Parameter Minimization using the Nelder-Mead algorithm

Do some fitting

1.1 Introduction

The Nelder-Mead plugin is used to fit an SBML model's 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 also come with estimated confidence limits.

The current implementation is based on the Nelder-Mead C implementation by Michael F. Hutt[1].

Plugin properties are documented in the next section.

[1] An implementation of the Nelder-Mead simplex method. Copyright (c) 1997-2011 Michael F. Hutt

1.2 Plugin Properties

Available properties in the Nelder-Mead 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 (Currently not used).
NrOfIter	int	N/A	Number of (internal outer loop) iterations.
NrOfFuncIter	int	N/A	Number of objective function iterations.

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

Epsilon	double	1.e-6	Convergence tolerance.
Scale	double	1	Scaling of vertices.
MaxNrOfIterations	int	1000	Maximum number of iterations.
Alpha	double	1.	Reflection coefficient.

Beta	double	0.5	Contraction coefficient.
Gamma	double	1	Expansion coefficient.

Table 1.1: Plugin Properties

1.3 Plugin Events

The Nelder-Mead plugin uses all of the 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 plugin's 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

1.4 Python example

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

```
1  from teplugins import *
2
3  # Load Plugins
4  chiPlugin       = Plugin("tel_chisquare")
5  nm              = Plugin("tel_nelder_mead")
6  modelPlugin     = Plugin("tel_test_model")
7  addNoisePlugin  = Plugin("tel_add_noise")
8
9  try:
10     #========== EVENT FUNCTION SETUP ============================
11     def myEvent(dummy): #We are not capturing any data from the plugin, so
           just pass a dummy
12         print 'Iteration, Norm = ' + `nm.getProperty("NrOfIter")`  + ', ' +
               `nm.getProperty("Norm")`
13
14     #Setup progress event function
15     progressEvent =  NotifyEventEx(myEvent)
16     assignOnProgressEvent(nm.plugin, progressEvent)
```

```
17      #============================================================
18
19      #Create model data, with and without noise using the test_model plugin
20      modelPlugin.execute()
21
22      #Setup plugin properties.
23      nm.SBML             = modelPlugin.Model
24      nm.ExperimentalData = modelPlugin.TestDataWithNoise
25
26      # Add the parameters that we're going to fit and an initial 'start'
            value
27      nm.setProperty("InputParameterList", ["k1", .3])
28      nm.setProperty("FittedDataSelectionList", "[S1] [S2]")
29      nm.setProperty("ExperimentalDataSelectionList", "[S1] [S2]")
30
31      # Start minimization
32      nm.execute()
33
34      print 'Minimization finished. \n==== Result ===='
35
36      print 'Hessian Matrix'
37      print nm.getProperty("Hessian")
38
39      print 'Covariance  Matrix'
40      print nm.getProperty("CovarianceMatrix")
41
42      print 'ChiSquare = '            + `nm.getProperty("ChiSquare")`
43      print 'Reduced ChiSquare = '    + `nm.getProperty("ReducedChiSquare")`
44
45      #This is a list of parameters
46      parameters = tpc.getPluginProperty (nm.plugin, "OutputParameterList")
47      confLimits = tpc.getPluginProperty (nm.plugin, "ConfidenceLimits")
48
49      #Iterate trough list of parameters and confidence limits
50      para  = getFirstProperty(parameters)
51      limit = getFirstProperty(confLimits)
52      while para and limit:
53          print getPropertyName(para) + ' = ' + `getPropertyValue(para)` + '
                +/- ' + `getPropertyValue(limit)`
54          para  = getNextProperty(parameters)
55          limit = getNextProperty(confLimits)
56
57
58      # Get the fitted and residual data
59      fittedData = nm.getProperty ("FittedData").toNumpy
60      residuals  = nm.getProperty ("Residuals").toNumpy
61
62      # Get the experimental data as a numpy array
63      experimentalData = modelPlugin.TestDataWithNoise.toNumpy
64
65      telplugins.plot(fittedData         [:,[0,1]], "blue", "-",    "",    "
            S1 Fitted")
66      telplugins.plot(fittedData         [:,[0,2]], "blue", "-",    "",    "
            S2 Fitted")
```

```
67      telplugins.plot(residuals           [:,[0,1]], "blue", "None", "x",    "
            S1 Residual")
68      telplugins.plot(residuals           [:,[0,2]], "red",  "None", "x",    "
            S2 Residual")
69      telplugins.plot(experimentalData    [:,[0,1]], "red",  "",     "*",    "
            S1 Data")
70      telplugins.plot(experimentalData    [:,[0,2]], "blue", "",     "*",    "
            S2 Data")
71      telplugins.plt.show()
72
73      #Finally, view the manual and version
74      nm.viewManual()
75      print 'Plugin version: ' + `nm.getVersion()`
76
77  except Exception as e:
78      print 'Problem.. ' + `e`
```

Listing 1.1: Minimization example.

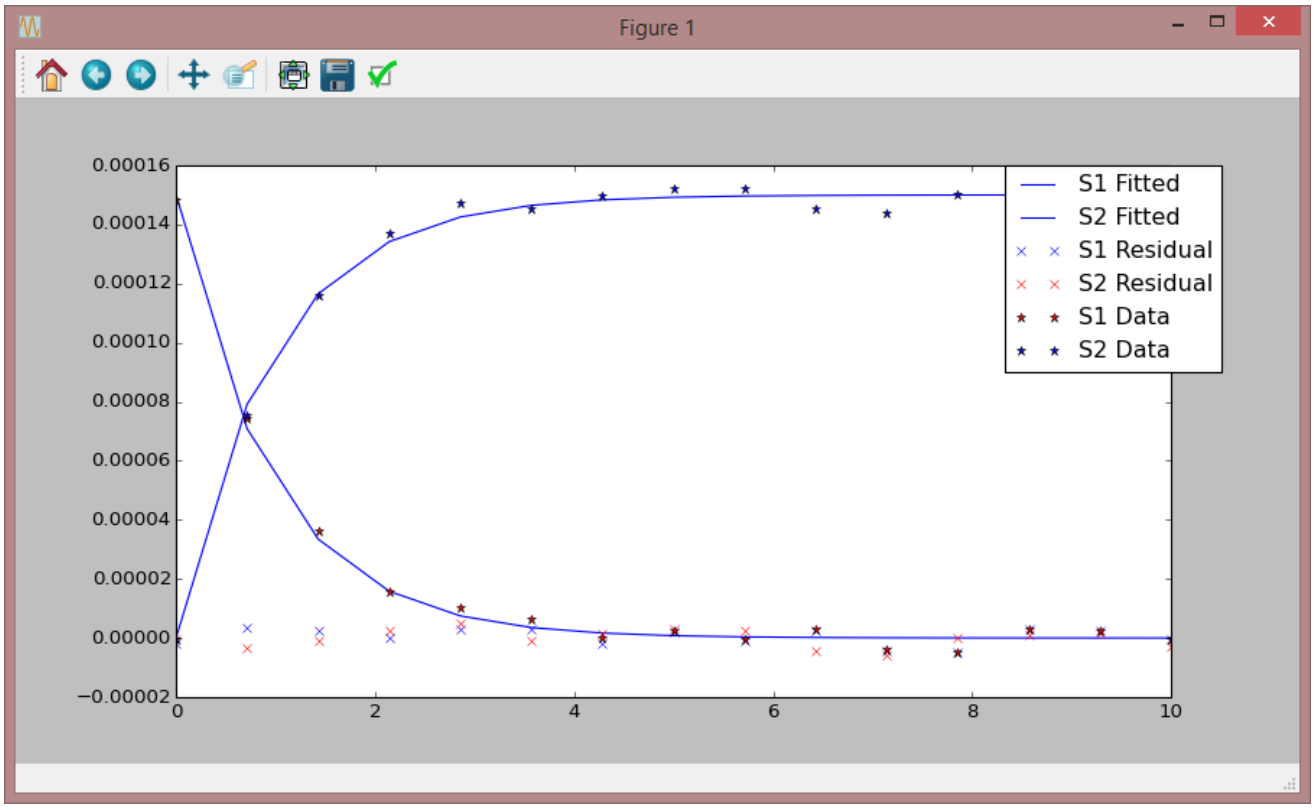

Figure 1.1: Typical output for the example script above.