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 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. In addition, various statistical properties, such as standardized residuals, Q-Q data, ChiSquare and reduced ChiSquare are made accessible to the user. The resulting parameter values do also come with estimated confidence limits.

The current implementation is based on the lmfit C library by Joachim Wuttke[1].

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.

Property 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.
InputParameterList	listOfProperties	N/A	Parameters to fit.
OutputParameterList	listOfProperties	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.
ConfidenceLimits	listOfProperties	N/A	Confidence limits for each fitted parameter. The confidence limits are calculated at a 95% confidence level.
Hessian	matrix	N/A	Hessian matrix. The Hessian is calculated using approximation at a found parameter minimum.

CovarianceMatrix	matrix	N/A	Covariance matrix. Calculated as the inverse of the Hessian.
Residuals	telluriumData	N/A	Residuals data.
StandardizedResiduals	telluriumData	N/A	Standardized Residuals.
NormalProbabilityOfResiduals	telluriumData	N/A	Normal Probability of Residuals.
ChiSquare	double	N/A	The ChiSquare at the minimum.
ReducedChiSquare	double	N/A	The Reduced ChiSquare at the minimum.
StatusMessage	string	N/A	Message from the internal fitting engine, communicating the status of the obtained fit.
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: Levenberg-Marquardt plugin properties

1.3 The `execute(bool inThread)` function

The `execute()` function will start the Levenberg-Marquardt algorithm. Depending on the problem at hand, the algorithm may run for a long time.

The `execute(bool inThread)`, do support a boolean argument indicating if the execution of the plugin work will be done in a thread, or not. Threading is fully implemented in the Levenberg-Marquardt plugin.

The inThread argument defaults to **false**.

1.4 Plugin Events

The Levenberg-Marquardt 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 assignOn() 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: Plugin events

1.5 Python example

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

```
from telplugins import *

# Load Plugins
chiPlugin       = Plugin("tel_chisquare")
lm              = Plugin("tel_levenberg_marquardt")
modelPlugin     = Plugin("tel_test_model")
addNoisePlugin  = Plugin("tel_add_noise")

try:
    #========== EVENT FUNCTION SETUP ============================
    def myEvent(dummy): #We are not capturing any data from the plugin, so just pass a dummy
        print 'Iteration, Norm = ' + `lm.getProperty("NrOfIter")`  + ', ' + `lm.getProperty("Norm")`

    #Setup progress event function
    progressEvent =  NotifyEventEx(myEvent)
    assignOnProgressEvent(lm.plugin, progressEvent)
    #============================================================

    #Create model data, with and without noise using the test_model plugin
    modelPlugin.execute()

    #Setup lmfit properties.
    lm.SBML             = modelPlugin.Model
    lm.ExperimentalData = modelPlugin.TestDataWithNoise

    # Add the parameters that we're going to fit and an initial 'start' value
    lm.setProperty("InputParameterList", ["k1", .3])
    lm.setProperty("FittedDataSelectionList", "[S1] [S2]")
    lm.setProperty("ExperimentalDataSelectionList", "[S1] [S2]")

    # Start minimization
    lm.execute()

    print 'Minimization finished. \n==== Result ===='
    print 'Fit engine status: ' + `lm.getProperty('StatusMessage')`

    print 'Hessian Matrix'
    print lm.getProperty("Hessian")

    print 'Covariance  Matrix'
    print lm.getProperty("CovarianceMatrix")

    print 'ChiSquare = '            + `lm.getProperty("ChiSquare")`
    print 'Reduced ChiSquare = '    + `lm.getProperty("ReducedChiSquare")`

    #This is a list of parameters
    parameters = tpc.getPluginProperty (lm.plugin, "OutputParameterList")
    confLimits = tpc.getPluginProperty (lm.plugin, "ConfidenceLimits")

    #Iterate trough list of parameters and confidence limits
    para  = getFirstProperty(parameters)
    limit = getFirstProperty(confLimits)
```

```
53      while para and limit:
54          print getPropertyName(para) + ' = ' + `getPropertyValue(para)` + '
               +/- ' + `getPropertyValue(limit)`
55          para  = getNextProperty(parameters)
56          limit = getNextProperty(confLimits)
57
58
59      # Get the fitted and residual data
60      fittedData = lm.getProperty ("FittedData").toNumpy
61      residuals  = lm.getProperty ("Residuals").toNumpy
62
63      # Get the experimental data as a numpy array
64      experimentalData = modelPlugin.TestDataWithNoise.toNumpy
65
66      telplugins.plot(fittedData          [:,[0,1]], "blue", "-",    "",    "
           S1 Fitted")
67      telplugins.plot(fittedData          [:,[0,2]], "blue", "-",    "",    "
           S2 Fitted")
68      telplugins.plot(residuals           [:,[0,1]], "blue", "None", "x",   "
           S1 Residual")
69      telplugins.plot(residuals           [:,[0,2]], "red",  "None", "x",   "
           S2 Residual")
70      telplugins.plot(experimentalData    [:,[0,1]], "red",  "",     "*",   "
           S1 Data")
71      telplugins.plot(experimentalData    [:,[0,2]], "blue", "",     "*",   "
           S2 Data")
72      telplugins.plt.show()
73
74  except Exception as e:
75      print 'Problem.. ' + `e`
```

Listing 1.1: Minimization example.

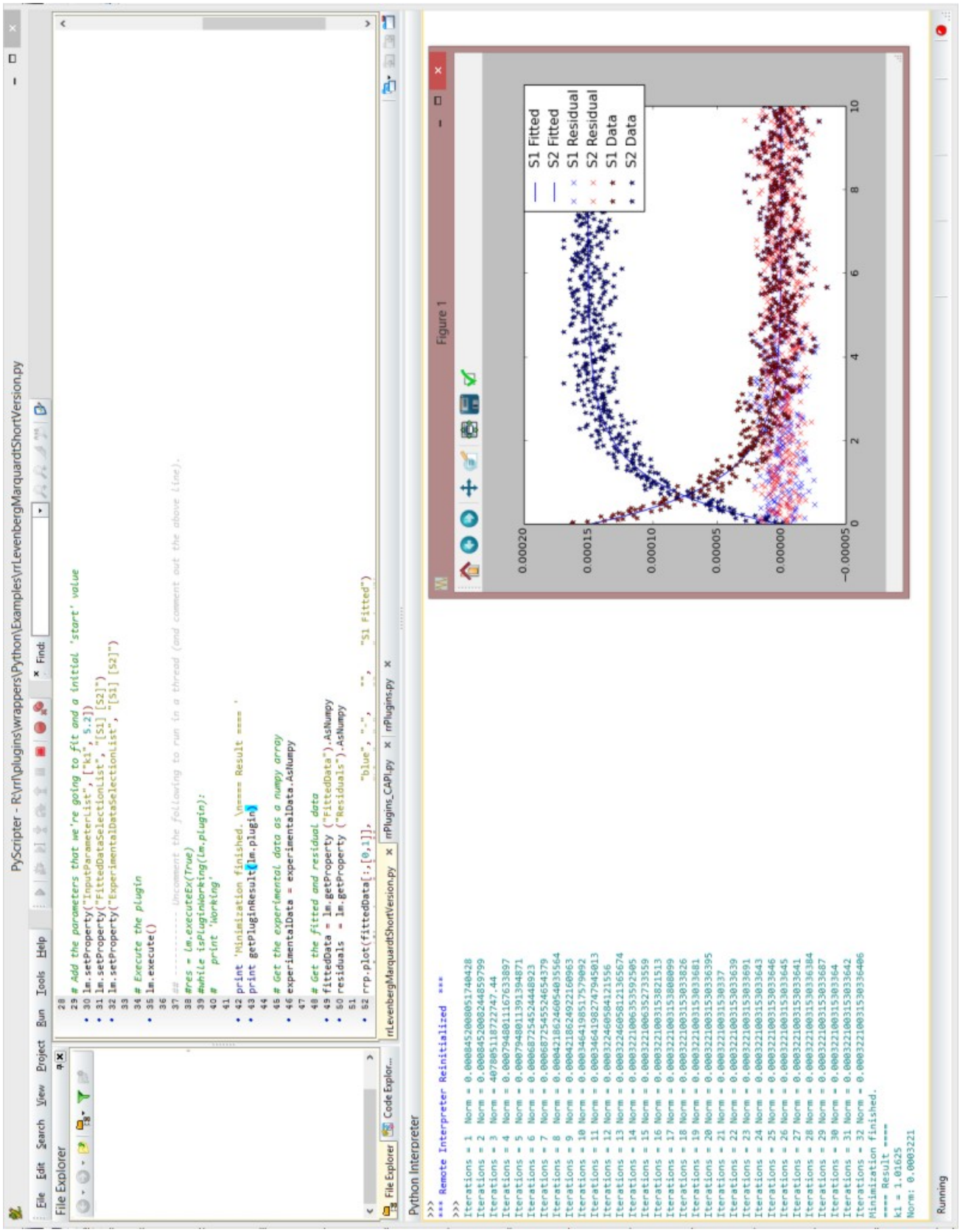

Figure 1.1: Typical output for the example script above.