Chapter 1

Parameter Minimization using the Levenberg-Marquardt algorithm

Do some fitting

1.1 Introduction

The Levenberg-Marquardt plugin is used to fit a SBML models parameters to experimental data.

The current implementation is based on the lmfit C library by Joachim Wuttke. See footnote[1] for license disclosure.

The plugin has numerous properties to allow the user full control over the internal fitting engine, as well as access to generated fitted data after a minimization session.

Plugin properties are documented in the next section.

[1]The package lmfit is distributed under the FreeBSD License:

1.2 Plugin Properties

Available properties in the Levenberg-Marquardt plugin are listed in the table below.

Parameter Name	Data Type	Default Value	Description
SBML	string	N/A	SBML document as a string. Model to be used in the fitting.
ExperimentalData	telluriumData	N/A	Input data.
FittedData	telluriumData	N/A	Output data.
Residuals	telluriumData	N/A	Residuals data.
InputParameterList	listOfParameters	N/A	Parameters to fit.
OutputParameterList	listOfParameters	N/A	List of fitted parameters.
Experimental-DataSelectionList	stringList	N/A	Species selection list for experimental data.
FittedDataSelectionList	stringList	N/A	Selection list for model data.
Norm	double	N/A	Norm of fitting. An estimate of goodness of fit.

Norms	telluriumData	N/A	The norm is calculated throughout a fitting session. Each Norm value is stored in the `Norms` (read-only) property. The dimension (rows x col) of the Norms Tellurium data object is maxNumberOfIterations x 1, where maxNumberOfIterations is obtained as $patience * (nrOfPars + 1)$. If a fitting session is finished before maxNumberOfIterations been reached, data beyond the actual number of iterations (mNrOfIter) should be discarded.
NrOfIter	int	N/A	Number of iterations.

The following properties are used internally by the fitting engine. They are preset with default values. Depending on the minimization problem at hand, they may need to be tweaked.

ftol	double	machine dep.	Relative error desired in the sum of squares.
xtol	double	machine dep.	Relative error between last two approximations.
gtol	double	machine dep.	Orthogonality desired between fvec and its derivs.
epsilon	double	machine dep.	Step used to calculate the jacobian.
stepbound	double	100.0	Initial bound to steps in the outer loop.

patience	double	100	Used for setting maximum number of iterations, calculated as `patience*(nr_of_parameters +1)`.

Table 1.1: LM fit plugin parameters

1.3 Plugin Events

The lmFit plugin are using all of a plugins available plugin events, i.e. the *PluginStarted*, *PluginProgress* and the *PluginFinished* events.

The available data variables for each event are internally treated as *pass trough* variables, so any data, for any of the events, assigned prior to the plugins execute function (in the assingOn.. family of functions), can be retrieved unmodified in the corresponding event function.

Event	Arguments	Purpose and argument types
PluginStarted	void*, void*	Signal to application that the plugin has started. Both parameters are *pass trough* parameters and are unused internally by the plugin.
PluginProgress	void*, void*	Communicating progress of fitting. Both parameters are *pass trough* parameters and are unused internally by the plugin.
PluginFinished	void*, void*	Signals to application that execution of the plugin has finished. Both parameters are *pass trough* parameters and are unused internally by the plugin.

Table 1.2: LMFit Plugin callbacks

1.4 Python example

The following Python script illustrate how the lmFit plugin can be used.

Line by line description of the code is given below.

```
1  import ctypes
2  from telPlugins_CAPI import *
3  import telPlugins as tel
4
5  #Get a lmfit plugin object
6  lm = tel.Plugin("tel_lm")
7
8  #========== EVENT FUNCTION SETUP ============================
9  def pluginIsProgressing(lmP, dummy):
10     # The plugin don't know what a python object is.
11     # We need to cast it here, to a proper python object
12     lmObject = cast(lmP, ctypes.py_object).value
13     print 'Iterations = ' + `lmObject.getProperty("NrOfIter")` \
14         + '\tNorm = ' + `lmObject.getProperty("Norm")`
15
16 progressEvent =  NotifyPluginEvent(pluginIsProgressing)
```

```
17
18 #The ID of the plugin is passed as the last argument in the
       assignOnProgressEvent.
19 #The plugin ID is later on retrieved in the plugin Event handler, see above
20 theId = id(lm)
21 assignOnProgressEvent(lm.plugin, progressEvent, theId, None)
22 #============================================================
23
24 #Setup lmfit properties.
25 lm.setProperty("SBML", lm.readAllText("lmFitTestModel.xml"))
26 experimentalData = lm.loadDataSeries ("testData.dat")
27 lm.setProperty("ExperimentalData", experimentalData)
28
29 # Add the parameters that we're going to fit and a initial 'start' value
30 lm.setProperty("InputParameterList", ["k1", 5.2])
31 lm.setProperty("FittedDataSelectionList", "[S1] [S2]")
32 lm.setProperty("ExperimentalDataSelectionList", "[S1] [S2]")
33
34 # Start minimization
35 lm.execute()
36
37 print 'Minimization finished. \n==== Result ===='
38 print getPluginResult(lm.plugin)
39
40 # Get the experimental data as a numpy array
41 experimentalData = experimentalData.AsNumpy
42
43 # Get the fitted and residual data
44 fittedData = lm.getProperty ("FittedData").AsNumpy
45 residuals  = lm.getProperty ("Residuals").AsNumpy
46
47 tel.plot(fittedData         [:,[0,1]], "blue", "-",    "",    "S1 Fitted")
48 tel.plot(fittedData         [:,[0,2]], "blue", "-",    "",    "S2 Fitted")
49 tel.plot(residuals          [:,[0,1]], "blue", "None", "x",   "S1 Residual"
       )
50 tel.plot(residuals          [:,[0,2]], "red",  "None", "x",   "S2 Residual"
       )
51 tel.plot(experimentalData   [:,[0,1]], "red",  "",     "*",   "S1 Data")
52 tel.plot(experimentalData   [:,[0,2]], "blue", "",     "*",   "S2 Data")
53 tel.plt.show()
```

Listing 1.1: Minimization example.

Line 1-3:

Import necessary python packages.

Line 6:

Create a lmFit python object.

Line 9-21:

This section defines an event function (line 9-14), creates a variable representing the function (line 16). Line 20 retrieves a unique ID for the plugin object. Line 21 assigns the event function variable and the ID of the plugin to the event functions first argument. When the plugin calls the event function, the ID of the plugin object can be retrieved in the functions first argument.

The sole purpose of the progress event function is to give the user information on how the minimization is progressing.

Line 25:

Read a sbml model from file to be used in the fitting.

Line 26-27:

Read experimental data from file, and assign it to the plugins experimental data property.

Line 30-32:

Setup required plugin properties. Line 30 assigns a parameter to be fitted, with an initial valur. Line 31 and 32 assigns selections for input and output data.

Line 35:

Execute the plugin.

Line 37-38:

After finishing the minimization, print the result.

Line 40-53:

These lines prepares data to be plotted, (line 41-52) and then plots it on line 53

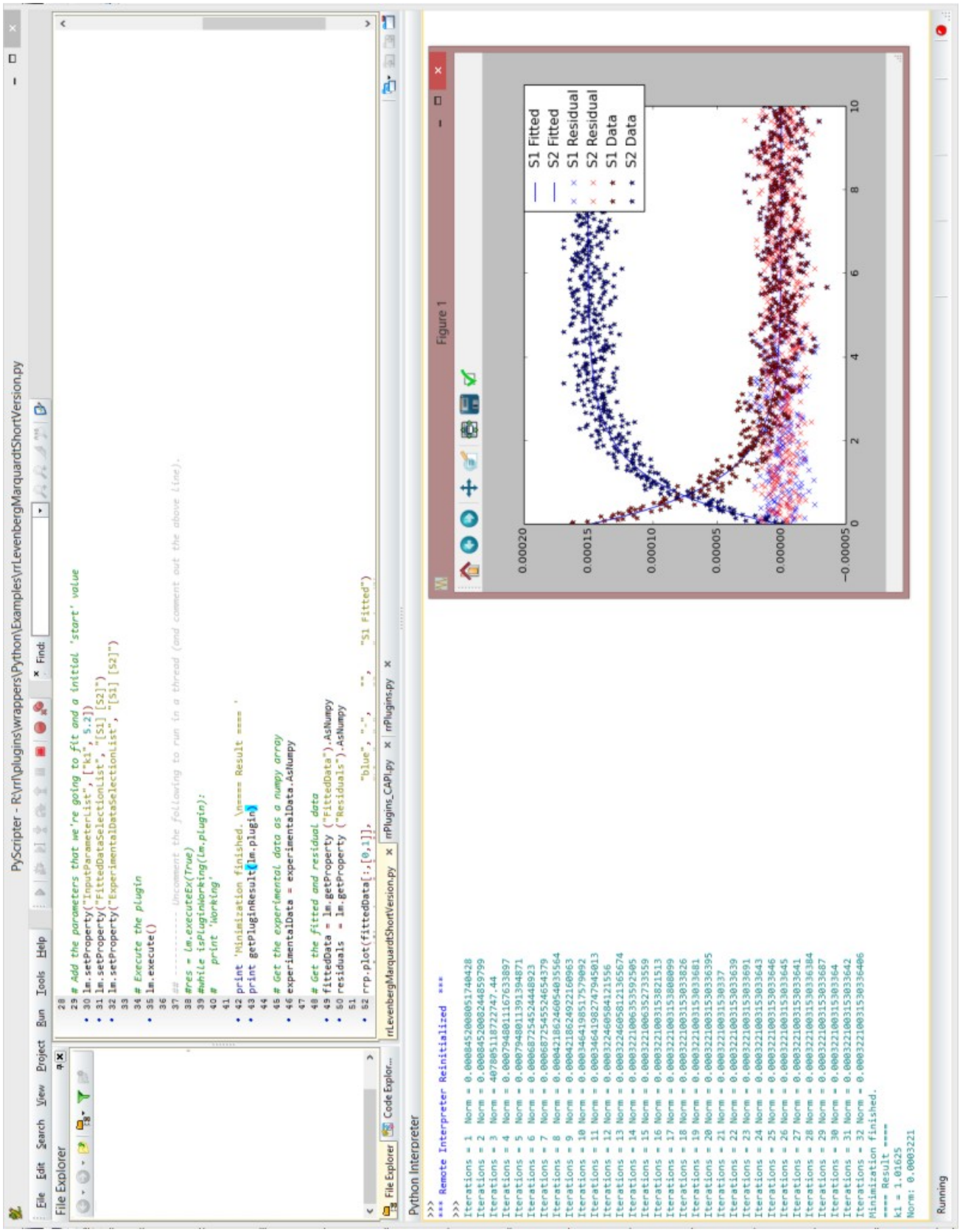

Figure 1.1: Output for the LMFit python example script discussed above.