**Dual feedback oscillator (Hasty et al. 2002.)**

**Model adapted from:**

Phys Rev Lett. 2002 Apr 8;88(14):148101. Epub 2002 Mar 22.

*Synthetic gene network for entraining and amplifying cellular oscillations.*

Hasty J, Dolnik M, Rottschäfer V, Collins JJ.

Plasmids with genes for activator protein CI and repressor protein LacI are inserted into a bacterial cell.

Both CI and LacI plasmids have promoter sites that can be activated by CI dimers (two binding sites), and repressed by LacI tetramers (one binding site).

**Model equations**

For analysis of the system Hasty et. al. developed an ODE model and rescaled the variables/parameters into the following form for ease of analysis.

The oscillatory behaviour of this circuit can be “tuned” because the degradation rates and can be manipulated by choosing particular experimental conditions.

**Variables:**

conc. activator protein (CI) (arbitrary units)

conc. repressor protein (LacI) (arbitrary units)

t time (1 unit in this model = 1/20 minutes)

**Parameters**

relates to activator degradation rate

relates to repressor degradation rate

relates to the ratio of production rates (repressor vs activator)

relates to the maximum activator production rate

relates to saturation level of activator-promoter binding

**Tasks**

1) Code and run the model using the values below. You should find the system exhibits oscillations.

**Initial conditions:**

=0.13

=2.8

**Parameters values:**

0

2) Demonstrate that the oscillation period is tunable by running the model with different values for ,

e.g. try

3) Plot a bifurcation diagram showing how the amplitude of [A] oscillations changes as is adjusted from 0.01 to 0.08.

To do this we can plot a graph of the maximum and minimum values of [A] once it has had time to settle into a limit cycle (or a steady state).

Template code to create this plot can be found in file:

hasty\_bifurcation1D.py

As before you need to complete the rate equations, parameter values and initial conditions (you can copy these from your work in the previous task).

You then need to write code to find the maximum, minimum and range of [A] in the recorded observations. When complete the code should produce output like this:

Input: k\_dA=0.15, k\_dR=0.010. Output [A]: max= 0.17, min= 0.17, range=0.00000

Input: k\_dA=0.15, k\_dR=0.020. Output [A]: max= 0.30, min= 0.30, range=0.00000

Input: k\_dA=0.15, k\_dR=0.030. Output [A]: max= 1.23, min= 0.17, range=1.05498

...

Looking at the lines above you can see that:

at k\_dR=0.010 and 0.020 [A] does not oscillate (range is zero)

at k\_dR=0.030 [A] oscillates (ranges between 0.17 and 1.23)

When you have this working, the next stage is to create a bifurcation plot.

To do this add the following code into the loop:

ax.plot(k\_dR, max\_A, 'b.')

ax.plot(k\_dR, min\_A, 'b.')

This draws the minimum/maximum level of [A] onto a plot of [A] vs

Once you have the code working correctly, you can increase the number of points plotted i.e. adjust k\_dR\_list to contain more points while keeping the same range.

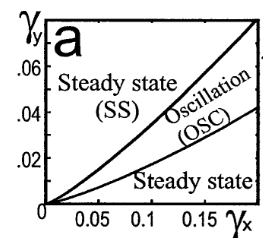
You should see that for:

[A] reaches steady state

[A] oscillates

[A] reaches steady state

4) (Optional) The bifurcation diagram

We can use a similar technique to reproduce the 2D bifurcation plot presented by Hasty et. al.

*Figure reproduced from original paper Hasty et al. 2002.*

*corresponds to*

*corresponds to .*

The plot created in the previous task corresponds to the scan across at in which oscillations are found between 0.025 < < 0.050.

We can create a similar plot using our code, by testing out parameter combinations of and and looking at the result to identify oscillating behaviour.

To do this we need a nested loop (one inside the other) that will loop over a set of k\_dA and k\_dR values and simulate the system using each combination.

With each simulation output we examine the behaviour of [A] and add the result onto a plot of vs. :

* if oscillations are detected we add a red ‘x’
* if no oscillations are detected we add a blue ‘x’

The following code is provided to help:

1. k\_dA\_list=np.linspace(0.01,0.2,20)
2. k\_dR\_list=np.linspace(0.005,0.08,20)
4. **for** k\_dR **in** k\_dR\_list:
5. **for** k\_dA **in** k\_dA\_list:
6. params=(v,K,Z,k\_dA,k\_dR)
8. #...
10. **if** range\_A>0.4:
11. ax.plot(k\_dA, k\_dR,'rx')
12. **else**:
13. ax.plot(k\_dA, k\_dR,'bx')

The resulting plot should look similar to the one produced in the paper, with red crosses in the oscillatory region, and blue crosses in the steady state region.

Note:

Once you have the code is working, you can improve your plot by increasing the number of points in the k\_dR\_list and k\_dR\_list arrays.

Do not go over ~100 values in each list unless you have a very fast computer as it may take several minutes (or longer) to run!