ICSB Tutorial 2008
Systems Biology Workbench

Frank T. Bergmann (fbergman@u.washington.edu) and
Kyung Hyuk Kim (kkim@u.washington.edu)

Tutorial website: http://www.sys-bio.org/sbwWiki/tutorials/icsb2008

Hands on Exercises – Part I

This set of exercises will demonstrate the use of Jarnac to create hypothetical biochemical models and how to analyze them with the tools included with the SBW.

Homeostasis

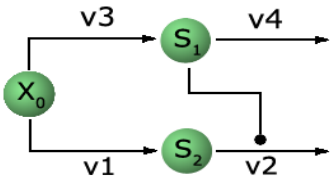
A homeostatic system is one that resists internal change when external parameters are perturbed. We will illustrate two such networks: one that shows perfect adaptation and another that near adaptation.

Perfect adaptation describes a system that recovers from a perturbation without any error (thus perfectly). There are a number of approaches to achieving perfect adaptation, one is via integral control and another, simpler approach, is via coordinate stimulation. In this tutorial we will illustrate perfect adaptation using coordinate stimulation.

A simpler and perhaps more common method for achieving homeostasis is to use negative feedback to resist external perturbations. Unlike systems which show perfect adaptation, systems which employ negative feedback cannot completely restore a disturbance.

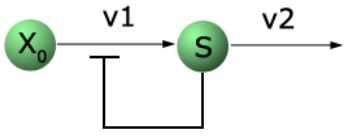
Reaction Diagrams

S_2 remains homeostatic



$$v1 = k_1 X_0 \qquad v2 = k_2 S_1 S_2$$
$$v3 = k_3 X_0 \qquad v4 = k_4 S_1$$

S remains homeostatic



$$v1 = \frac{X_0}{K_m + S^4} \qquad v2 = k_1 S$$

$$\mathbf{v1 = \frac{x_0}{Km + s^4}}, \qquad \mathbf{v2 = ks}$$

Perfect Adaptation **Negative Feedback**

Exercise - Perfect adaptation

In the following example we will load a model corresponding to the Perfect Adaptation Reaction Scheme above into Jarnac / JarnacLite, and then perturb the model and see whether / how the model recovers from those perturbations.

1. Enter the following Jarnac code into Jarnac / JarnacLite. (Or download PerfectAdaptation.jan from the tutorial website):

   Jarnac code:

```
p = defn PerfectAdaptation
      $Xo  -> S2;  k1*Xo;
       S2  -> $w;  k2*S1*S2;
       $Xo -> S1;  k3*Xo;
       S1  -> $w;  k4*S1;

end;

// initialize
p.k1 = 1;
p.k2 = 1;
p.k3 = 1;
p.k4 = 1;
p.Xo = 1.0;
```

2. From the SBW menu in Jarnac, select “edu.kgi.roadRunner Simulation Service”.
3. Here select “Pulse / Scan”
   a. Set Time End to 60.0
   b. Pulse Start Time to 30.0
   c. Pulse Duration to 20
   d. Pulse Value to 3

   After pressing “Pulse” the simulation will be carried out and displayed.

4. A similar effect can be studied from the “Timecourse (continuous)” tab:
   a. Select the “Time course (continuous)” tab. Click Start.
   b. From the Options menu, select “Sliders”. Under the boundary variables, you will find X_0
   c. Change X_0 and observe the manner in which the two concentrations are disturbed, and then observe how S_2 recovers to the original state.
5. Of course Jarnac itself could have been easily used to perform the simulation, if we just add the following code lines:

```
m1 = p.sim.eval(0, 10,100, [<p.time>, <p.Xo>, <p.S2>]); graph(m1);
setghold(true)

p.Xo = 5;

m2 = p.sim.eval(10, 20,100, [<p.time>, <p.Xo>, <p.S2>]); graph(m2);
setghold(false)
```

Exercise - Negative feedback system

1. Enter the following Jarnac code into Jarnac / JarnacLite. (Or download NegativeFeedback.jan from the tutorial website):

   Jarnac code:

```
p = defn NegativeFeedback
      $Xo   -> S;   Xo/(km + S^h);
       S    -> $w;  k1*S;

end;


// initialize
p.h   = 4;   // Hill coefficient
p.k1  = 1;
p.km  = 1;
p.S   = 1.5;
p.Xo  = 5;
```

2. From the SBW menu in Jarnac, select "edu.kgi.roadRunner Simulation Service".

1. In order to see why a negative feedback produces homeostatic behavior, one needs to see how the flow into S_2 and the flow out of S_2 are connected to the concentration of S_2. Open the Simulation Service (SBW menu -> edu.kgi.roadRunner Simulation Service).
2. Select the "Rate vs. Species" tab. Select S for the "Species", and plot. The two curves indicate the flow in and flow out of S.
3. Use the sliders (Options -> Sliders) to see how the curves are affected by the parameters.

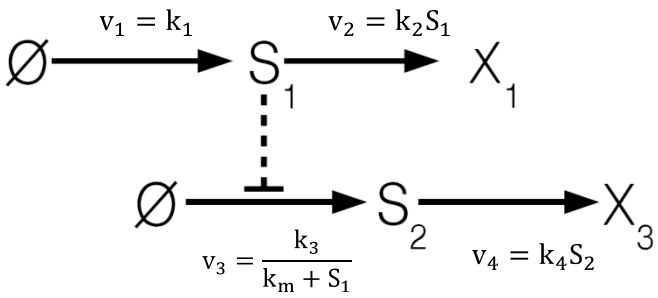
Hands on Exercises – Part II

Stochastic Focusing

Stochastic focusing describes a system that shows increased sensitivities due to the stochastic fluctuations. When a chemical reaction system is perturbed by changing its parameters such as rate constants, the chemical concentrations and fluxes can be changed. Sensitivities quantify such changes in concentrations or fluxes due to the changes of the parameters.

Stochastic focusing can be achieved in a reaction system where noise generated from upstream of the network inhibits the downstream reactions.

Reaction Diagram



Exercise – Stochastic Simulation of Stochastic Focusing

[Jarnac file names: stochastic_focusing.jan, stochastic_focusing_perturbation.jan]

The following Jarnac script describes a model that shows stochastic focusing:

Jarnac code:

```
p = defn stochastic_focusing
     v1: $Xo -> S1;  k1*Xo;
     v2:  S1 -> $w;  k2*S1;
     v3: $Xo -> S2;  k3*Xo/(km +S1);
     v4:  S2 -> $w;  k4*S1*S2;
end;

p.k1 = 20;  p.k2 = 10;   p.k3 =1;
p.k4 = 1;   p.km =0.1;   p.S1 =1;
p.S2 =10;   p.Xo=1;
```

0. Perform a Gillespie stochastic simulation. First load the model by running

```
gillLoadSBML(p.xml)
```

Second, store all the time series data of the numbers of each species of molecules in a matrix m1 by executing

```
m1 = gillSimulate(0,100,1)
```

The start time is set to be 0 and the end time to be 100. Every period of sample is stored.

`For example gillSimulate(0,100,2)` will store every other points in the time series data.

1. Plot the data in time by running

```
graph(m1)
```

2. Roughly estimate the time t1 when the numbers of molecules fluctuate with respect to constant levels (called in the stationary state).

t1=_____

3. For the rigorous estimation of t1, first, comment out "`m1=gillSimulate(…)`". In its place type

```
m1 = gillsimulateMeanAndSDOnGrid(0, your t1 value here, 0.1, 10)
```

The argument 0.1 is the grid size, and the argument 10 means take 10 independent runs of simulations and average them. Increase the run sample size from 10 to 1000. How does the graph of S1 vs. time change?

Next comment out `gillsimulateMean….` and uncomment "`m1=gillsimulate(…)`"

4. Perform a deterministic simulation and store all time series data to a matrix m2 and plot them:

```
m2 = p.sim.eval (0,100, 100, [<p.Time>,  <p.S1>, <p.S2>])
graph(m2)
```

5. Compare qualitatively S2 from the two simulations. The mean level of S2 in the stochastic result will be larger than that from the deterministic simulation. You can use a command `setghold` to overlap two graphs corresponding to m1 and m2.

6. Perturb a parameter k1 by setting it to 40:

```
gillSetParameter ("k1", 40);
```

Compare again how S2 changes relatively to the original unperturbed case (k1=20) qualitatively. The change of S2 from the stochastic simulation will be larger than that from the deterministic simulation. Such increase in the change is closely related to the stochastic focusing.

Now, we move on to a quantitative analysis for the stochastic focusing.

7. Set k1 to be 20 again:

```
gillSetParameter ("k1", 20);
```

8. Click the upper console panel in Jarnac.

9. Slice out the stationary state portion of the matrix m1 and store it m1stt by using

```
m1stt = timeslice(m1, t1, t2)
```

   where t2 is the end time that the matrix m1 is sliced out.

10. Measure the mean value of S1 and S2 in the stationary state by using commands `mean` and `getColumn`. Store the mean values of S1 and S2 in stoS1 and stoS2.

    "?" gives help a help message, e.g.,

    `->?mean`

    `mean (x)` **:** Returns the mean of elements in vector or matrix.

    Hint: `mean (getcolumn(m1stt,2))` gives the mean value of s1 in the stationary state.

11. Set k1 to be 40.

12. Store the mean values of S1 and S2 in variables stoS1a and stoS2a after slicing the stationary state portion.

13. Compute the percentage change of S1 and S2: e.g., (stoS1a-stoS1)/stoS1.

14. Compute the sensitivity of S2 due to the change of S1, which is equal to the percentage change of S2 over that of S1.

15. Repeat the same procedure as described above for the deterministic simulations. Compute the sensitivity.

16. Compare the sensitivity in the stochastic and deterministic cases.

17. (Challenge) Set the values of k1 and k2 to 2 and 1, respectively. Run the stochastic and deterministic simulations. Any difference? Does the mean of S2 change? Why not? Does the fluctuation of S2 change? Why?

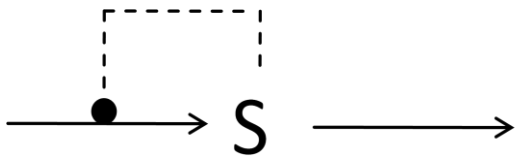
Stochastic_focusing.jan

```
//Stochastic Focusing
//model
p = defn stochastic_focusing
     v1: $Xo -> S1;     k1*Xo;
     v2: S1  -> $Null;  k2*S1;
     v3: $X1 -> S2;     k3*X1/(km +S1);
     v4: S2  -> $w;     k4*S2;

end;

//initialize
p.k1 = 20;
p.k2 = 10;
p.k3 = 1;
p.k4 = 1;
p.km =0.1;
p.S1 =1;
p.S2 =10;
p.Xo=1;
p.X1=1;


//Perform 2 different stochastic simulations for 2 different values of
k1
gillLoadSBML(p.xml);
m11 = gillSimulateOnGrid (0, 500, 1);
setColumnName(m1, 2, "Gillespie: S1");
setColumnName(m1, 3, "Gillespie: S2");

gillSetParameter ("k1", 40);
m12 = gillSimulateOnGrid(500,1000,1);

//Merge data of m11 and m12 into a matrix m1.
m1 = augr(m11,m12)


//Deterministic processes
m21 = p.sim.eval (0,500, 50, [<p.Time>,  <p.S1>, <p.S2>]);
p.k1 = 40;
m22 = p.sim.eval (500,1000,50,[<p.Time>, <p.S1>, <p.S2>]);
m2 = augr(m21,m22);
setColumnName(m2, 2, "Deterministic: S1");
setColumnName(m2, 3, "Deterministic: S2");


//plot all the results together
graph(m1);setghold(true);
graph(m2);setghold(false);
```

Stochastic Switching

Positive feedback is a common approach to generate a system with two stable states, often called bistable systems. In the stochastic framework, such bistable system can show spontaneous switching from one stable stationary state to the other. This is caused by stochastic fluctuations. We examine this stochastic switching phenomenon.

Reaction Diagram



$$v_1 = \frac{100(1+S^2)}{100+S^2}, \qquad v_2 = 3.8\,S$$

Exercise – Stochastic Simulation of Bistability

[Jarnac file name: bistability.jan, bistability-stoch.jan]

Jarnac code:

```
//Bistability
//model
p = defn bistability
     v1: $Xo -> S1; Xo*(100+100*S1*S1)/(100+S1*S1);
     v2: S1 -> $w;  k2*S1;
end;
//initialize
p.k2 = 3.8; p.Xo=1;
```

1. Let us first run model deterministically. Set the value of S1 equal to 1.0 and simulate the model by executing

```
m1 = p.sim.eval(0,10,100,[<p.time>, <p.S1>])
```

2. Generate its graph by using the command `graph(m)`

3. Choose a different value of S1 and generate a graph of S1 vs. time. Observe that the system can reach two different stationary states depending on the initial value of S1.

4. Next we run the model stochastically by using the gillSimulateOnGrid command and plot the probability density function of S1. [Refer to commands: pdf, pdfPlot, timeslice, getColumn, getColumns]. A summary of these experiments is given in the scripts below.

bistability.jan

```
//Bistability

//model
p = defn bistability
     v1: $Xo -> S1;  Xo*(100+100*S1*S1)/(100+S1*S1);
     v2: S1 -> $w;   k2*S1;
     end;

//initialize
p.k2 = 3.8;
p.Xo = 1;

p.S1 = 1;
m1 = p.sim.eval(0,10,100,[<p.time>, <p.S1>])
graph(m1)
setghold(true);

p.S1 = 10;
m2 = p.sim.eval(0,10,100,[<p.time>, <p.S1>])
graph(m2)
setghold(false);
```

bistability-stoch.jan

```
//Bistability

//model
p = defn bistability
     v1: $Xo -> S1;  Xo*(100+100*S1*S1)/(100+S1*S1);
     v2: S1 -> $w;   k2*S1;
     end;

//initialize
p.k2 = 3.8;
p.S1 =1;
p.Xo=1;

gillLoadSBML(p.xml);
m1 = gillSimulate (0, 500, 1);
graph(m1)
```