THE UNIVERSITY *of* EDINBURGH
School of Biological Sciences

# Assignment 2
# Practical Systems Biology

**Student Exam Number: B132062**

**Sharing code with B139748**

# A genetic toggle switch

**Question 1.1:**

Create a function that returns $du/dt$ and $dv/dt$ given $u$ and $v$ and $t$. As we will vary the concentration of inducers, the function should also take the concentrations of the inducers as inputs. Let $a_u$=10, $a_v$=9, $K$=3, and both Hill numbers equal 2. **[2]**

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

#creates function
def toggleswitch(y, t, Iu, Iv):
    au=10 ; av=9 ; k=3 ; b=2 ; n=2
    (u,v)=y
    dudt= -u+(au/(1+(v/(1+Iv/k)**n)**b))
    dvdt= -v+(av/(1+(u/(1+Iu/k)**n)**b))
    return np.array([dudt,dvdt])
```

**Question 1.2:**

Starting with $u$=$v$=0, simulate the application of $I_u$ for 50 time units (set $I_u$=100 and $I_v$=0), then remove $I_u$for another 50 time units. Following this removal, apply $I_v$ for 50 time units (set $I_u$=0 and $I_v$=100) and then remove $I_v$ for another 50 time units. [4] Do you see hysteresis (history-dependent behaviour)? Why?

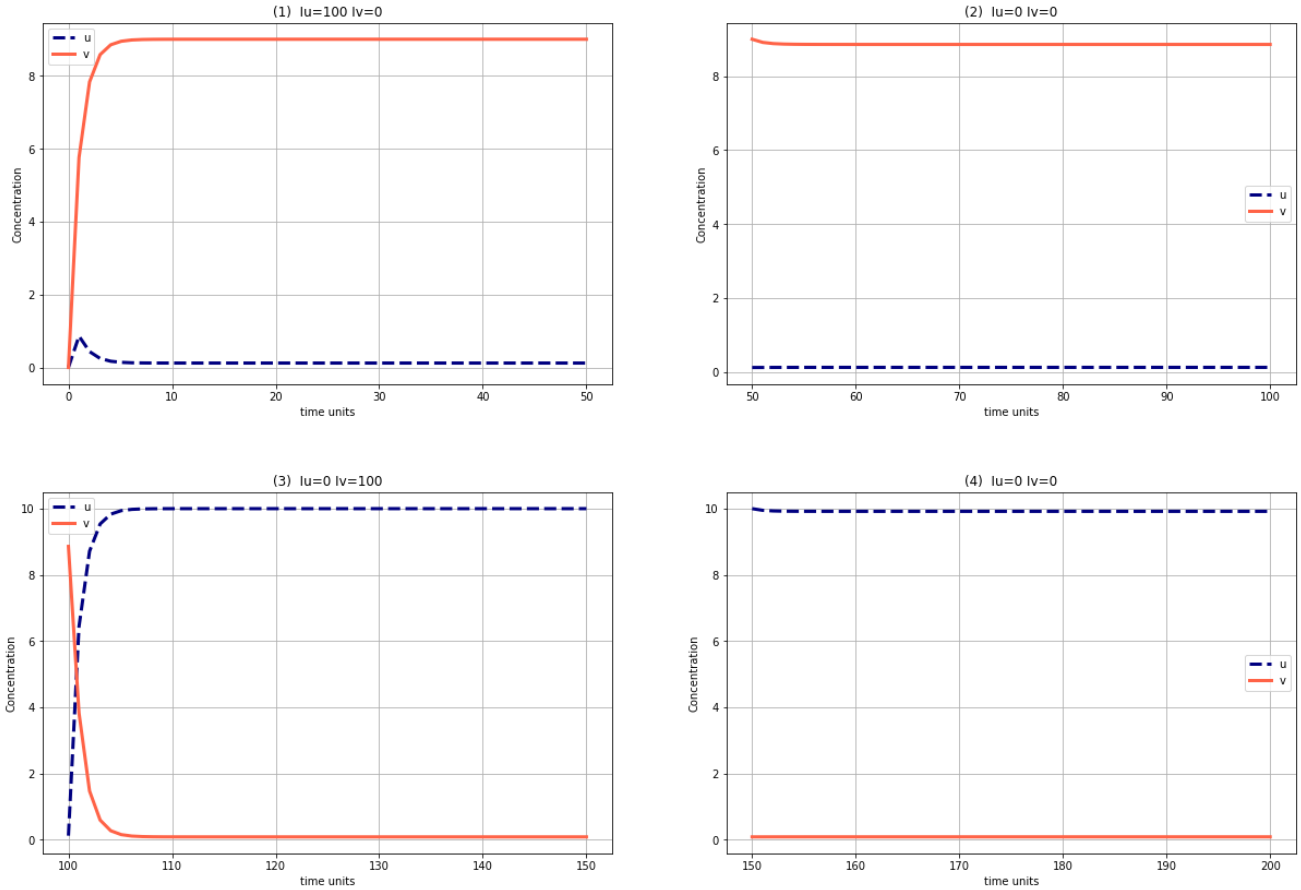


Figure 1.

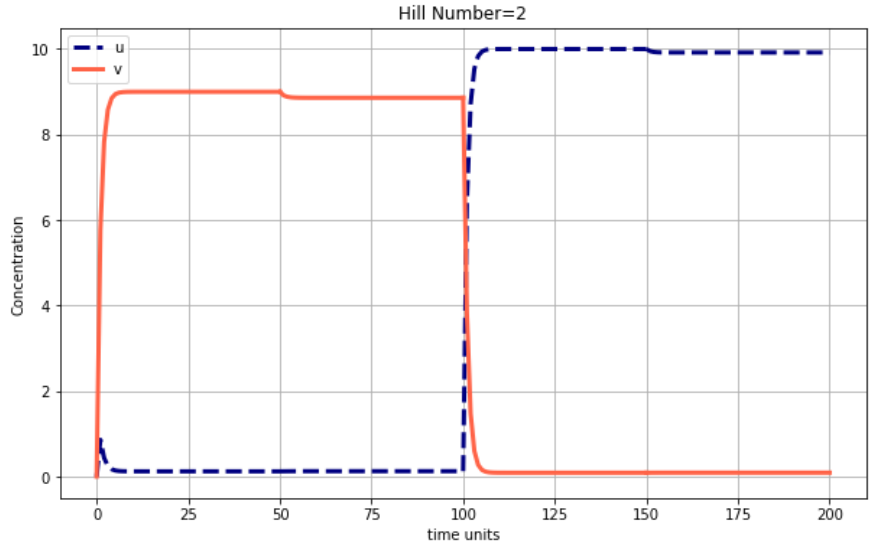


**Figure2. Toggle Switch Behavior with a hill number of 1.**

In this graphs we can see that the system behaves with a hysteresis dependent behavior.
When the system gets an input of *Iu* but no *Iv* the concentration of the repressor V increases reaching a value of ~9 ( Panel A) ; however we can see in Panel B that when *Iu* is removed the value of V is maintained even when the input signals have been removed completely, thus meaning that the switch it has a sort of memory.
We can see this better in figure 2, although it is interesting to see that there is a small decrease each time the inducers get removed, meaning that the hysteresis is not maintained 100% and could be do to protein degradation. A similar behavior can be see when we start with the other inducer *Iv* , but this time the protein U is the abundant one. This clearly shows that the system has 2 stable states and that it has strong enough positive feedback to produce bistability.
Bistability can also explain why the system has memory.

### Code Figure 1:

```
y0=[0,0]
t1=np.linspace(0,50)
t2=np.linspace(50,100)
t3=np.linspace(100,150)
t4=np.linspace(150,200)
plt.figure(figsize= (14, 14))
plt.subplots_adjust(left= 0.06, right=1.2, wspace=0.2, hspace=0.3)
Iu=100
Iv=0
y1=odeint(toggleswitch,y0,t1,(Iu,Iv))
plt.subplot(2,2,1)
plt.plot(t1,y1[:,0],color='navy',label='u',linewidth=3,linestyle='--')
plt.plot(t1,y1[:,1],color='tomato',label='v',linewidth=3)
plt.legend()
plt.xlabel('time units')
plt.ylabel('Concentration')
plt.title('(1)  Iu=100 Iv=0')
plt.grid()
Iu=0
Iv=0
```

```
y2=odeint(toggleswitch,y1[-1],t2,(Iu,Iv))
plt.subplot(2,2,2)
plt.plot(t2,y2[:,0],color='navy',label='u',linewidth=3,linestyle='--')
plt.plot(t2,y2[:,1],color='tomato',label='v',linewidth=3)
plt.legend()
plt.xlabel('time units')
plt.ylabel('Concentration')
plt.title('(2)  Iu=0 Iv=0')
plt.grid()
Iu=0
Iv=100
y3=odeint(toggleswitch,y2[-1],t3,(Iu,Iv))
plt.subplot(2,2,3)
plt.plot(t3,y3[:,0],color='navy',label='u',linewidth=3,linestyle='--')
plt.plot(t3,y3[:,1],color='tomato',label='v',linewidth=3)
plt.legend()
plt.xlabel('time units')
plt.ylabel('Concentration')
plt.title('(3)  Iu=0 Iv=100')
plt.grid()
```

```
Iu=0
Iv=0
y4=odeint(toggleswitch,y3[-1],t4,(Iu,Iv))
plt.subplot(2,2,4)
plt.plot(t4,y4[:,0],color='navy',label='u',linewidth=3,linestyle='--')
plt.plot(t4,y4[:,1],color='tomato',label='v',linewidth=3)
plt.legend()

plt.xlabel('time units')
plt.ylabel('Concentration')
plt.title('(4)  Iu=0 Iv=0')
plt.grid()
plt.show()
```

## Code for Figure 2:

```
plt.figure(figsize= (10, 6))
plt.plot(t1,y1[:,0],color='navy',label='u',linestyle='--',linewidth=3)
plt.plot(t1,y1[:,1],color='tomato',label='v',linewidth=3)
plt.plot(t2,y2[:,0],color='navy',linestyle='--',linewidth=3)
plt.plot(t2,y2[:,1],color='tomato',linewidth=3)
plt.plot(t3,y3[:,0],color='navy',linestyle='--',linewidth=3)
plt.plot(t3,y3[:,1],color='tomato',linewidth=3)
plt.plot(t4,y4[:,0],color='navy',linestyle='--',linewidth=3)
plt.plot(t4,y4[:,1],color='tomato',linewidth=3)
plt.xlabel('time units')
plt.ylabel('Concentration')
plt.title('Hill Number=2')
plt.legend()
plt.grid()
plt.show()
```

**Change both Hill numbers to 1. Rerun the experiment and explain what you observe. [3]**

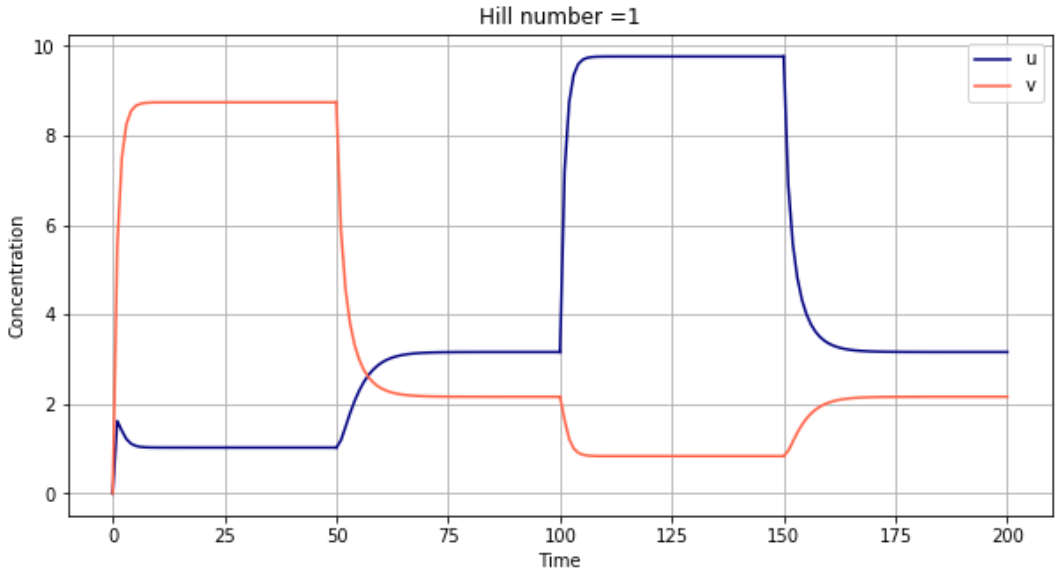


**Figure3. Toggle Switch Behavior with a hill number of 1.**

In the case where the hill number is lowered to 1 the system no longer shows positive cooperativity and it doesn't have memory anymore. We can see this in Figure 3, where the concentration of V decreased and it did not begging at the same value of a ~9 when the input of *Iu* was removed as shown on figure 2. Asides from the decrease of concentration of V we can also see there is a small increase of concentration of U even though there isn't any inducer molecules to increase its production. This could be due to the fact that the binding affinity is lower so the repression is therefore lower and leaky.

## Code Figure 3:

```
def toggleswitch2(y, t, Iu, Iv):
    au=10 ; av=9 ; k=3 ; b=1 ; n=1
    (u,v)=y
    dudt= -u+(au/(1+(v/(1+Iv/k)**n)**b))
    dvdt= -v+(av/(1+(u/(1+Iu/k)**n)**b))
    return np.array([dudt,dvdt])

plt.figure(figsize= (10, 6))
y0=[0,0]
t1=np.linspace(0,50)
t2=np.linspace(50,100)
t3=np.linspace(100,150)
t4=np.linspace(150,200)
Iu=100
Iv=0
y1=odeint(toggleswitch2,y0,t1,(Iu,Iv))
plt.plot(t1,y1[:,0],color='navy',label='u')
plt.plot(t1,y1[:,1],color='tomato',label='v')
Iu=0
Iv=0
y2=odeint(toggleswitch2,y1[-1],t2,(Iu,Iv))
plt.plot(t2,y2[:,0],color='navy')
plt.plot(t2,y2[:,1],color='tomato')
Iu=0
Iv=100
```

```
y3=odeint(toggleswitch2,y2[-1],t3,(Iu,Iv))
plt.plot(t3,y3[:,0],color='navy')
plt.plot(t3,y3[:,1],color='tomato')
Iu=0
Iv=0
y4=odeint(toggleswitch2,y3[-1],t4,(Iu,Iv))
plt.plot(t4,y4[:,0],color='navy')
plt.plot(t4,y4[:,1],color='tomato')
plt.grid()
plt.legend()
plt.ylabel('Concentration')
plt.xlabel('Time')
plt.title('Hill number =1')
plt.show()
```

**Question 1.3:**

Determine the nullclines of the model system. Explain which fixed points are stable and, from the trajectories you plotted, why. [5]

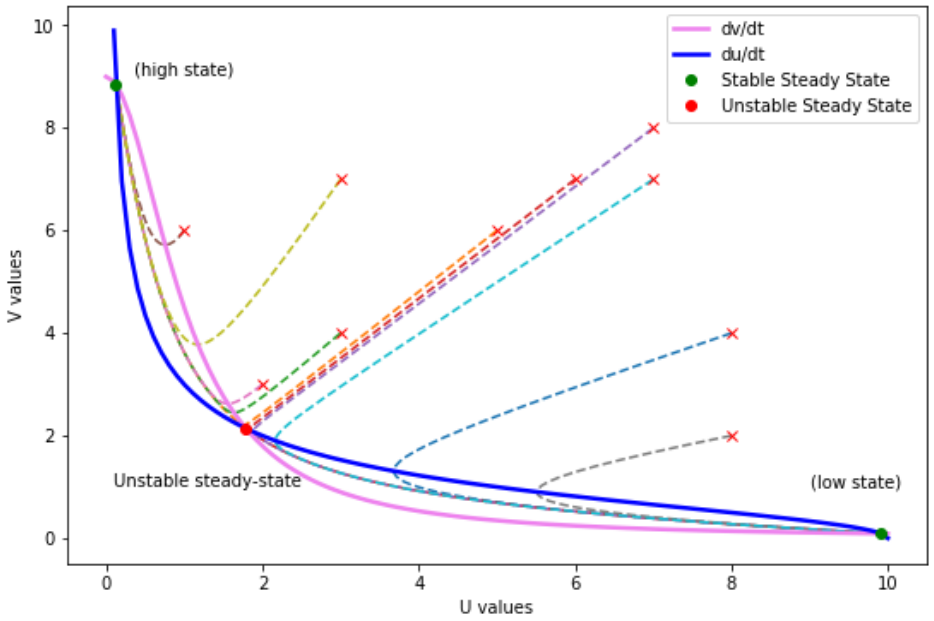


**Figure 4. Nullclines of the model**

In figure 4 we can see that system has two stable states (Green Dots) where one is a high and the other is low. These can be concluded because the trajectories of the random initial conditions used for u and v migrate always to the nearest stable point ,meaning that this system will always try to reach a stable state.

**Code Figure 4:**

```
def st_st(u):
    return -u+(au/(1+((av/(1+(u/(1+(Iu/K))**n)**b))/(1+(Iv/K))**n)**b))
```

```
random_y0=[]
for i in range(10):
    alpha=np.random.randint(1,9,2)
    random_y0.append(alpha)
au=10
av=9
K=3
b=2
n=2
Iu=0
Iv=0
npts=100
vmin,vmax=0,10
umin,umax=0,10
v=np.linspace(vmin,vmax,npts)
u=np.linspace(umin,umax,npts)
time2=np.linspace(0,100,1000)
plt.figure(figsize= (9, 6))
```

```
for i in random_y0:
    yi=odeint(toggleswitch,i,time2,(0,0))
    plt.plot(yi[:,0],yi[:,1],linestyle='--')
    plt.plot(i[0],i[1],'rx')
plt.plot(u,(av/(1+(u/(1+(Iu/K))**n)**b)),linewidth=2.5,color='violet',label='dv/dt')
plt.plot(u,((1+(Iv/K))**n)*((au/u)-1)**(1/b),linewidth=2.5,color='blue',label='du/dt')
sol1=fsolve(st_st, K/100)
sol2=fsolve(st_st, K)
sol3=fsolve(st_st, 23*K)
plt.plot(sol1 , (av/(1+(sol1/(1+Iu/K)**n)**b)), 'go',label='Stable Steady State')
plt.plot(sol2, (av/(1+(sol2/(1+Iu/K)**n)**b)), 'ro',label='Unstable Steady State')
plt.plot(sol3, (av/(1+(sol3/(1+Iu/K)**n)**b)), 'go')
plt.text(0.3,9, ' (high state)')
plt.text(9,1,'(low state)')
plt.text(0.1,1, 'Unstable steady-state')
plt.legend(loc='upper right')
plt.ylabel('V values')
plt.xlabel('U values')
plt.show()
```

**Question 1.4:Determining the bifurcation diagram for a bistable system.**
Plot on the same graph the steady-state value of $u$ against $a_u$ when $a_u$ is increased and when $a_u$ is decreased. Explain what you see. Is there a range of values for $a_u$ for which the switch once thrown on cannot be reset? [6]

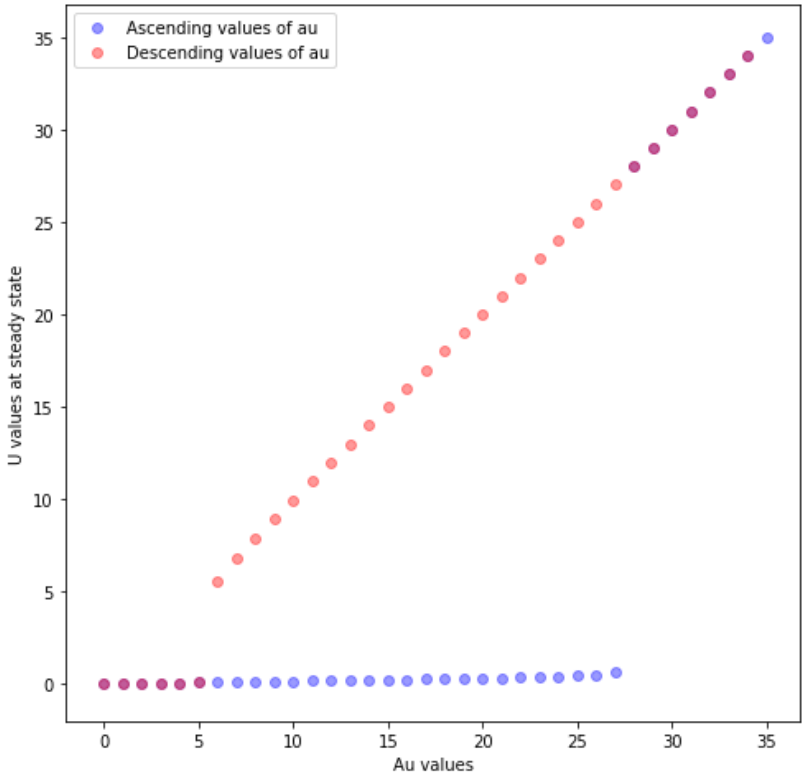


**Figure 5. Nullclines of the model**

With the diagram on figure 5 we can also see the history dependant behavior. When the au value increases to 27 the system switches to the other steady state, but when the value of au decreases below this number it remains on until it reaches 5 and then switches off again.

Code for Figure 5:

```
au_values=np.linspace(0,35,36)
meh_values=np.linspace(34,0,35)
```

```
def toggleswitch3(y, t, Iu, Iv, au):
    av=9 ; k=3 ; b=2 ; n=2
    (u,v)=y
    dudt= -u+(au/(1+(v/(1+Iv/k)**n)**b))
    dvdt= -v+(av/(1+(u/(1+Iu/k)**n)**b))
    return np.array([dudt,dvdt])
```

```
def bifurcation(au1,au2):
    steady_state_graph1=[]
    steady_state_graph2=[]
    y0=[0,0]
    for i in au1:
        y=odeint(toggleswitch3,y0,t1,(Iu,Iv,i))
        steady_state_graph1.append(y[-1,0])
        y0=y[-1]
    for j in au2:
        c=odeint(toggleswitch3,y0,t1,(Iu,Iv,j))
        steady_state_graph2.append(c[-1,0])
        y0=c[-1]
    plt.figure(figsize= (8, 8))
    plt.scatter(au1,steady_state_graph1,alpha=0.4,color='blue',label='Ascending values of au')
    plt.scatter(au2,steady_state_graph2,alpha=0.4,color='red',label='Descending values of au')
    plt.legend()
    plt.ylabel('U values at steady state')
    plt.xlabel('Au values')
    plt.show()
```

```
bifurcation(au_values,meh_values)
```

# A genetic oscillator

**Question 2.1: Explain why this system has negative feedback. [2]**

The Elowitz repressilator was made of 3 proteins LacI, tetR and CI. As shown on figure the production of one protein (e.g.TetR) after a cycle is complete will cause this same protein to be decreases creating a negative feedback loop which ultimately leads to temporal oscillations

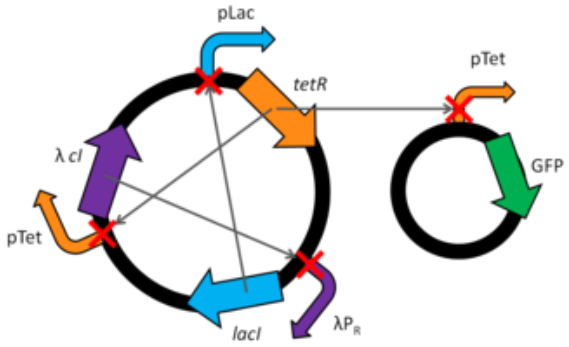


**Figure 6 . The Repressilator Source:https://www.nature.com/scitable/blog/bio2.0/the_repressilator**

**Question 2.2:**

Fix $b$=10 and by varying $a$, starting around the value of 14, produce six plots of the concentration of one of the proteins as the system goes through a bifurcation from a steady-state to a limit cycle. Make sure that at least one of the repressors has a positive initial value. If there are no repressors and no mRNAs, the system stays at this unstable steady-state. Label each subplot with its corresponding value of $a$. [4]

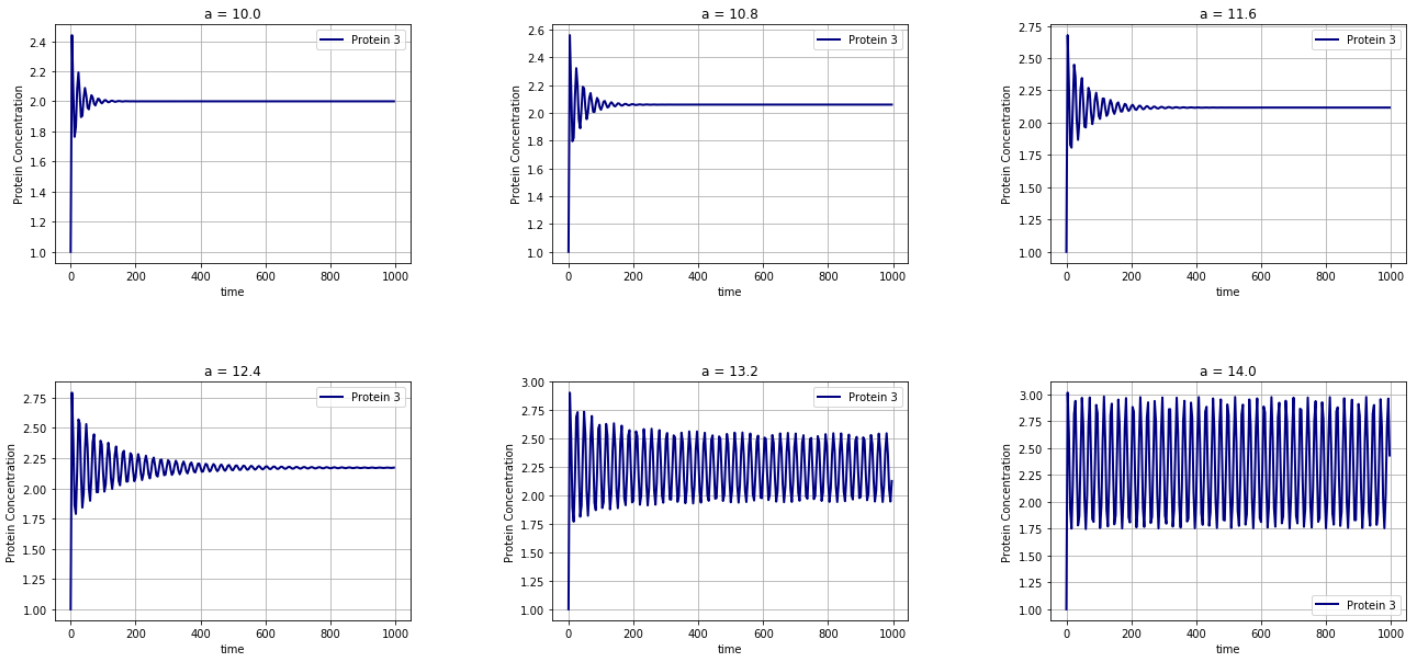


**Figure 7. Plots of protein concentration of protein 3 with a starting condition of 1, b=10 & a ranging from 10 to 14.**

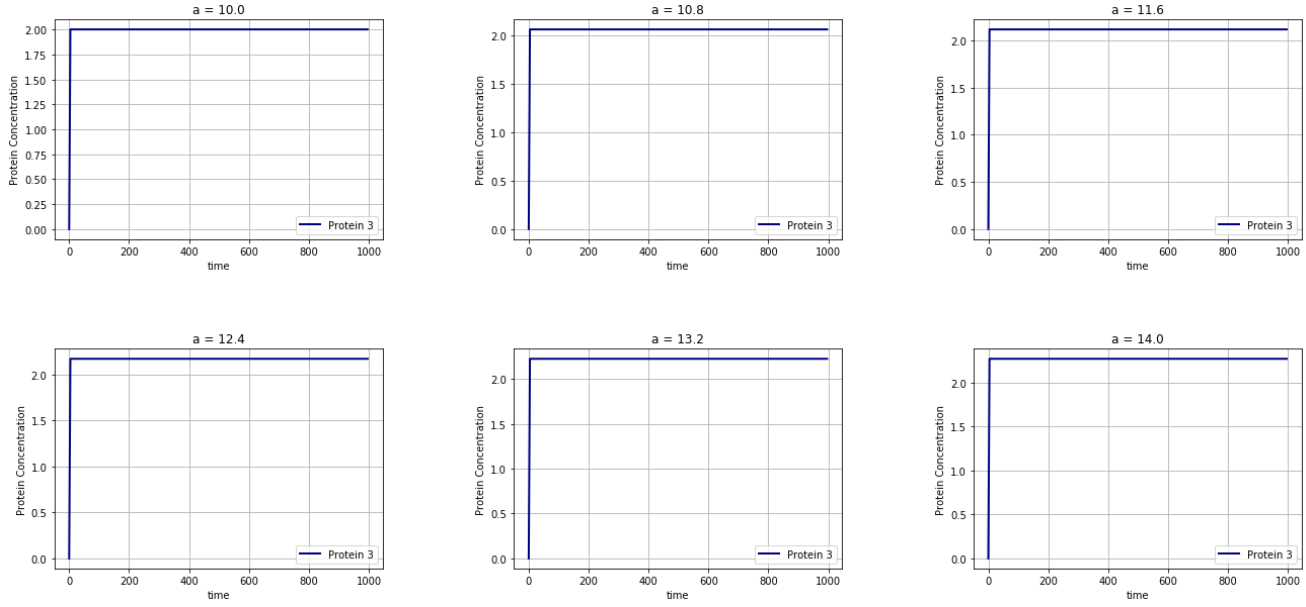


**Figure 8. Plots of protein concentration of protein 3 with a starting condition of 0, b=10 & a ranging from 10 to 14.**

On figure 7 we can see the transition from a steady state ( a=10) to an oscillating state(a=14) meaning that when the alpha number is increases the system starts to oscillate. If there are no repressors and no mRNAs, the system stays at a unstable steady-state as shown on Figure 8.

Code:

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# y[0], y[1], y[2] = conc of m1, m2 m3
#y[3], y[4], y[5] = p1 p2 p3

def f(y, t, a, n, b):
    return (-y[0]+a/(1+y[5]**n),
            -y[1]+a/(1+y[3]**n),
            -y[2]+a/(1+y[4]**n),
            b*(y[0]-y[3]),
            b*(y[1]-y[4]),
            b*(y[2]-y[5]))

a= np.linspace(10,14,6)
t= np.arange(0,1000,4)
ini= (0,0,0,0,0,1)
sp=1

plt.figure(figsize= (15, 10))
for i in a:
    rep= odeint(f, ini, t, args=(i,2,10))
    plt.subplot(2,3,sp)
    sp = sp+1
    plt.plot(t,rep[:,5],linewidth=2,color='navy')
    plt.title('a = %s' %(i))
    plt.ylabel('Protein Concentration')
    plt.xlabel('time')
    plt.legend(['Protein 3'])
    plt.subplots_adjust(left= 0.06, right=1.2, wspace=0.4, hspace=0.5)
    plt.grid()
```

**Question 2.3:** Write some Python code to automate this procedure. Let $a$ run between 1 and $10^3$ and $b$ run between 0.1 and $10^3$ (use np.logspace). Loop through each point on the grid (each particular value of $a$ and $b$) and use odeint to simulate the dynamics with these parameter values. Select a portion of the time-series of one of the proteins and use np.diff to determine if the protein is either at steady-state or is oscillating. To plot your bifurcation diagram, use the plt.contourffunction. To set the axes to a log-scale, you can use, for example,plt.xscale('log') and 'linear' to change the x-axis back. **[4]** Explain why your bifurcation diagram has the shape it does. **[4]**

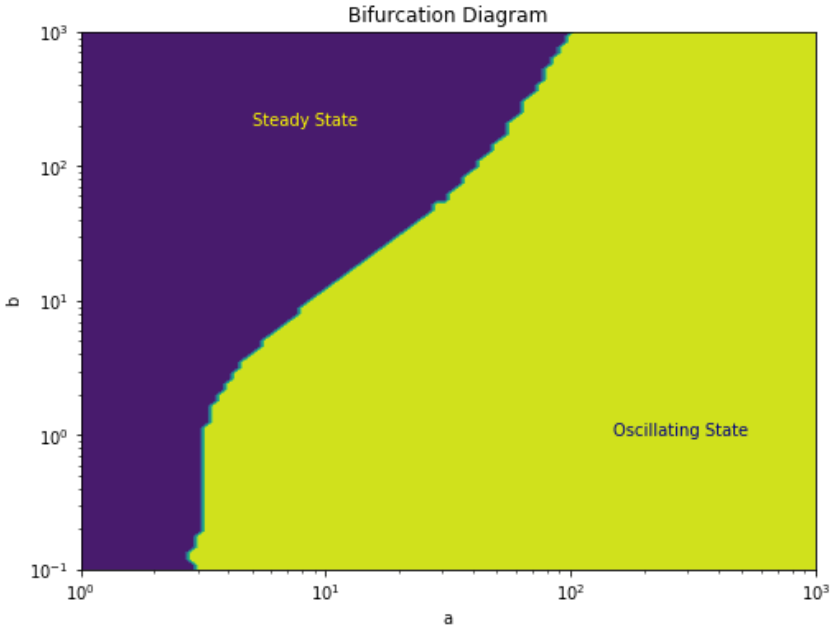


**Figure 9. Bifurcation Diagram**

On figure we can see a bifurcation diagram that shows how the system behaves with different values of a ( mRNA transcribed) and b (ratio of mRNA to protein lifetime) .With a high value of b and low values of a the system tends to a stable steady state. However when the system has high values of a and low values of b the system behaves in an oscillatory manner.

## Code:

```
a=np.logspace(0,3,100)
b=np.logspace(-1,3,100)
t = np.arange(0,100,1)
```

```
sw=[]
for i in b:
    for j in a:
        results=odeint(f,ini,t,(j,2,i))
        if np.sum(np.absolute(np.diff(results[-20:,5]))) <= 0.05:
            sw.append(0)
        else:
            sw.append(1)
sw_array=np.asarray(sw)
sw_grid=sw_array.reshape(100,100)
plt.figure(figsize= (8, 6))

plt.contourf(a,b,sw_grid)
plt.xscale('log')
plt.yscale('log')
plt.text(5,200,'Steady State',color='yellow')
plt.text(150,1,'Oscillating State',color='navy')
plt.ylabel('b')
plt.xlabel('a')
plt.title('Bifurcation Diagram')
plt.show()
```