Useful Python Functions within the General Simulation Tool Script Console

The Python script console exposes some useful libraries and functions. The most useful include graphing, loading models, ODE simulation (**sim**), stoichiometry analysis (**StructAnalysis**) and stochastic simulation (**gil**).

To list the function in the libraries, type the Python command dir(), eg dir (sim)

Graphing

ClearGraph()

Clears the main graph

Graph(m)

Clears the graphing area then plot a graph using the columns form the array, m

AddGraph (m)

Add a new graph to an existing plot, i.e it does not clear the graph first. Good for overlying multiple data sources.

Loading Models

All loading function, load the model into the host object and the simulator object (sim). To retrieve a loaded model, use host.GetChangedSBML()

LoadSBML()

Loads either a SBML model or a Jarnac model string. eg

```
LoadSBML(
"""p = defn model
   J1: S1 -> $S2; Vm*(S1/Ks)*(1-S2/S1/Keq)*pow(S1/Ks+S2/Kp,n-
1)/(1+pow(S1/Ks+S2/Kp,n));
```

```
end;

p.S2 = 0.000001;
p.S1 = 1E-4;
p.Vm = 1;
p.Ks = 0.4;
p.Keq = 5;
p.Kp = 1.4;
p.n = 4;""");
```

Open()

Brings up an open file dialog box to allow a user to either select a SBML model or a Jarnac model.

OpenFile('/path/to/jarnacfile.jan')

Loads either a Jarnac or SBML file from the file system

sbmlStr = host.GetChangedSBML ()

Gets the currently loaded SBML. Use this to retrieve the SBML of a loaded model so that it can be loaded in to other libraries, eg the Gillespie or structural analysis library.

Simulation

ODE simulation is provided by sim library. Type dir (sim) to get the full list.

result = sim.simulateEx (startTime, endTime, numberOfPoints)

Carry out a simulation on the current model, returns and array of all floating species values with time in the first column.

```
result = sim.simulateEx (0, 10, 100)
```

result = sim.oneStep(currentTime, timeStep)

Performs one step simulation using the current model. Returns the new time (currentTime + timeStep)

```
currentTime = sim.oneStep (currentTime, 2.0)
```

result = sim.steadyState()

Find the steady state for the current model, returns value indicating how close the solution is to the steady state.

```
d = steadyState()
```

sim.setValue (name, value)

Set the value for a parameter or variable. For example:

```
sim.setValue ('S1', 3.4)
```

result = sim.getValue (name, value)

Obtain the value for a parameter or variable. For example:

```
d = sim.getValue ('S1')
```

Stoichiometry Analysis

Analysis on the stoichiometry matrix can be carried out using the StuctAnalysis library. Type dir (StructAnalysis) to get the full list. Here are some selected functions.

StructAnalysis.LoadModel (SBML)

Load the sbml string into the structural analysis library. For example:

```
print StructAnalysis.loadSBML (host. getChangedSBML())
```

StructAnalysis.GetReorderedStoichiometryMatrix ()

Returns the reordered stoichimetry matrix. If you wish to have the original stoichiometry matrix then use GetStoichiometryMatrix()

StructAnalysis.GetReorderedStoichiometryMatrixLabels ()

Returns the row and column labels in the reordered matrix. Use GetStoichiometryMatrixLabels to get the labels for the original matrix.

StructAnalysis.GetGammaMatrix ()

Returns the conservation laws for the given matrix. Note that the column labels corresponds to the row labels in the reordered stoichiometry matrix.

Stochastic Simulation

Stochastic simulation can be carried out using the gil library which implements a standard Gillespie stochastic simulator. Type dir (gil) to get the full list.

gil.loadSBML (sbml)

Load the sbml string into the structural analysis library. For example:

```
gil.loadSBML (host. getChangedSBML())
```

result = gil.simulate (startTime, endTime, samplePeriod)

Carry out a Gillespie simulation from time start to time end. The samplePeriod indicates how often to take a measurement. A samplePeriod = 1 means take every data point that the Gillespie algorithm generates, whereas samplePeriod = 10 means take every 10th point. Note that time intervals in a Gillespie simulation are not regular, to get regular time points use the grid based calls.

newTime = gil.moveOneStep (currentTime, timeStop)

Move the Gillespie simulator on one step. The call returns the new time. If the simulator is unable to move forward then it returns timeStop.

result = gil.simulateOnGrid (startTime, endTime, gridSize);

Carry out a Gillespie simulation from time start time end. The gridSize indicates when to record data from the simulation. A gridSize = 1 means record data at intervals of time one. The units of time will depend on the units given to the rate constants in the model.

result = gil.simulateMeanOnGrid (startTime, endTime, gridSize, populationSize)

The above call is the same as the previous one except it will do multiple simulations indicated by the populationSize argument. From the population of runs, the method will return columns corresponding to the mean changes. For example, the following call will run 10,000 simulations on a grid of width 0.1 and returns the mean change in species levels:

```
gil.simulateMeanOnGrid (0.0, 10.0, 0.1, 10000);
```

result = gil.simulateMeanAndSDOnGrid (startTime, endTime, gridSize, populationSize);

The above call is the same as the previous one except it will do multiple simulations indicated by the populationSize argument. From the population of runs, the method will return columns corresponding to the mean changes. For example, the following call will run 10,000 simulations on a grid of width 0.1 and returns the mean change in species levels:

```
gil.SimulateMeanOnGrid (0.0, 10.0, 0.1, 10000);
```

gil.setParameter (parameterName, value)

This call enables a parameter in the model to be changed without having to reload the SBML model into the Gillespie solver. For example:

```
gil.SetParameter ("k1", 0.1234);
```

gil.setSeed (integer seed)

There are times when simulations runs need to be repeated exactly, this can be accomplished by setting the random number generator seed. This ensures that runs with a given random number seed will be identical.

result = gil.getNamesOfParameters()

Return parameters names in the model.

result = gil.getNamesOfVariables()

Return variables names in the model

result = gil.getNamesOfBoundarySpecies()

Return list of boundary species names.

result = gil.getNumberOfFloatingSpecies()

Returns the number of floating species in the model

Examples:

1. How to access biomodels from the General Simulation Tool (GST) and simulate it?

```
model = biomodels['BIOMD0000000204']
LoadSBML (model.SBML)
d = sim.simulateEx(0, 10, 100)
Graph (d)
```

2. How to run a stochastic simulation

```
gil.loadSBML(sbml)
gil.setSeed(0)
startTime = 0
endTime = 10
result = gil.simulate(startTime, endTime)
graph (m)
```

3. How to compute the elasticity of a reaction over a range of concentrations

```
m = Util.CreateArray(100,2);
sim.reset();
for i in range(100):
       m[i,0] = sim.getValue('S1');
       m[i,1]= sim.getValue('EE:J1,S1');
       print  sim.setValue('S1', m[i, 0] + 0.2);
Graph(m);
```

4. How to compute the reaction rate/flux through a step

```
sim.setValue('S1', 0.45);
sim.Instance.model.computeAllRatesOfChange();
flux = sim.getValue ('J1');
```

5. How to compute the control coefficient for a reaction

```
sim.Instance.computeSteadyStateValue('CC:J1,E1');
```

6. How to specify a Jarnac script

```
LoadSBML(
"""p = defn NewModel
 J1: $S1 -> S2; E1*(k1*S1 - k2*S2);
 J2: S2 -> $S3; E2*(k3*S2 - k4*S3);
end;

p.S1 = 1;
p.S2 = 2;
p.S3 = 0;
p.k1 = 0.4;
p.k2 = 3.5;
p.k3 = 0.23;
p.k4 = 0.11;
p.E1 = 1;
p.E2 = 1;""");
```

7. How to load a Jarnac script from a file

```
OpenFile('/path/to/jarnacfile.jan')
```

Or

```
Open()
```

Use the following to obtain the current directory:

```
import os
print os.getcd()
```