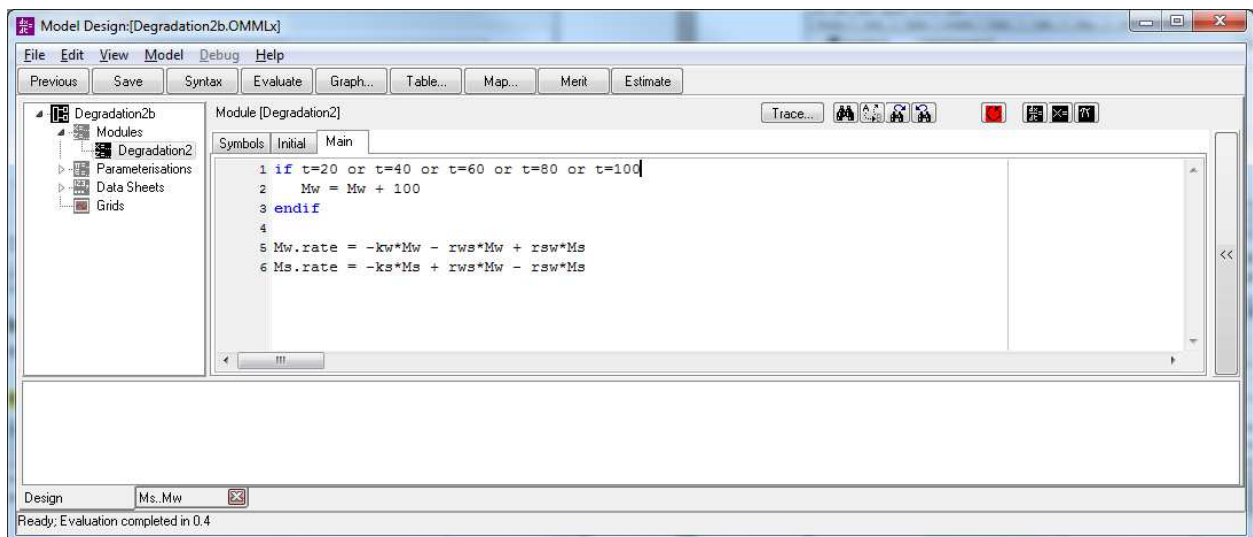
The model file **Degradation2a.OMMLx** implements this variation of the model for the case of **Input** = 0.2 per unit time.

The resulting behaviour of Mw and Ms are shown below.

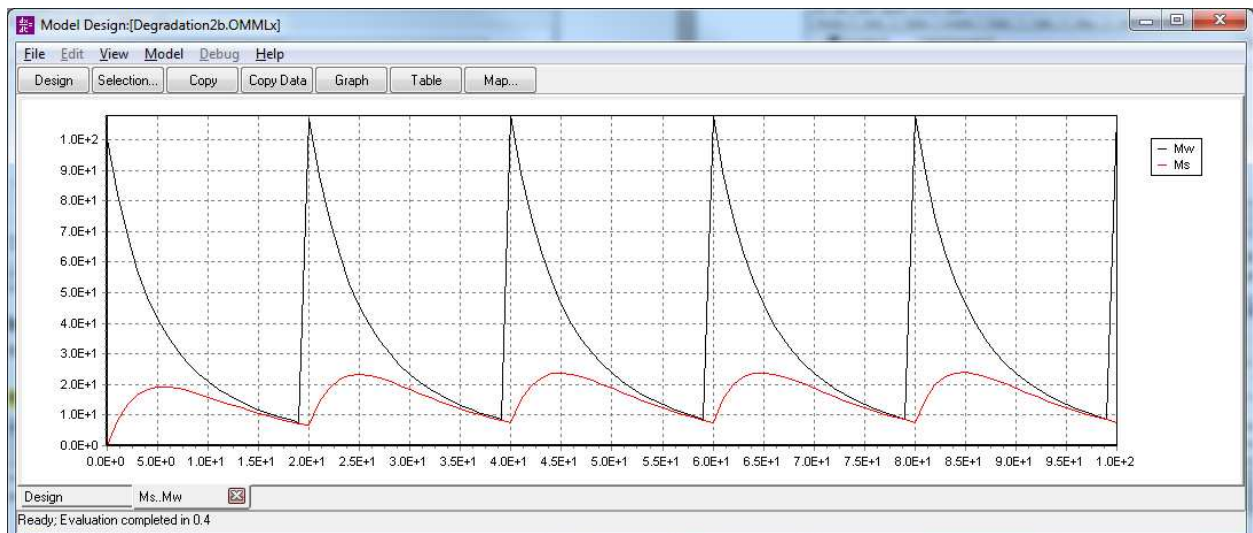


Multiple Inputs

Another variation is to consider the effect of regular inputs of the contaminant chemical, for example the Main script for **Degradation2b.OMMLx** has been modified to add a 100 units of chemical to the water compartment (M_w) every 20 time units.



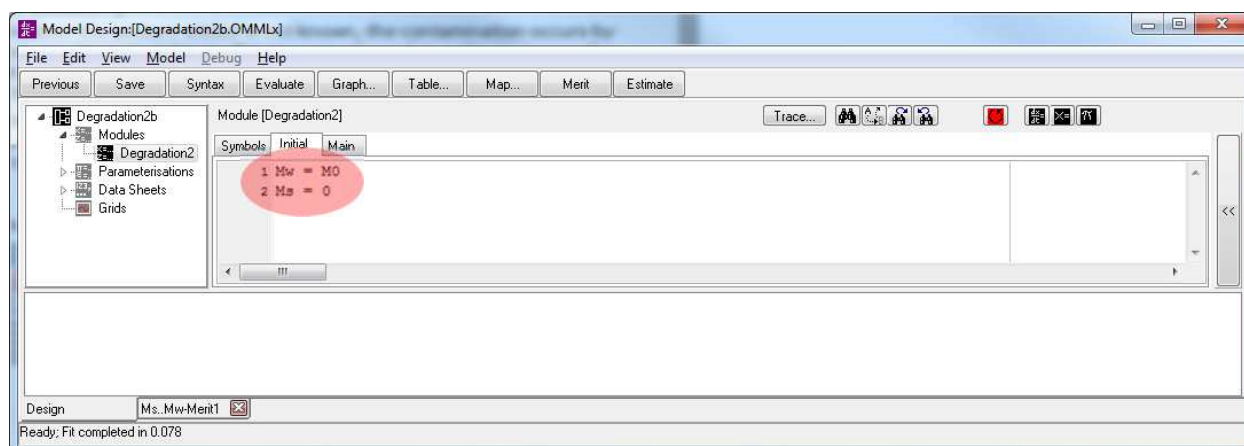
The resulting prediction for M_w and M_s is shown below; a succession of decaying pulses which have established a repeating pattern after the 3rd input event.



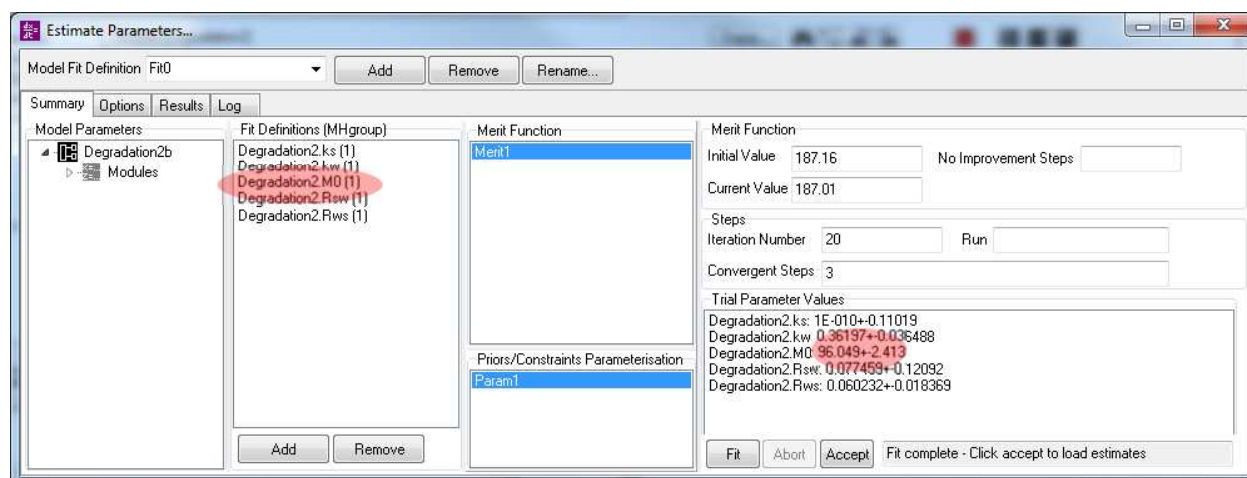
How much was added?

Maybe in our application the amount added is not known, the contamination occurs by some accident and our aim is to estimate the amount released in order to assess the negligence of whoever was responsible! We can do this using our subsequent measurements of the chemical in the system.

Degradation2c.OMMLx implements this case, extending the fitting described earlier to include an additional parameter M_0 which is used as the initial value of M_w as shown in the screenshot of the **Initial** script below.



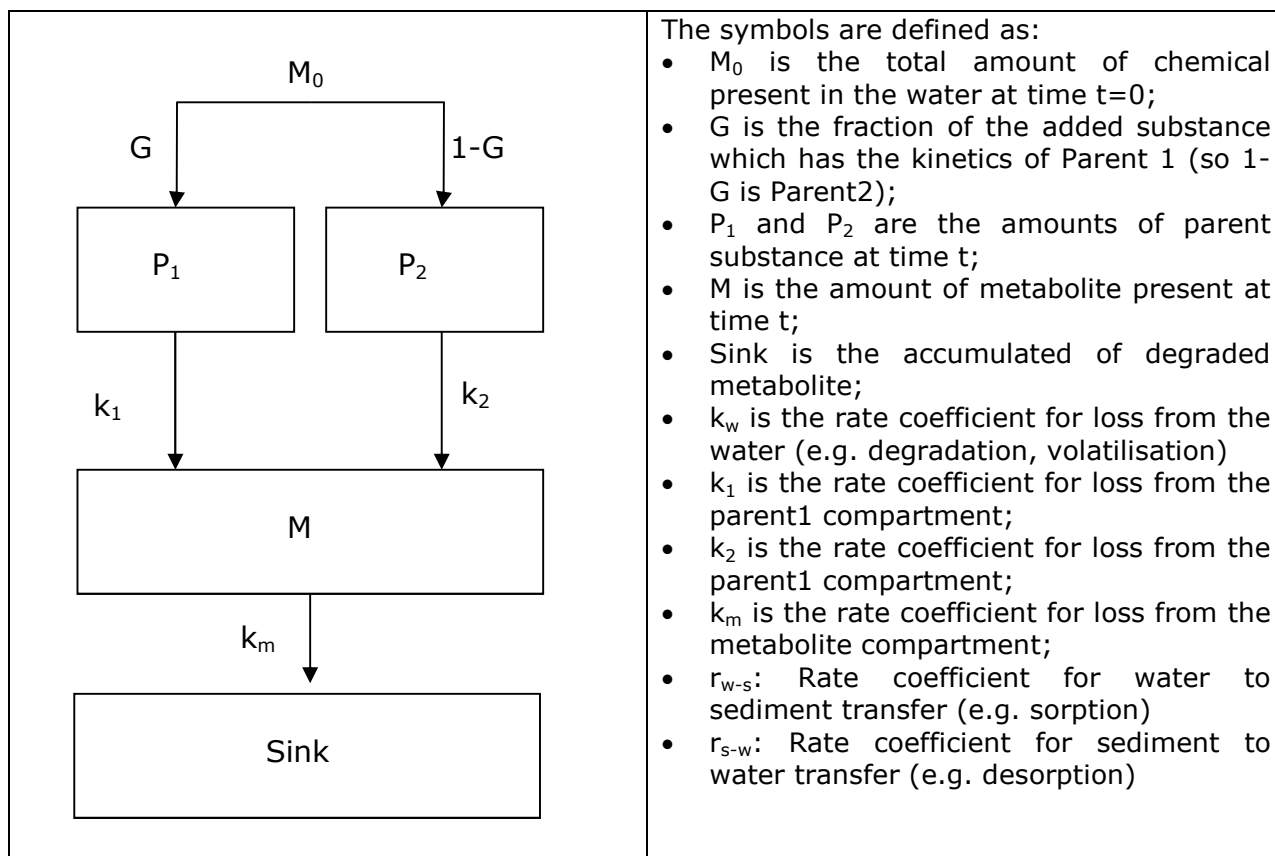
We now have 5 adjustable parameters and it turns out the M_0 is estimated to be 96 ± 2.4 , quite close to the value of 100 we used before.



3.3 Parallel Kinetics

In this example we consider a substance added to a system where there are 2 parallel degradation pathways to a metabolite form (M). The 2 pathway as are represented by 2 parent compartments (P_1 and P_2). The substance is partitioned between the 2 parent forms by a fraction α . Losses from the parent compartments and the subsequent metabolite compartment are described by rate coefficients k_1 , k_2 , and k_m . In the model a sink compartment is included, this is not really needed but is useful to give us a cross check against the behaviour of the model (it can be useful to check that after a long time $sink = M_0$).

As before, it is useful to have a conceptual diagram for these types of models.



The example model is in **Degradation3.OMLx** and the model scripts are shown below. The partitioning fraction G is implemented in the initial script.

Main script

```

Parent1.rate = -k1*Parent1
Parent2.rate = -k2*Parent2

Metabolite.rate = k1*Parent1 + k2*Parent2 - km*Metabolite

Sink.rate = km*Metabolite

```

Initial script

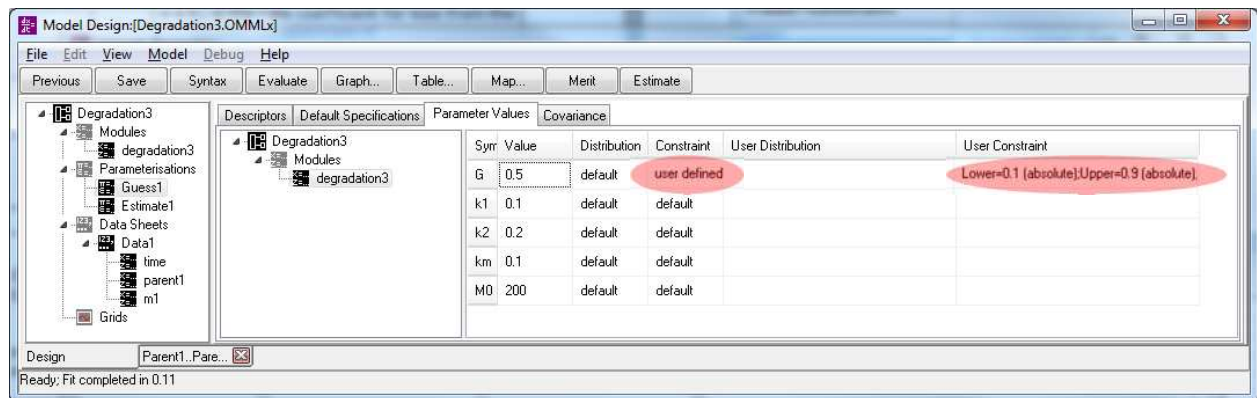
```

Parent1 = G*M0
Parent2 = (1-G)*M0
Sink     = 0

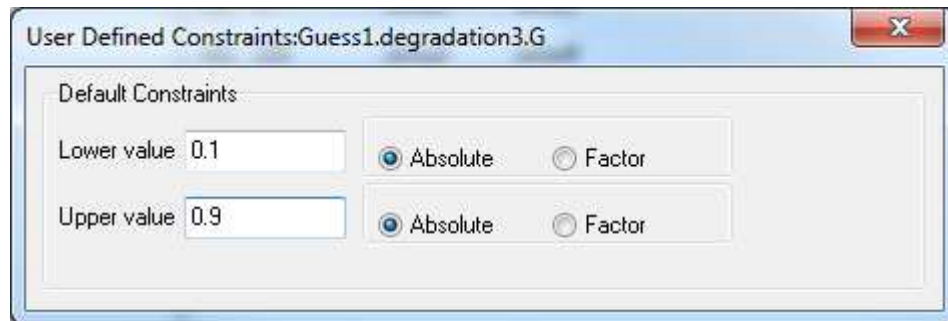
```

The model is parameterised using observed data for **Parent1** and **Metabolite** (observations for **Parent2** are not available). The parameter for **Parent2** degradation, k_2 , is therefore inferred in the fitting of the other model compartments.

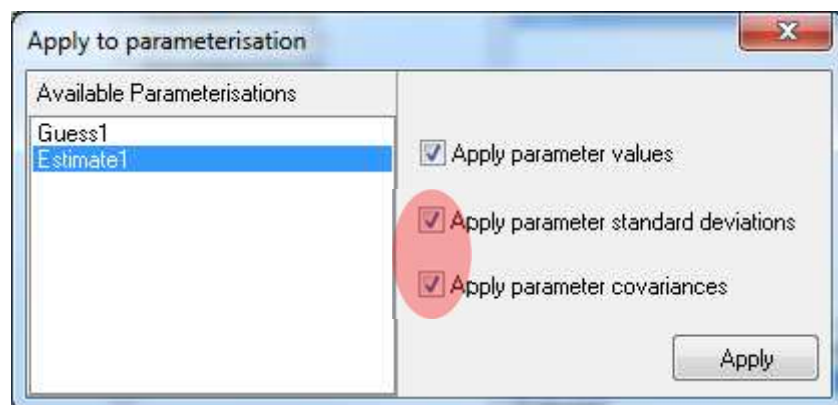
G is intended to be used as a fraction, values outside the range of 0 to 1 would not be physically reasonable. OpenModel does not know this so we define a constraint for the parameter within the 'Guess' parameterisation. Double click on the 'Constraint' cell to change its state to 'User Defined'.



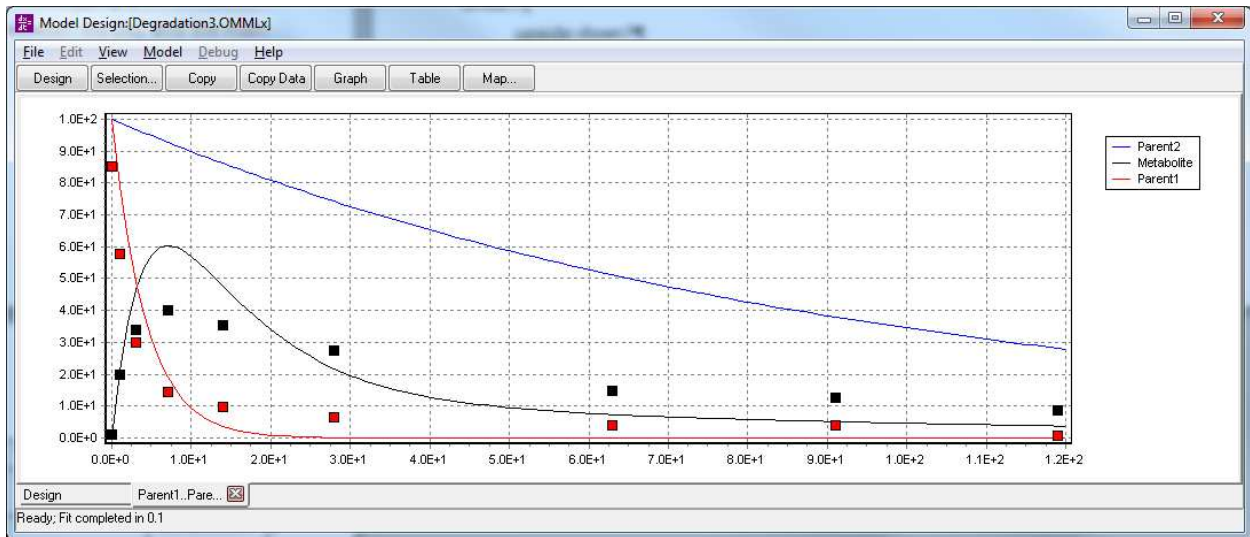
Double clicking on the 'User Constraint' cell displays the dialog box shown below allowing upper and lower constraints to be specified. As you can see we have set the range to be 0.1 to 0.9 rather than 0.0 to 1.0. Why? It is all too easy for the fitting procedure to get 'stuck' at values of 0.0 and 1.0 as they have the effect of removing one of the compartments. We therefore slightly over-constrain the parameter. The precise value should not be critical, 0.05 to 0.95 works equally well (you could check this).



The implemented model has 2 parameterisation objects, **Guess1**, which as the name suggests is the initial estimates. This was then copied (by right clicking it in the tree) to create another parameterisation object which we named **Estimate1**. The fitted parameters were 'Applied' to the **Estimate1** object together with their standard deviations and co-variances.



Running the model using the **Estimate1** parameterisation gives the agreement between prediction and observation shown below.



3.4 Temperature and Moisture Dependent Degradation

This model (**Degradation4.OMMLx**) is another extension of the single first order model introduced in 3.1. A chemical product, applied to the ground, is degraded. The degradation rate k depends on temperature and moisture (previously k was regarded as a constant).

The overall model equation is the same as before

$$\frac{d\text{Parent}}{dt} = -k(T, M)\text{Parent}$$

where the rate coefficient k (d^{-1}) is a function of temperature (T) and moisture content (M). There are numerous methods for describing temperature and moisture effects, in this case a combined exponential factor is used

$$k(T, M) = k_{\text{ref}} \exp\left(\frac{E_a(T - T_{\text{ref}})}{RTT_{\text{ref}}}\right) \left(\frac{M}{M_{\text{ref}}}\right)^B$$

Where:

T is the temperature (C)

M the moisture content (m^3 water m^{-3} soil),

k_{ref} is the degradation rate at a standard temperature water moisture content (d^{-1})

E_a the activation energy for the process (J mol^{-1})

T_{ref} the reference temperature (C)

R the gas constant ($\text{J mol}^{-1} \text{K}^{-1}$)

M_{ref} the reference soil moisture content (m^3 water m^{-3} soil)

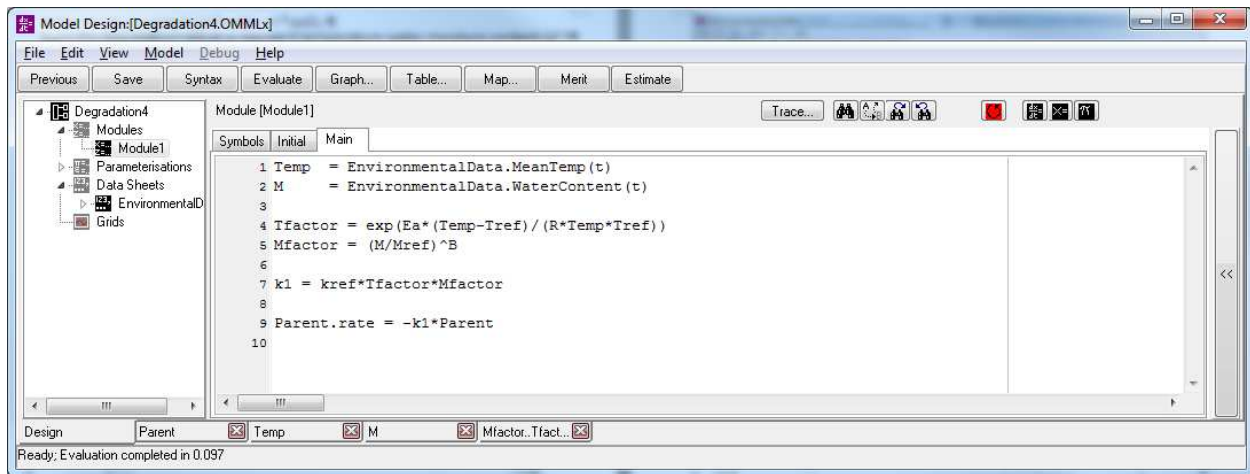
B exponent for the moisture effect

In the OpenModel implementation k_{ref} , T_{ref} , M_{ref} , E_a , R , and B are model parameters. Of these R is a universal constant and does not require situation specific estimation.

The model equations are readily implemented in OpenModel. Previously k_1 was a parameter, in this case it becomes a variable in order for it to vary with temperature and soil moisture. It is helpful to introduce separate factors for the effect of temperature (**Tfactor**) and moisture (**Mfactor**) in order to see their individual effects. Note that the

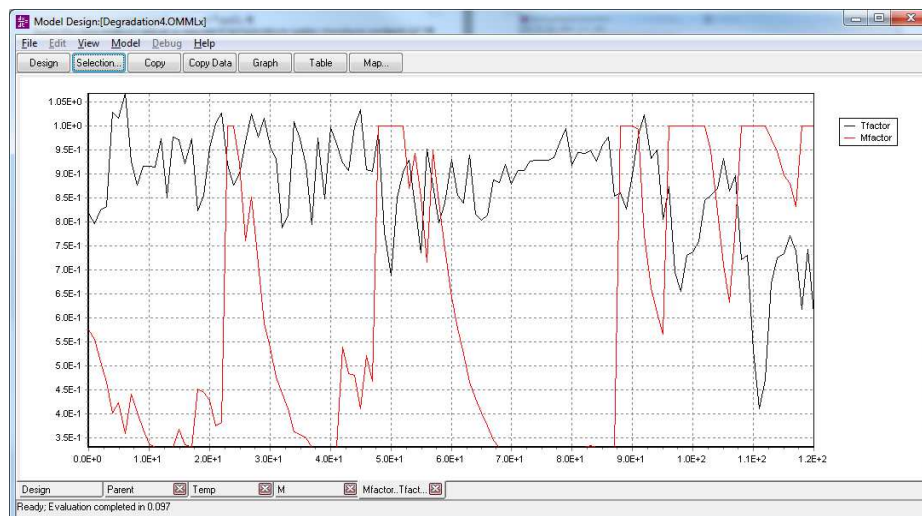
temperature is represented by variable **Temp**, not **t**. This is done because **t** is the symbol for the models independent variable, time.

The model has external driving variables, **Temp** and **M**, these are set to values held in a data sheet.



The data sheet values are included in the example file included in the OpenModel installation. For the purposes of the example we have assumed that the degradation is controlled by environmental conditions in the upper 10cm of soil.

Plotting **Mfactor** and **Tfactor** you can see that the degradation process is strongly controlled by the environmental conditions in this case. The effect of moisture is especially significant.



The overall effect on the amount of chemical present is shown below, the underlying exponential decline remains visible but is overlaid with environmentally controlled fluctuation.



4. Radiocaesium in the tissues of sheep

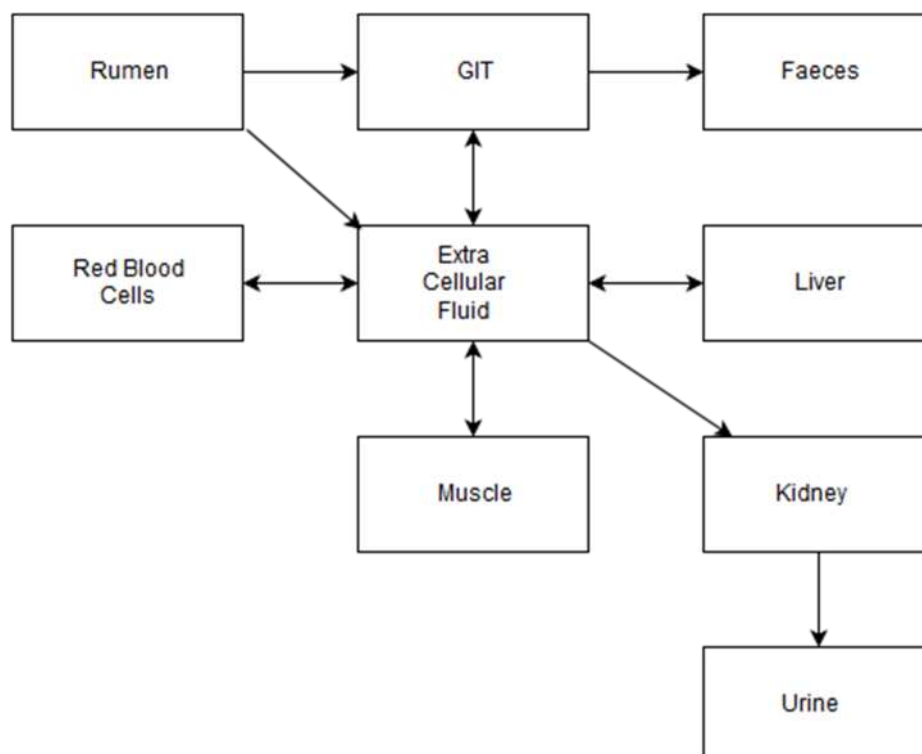
This example uses OpenModel to solve a set of simultaneous differential equations and fits unknown rate coefficients (model parameters) to observations. It provides a good illustration of some of the practical issues that arise when using observed values for fitting and demonstrates the use of multiple parameterisations in OpenModel.

4.1 Background

Radiocaesium can be released into the environment following nuclear accidents (or, heaven forbid, by the detonation of nuclear weapons). It is harmful and readily transferred through the food chain. After the Chernobyl nuclear accident in 1986 there was widespread radiocaesium contamination of the landscape and regulatory controls were introduced for various food products. In particular grazing animals were affected in large parts of Europe.

4.2 Model Overview

The sheep radiocaesium model we consider was presented by Galer et al (1993). A series of inter-connected compartments are used to represent the various tissues of the animals. Experimental measurements were made of the radiocaesium present in different tissues and those measurements are used to fit the model. The model is now rather old but nevertheless makes a good example for the use of model fitting in OpenModel. It is interesting to note in re-implementing the model that a 'better' parameter set is obtained than that presented by Galer et al way back in 1993. They did not have the benefit of readily available model fitting!



The corresponding model equations are

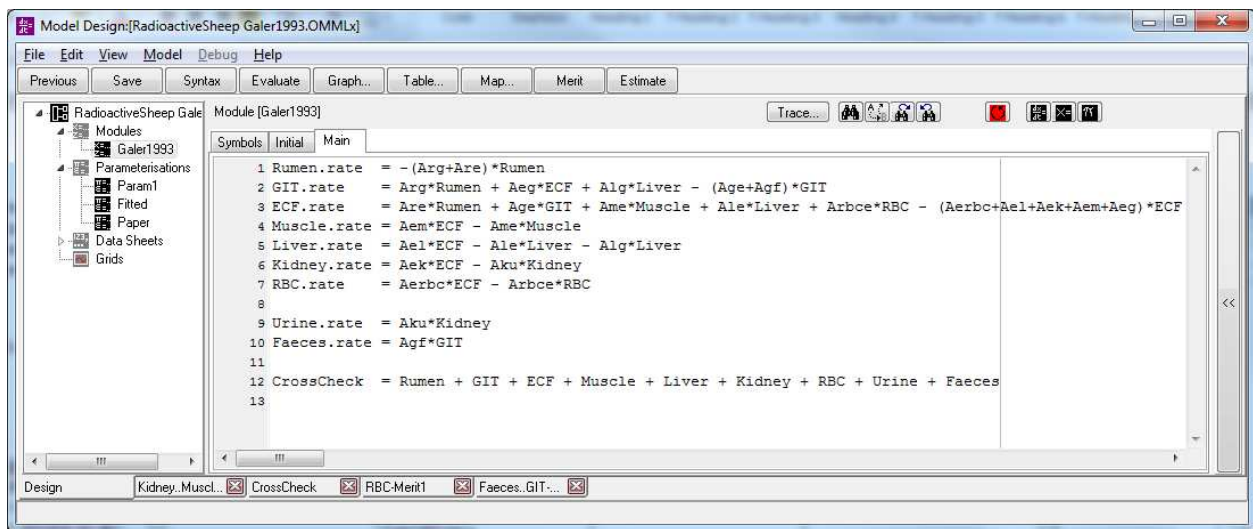
$$\begin{aligned}\frac{dR}{dt} &= -(a_{re} + a_{rg})R \\ \frac{dGIT}{dt} &= a_{re}R + a_{eg}ECF - (a_{ge} + a_{gf})GIT\end{aligned}$$

$$\begin{aligned}
\frac{dRBC}{dt} &= a_{erbce}ECF - a_{rbce}RBC \\
\frac{dM}{dt} &= a_{em}ECF - a_{me}M \\
\frac{dL}{dt} &= a_{el}ECF - a_{le}L \\
\frac{dK}{dt} &= a_{ek}ECF - a_{ku}K \\
\frac{dECF}{dt} &= a_{re}R + a_{ge}GIT + a_{rbce}RBC + a_{me}M + a_{le}L - (a_{eg} + a_{erbce} + a_{em} + a_{ek} + a_{el})ECF \\
\frac{dF}{dt} &= a_{gf}GIT \\
\frac{dU}{dt} &= a_{ku}K
\end{aligned}$$

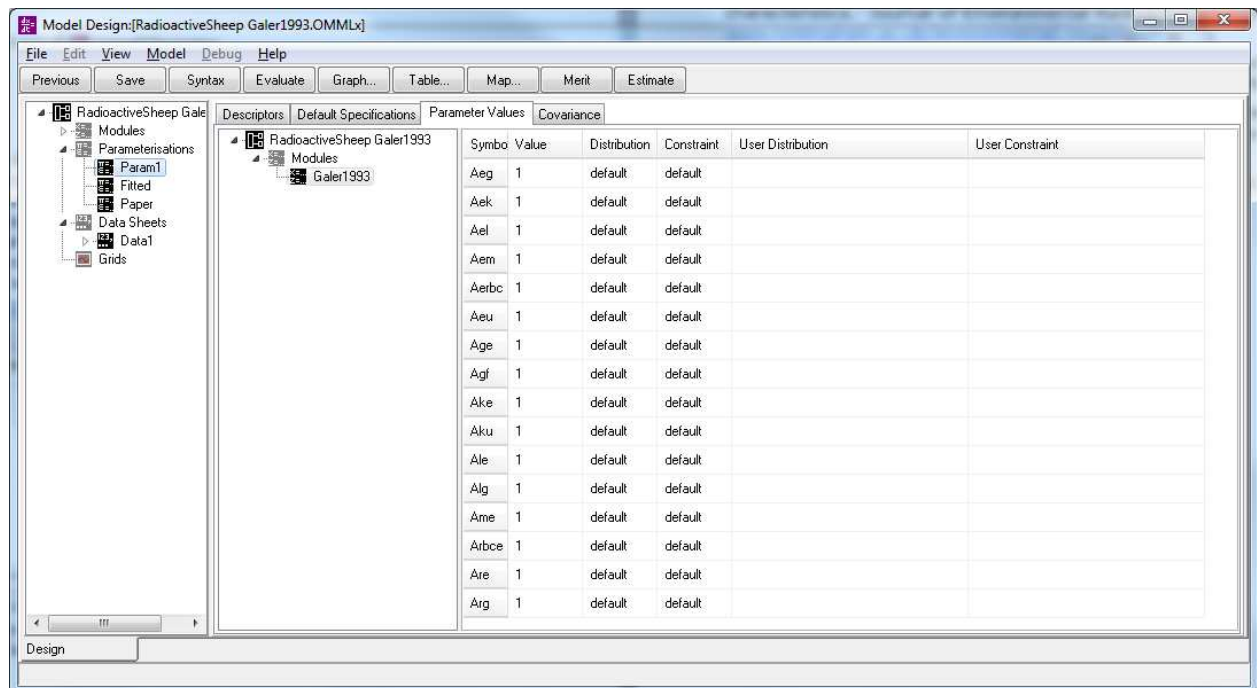
Where the rate coefficient notation a_{xy} implies transfer from compartment x to compartment y .

4.3 Model Implementation

These equations are readily set up in OpenModel using ODEs and parameters. In the example file a cross-check variable is also included; this sums the values of the differential equations as a mass balance check. In closed system models this can be a useful precaution against silly errors in arranging the inter-connections between the model compartments.



In order to evaluate the model we have to specify values of the parameters, even though we don't know them. The idea is to have some initial guesses and then fit the model. In the absence of any firm information they are all set to a value of 1.0.



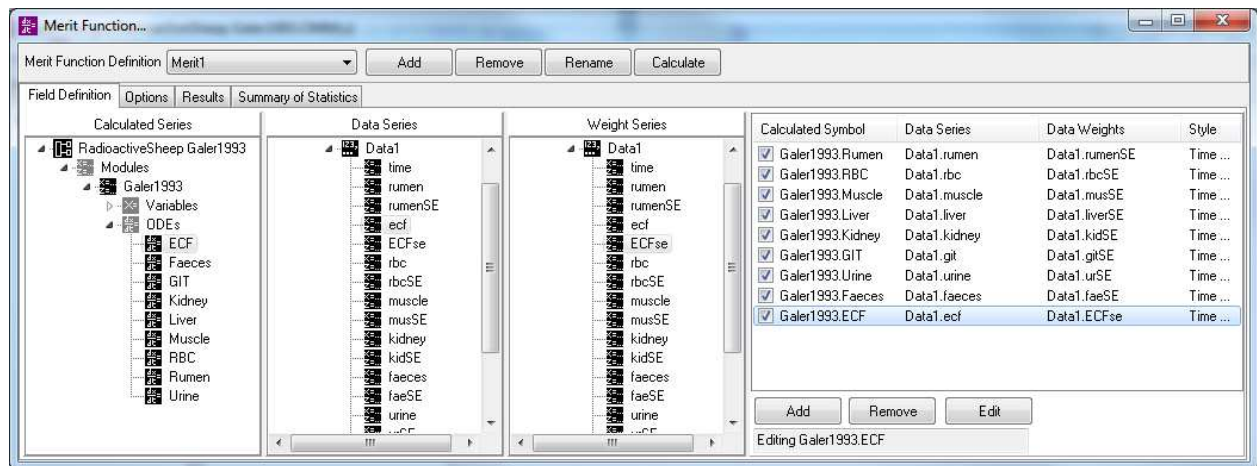
We can now evaluate the model and check it works from a technical perspective. Of course the results aren't realistic, that requires the fitting of the parameters.

4.4 Using the Observations to Measure the Fit

The observations used to fit the model are incorporated as a data sheet (see below). This comprises a time series of measurements of radiocaesium in each tissue together with estimated standard errors for each value. Each measurement is the mean of 3 replicates; the standard errors are based on variation between the replicate values. This arrangement of means values and standard errors is typical for many experimental data sets. Although the standard errors are useful they do need to be interpreted with a little caution when used to 'weight' the model fitting as will be discussed in a moment.

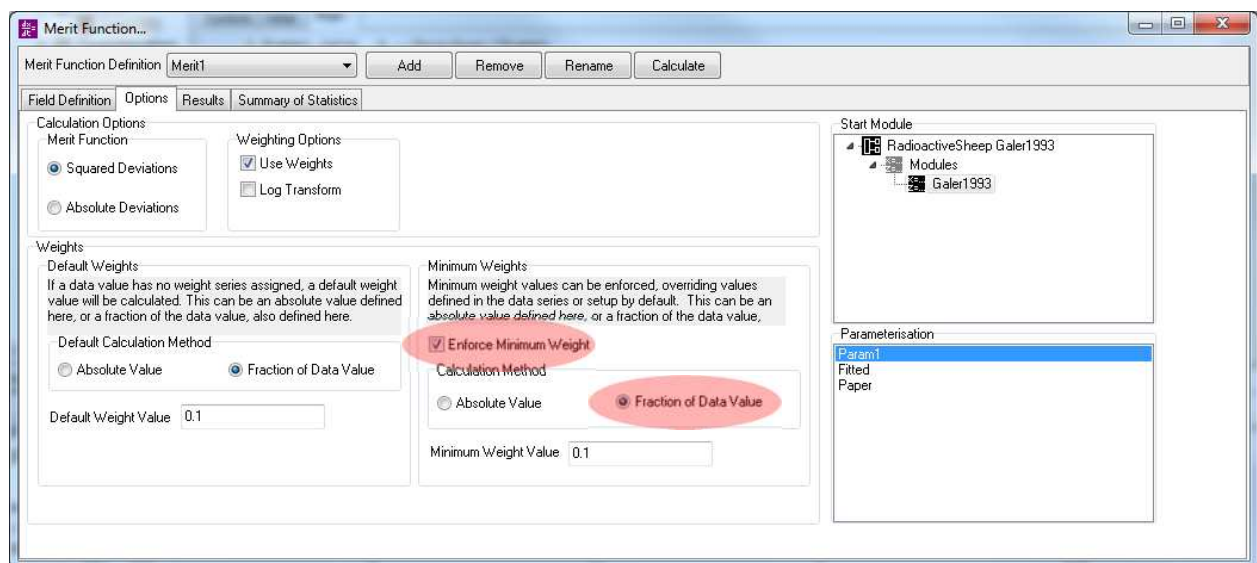
time	rumen	rumenSE	ecf	ECFse	rbc	rbcSE	muscle	musSE	kidney	kidSE
0.17	*	*	1.354	0.1	0.06	0.15	*	*	*	*
0.33	*	*	2.137	0.08	0.08	0.11	*	*	*	*
0.5	61.517	0.27	2.373	0.08	0.095	0.05	5.15	0.195	1.261	0.18
0.83	*	*	1.994	0.06	0.117	0.07	*	*	*	*
1.25	23	0.26	1.614	0.07	0.113	0.06	6.03	0.516	1.046	0.23
1.96	*	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*
4	0.923	0.17	0.366	0.08	0.161	0.17	9.012	0.33	0.302	0.04
12	*	*	0.052	0.13	0.131	0.17	9.507	0.074	0.058	0.034
20	*	*	0.023	0.52	0.054	0.27	6.265	0.23	0.03	0.07

The next step is to define the merit object. This links the predicted model results to the observed values in the data sheet to calculate a residual sum of squares. You can see below that each connection has a calculated symbol, a data series and a data weight.



The weights are optional, but have been chosen here as standard errors are available and using these allows the certainty of the measurement to be taken into account. OpenModel will use $1/(\text{weight series})^2$ as the weight for each data-model pair in the residual sum of squares.

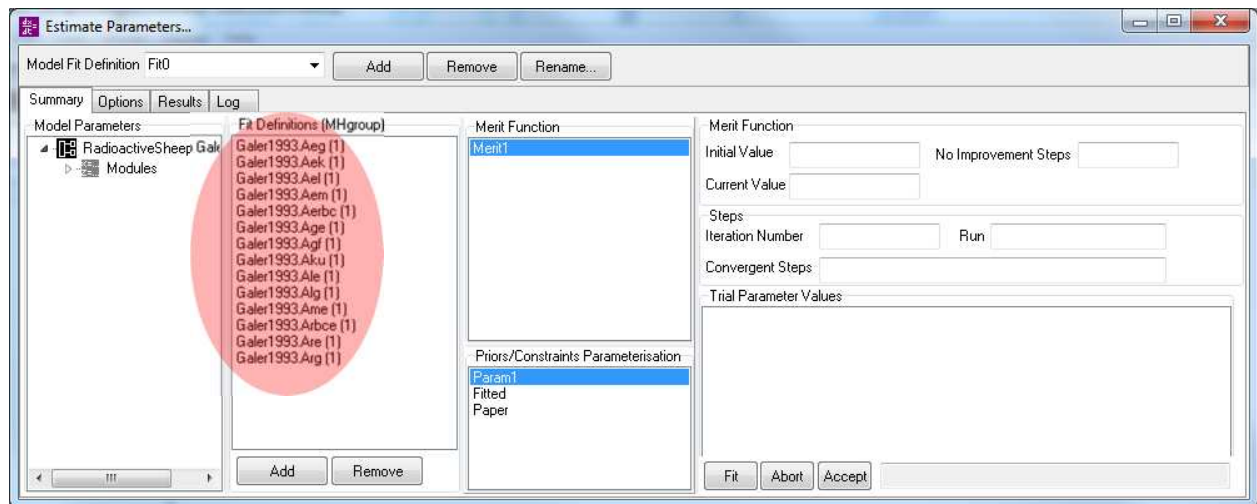
Implicitly this approach allows for the different magnitudes of the observations for the different tissues (a good thing). However a little care must be taken; as mentioned before the standard errors are only estimates, any anomalously small value of standard error can distort the residual sum of squares. To protect against this the **minimum weights** option is used. This enforces a minimum weight rule, overriding the weight series value if it is smaller than the specified threshold. In this example the minimum is set as 10% of the mean observed value.



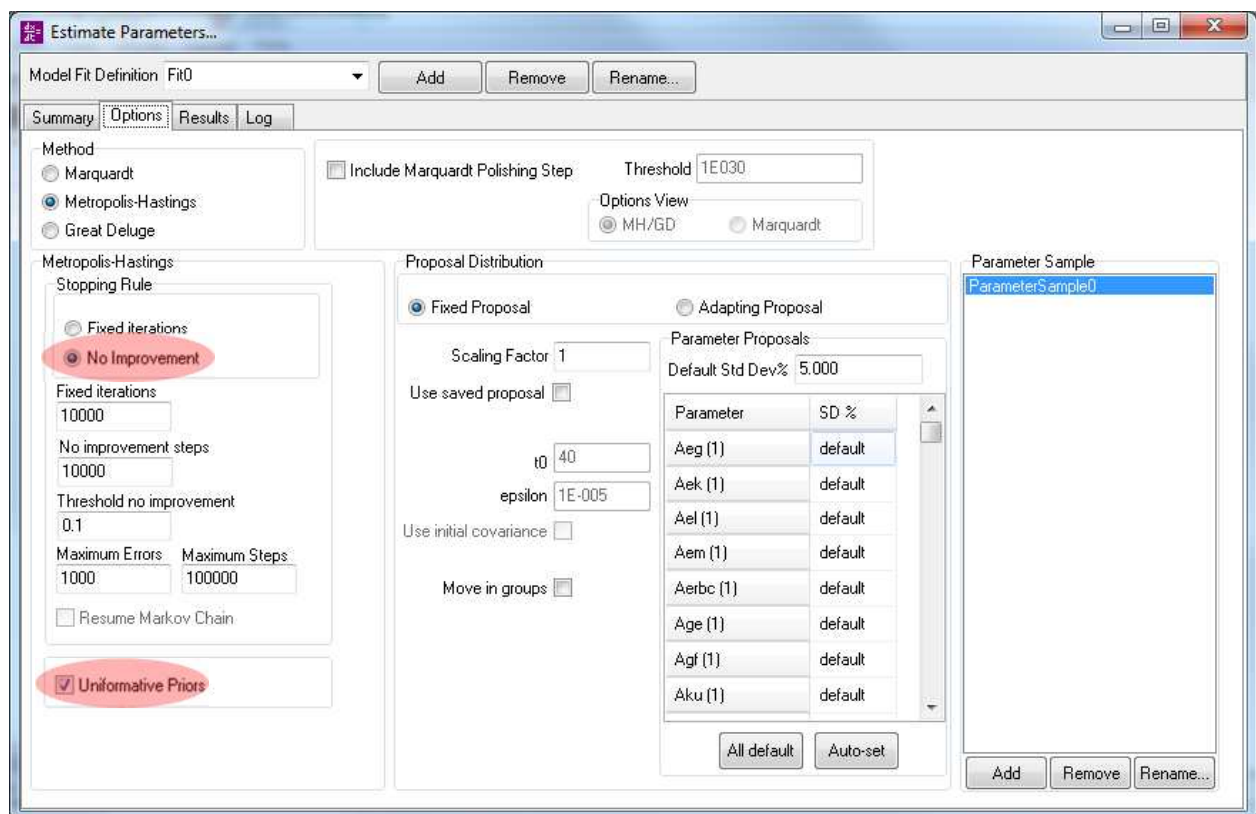
An alternative approach would be to use a pooled standard error as the weight, estimated over the observational data set as a whole, or perhaps for each tissue. There is not 'correct' approach, some careful judgement is required. It can be very helpful to investigate whether the alternatives make much difference to the final result and OpenModel makes this relatively straightforward by allowing you to define any number of merit and fit objects.

4.5 Fitting the Model

Having set up the merit object we are ready to try and fit the model. In the estimate parameters dialog the adjustable parameters are selected. Model fitting always requires a merit function object to define how the residual sums of squares are to be calculated, **Merit1** in the example shown below.



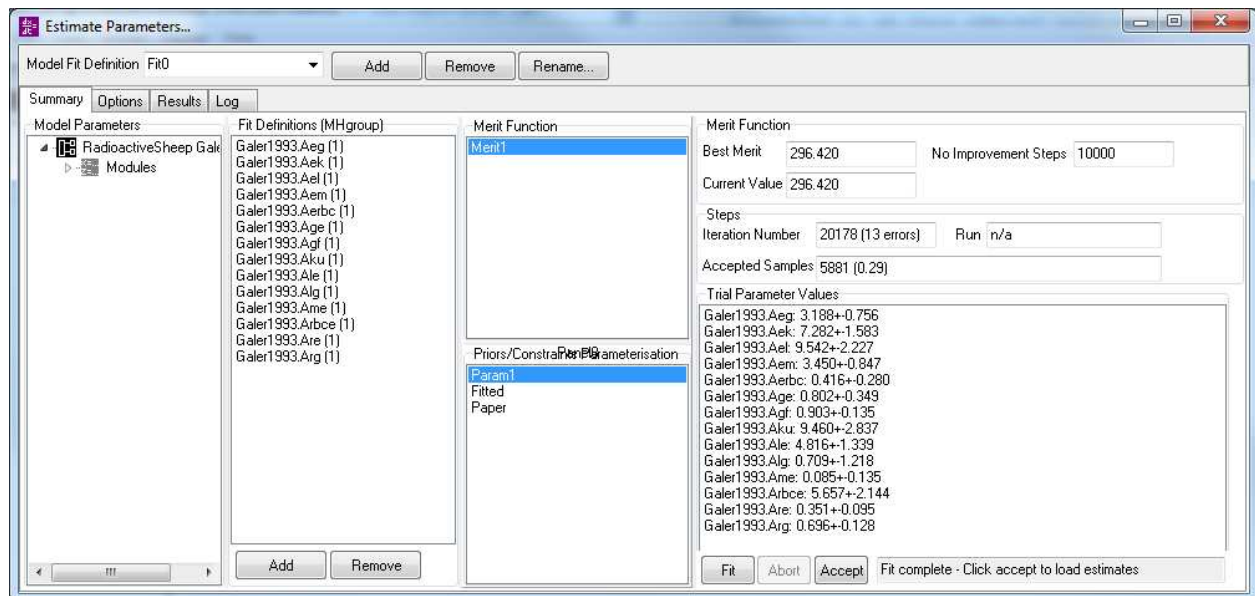
OpenModel has several fitting methods available. These are selected on the options tab as shown below. In this case we have used completely arbitrary initial parameter values. Therefore we start the fitting with metropolis-hastings. This random walk search procedure can avoid getting stuck in false minima when adjusting the parameters. This is very likely when starting from guessed parameter values. Metropolis-Hastings can make use of prior knowledge of the parameters (set via the distributions of the parameters). However we have no useful prior knowledge so we have selected **Uninformative Priors**. The search will respect any constraints to the parameters that we have set but not assume anything else.



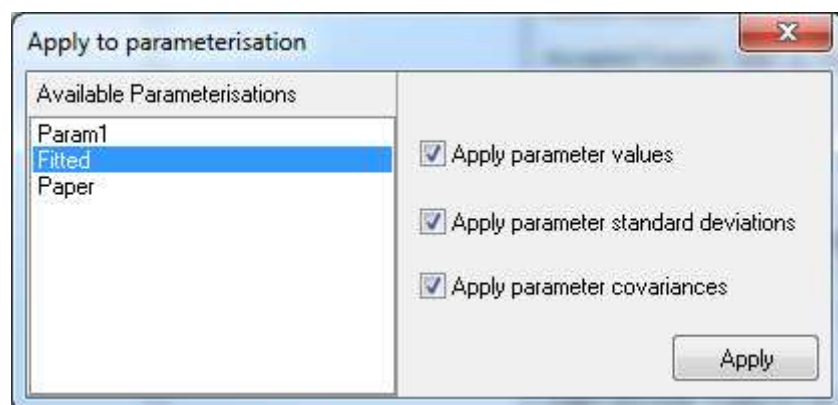
Metropolis-Hastings is an iterative search. It will take 1000s of steps and we must tell it when to stop! A simple way is to limit it to a specified number of steps (this is OpenModel's default). The alternative is to set it to stop after a specified number of iterations without improvement in the residual sums of squares. This is the option chosen here. We have selected 10000 steps without improvement as the end point. Why 10000? It is a trade-off; the larger the number of step the greater the chance of finding the global minimum ☺. On the other hand a high number requires more computational

effort ☹️. After some experimentation we judged that this amount of calculation required an acceptable amount of time and gave similar outcomes after a few repeats. The process is pragmatic. Other options have been left at their default values.

After running you should see a result such as that shown below



In order to use these better fitting parameters you need to 'accept' them into a parameterisation. The accept dialog gives the options to apply values, standard errors and co-variances. In this case we apply all of these to the parameterisation **Fitted**. We have added **Fitted** to the model, making a copy of the initial **Param1** parameterisation.



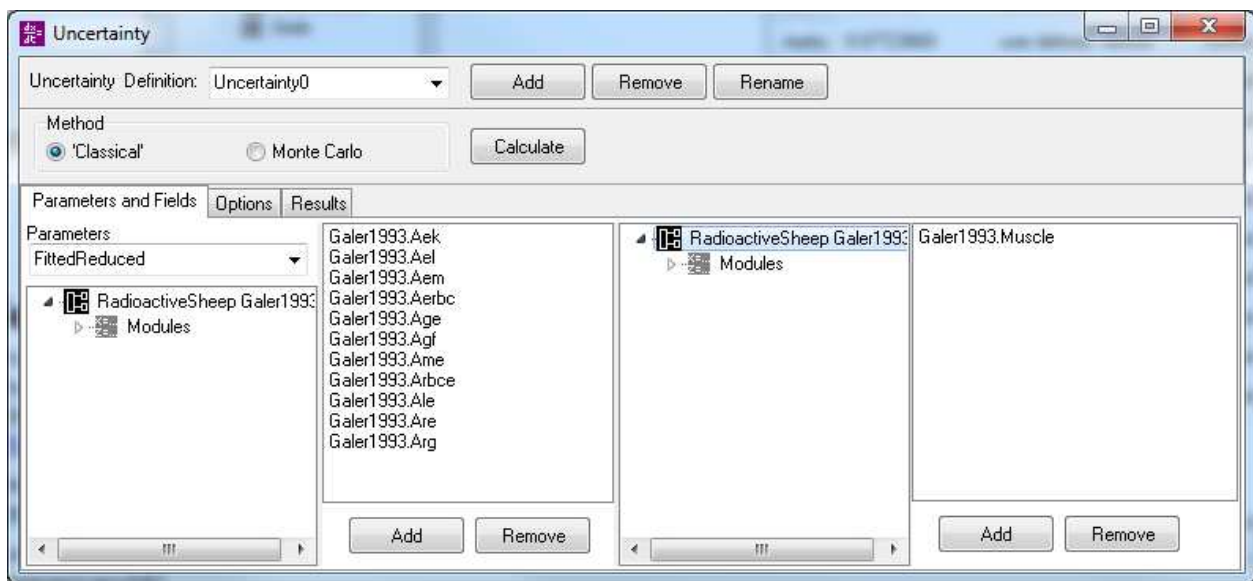
This may not be the best fitting parameter set, but hopefully it is close. A useful 'trick' is now to use the deterministic Marquardt procedure to tune the parameter values. Simply change the method and set the merit object to use the new **Fitted** parameterisation. You will probably find that there is modest further improvement in the residual sums of squares. Because of the way in which the parameters are constrained you may find this can be repeated a few times to make small improvements.

Inspecting the resultant parameter values you should see that 2 of them (A_{eg} and A_{lg}) have very small values and large standard errors. They are not well defined, not significantly different from zero. These parameters can be set at zero and the model refitted excluding them. This will not change the other fitted values but will improve the estimate of the parameter covariance matrix made by Marquardt. The resulting parameterisation in the example file is called **FittedReduced**.

4.6 Classic Confidence Intervals

Now we have a nicely parameterised model. It can be used to make predictions, perhaps varying the input of radiocaesium into the sheep. Perhaps you need to consider the effect of various modelled countermeasures. Generally predictions are more valuable if they have uncertainty estimates. OpenModel can do some uncertainty estimation, presenting the results as confidence intervals. Several methods are available.

Here we use the Uncertainty/Confidence Interval method. This uses the estimated standard errors (or where available the full covariance matrix) to estimate uncertainty in any predictions. The procedure is to select the parameters whose uncertainty is to be included in the analysis (using the left hand tree view/list) and the model variables or ODEs you wish to calculate the confidence intervals for (in the right hand/tree-list). The method choices are 'classic' or 'monte carlo'. We use classic in this case to calculate the confidence interval for muscle due to all the adjustable parameters considered in the **FittedReduced** parameterisation.



'Classic' uncertainty estimation treats the parameter uncertainties as multi-normal and (optionally) includes the effect of covariance (if this is defined in the parameterisation, which it is in our example). The method is computationally efficient, only requiring a single model evaluation for each parameter considered.

If the parameters are independent (i.e. covariance is not considered), then the uncertainty in muscle at each point in time is calculated as, ΔM is,

$$\Delta M^2 \approx \sum_i \left(\frac{dM}{dP_i} \Delta P_i \right)^2$$

Where ΔP_i is the uncertainty in the i^{th} parameter.

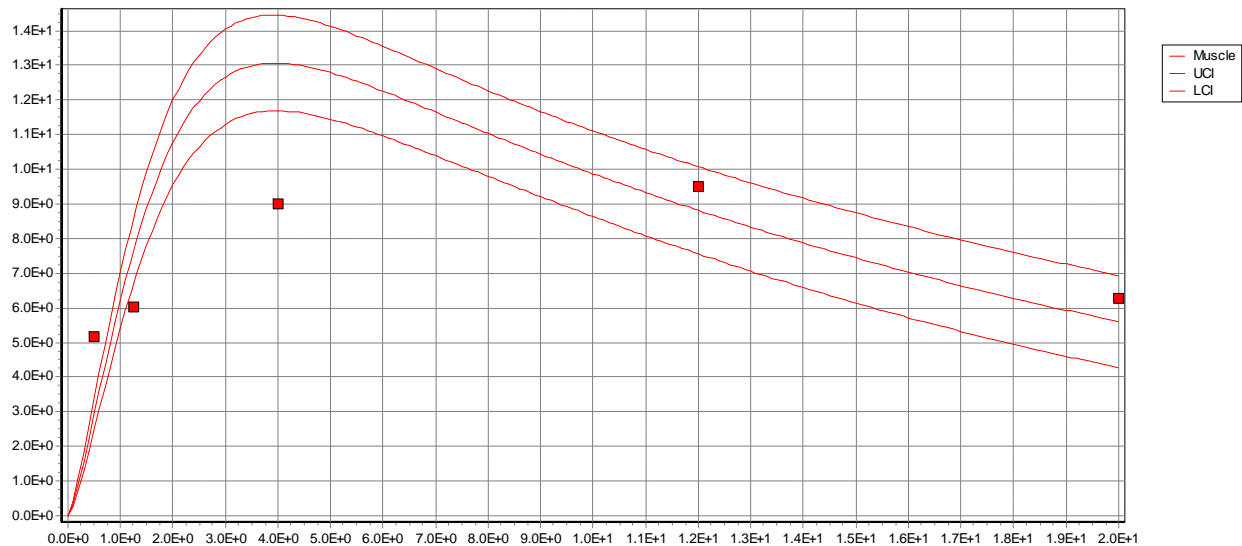
If the full covariance case is used then this is the matrix multiplication of the covariance matrix, C , and the vector of dM/dP_i , i.e.

$$\Delta M^2 \approx \left[\frac{dM}{dP_i} \right]^2 [C]$$

The limitations are the assumption that the distribution of the parameters is neatly wrapped up in the multi-normal distribution defined by the covariance matrix and that the response of the model to variation in the parameters is estimable from a simple difference taken at the mean prediction. These can be rather significant limitations!

The monte carlo approach is more computationally demanding but does not rely on the uncertainty being estimable from the simple difference calculation. However it is still reliant on a realistic parameter distribution, and does not allow for covariance (i.e. parameter interactions in the fitting).

The 'classic' result is shown below as an interval around the mean model prediction.



4.7 Metropolis-Hastings Confidence Intervals

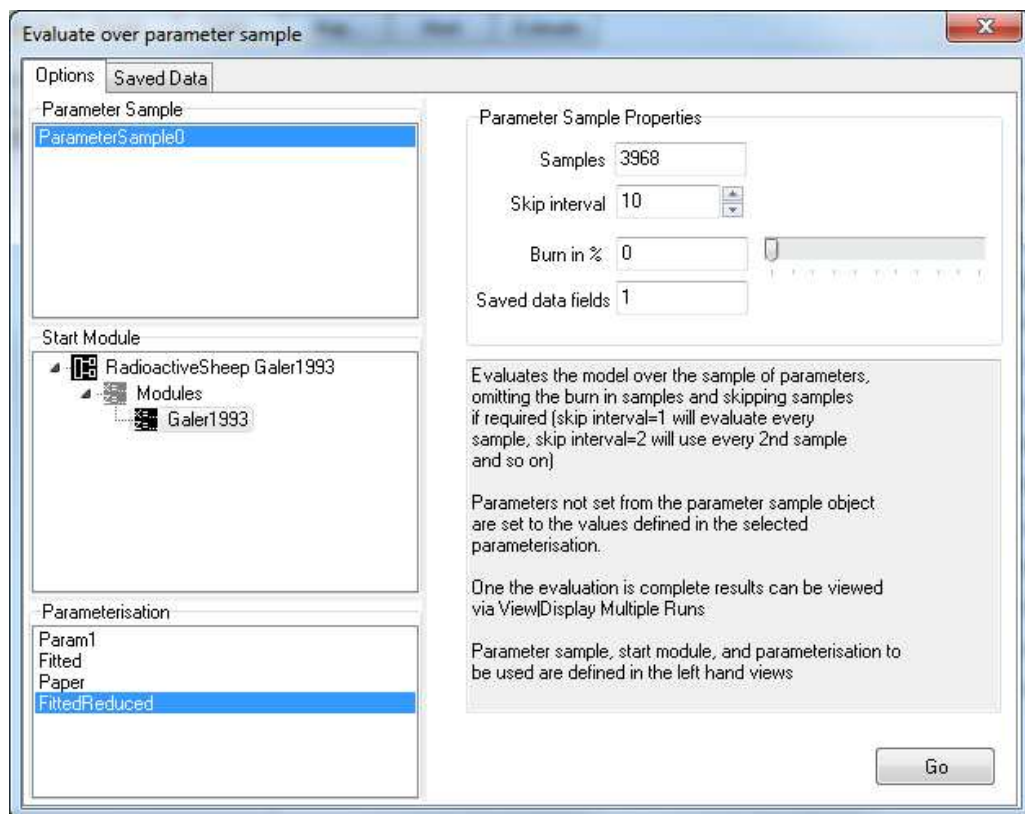
An alternative approach to confidence interval estimation is to create a sample of parameters using the Metropolis-Hastings random walk. If you have a suitable merit object, such that the likelihood is realistically calculated this will create a sample of parameter values drawn from the joint posterior distribution of the parameters. You can then run the model (in any set up that is appropriate for your purposes) using these samples and get a distribution of predictions which can be summarised as a confidence interval.

This is really quite a neat approach. It takes into account the relationships between the estimated parameters, but does not rely on the single point estimations in the classic approach. Nevertheless there are some practical issues to be aware of:

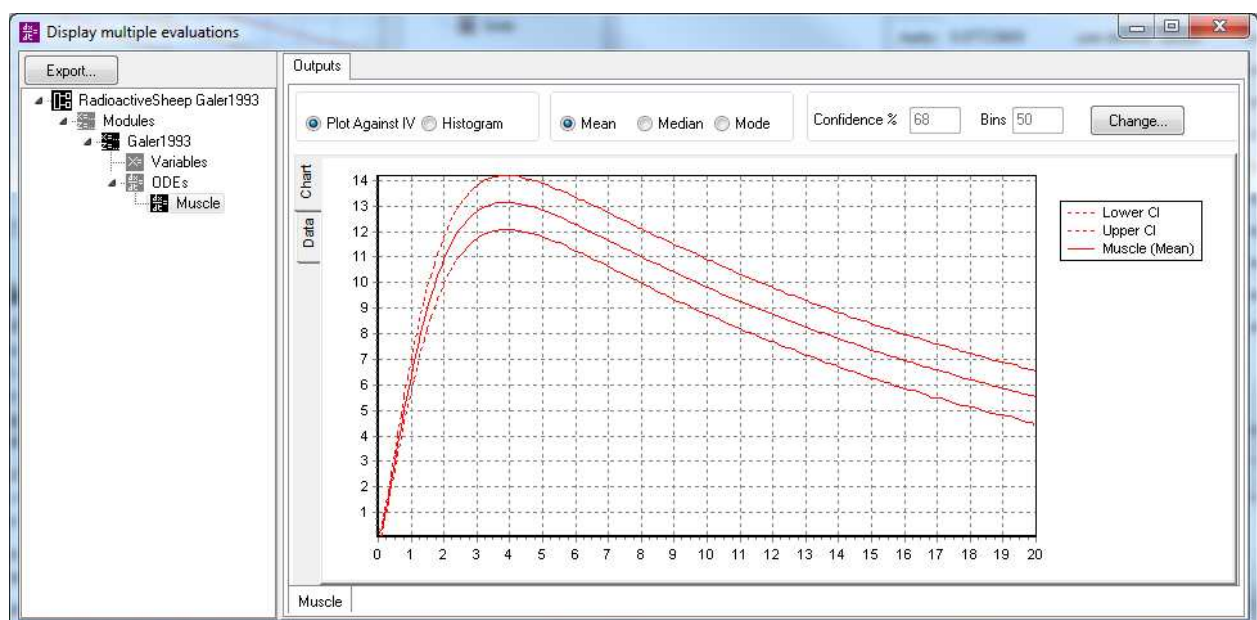
- It is computationally demanding. Lots of iteration is required to create the sample and then run the model over the sample.
- A realistic likelihood estimate is required. In practice this means that the use of weights in the merit object must realistically reflect uncertainty in the observations.
- The parameter sample is created by a series of small random steps. Consequently the sample is likely to be autocorrelated. It is therefore unwise to use all the sample, instead one evaluates the model over a subset, for example skipping to every 10th or 100th value.
- The sample is best created by running a Metropolis-Hastings search starting at the fitted parameters. This creates a sample around best fitting parameter values and avoids the need to think about which part of the sequence of parameter values (often called a chain) it is best to use.

In the example provided the sample `parametersample0` has been created by running 10000 steps (fixed length). The model was then evaluated over the resultant sample, skipping to every 10th set of parameter values.

The variables or ODEs to be saved (muscle in this case) are set on the Saved Data tab (not shown)



The result is shown below via the display multiple evaluations dialog. The confidence interval has been set to 68% to match that of the classic evaluation shown earlier. They are reassuringly similar.



4.8 More things to try...

If you want to take this example further you may find it instructive to:

- Repeat the fitting without using the minimum weights setting.

- Similarly you could try using no weights (i.e. pure residual sums of squares). You should get a funny fit for most of the tissues. Can you figure out why?
- Investigate the effect of the covariance on uncertainty/confidence interval calculation. You should see that ignoring the covariance increases the uncertainty in the predictions. Many people are surprised by that until they think it through!
- Try using all the parameters in the **Fitted** parameterisation for the classic uncertainty analysis. You should see a weird result. This is the effect of the parameters which are really zero being included in the uncertainty analysis.
- As it happens the biology of the model was later revised by [Crout et al \(1996\)](#). The arrangement of the tissues was made more realistic and concentrations used as state variables rather than total radiocaesium. You may wish to set up this version of the model.

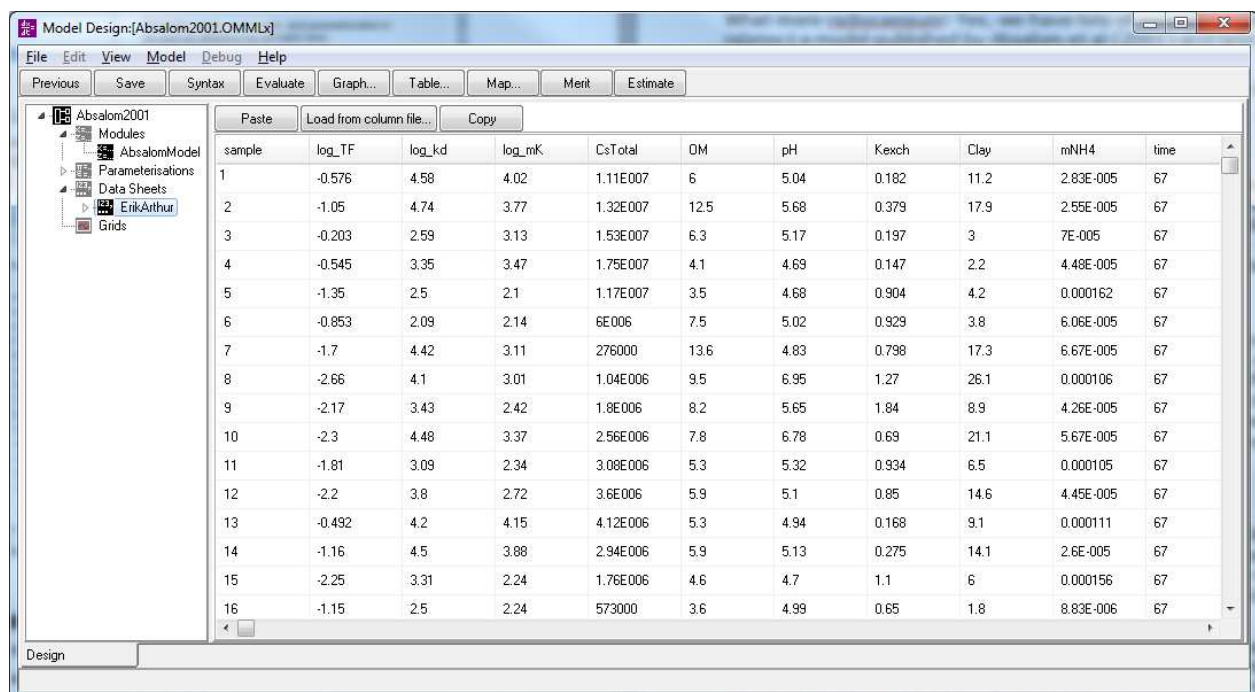
5. Uptake of Radiocaesium from Soil to Plants

What more radiocaesium! Yes, we have lots of examples using radiocaesium. This one relates to a model published by Absalom et al (2001) and later revised by Tarsitano et al (2011).

Here we will only briefly review some features of the model from the perspective of its implementation and evaluation in OpenModel.

5.1 Iterating over Samples

An interesting feature of this model is that the independent variable is not time, nor is it any other continuous variable. In this case OpenModel's iteration over the independent variable is used to iterate the model over a series of independent data records (each of which represents a different site where the model is being applied). The aim is to simultaneously parameterise the model over the 53 sites considered. These are defined as a data sheet of 53 rows.

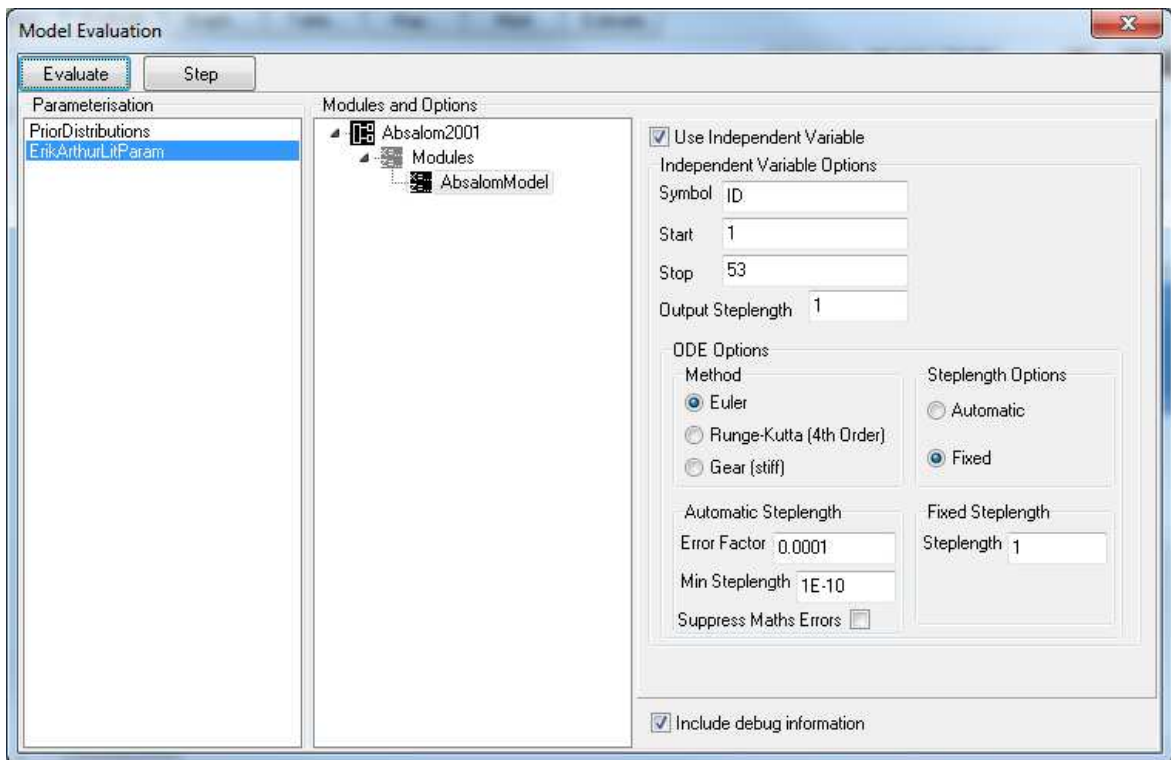


sample	log_TF	log_kd	log_mK	CsTotal	OM	pH	Kexch	Clay	mNH4	time
1	-0.576	4.58	4.02	1.11E007	6	5.04	0.182	11.2	2.83E-005	67
2	-1.05	4.74	3.77	1.32E007	12.5	5.68	0.379	17.9	2.55E-005	67
3	-0.203	2.59	3.13	1.53E007	6.3	5.17	0.197	3	7E-005	67
4	-0.545	3.35	3.47	1.75E007	4.1	4.69	0.147	2.2	4.48E-005	67
5	-1.35	2.5	2.1	1.17E007	3.5	4.68	0.904	4.2	0.000162	67
6	-0.853	2.09	2.14	6E006	7.5	5.02	0.929	3.8	6.06E-005	67
7	-1.7	4.42	3.11	276000	13.6	4.83	0.798	17.3	6.67E-005	67
8	-2.66	4.1	3.01	1.04E006	9.5	6.95	1.27	26.1	0.000106	67
9	-2.17	3.43	2.42	1.8E006	8.2	5.65	1.84	8.9	4.26E-005	67
10	-2.3	4.48	3.37	2.56E006	7.8	6.78	0.69	21.1	5.67E-005	67
11	-1.81	3.09	2.34	3.08E006	5.3	5.32	0.934	6.5	0.000105	67
12	-2.2	3.8	2.72	3.6E006	5.9	5.1	0.85	14.6	4.45E-005	67
13	-0.492	4.2	4.15	4.12E006	5.3	4.94	0.168	9.1	0.000111	67
14	-1.16	4.5	3.88	2.94E006	5.9	5.13	0.275	14.1	2.6E-005	67
15	-2.25	3.31	2.24	1.76E006	4.6	4.7	1.1	6	0.000156	67
16	-1.15	2.5	2.24	573000	3.6	4.99	0.65	1.8	8.83E-006	67

The model has an independent variable ID and for each iteration this is used to read the relevant inputs from the data sheet as illustrated by the code extract below.

```
thetaHumus = ErikArthur.OM (ID)/100
thetaClay  = ErikArthur.Clay(ID)/100
Kx_soil    = ErikArthur.kexch(ID)
mNH4       = ErikArthur.mNH4(ID)
```

The independent variable is defined in the Model Evaluation dialog and set to run from 1 to 53 in steps of 1. Obviously ID is not continuous; a value of 1.5 would have no meaning. However OpenModel will happily interpolate between the rows of the data sheet so we have to make sure we don't ask it to do that!



5.2 Variable Replacement

Variable replacement (also termed model reduction, or model simplification) is a useful method to test whether the structure of a model is justified by the observations you have. OpenModel was developed into its current form largely because of work to develop the variable replacement methods so it seems sensible to show an example. The Absalom model of radiocaesium uptake was exploited heavily in the development of the methods and this work is reported by Crout et al (2009) and Tarsitano et al (2011).

5.3 Method Overview

The principle of the method is simple. Selected model variables are fixed to a constant value within OpenModel and the performance of the model assessed using a merit object evaluation of residual sums of squares. The variables may be fixed individually or in combination. If a variable is important in the model one would expect that fixing it to a constant would result in a poor comparison to the observations.

Interacting effects are often found between the variables replaced. Therefore it is useful to be able to explore different combinations of replacement variables. If you have a large number of variables to consider this can become quite time consuming. The method therefore usually operates in several steps, each of which is outlined below:

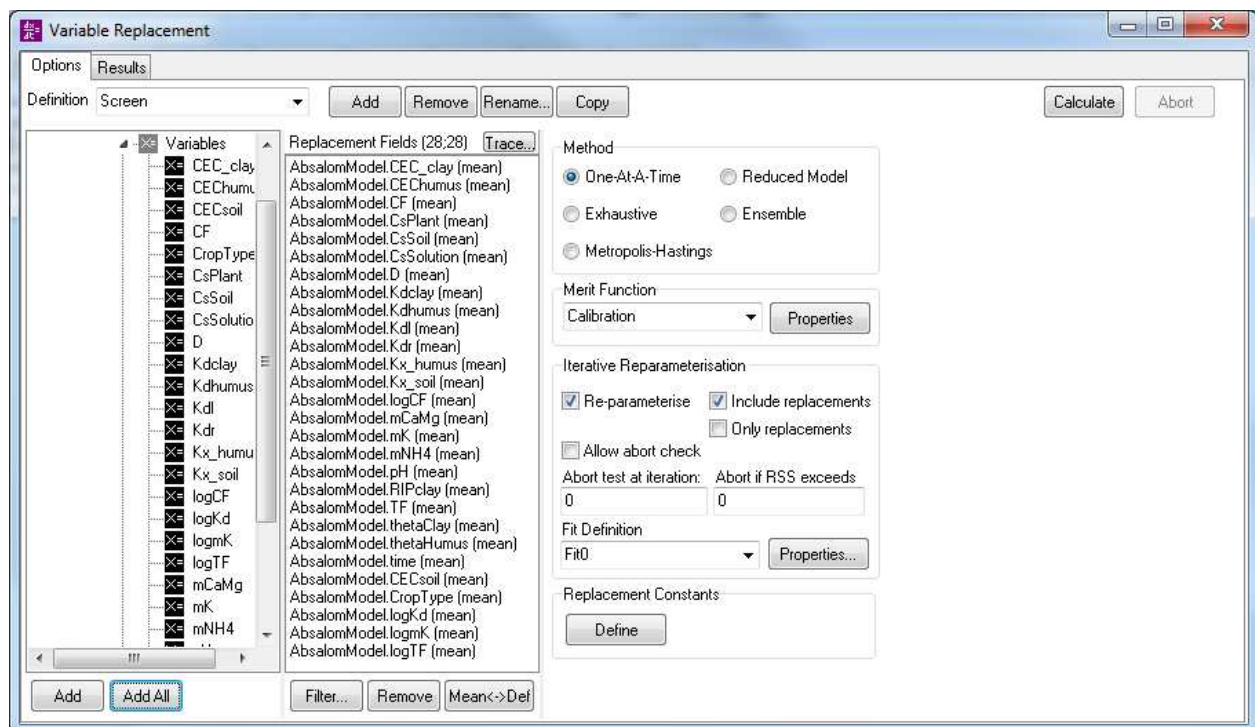
- Identification of candidate replacement variables based on their function in the model.
- Screening of candidate variables individually (i.e. ignoring any interactions) to identify a shortlist of potential replacement variables.
- A search of the shortlisted variables in combination to test the performance of the model structure against the observations.

5.4 Practical Implementation

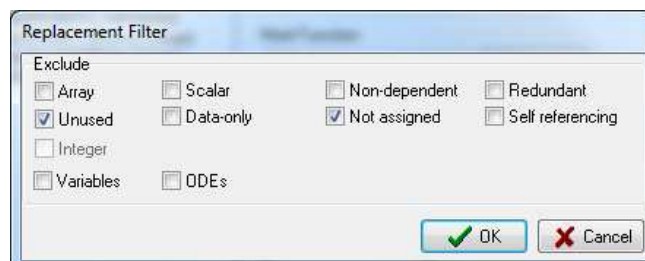
5.4.1 Identifying Candidate Variables

A typical model will have many variables, some of which are not really part of the conceptual model but serve some practical function within the model code. For example, you may have a variable that is only used for calculating an output (perhaps taking a log of another variable), or a variable that is included to convert to more convenient units. Variable replacement is usually only relevant for variables which form part of the conceptual model; that is they are part of the model's description of the system modelled. OpenModel provides some tools to assist in identifying these systematically.

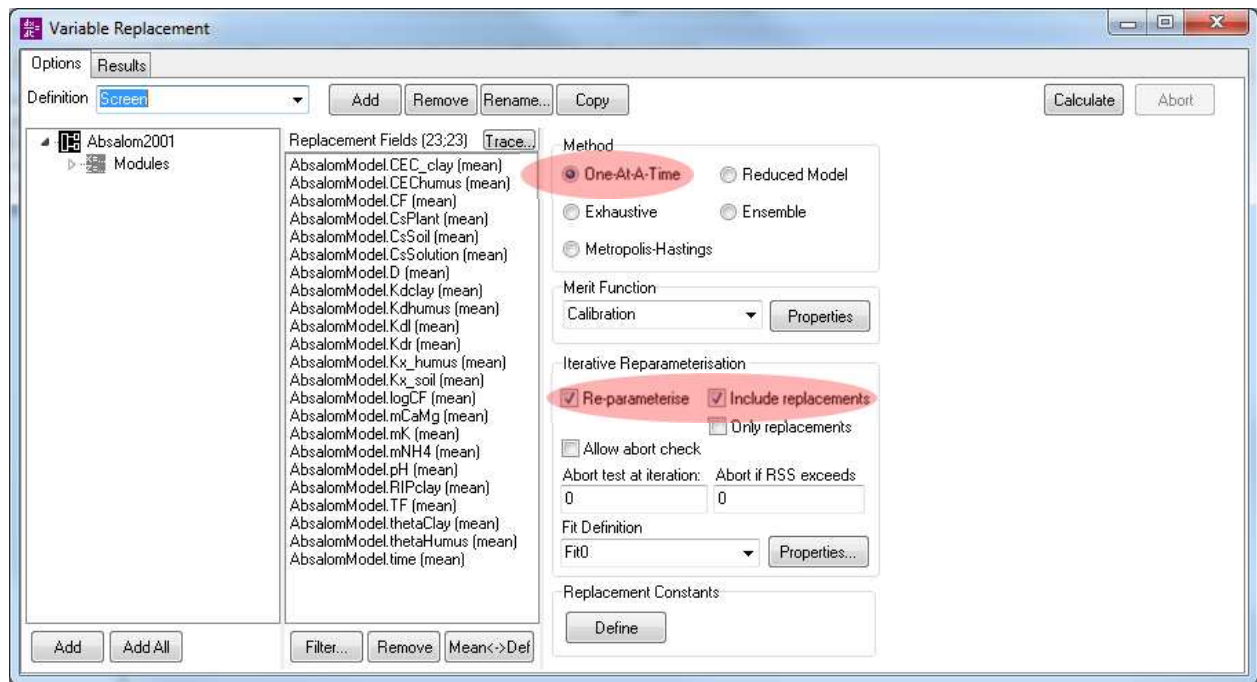
In the variable replacement dialog click **Add All**. This will include all the model variables as replacement variables as shown below.



Use the filter button to display the variable filter dialog (this is similar to the **Categorise Symbols** facility available from the main menu). In the example shown **Unused** and **Not Assigned** have been checked. Variables which OpenModel places in these categories will be filtered out of the replacement variables list on the replacement variable dialog (note this facility only works fully if the model has been compiled).



This gives us a reduced list of 23 shortlisted variables (below) for the next step in the analysis.



5.4.2 Screening variable replacement 1 at a time

When One-At-A-Times is selected in the Variable Replacement dialog running the procedure (via the Calculate button) evaluates the model once for each replacement variable, replacing it with a constant value. The replacement value can be set by the user (doubling clicking on the variable in the reduction field list), be the mean value obtained over the unreduced run of the model or be fitted as part of an overall re-parameterisation of the model for each step of the procedure. In this example the replacement values are obtained by re-parameterisation. This is more computationally intensive (there is multiple running of the model for each step in the reduction procedure) but it provides a stricter test of the model structure.

The Absalom model is very rapidly evaluated so the replacement procedure runs quite quickly. The output is presented on the Results tab of the Variable Replacement dialog as shown below.

This output also saved to C:\My Work\OpenModel\OpenModel\Install Examples\Absalom2001_Screen.backup

Merit Value prior to simplification: 38.58

Omitted from replacement:
None

Variable	Fixed Replacement Constant	N	Mean	Min	Max	SD	CV(%)	
AbsalomModel.CEC_clay (mean)	6.1415	53	6.1415	0.25	28.8	5.1785	84.32	
AbsalomModel.CEChumus (mean)	97.448	53	97.448	36.668	171.89	38.539	39.548	
AbsalomModel.CF (mean)	2712.5	53	2712.5	16.143	34321	5891.2	217.19	
AbsalomModel.CsPlant (mean)	2.2718E006	53	2.2718E006		45.566	2.2341E007	4.8186E006	212.1
AbsalomModel.CsSoil (mean)	5.9812E006	53	5.9812E006		2170	2.35E007	7.3431E006	122.77
AbsalomModel.CsSolution (mean)	20318	53	20318	0.067559		8.0063E005	1.0962E005	539.52
AbsalomModel.D (mean)	0.91041	53	0.91041	0.89458	0.98907	0.021491	2.3606	
AbsalomModel.Kdclay (mean)	10390	53	10390	2.6792	1.9643E005		28884	278
AbsalomModel.Kdhumus (mean)	11.379	53	11.379	1.4835	29.982	9.8849	86.869	
AbsalomModel.Kdl (mean)	10401	53	10401	19.658	1.9644E005		28883	277.68
AbsalomModel.Kdr (mean)	0.92559	53	0.92559	0.097476	0.99997	0.18588	20.082	
AbsalomModel.Kx_humus (mean)	3.4795	53	3.4795	0.095851		13.555	4.1573	119.48
AbsalomModel.Kx_soil (mean)	0.56777	53	0.56777	0.1	1.84	0.35439	62.417	
AbsalomModel.logCF (mean)	2.7404	53	2.7404	1.208	4.5356	0.87257	31.841	
AbsalomModel.mCaMg (mean)	0.0024618	53	0.0024618		0.0010375		0.0055463	0.0011546
AbsalomModel.mK (mean)	0.00076984	53	0.00076984	3.3E-005		0.003166	0.00091985	119.49
AbsalomModel.mNH4 (mean)	0.00072764	53	0.00072764	0		0.00642	0.0013549	186.2

There is a lot of output, but it is readily copied and pasted into excel for interpretation. The key information is shown opposite, the residual sums of squares for the model when each of the variables is replaced with the best fitting replacement (or at least the best that could be found).

The full model (as published by Absalom et al) has an RSS of 38.58. If CECclay is replaced this increases to 42.58 and so on for the other variables considered.

You will note that several the replacement of several variables produce a lower RSS than the model without replacement. This telling us that the variable has a detrimental effect on the comparison of the model with observation.

In the example model provided this One-At-A-Time screen is defined in the replacement object **screen**.

Field	RSS
No-Replacement	38.58
CEC_clay	42.23
CEChumus	39.33
CF	67.12
CsPlant	96.02
CsSoil	38.58
CsSolution	96.03
D	38.57
Kdclay	97.37
Kdhumus	37.23
Kdl	96.23
Kdr	38.44
Kx_humus	89.62
Kx_soil	50.26
logCF	67.32
mCaMg	39.65
mK	77.11
mNH4	56.08
pH	36.56
RIPclay	45.94
TF	97.55
thetaClay	52.49
thetaHumus	72.07
time	38.66

5.4.3 Factorial replacement

Having completed the One-At-A-Time screen we identify 10 variables which are the most likely candidates for inclusion is the multiple replacement search. This is defined in the replacement object **Factorial**.

There are two options for searching the factorial combinations of variable replacements, Exhaustive (used here) or Metropolis-Hastings.

Exhaustive searches every possible permutation of the replacements ($2^n - 1$). There is no concern about the reliability of the search procedure, as everything is searched! However the approach is only feasible if the number of replacement variables to be considered is not too large and the model not unduly time consuming to evaluate. For larger numbers of replacement variables, or more computationally time consuming models the Metropolis-Hastings random walk may be more feasible.

Running the exhaustive procedure for this model reveals several permutations of the replacement variables which improve the model agreement with observations. These must then be interpreted in the context of the model design. In the case of the Absalom model this is discussed in more detail by Crout *et al* (2009). The key point is the identification of several aspects of the model which add unnecessary uncertainty to the predictions. Tarsitano *et al* (2011) extends the analysis, using the outputs to guide a re-design of the model.

6. Metal Fixation in Soils – Same Model, Different Samples (Arrays)

This example uses the model of metal kinetics in soils presented by Crout *et al* (2006). The model itself is rather simple, exponential fixation to an equilibrium controlled by soil pH. The model was applied to 23 different soils using 2 metals, Zinc and Cadmium.

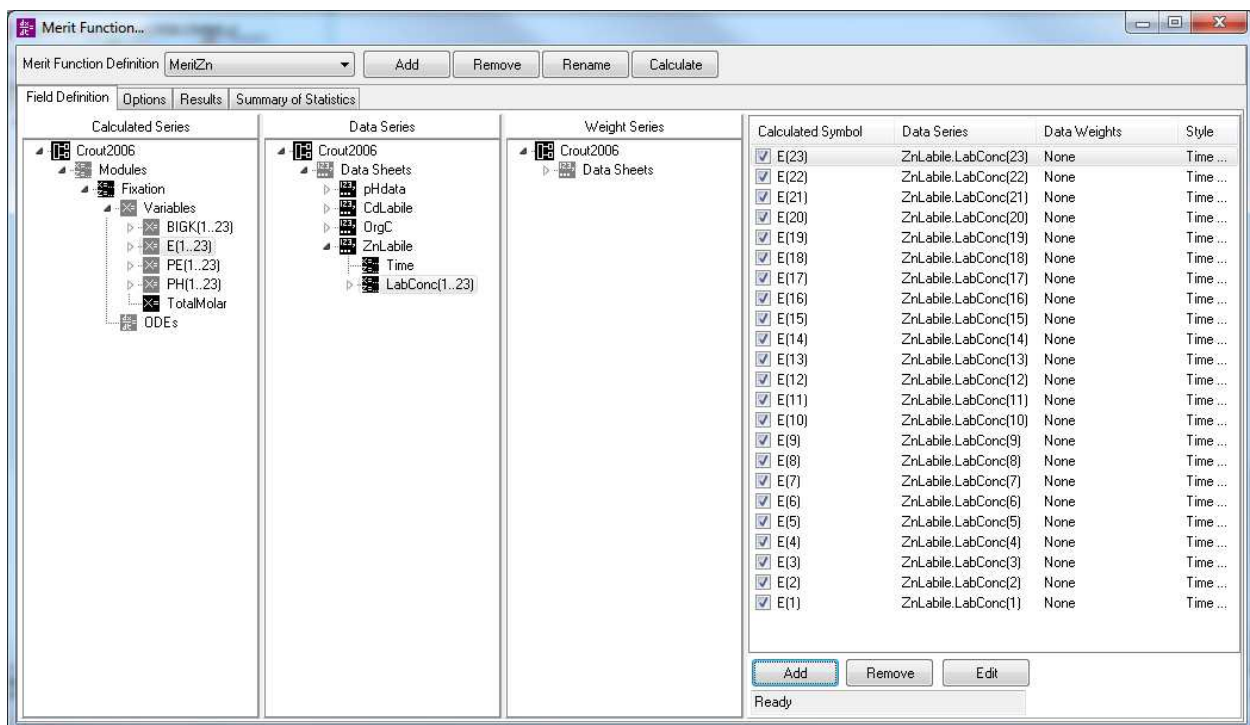
It is shown here as it illustrates a typical situation, namely a common model that we wish to parameterise over a number of sample situations (23 in this case). Furthermore the same model (and samples) are used with two different parameterisations (one for each metal). OpenModel is well adapted for implementing models in this situation. We will focus on these aspects of the model implementation.

The 23 soils are represented in the model by using arrays for the model variables as illustrated in the script segment below (taken from the main view of the model).

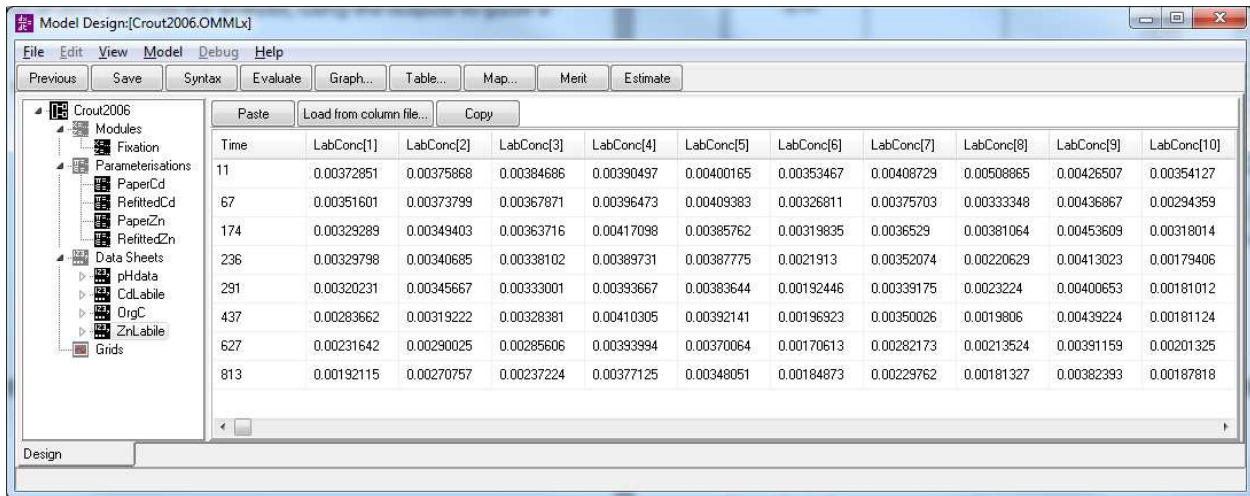
```
for i=1,23
  pH(i)    = pHdata.column(i,t)
  Pe(i)    = a1-a2*pH(i)
  BigK(i)  = b1*exp(-b2*pH(i))
  E(i)     = (Pe(i)+(1-Pe(i))*exp(-BigK(i)*t))*TotalMolar
Endfor
```

The two metals are allowed for by using different parameterisations and evaluating the model with the appropriate parameterisation. During model development, when typically different model designs are explored this greatly reduces the work involved.

For each metal a merit object is defined. This links the array **E(1..23)** with the observed data, accessed from data sheets. As shown below the data sheet entries have been arranged in an array format. The series for the sheet **ZnLabile** are shown. This data sheet has values for each of the 23 soils, arranged under array headers allowing OpenModel to treat them as an array. The merit object allows the arrays **E** and **LabConc** to be linked in one step creating the 23 comparison fields shown.



This approach requires that the headers of the data sheet adhere to an array format (below) but these is readily accomplished in excel before pasting the data into OpenModel



Time	LabConc[1]	LabConc[2]	LabConc[3]	LabConc[4]	LabConc[5]	LabConc[6]	LabConc[7]	LabConc[8]	LabConc[9]	LabConc[10]
11	0.00372851	0.00375868	0.00384686	0.00390497	0.00400165	0.00353467	0.00408729	0.00508865	0.00426507	0.00354127
67	0.00351601	0.00373799	0.00367871	0.00396473	0.00409383	0.00326811	0.00375703	0.00333348	0.00436867	0.00294359
174	0.00329289	0.00349403	0.00363716	0.00417098	0.00385762	0.00319835	0.0036529	0.00381064	0.00453609	0.00318014
236	0.00329798	0.00340685	0.00338102	0.00389731	0.00387775	0.0021913	0.00352074	0.00220629	0.00413023	0.00179406
291	0.00320231	0.00345667	0.00333001	0.00393667	0.00383644	0.00192446	0.00339175	0.0023224	0.00400653	0.00181012
437	0.00283662	0.00319222	0.00328381	0.00410305	0.00392141	0.00196923	0.00350026	0.0019806	0.00439224	0.00181124
627	0.00231642	0.00290025	0.00285606	0.00393994	0.00370064	0.00170613	0.00282173	0.00213524	0.00391159	0.00201325
813	0.00192115	0.00270757	0.00237224	0.00377125	0.00348051	0.00184873	0.00229762	0.00181327	0.00382393	0.00187818

7. References

- Absalom JP, Young SD, Crout NMJ, Sanchez AL, Wright SM, Smolders E, Nisbet A, Gillett AG. (2001) Predicting the transfer of radiocaesium from organic soils to plants using soil characteristics. *Journal of Environmental Radioactivity* 52:31-43.
- Cox GM, Gibbons JM, Wood ATA, Craigon J, Ramsden SJ, Crout NMJ (2006). Towards the systematic simplification of mechanistic models. *Ecological Modelling* 198:240-246.
- Crout NMJ, Beresford NA, Howard BJ, Mayes RW, Assimakopoulos PA, Vandecasteele C. (1996). The development and testing of a revised model of radiocaesium transfer to sheep tissues. *Radiation & Environmental Biophysics* 35:19-24
- Crout NMJ, Tye AM, Zhang H, McGrath SP, Young SD (2006). Kinetics of metal fixation in soils: measurement and modelling by isotopic dilution. *Environmental Toxicology and Chemistry* 25:659-663.
- Crout NMJ, Tarsitano D, Wood AT(2009). Is my model too complex? Evaluating model formulation using model reduction. *Environmental Modelling & Software*, 24:1-7.
- Evans GT, Parslow JS (1985). A model of annual plankton cycles. *Biol. Oceanogr.* 3:327-347.
- Galer AM, Crout NMJ, Beresford NA, Howard BJ, Mayes RW, Barnett CL, Eayres H, Lamb CS (1993). Dynamic radiocaesium distribution in sheep: measurement and modelling. *J. Environ. Rad.* 20:35-48.
- Tarsitano D, Young SD, Crout NMJ. (2011) Evaluating and reducing a model of radiocaesium soil-plant uptake. *Journal of Environmental Radioactivity*, 102, 262-269.