Introduction to Tellurium

Herbert M Sauro

University of Washington

hsauro@uw.edu

October 6, 2021

Tellurium

Tellurium is an integrated platform based on Python and spyder2. It runs on Mac, Windows and Linux. It includes the following libraries:

libRoadRunner: A high performance SBML simulation library.

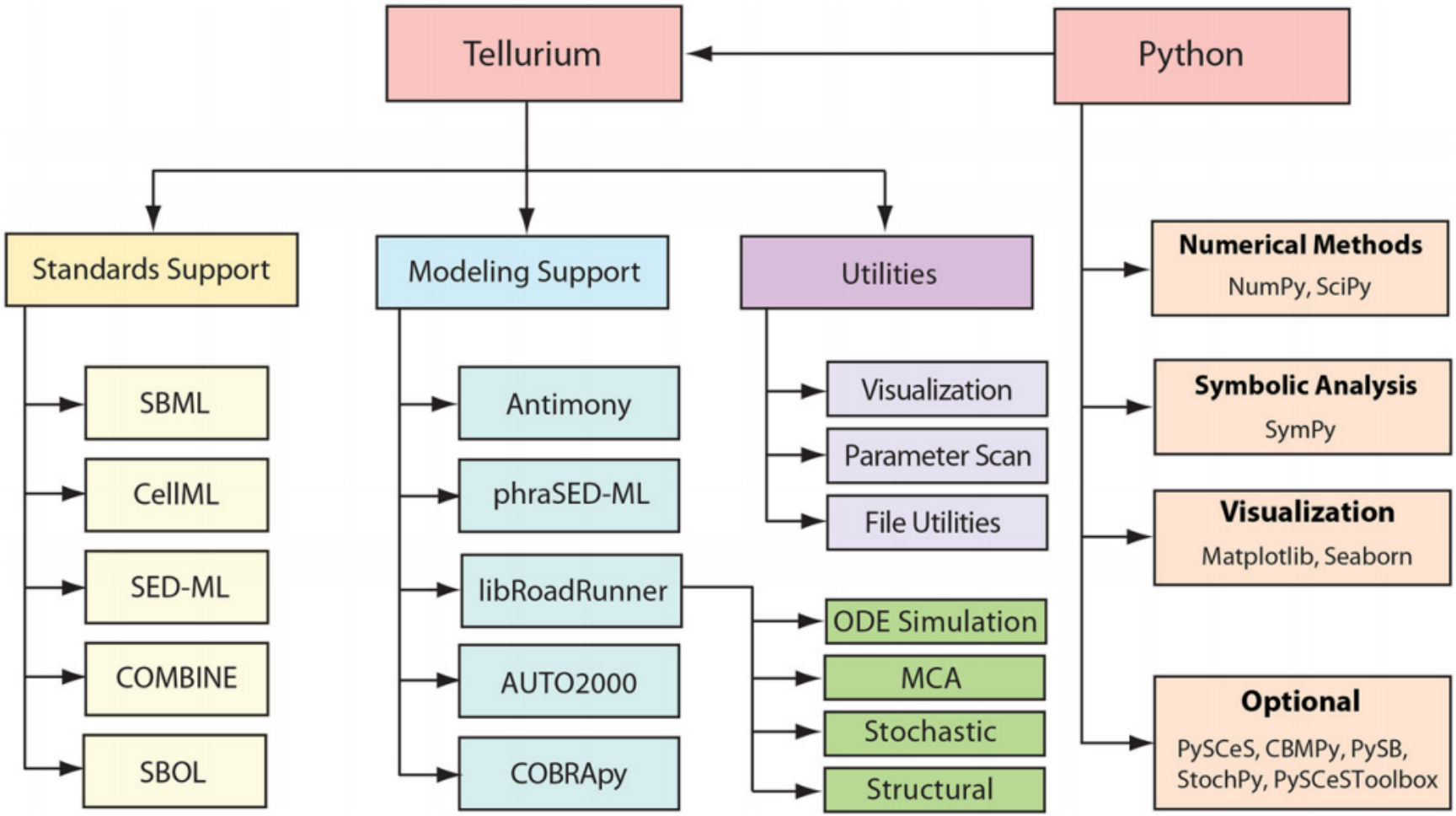
Antimony: Allows users to write models in a more human readable form.

SBML2Matlab: Allows users to export models in Matlab format

In addition Tellurium comes preloaded with the Python plotting library **Matplotlib**, the array package **numpy**, **scipy** as well as other useful additions. Tellurium also comes with a small number of helper subroutines to make it easier for the average modeler.

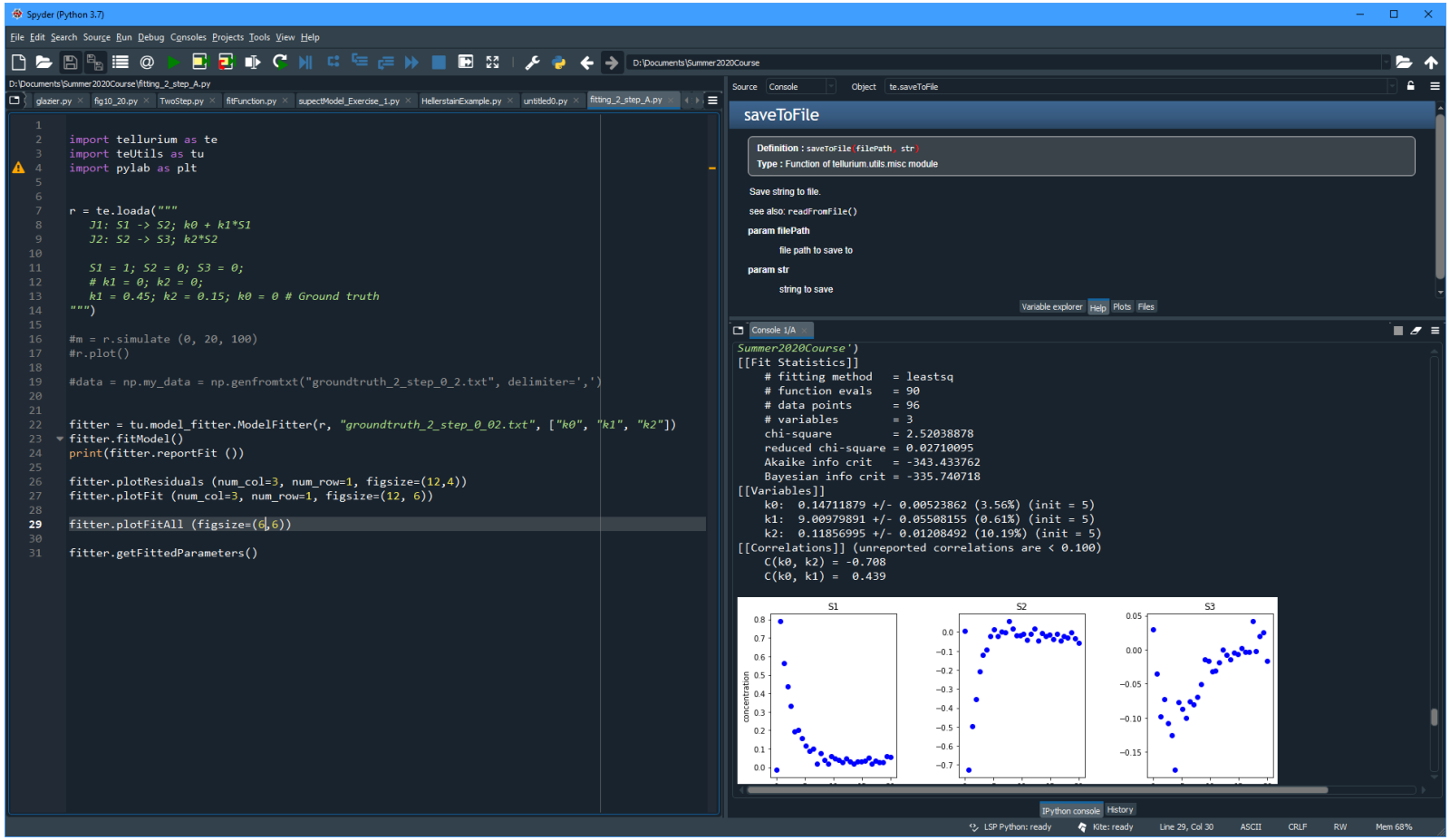
Overall Design

Tellurium Structure



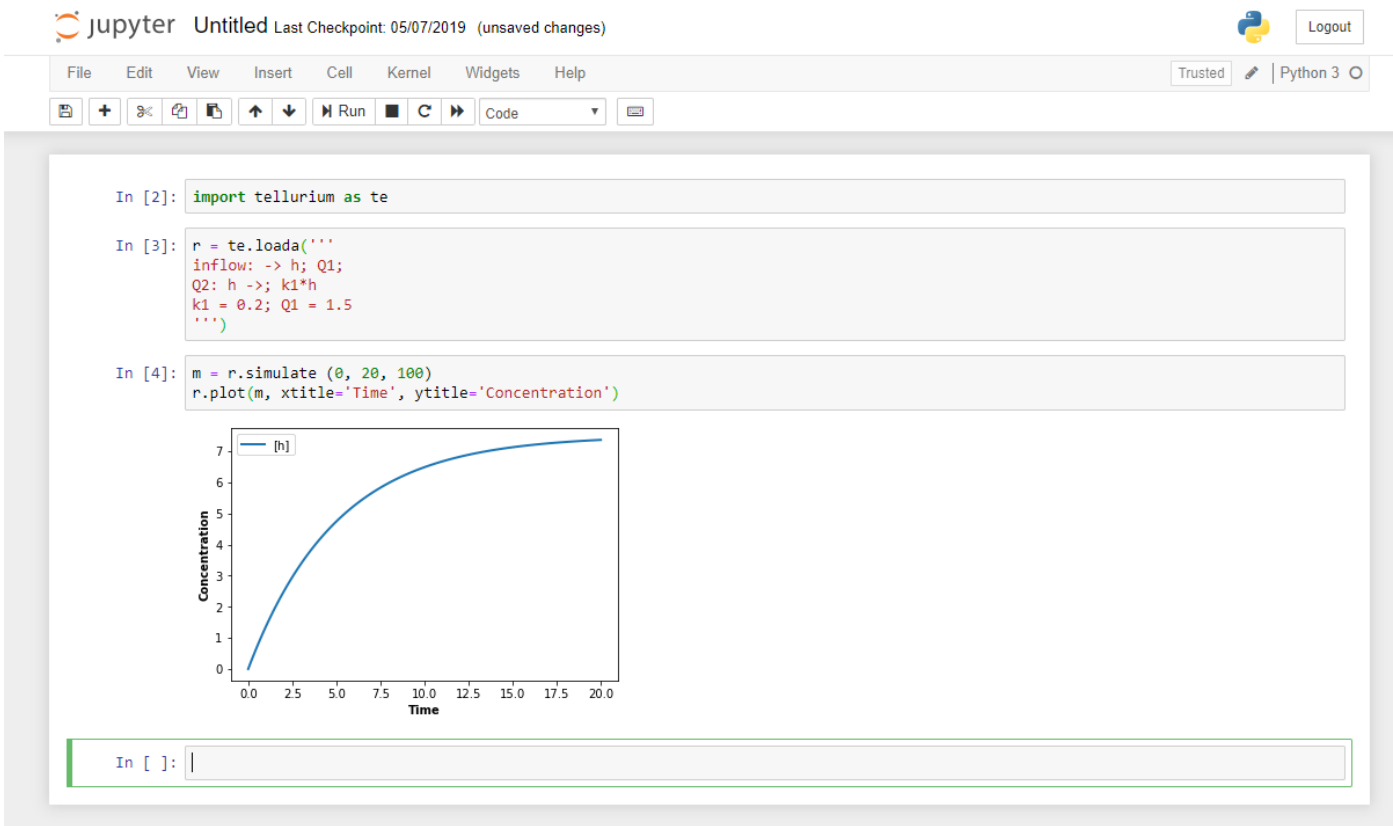
Interface

Tellurium IDE Interface using Spyder



Interface

Tellurium Jupyter Notebook Interface



First Example - running a simulation

Example

```
import tellurium as te

r = te.loada ('''
    S1 -> S2; k1*S1;
    S2 -> S3; k2*S2;

    k1 = 0.1; k2 = 0.45;
    S1 = 10; S2 = 0; S3 = 0
''')

result = r.simulate (0, 40, 100)
r.plot ()
```

Antimony

Example (Simple Model)

```
S1 -> S2; k1*S1;

k1 = 0.1; S1 = 10; S2 = 0
```

$$\frac{dS_1}{dt} = -k_1 S_1$$

$$\frac{dS_2}{dt} = k_1 S_1$$

Antimony

Example (Multiple Reactions)

```
S1 -> S2; k1*S1;
S2 -> S3; k2*S2;

k1 = 0.1; k2 = 0.2;
S1 = 10; S2 = 0; S3 = 0
```

$$\frac{dS_1}{dt} = -k_1 S_1$$

$$\frac{dS_2}{dt} = k_1 S_1 - k_2 S_2$$

$$\frac{dS_3}{dt} = k_2 S_2$$

Antimony

Example (Rate Laws)

```
S1 -> S2; k1*S1 - k2*S2;      # Reversible
S2 -> S3; Vmax*S3/(Km + S3); # Michaelis-Menten

k1 = 0.1; k2 = 0.2; Vmax = 10; Km = 0,4
S1 = 10; S2 = 0; S3 = 0
```

Antimony

Example (Bimolecular Reactions)

```
S1 + S2 -> S3; k1*S1*S2;
S3 -> S4 + S4; k2*S3;

k1 = 0.1; k2 = 0.2;
S1 = 10; S2 = 0; S3 = 0
```

Antimony

Example (Boundary Species)

```
# This is a comment
# A $ means FIX the concentration of the
# species *unless* I say otherwise
$S1 -> S2; k1*S1;
S2 -> $S3; k2*S2;

k1 = 0.1; k2 = 0.2;
S1 = 10; S2 = 0; S3 = 0
```

Tellurium

Example (Loading a Model into the Simulator libRoadRunner)

```
import tellurium as te
r = te.loada ('''
  J1: $S1 -> S2; k1*S1;
  J2: S2 -> $S3; k2*S2;

  k1 = 0.1; k2 = 0.2;
  S1 = 10; S2 = 0; S3 = 0
''')
```

Tellurium

Example (Run a Simulation)

```
r = te.loada ('''
  J1: $S1 -> S2; k1*S1;
  J2: S2 -> $S3; k2*S2;

  k1 = 0.1; k2 = 0.2;
  S1 = 10; S2 = 0; S3 = 0
''')

result = r.simulate (0, 10, 100)
```

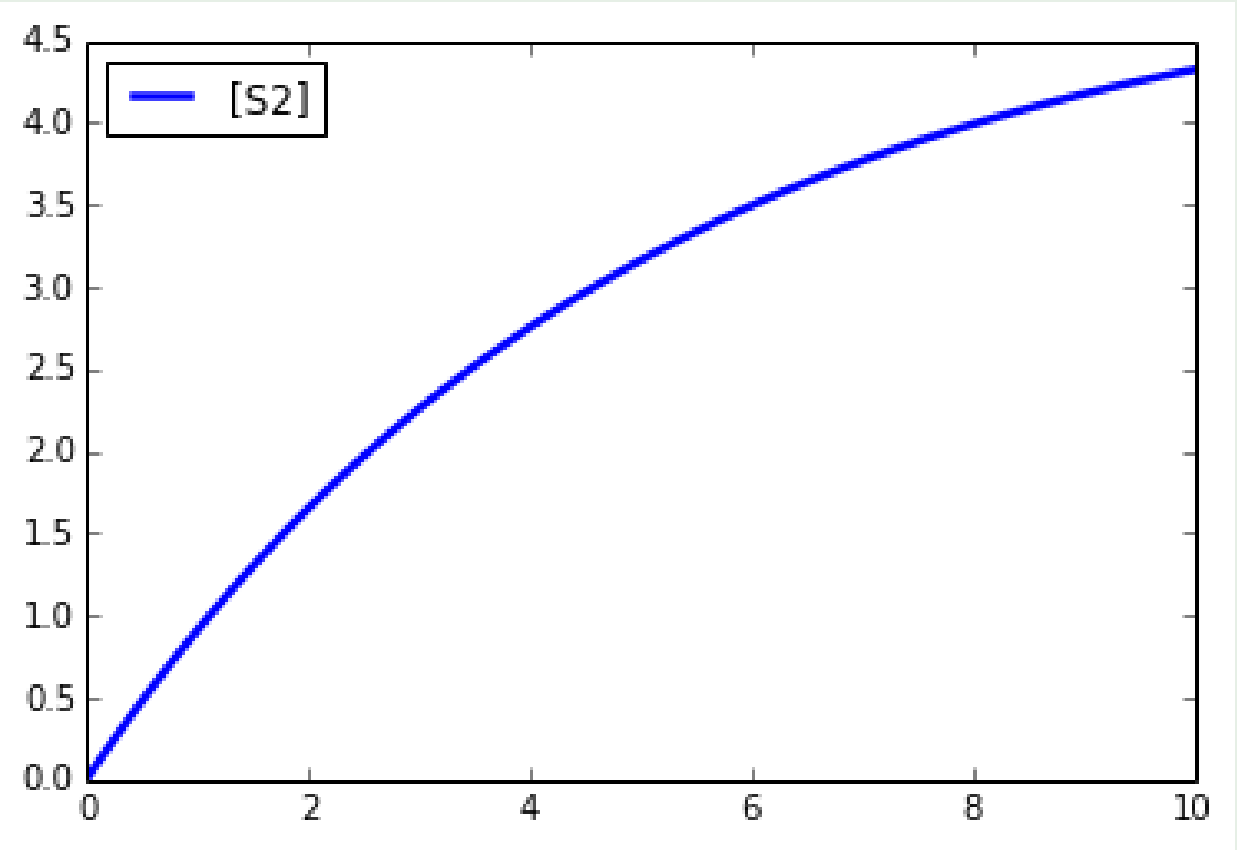
Tellurium

Example (Plotting Results)

```
r = te.loada ('''
  J1: $S1 -> S2; k1*S1;
  J2: S2 -> $S3; k2*S2;

  k1 = 0.1; k2 = 0.2;
  S1 = 10; S2 = 0; S3 = 0
''')

result = r.simulate (0, 10, 100)
r.plot ()
```

Tellurium

Example (Plotting Results)



Example (Exercise)

Run a simulation of this model (t = 0 to 20):

```
S1 -> S2; k1*S1
S2 -> S3 + 2 S4; k1*S2

k1 = 0.1; k2 = 0.2
S1 = 10
```

Find time and value for S2 when it reaches a maximum:

```
rowIndex = np.argmax (m[:,2])
print (m[rowIndex])
```

Tellurium

Example (Changing Values)

```
r = te.loada ('''
  J1: $S1 -> S2; k1*S1;
  J2: S2 -> $S3; k2*S2;

  k1 = 0.1; k2 = 0.2;
  S1 = 10; S2 = 0; S3 = 0
''')

r.k1 = 12.3
r.S1 = 20
result = r.simulate (0, 10, 100)
r.plot ()
```

Pause

Example (Controlling Boundary Species)

```
# A $ means FIX the concentration of the
# species *unless* I say otherwise

# **NOTE** the := symbol
# + 2 is to make sure it doesn't go negative
S1 := 4*sin (time*1.5) + 2;
$S1 -> S2; k1*S1;
S2 -> $S3; k2*S2;

k1 = 0.1; k2 = 0.2;
S2 = 0; S3 = 0 # S1 is NOT initialized
```

Antimony

Example (Adding extra calculations to the model)

```
switch := 1.0;
S1 := switch*(4*sin (time*1.5) + 2);

$S1 -> S2; k1*S1;
S2 -> $S3; k2*S2;

k1 = 0.1; k2 = 0.2;
S2 = 0; S3 = 0 # S1 is NOT initialized
```

Antimony

Example (Events)

```
# A $ means FIX the concentration of the
# species *unless* I say otherwise
$S1 -> S2; k1*S1;
S2 -> $S3; k2*S2;

at (time > 5): k2 = k2*2;

k1 = 0.1; k2 = 0.2;
S1 = 10; S2 = 0; S3 = 0
```

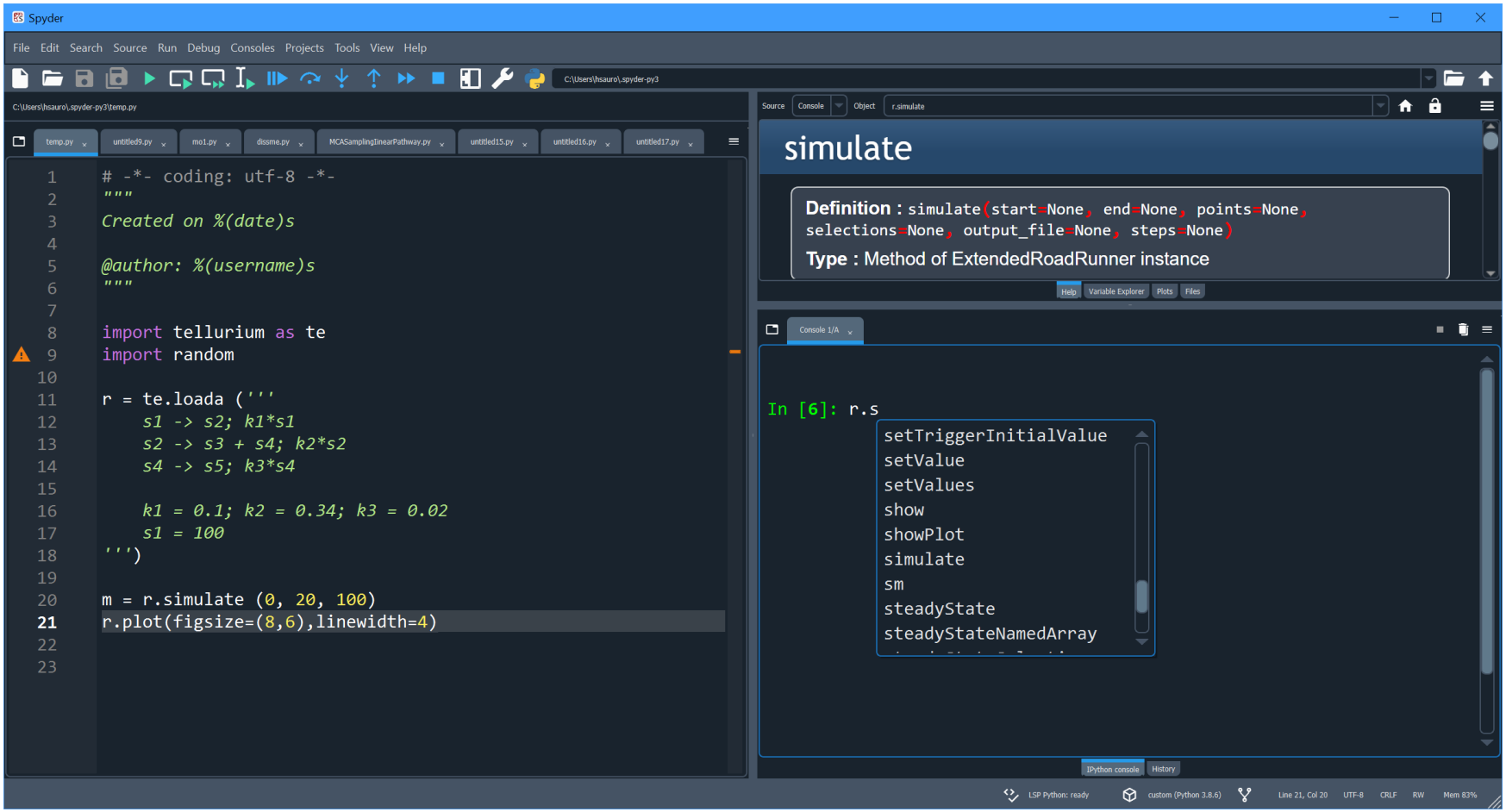
Antimony

Example (Named Reactions)

```
# Name reactions are useful for getting the reaction rates
J1: $S1 -> S2; k1*S1;
J2: S2 -> $S3; k2*S2;

k1 = 0.1; k2 = 0.2;
S1 = 10; S2 = 0; S3 = 0
```

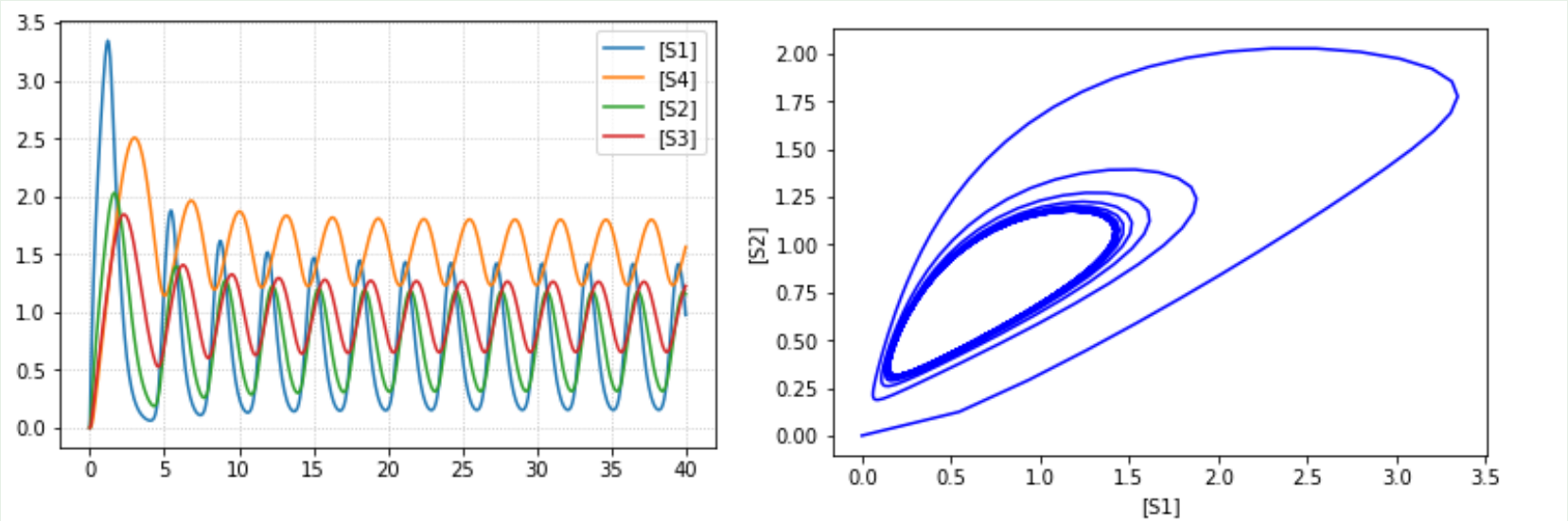
TAB key to bring up options

When typing a command, use the tab key to bring up the possible options:



Try it by typing `r.` at the console then hit TAB.

Example (Plotting Results)



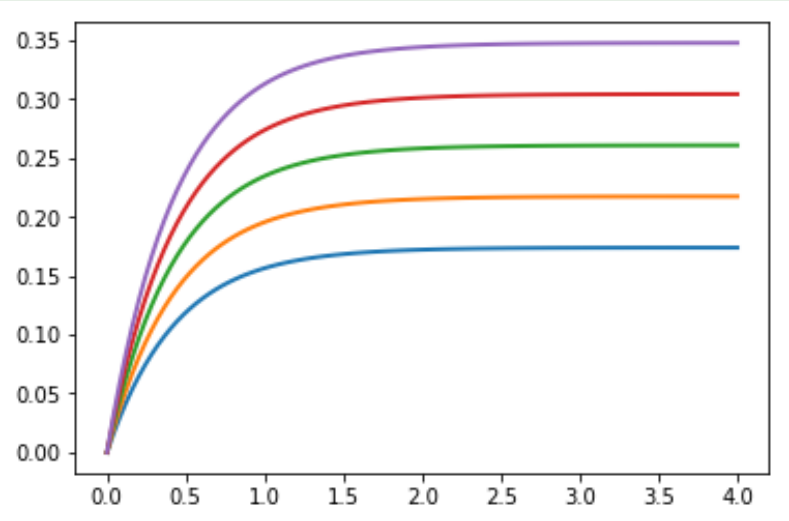
Example (Resetting the Model)

```
r = te.loada ('''
  J1: $S1 -> S2; k1*S1;
  J2: S2 -> $S3; k2*S2;

  k1 = 0.1; k2 = 0.2;
  S1 = 10; S2 = 0; S3 = 0
''')

result = r.simulate (0, 10, 100)

r.reset()  # Reset to species initial conditions
r.resetAll() # Reset initial conditions and parameter values
r.resetToOrigin() # Reset back to when the model was loaded
```

Example (Parameter Scan)

```
import tellurium as te
import numpy as np

r = te.loada ('''
    J1: $X0 -> S1; k1*X0;
    J2: S1 -> $X1; k2*S1;

    X0 = 1.0; S1 = 0.0; X1 = 0.0;
    k1 = 0.4; k2 = 2.3;
''')

m = r.simulate (0, 4, 100, ['Time', 'S1'])
for i in range (0,4):
    r.k1 = r.k1 + 0.1
    r.reset()
    m = np.hstack([m, r.simulate(0, 4, 100, ['S1'])])

# use plotArray to plot merged data
te.plotArray(m)
```

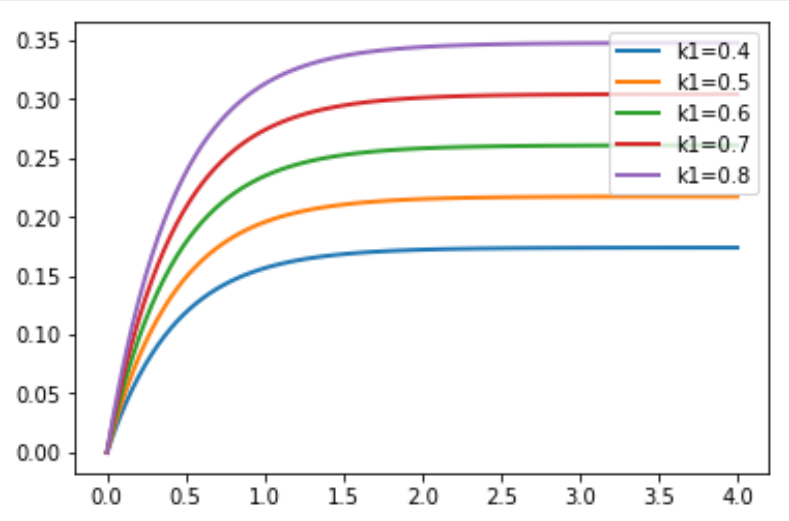
Example (Plotting Results)



Example (Parameter Scan with Legend)

```
import tellurium as te
import numpy as np

r = te.loada ('''
    J1: $X0 -> S1; k1*X0;
    J2: S1 -> $X1; k2*S1;

    X0 = 1.0; S1 = 0.0; X1 = 0.0;
    k1 = 0.4; k2 = 2.3;
''')

label = ['k1='+str(r.k1)]
m = r.simulate (0, 4, 100, ['Time', 'S1'])
for i in range (0,4):
    r.k1 = r.k1 + 0.1
    label.append ('k1='+str(r.k1))
    r.reset()
    m = np.hstack([m, r.simulate(0, 4, 100, ['S1'])])

# use plotArray to plot merged data
te.plotArray(m, labels=label)
```

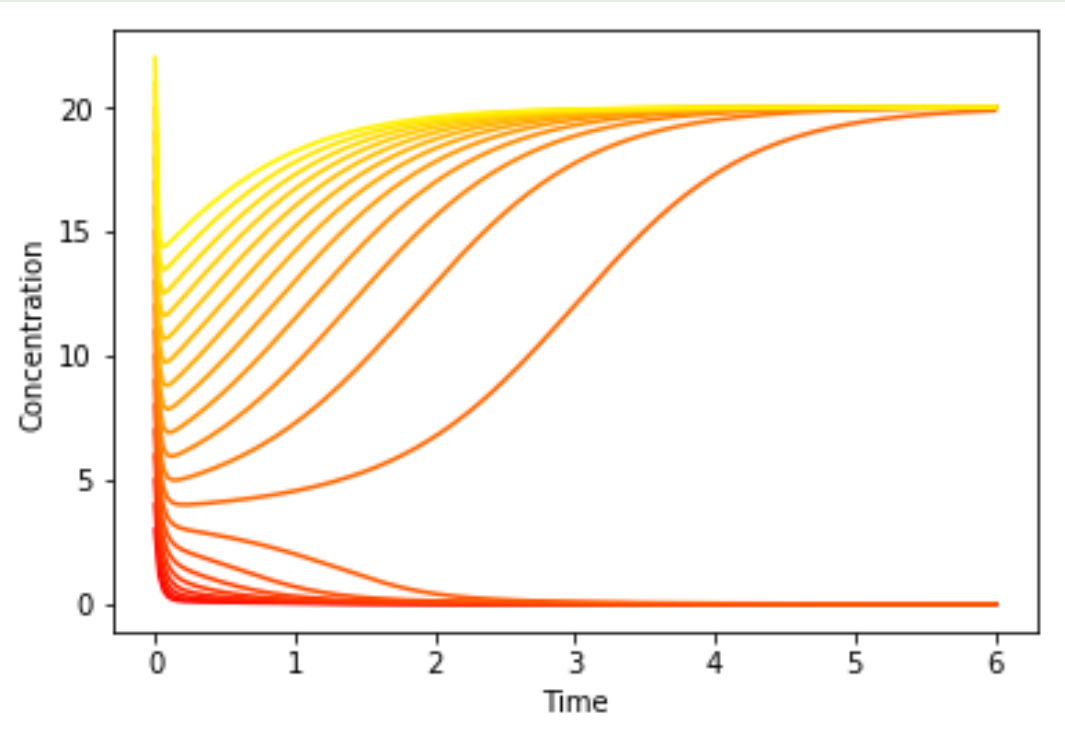
Example (Plotting Results)



Example (Parameter Scan with Legend)

```
import tellurium as te, roadrunner, numpy as np, matplotlib as mpl, pylab

# Bistable switch
r = te.loada('''
   J1: $A + X -> 2 X; J1_k1*A*X-J1_k2*X*X;
   J2: X + Y -> Z; J2_k1*X*Y-J2_k2*Z;
   J3: Z -> $B + Y; J3_k1*Z-J3_k2*B*Y;

   A = 10.47; B = 0; X = 0.874; Y = 8.302; Z = 0;
   J1_k1 = 0.23; J1_k2 = 0.1;
   J2_k1 = 4.88; J2_k2 = 1;
   J3_k1 = 1; J3_k2 = 1;
''')

def scan (r, parameter, lowRange, stepSize, numberOfScans):
    r[parameter] = lowRange
    result = r.simulate(0, 6, 201, ['Time', 'X'])
    for i in range (numberOfScans):
        r.reset()
        r[parameter] = r[parameter] + stepSize
        m = r.simulate(0, 6, 201, ['X'])
        result = np.hstack([result, m])
    return result

result = scan (r, 'init([X])', 3, 1, 20)
pylab.xlabel ('Time'); pylab.ylabel ('Concentration')
mpl_color = pylab.figure(); cmap = mpl.cm.autumn
for i in range (20):
    pylab.plot(result[:,0], result[:,i+1], color=cmap(i / float(20)))
```

Example (Plotting Results)



Documentation

Go to:

tellurium.analogmachine.org

for the complete package or

libroadrunner.org

for just libRoadRunner.

Exercise

Build a model that describes two consecutive reactions, each reaction governed by the simple Michaelis-Menten rate law

$$v = V_m \frac{S}{K_m + S}$$

$$\$S_1 \rightarrow S_2 \rightarrow \$S_3$$

Note S_1 and S_3 are FIXED. Set the parameters and species to:

```
Km1 = 0.5; Km2 = 0.5;
S1 = 10; S2 = 0; S3 = 0;
Vm1 = 30; Vm2 = 20;
```

Load the model into simulator and run a simulation from time zero to time 10 time units. Plot the results. Explain what you observe. Set `Vm1 = 18` and rerun the simulation, explain the results.

Use ODEs directly in a model

Example (ODEs)

How to specify differential equations in Tellurium (Lorenz system):

```
import tellurium as te
r = te.loada('''
    R1: -> x; sigma*(y - x);
    R2: -> y; x*(rho - z) - y;
    R3: -> z; x*y - beta*z;

    x = 0.96259; y = 2.07272; z = 18.65888;

    sigma = 10; rho = 28; beta = 2.67;
''')

m = r.simulate (0, 60, 200)
r.plot()
```

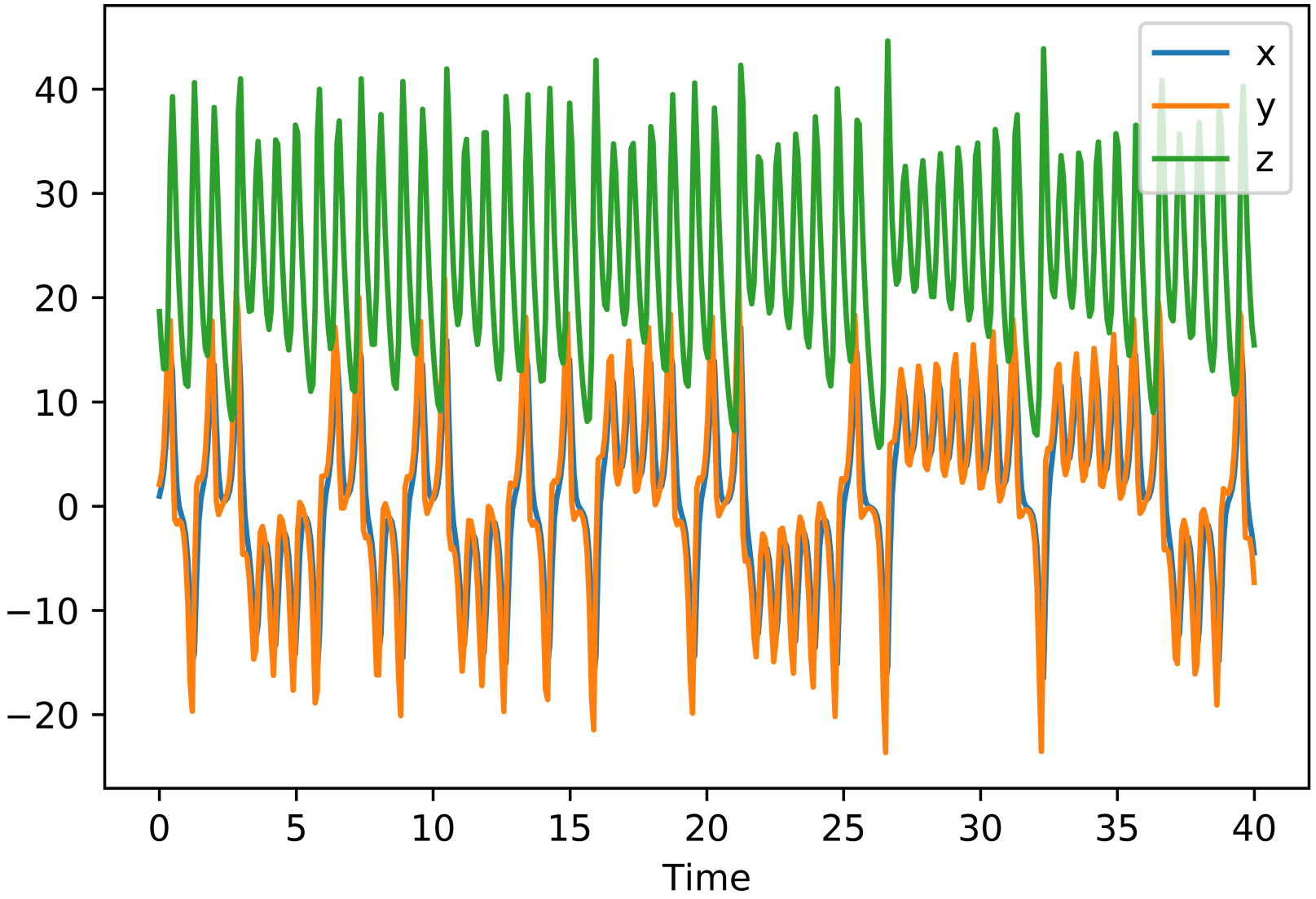
Use ODEs directly in a model

Example (ODEs)

Alternative: How to specify differential equations in Tellurium (Lorenz system):

```
import tellurium as te
r = te.loada('''
    x' = sigma*(y - x);
    y' = x*(rho - z) - y;
    z' = x*y - beta*z;

    x = 0.96259; y = 2.07272; z = 18.65888;

    sigma = 10; rho = 28; beta = 2.67;
''')

m = r.simulate (0, 60, 200)
r.plot()
```

Use ODEs directly in a model



SBML

The Systems Biology Markup Language (SBML) is a representation format, based on XML, for communicating and storing computational models of biological processes. It is a free and open standard with widespread software support. SBML can represent many different classes of biological phenomena, including metabolic networks, cell signaling pathways, regulatory networks, infectious diseases, and many others. As an XML format, SBML is not meant to be read or written by Humans.

Importing and Exporting SBML

To import SBML into a model use:

```
r = roadrunner.RoadRunner('mymodel.xml')
```

or (perhaps easier to remember)

```
import tellurium as te
r = te.loadSBMLModel('mymodel.xml')
```

To export SBML from a model use:

```
sbmlStr = r.getSBML()
```

To save the SBML to a file you can use the tellurium helper method:

```
import tellurium as te
te.saveToFile ('mymodel.xml', r.getSBML())
```

Where are files saved to?

When you typed this:

```
te.saveToFile ('mymodel.xml', r.getSBML())
```

where did the file go?

It went to the current working directory:

```
import os
print os.getcwd()
```

To get the list of files:

```
import glob
print glob.glob('*.xml')
```

Exercise: Export SBML

Enter the following model in to Tellurium and save it as SBML

Example (SBML Exercise)

```
A + B -> C; k1*A*B
C -> D; k2*C
k1= 0.2; k2 = 0.6;
A = 10; B = 12;
C = 0; D = 0
```

Exercise: Export SBML

Example (Export SBML)

```
import tellurium as te
r = te.loada('''
  A + B -> C; k1*A*B;
  C -> D; k2*C

  k1= 0.2; k2 = 0.6;
  A = 10; B = 12; C = 0; D = 0
''')

te.saveToFile ('mymodel.xml', r.getSBML())
```

Exercise: Import SBML

Example (Import SBML)

Import the SBML model you saved in the last exercise and run a simulation.

Exercise: Import SBML

Example (Import SBML)

```
import tellurium as te
r = te.loadSBMLModel ('mymodel.xml')

m = r.simulate (0, 10, 100)
r.plot()
```

Answer to exercise on slide: 32

Example (ODEs)

```
import tellurium as te
r = te.loada('''
        $S1 -> S2; Vm1*S1/(Km1 + S1)
        S2 -> $S3; Vm2*S2/(Km2 + S2)

        Km1 = 0.5; Km2 = 0.5;
        S1 = 10; S2 = 0; S3 = 0;
        Vm1 = 30; Vm2 = 20;
''')

m = r.simulate (0, 10, 200)
r.plot()
```