Add Noise Plugin

Add some noise

1.1 Introduction

The purpose of the *AddNoise* plugin is to introduce random noise to *Tellurium* data.

Generation of actual noise is using the fact that a Rayleigh-distributed random variable R, with the probability distribution $F(R) = 0$ if $R < 0$ and $F(R) = 1 - exp(-R^2/2 * \sigma^2)$ if $R >= 0$, is related to a pair of Gaussian variables C and D through the transformation $C = R * cos(\theta)$ and $D = R * sin(\theta)$, where θ is a uniformly distributed variable in the interval $(0, 2 * \pi())$ [1]

Currently only Gaussian noise is implemented.

1.2 Plugin Parameters

Table 1.1 lists available plugin property names, along with their data type and purpose.

1.3 Plugin Callbacks

The AddNoiseplugin 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

[1] From Contemporary Communication Systems USING MATLAB(R), by John G. Proakis and Masoud Salehi, published by PWS Publishing Company, 1998, pp 49-50.

Parameter Name	Data Type	Purpose
InputData	TelluriumData	Data on which noise will be applied to.
Sigma,(σ)	double	Size of applied noise. Noise is generated for each single data value, with a probability corresponding to a Gaussian distribution, centered around the value, and with a variance equal to σ^2 .
NoiseType	int	Type of noise applied on data. Only Gaussian noise is currently supported.
Progress	double	The progress property communicates the progress (in percent) of Noise application.

Table 1.1: Add noise Plugin Parameters

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.

Callback	Arguments	Purpose
PluginStarted	void*, void*	Signal to application that the plugin has started applying noise on data. Both parameters are *pass trough* parameters and are unused internally by the plugin.
PluginProgress	void*, void*	Communicating progress of noise generation. 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: AddNoise Plugin callbacks

1.4 The `execute(bool inThread)` function

The `execute()` function will apply noise to all rows and columns of the assigned data, with one exception. Data not affected are data in the first column, and if, and only if, its column header equals "time" (case insensitive).

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 AddNoise plugin.

The inThread argument defaults to **false**.

1.5 Python examples

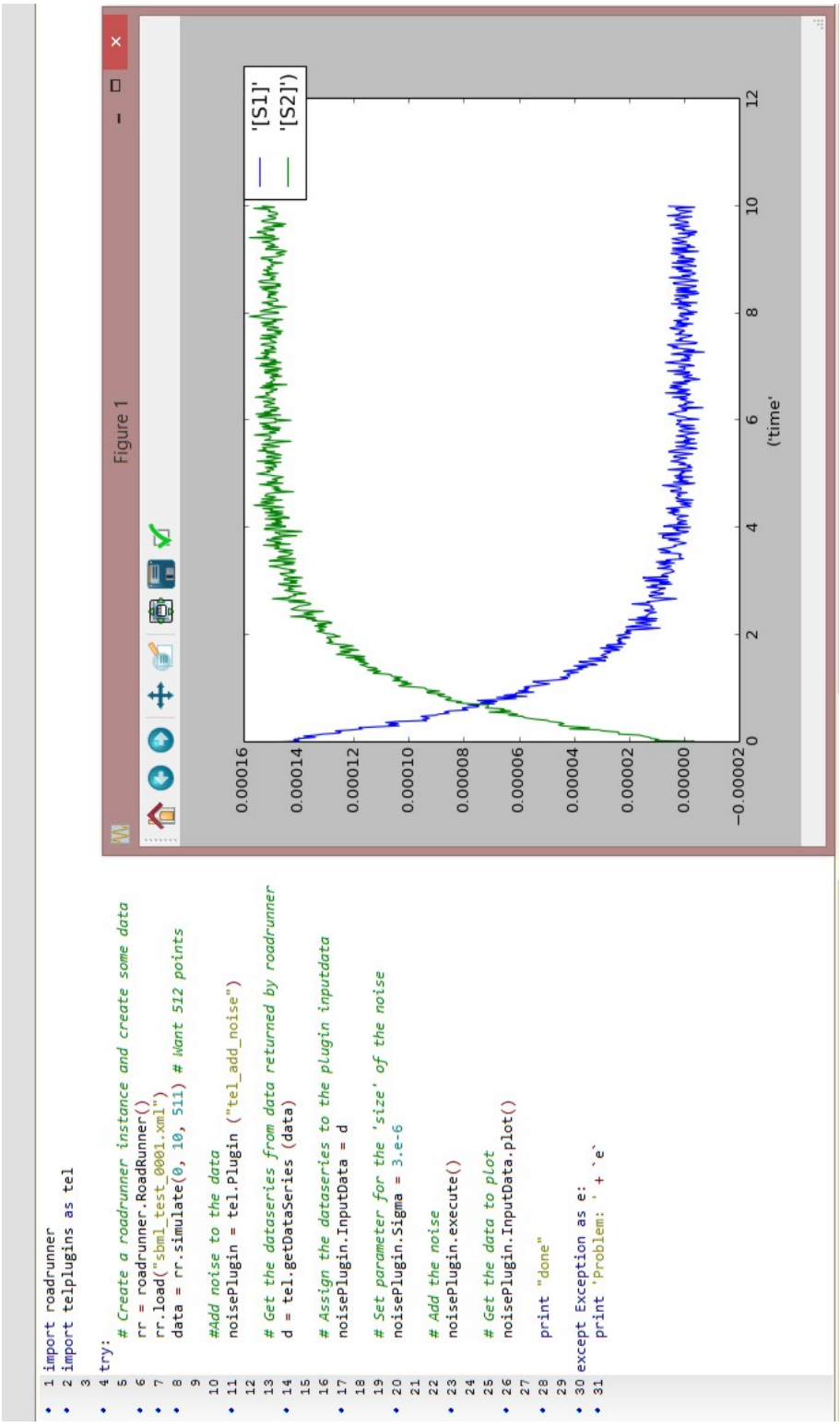
1.5.1 Add noise to data acquired from RoadRunner

The python script below shows how to acquire simulation data from RoadRunner and pass it to he noise plugin. The format of the data, that is obtained from the `simulate()` function (line 8), is not directly compatible with the Noise plugins InputData property. This incompatibility is handled by an intermediate data structure in Python, that is called DataSeries (line 14). The Plugins properties, InputData and Sigma, is assigned on line 17 and 20 respectively. Line 23 denote the execution of the noise plugin, and after that has finished, data can be visualized by using the `plot` function (line 26). The output is shown below the script.

```
1  import roadrunner
2  import telplugins as tel
3
4  try:
5      # Create a roadrunner instance and create some data
6      rr = roadrunner.RoadRunner()
7      rr.load("sbml_test_0001.xml")
8      data = rr.simulate(0, 10, 511) # Want 512 points
9
10     #Add noise to the data
11     noisePlugin = tel.Plugin ("tel_add_noise")
12
13     # Get the dataseries from data returned by roadrunner
14     d = tel.getDataSeries (data)
15
16     # Assign the dataseries to the plugin inputdata
17     noisePlugin.InputData = d
18
19     # Set parameter for the 'size' of the noise
20     noisePlugin.Sigma = 3.e-6
21
22     # Add the noise
23     noisePlugin.execute()
24
25     # Get the data to plot
26     noisePlugin.InputData.plot()
```

```
27
28 except Exception as e:
29     print 'Problem: ' + `e`
```

Listing 1.1: Add noise example.




```
1 import roadrunner
2 import telplugins as tel
3
4 try:
5   # Create a roadrunner instance and create some data
6   rr = roadrunner.RoadRunner()
7   rr.load("sbml_test_0001.xml")
8   data = rr.simulate(0, 10, 511) # Want 512 points
9
10  #Add noise to the data
11  noisePlugin = tel.Plugin ("tel_add_noise")
12
13  # Get the dataseries from data returned by roadrunner
14  d = tel.getDataSeries (data)
15
16  # Assign the dataseries to the plugin inputdata
17  noisePlugin.InputData = d
18
19  # Set parameter for the 'size' of the noise
20  noisePlugin.Sigma = 3.e-6
21
22  # Add the noise
23  noisePlugin.execute()
24
25  # Get the data to plot
26  noisePlugin.InputData.plot()
27
28  print "done"
29
30 except Exception as e:
31   print 'Problem: ' + `e`
```

Figure 1.1: Output for the AddNoise python example script discussed above

1.5.2 Visualization of the noise distribution used in the AddNoise plugin

The Python script below demonstrate how to obtain and visualize the actual distribution (Gaussian) of noise that is applied on data.

```
# Show that add noise plugin correctly computes Sigma (standard deviation)
import matplotlib.pyplot as plt
import scipy.stats as stats
import telplugins as tel
import numpy as np

p = tel.Plugin ("tel_add_noise")

value = 2.34    #This will be the mean
n = 80000
inputData  = np.zeros (shape=(1,2))
inputData[0] = [0, value]

data = tel.DataSeries.fromNumPy (inputData)
p.Sigma = 0.25

outArray = []
for i in range(n):
    p.InputData = data
    p.execute()
    outValues = p.InputData.toNumpy
    outArray.append(outValues[0][1])

plt.hist(outArray, 200, normed=True)

# Overlay analytical solution
aRange = np.arange(min(outArray), max(outArray), 0.001)
plt.plot(aRange, stats.norm.pdf(aRange, value, p.Sigma), linestyle='--',
   linewidth='2', color='red')

plt.show()
```

Listing 1.2: Noise distribution example.

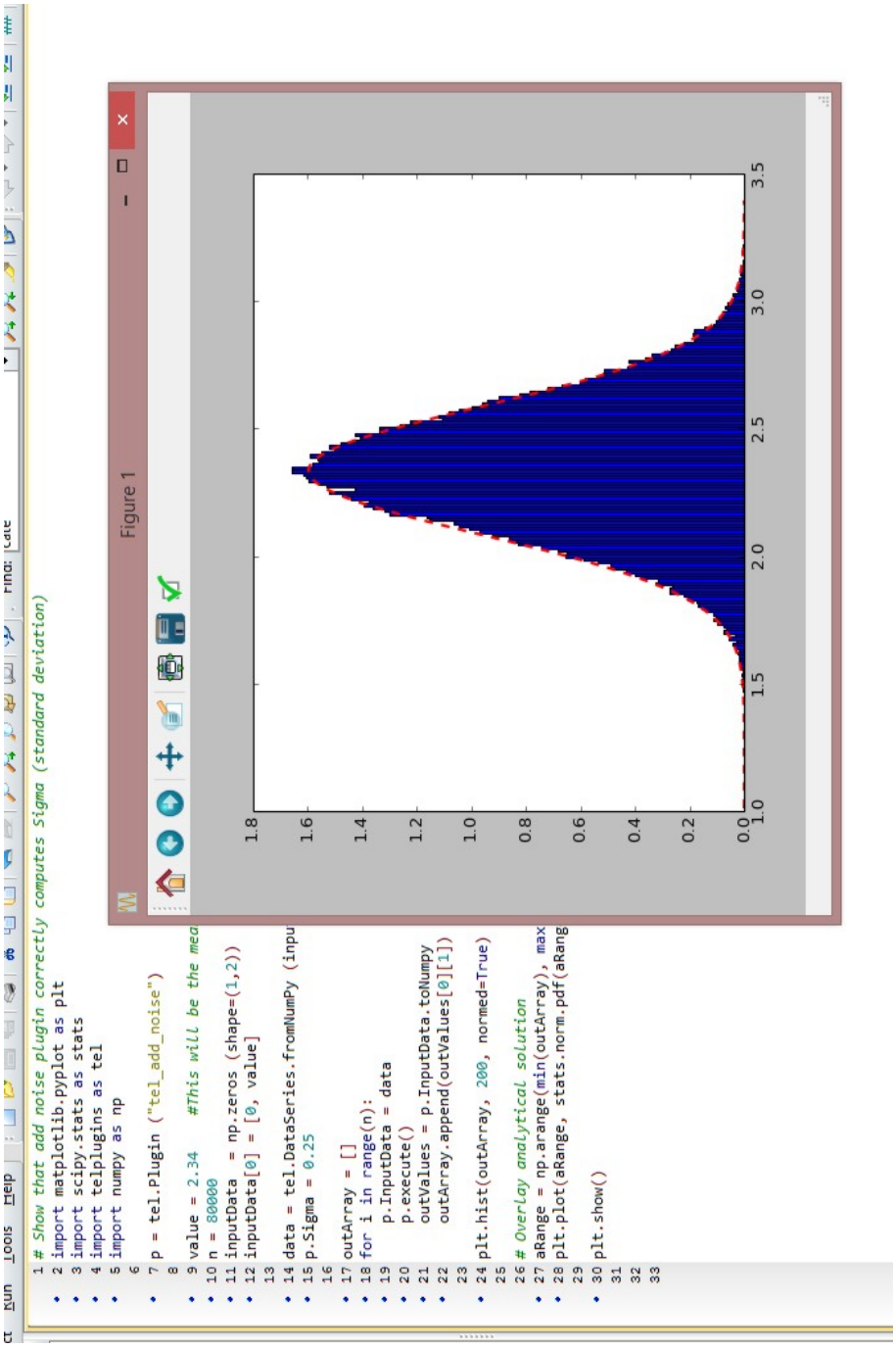


Figure 1.2: Output for the AddNoise python example script discussed above