Stochastic Simulations of Simple Chemical Reaction Networks Using Python

Diego Ortega Hernandez

January 12, 2019

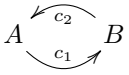
The following is a write up for a project I performed while in the M.S Mathematics program at San Jose State under Matthew D. Johnston presented with modifications for a general audience.

Introduction:

Chemical reaction networks consist of exchanges between reactants and products. In a stochastic simulation, we assume the molecules are bouncing around in some closed box with some probability that an reaction occurs is with respect to an exponential distribution. These events are memory-less, meaning the probability of a reaction does not increase as time moves forward (think of a really shitty bus where if you wait at the bus stop for 10 minutes, you do not have a higher chance of the bus appearing). The probability of a reaction occurring depends on two factors, initial concentration rates (the likelihood of a reaction to occur) and the amount of molecules floating around. Once it is time for a reaction occur, one of the reactions will occur with equal probability. We will be using the code provided by Matthew D. Johnston of San Jose State.

Simulations:

Example 1. Consider



with a conservation relation

$$A(t) + B(t) = N$$

where $A(t), B(t)$ are molecular counts of A and B, and N is some fixed natural (counting) number.

First, we will sketch the state space of the Markov Chain for $N = 5$

$$\begin{bmatrix} A=5 \\ B=0 \end{bmatrix} \rightleftarrows \begin{bmatrix} A=4 \\ B=1 \end{bmatrix} \rightleftarrows \begin{bmatrix} A=3 \\ B=2 \end{bmatrix} \rightleftarrows \begin{bmatrix} A=2 \\ B=3 \end{bmatrix} \rightleftarrows \begin{bmatrix} A=1 \\ B=4 \end{bmatrix} \rightleftarrows \begin{bmatrix} A=0 \\ B=5 \end{bmatrix}$$

There are 6 accessible states, or situations we can expect to see.

Let's simulate the system for $A(0) = 5$, $B(0) = 0$, $c_1 = c_2 = 1$ up to $t = 25$ and provide the plot. What can we observe?

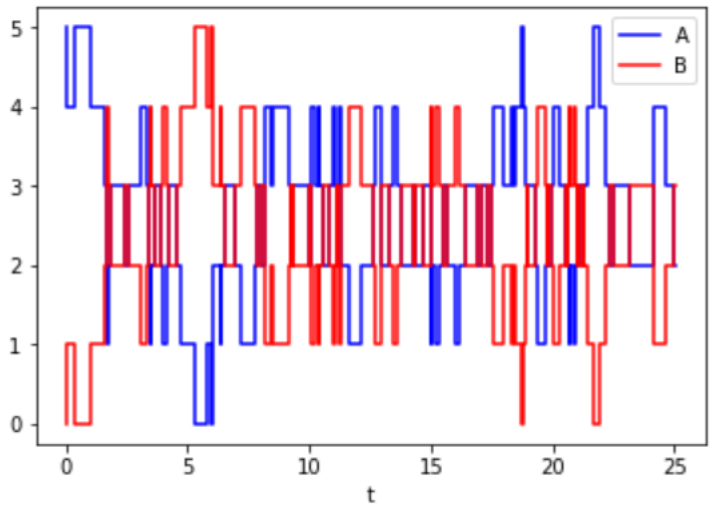


Figure 1: Plot of the system with $A(0) = 5$, $B(0) = 0$, $c_1 = c_2 = 1$ up to $t = 25$

Notice we have a symmetric plot where most of the concentrations of A and B occur between 2 and 3.

Let's generate 1000 independent simulations up to $t = 25$. We will store each value of A and plot a histogram of those values. What can we observe?

We modified the code to include the entire algorithm in a for-loop and appended the last value of A to a list

```
A_values = []
for i in range(1000):
        '''algorithm'''
        A_values.append(A[-1])

>>len(A_values)
1000
```

Now we will simulate one trajectory up to the time t = 1000 and produce a bar graph of the dwell times for the different values of A. The Dwell time is the amount of time spent in a state over the total time.

We modified the code to change t_end = 1000, added a list of 0's where each index corresponds directly to an A value. We add the time step to the index when the last A value occurred after all reactions in one time step.

```
#part C(ii)
t_end = 1000

#we need dwell values for each state of A
#each index corresponds to that state of A
#e.g. Dwell[2] is when A = 2
Dwell = [0,0,0,0,0,0]

if total_prop != 0:
        '''other code'''
```

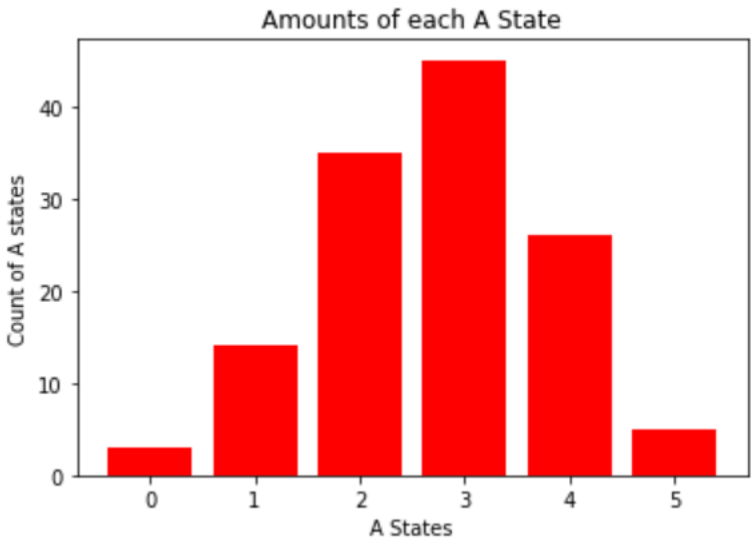


Figure 2: Histogram of A-values after 1000 independent simulations

```
        #part c(ii) record the time value
        Dwell[int(A[-1])] = Dwell[int(A[-1])] + t_temp
```

```
>>Dwell
[30.1992373180904,
157.971417472355,
300.94488539163694,
314.8312621066903,
161.33768178569568,
34.85092789014552]

>>sum(Dwell)
1000.1354119646138
```

Notice the sum of the dwells is slightly past 1000 time counts, which is fine since the while loop in the code stops slightly past 25. Also, observe we spent most of our time in states $A = 2, A = 3$, and spent the least amount of time at states $A = 0, A = 5$, which is expected since there is a less chance of a single reaction to happen.

Now let's calculate the dwell times using the following code:

```
Dwell_times = []
for time in Dwell:
        Dwell_times.append(time/1000)
```

```
>>Dwell_times
[0.0301992373180904,
0.157971417472355,
0.30094488539163694,
0.3148312621066903,
0.16133768178569569,
0.034850927890145524]
```

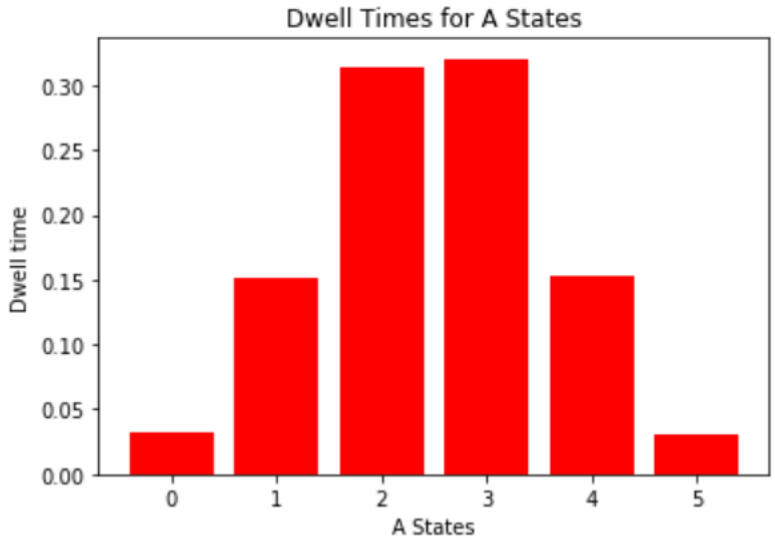
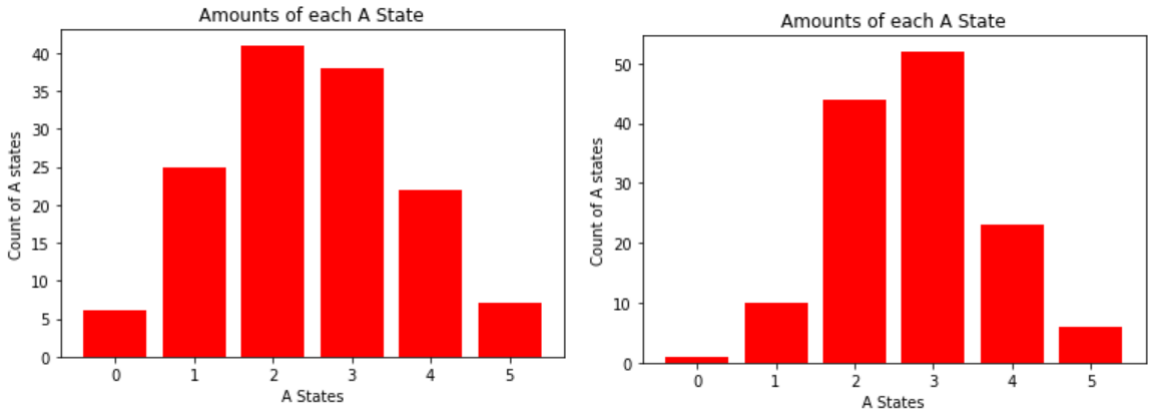


Figure 3: Dwell times for $A = 0, ..., 5$

Notice the two plots are roughly the same, which should not be surprising since there is a direct correlation between the average time spent in a state vs the frequency of the states. That means the A's with the longest dwell times also occur the most often, and vice versa. We also noticed if there are more 2's then 3' and vice versa, the other states roughly reflect that relationship too. i.e. more of the left side than right side. See Figure 4.



(a) A-values with $A = 2$ larger than $A = 3$

(b) A values with $A = 2$ smaller than $A = 3$

Figure 4: Higher frequency of $A = 2$ than $A = 3$ and vice versa

Example 2. Consider the reaction network

$$D_u + A \underset{c_1}{\overset{c_2}{\rightleftarrows}} D_b \xrightarrow{c_3} D_b + R$$

$$R \xrightarrow{c_4} R + P$$

$$R \xrightarrow{c_5} \emptyset$$

$$P \xrightarrow{c_6} \emptyset$$

In this network, A is an activator, D_u is unbound DNA, D_b is bound DNA, R is RNA, and P is a protein. This network represents activation of a gene. When activated, the DNA will produce the protein P while when inactivated it produces nothing.

(a) Suppose that $D_u(0) = 1$, $A(0) = 1$, $D_b(0) = R(0) = P(0) = 0$, $\{c_i\}_{i=1}^4 = c_6 = 25$, and $c_5 = 1$. We will use the stochastic simulation algorithm to simulate the trajectory up to $t = 100$ and display the evolution over time for the five species. What can we observe?

We modified the code to include all of the initial data, and added extra elif statements in order to determine the reaction that activates. We do this by shifting the reaction by λ_i to the interval

$$\frac{\lambda_i}{\sum \lambda_i} \leq \gamma \leq \frac{\lambda_{i+1}}{\sum \lambda_i}$$

Since each reaction is chosen off the uniform distribution from $(0, \sum \lambda_i)$ with propensities λ_i, we shift enough propensities. We converted some of the code to list formatting in order to increase its runtime speed by reducing the amount of floating point operations (see Issues).

```
#Parameters
c1 = 25.0
c2 = 25.0
c3 = 25.0
c4 = 25.0
c5 = 1.0
c6 = 25.0
#Initial data
Du = [1.0]
A = [1.0]
Db = [0.0]
R = [0.0]
P = [0.0]

t =

while loop:

        prop = [0,0,0,0,0,0]
        #forward reaction
        prop[0] = c1*Du[-1]*A[-1]
        #backward reaction
        prop[1] = c2*Db[-1]
        #forming RNA
        prop[2] = c3*Db[-1]
        #forming protein
        prop[3] = c4*R[-1]
        #decaying reactions
        prop[4] = c5*R[-1]
        prop[5] = c6*P[-1]
        #total propensities
        total_prop = sum(prop)

        Rtemp = R[-1]+0
        Ptemp = P[-1]+0
        if total_prop != 0:
```

```
        t_temp = -1/total_prop*np.log(rand_wait)
        #if reaction wait is from (0,prop1)
        if rand_reaction <= prop[0]/total_prop:
                Dutemp = Du[-1]-1
                Atemp = A[-1]-1
                Dbtemp = Db[-1]+1


        #if reaction is from (prop1,prop2)
        elif rand_reaction <= (sum(prop[0:2]))/total_prop:
                Dutemp = Du[-1]+1
                Atemp = A[-1]+1
                Dbtemp = Db[-1]-1

        #if reaction is from (prop2,prop3)
        elif rand_reaction <= (sum(prop[0:3]))/total_prop:
                #Dbtemp = Db[-1]+0
                Rtemp = R[-1]+1

        #if reaction is from (prop3,prop4)
        elif rand_reaction <= (sum(prop[0:4]))/total_prop:
                Rtemp = R[-1]+0
                Ptemp = P[-1]+1

        #if reaction is from (prop4,prop5)
        elif rand_reaction <= (sum(prop[0:5]))/total_prop:
                Rtemp = R[-1]-1

        #if reaction is from (prop5,prop6)
        elif rand_reaction <= 1:
                Ptemp = P[-1]-1

        t.append(t[-1]+t_temp)
        Du.append(Dutemp)
        A.append(Atemp)
        Db.append(Dbtemp)
        R.append(Rtemp)
        P.append(Ptemp)
```

We generated some of the graphs with different percentages of time steps data to make it easier to see the behaviour of states using the function.

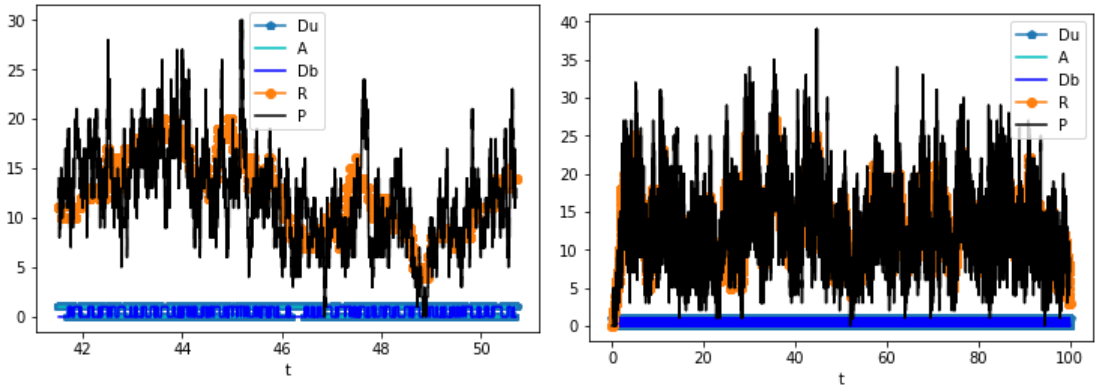
This is the code to plot the data from the previous Stochastic algorithm.

```
#n's and d's are numerators and denominators respectively
#must be integers
def scale_list(L, n1,d1, n2,d2):
        L = L[n1*len(L)//d1: n2*len(L)//d2]
        return L

scales = [1,500,4,5]
t = scale_list(t,scales[0],scales[1],scales[2],scales[3])
Du = scale_list(Du,scales[0],scales[1],scales[2],scales[3])
A = scale_list(A,scales[0],scales[1],scales[2],scales[3])
```

```
Db = scale_list(Db, scales[0], scales[1], scales[2], scales[3])
R = scale_list(R, scales[0], scales[1], scales[2], scales[3])
P = scale_list(P, scales[0], scales[1], scales[2], scales[3])

plt.plot(t,Du,'g',linestyle='steps',label='Du')
plt.plot(t,A,'c',linestyle='steps',label='A')
plt.plot(t,Db,'b',linestyle='steps',label='Db')
plt.plot(t,R,'r',linestyle='steps',label='R')
plt.plot(t,P,'k',linestyle='steps',label='P')
plt.xlabel('t')  plt.legend()
```



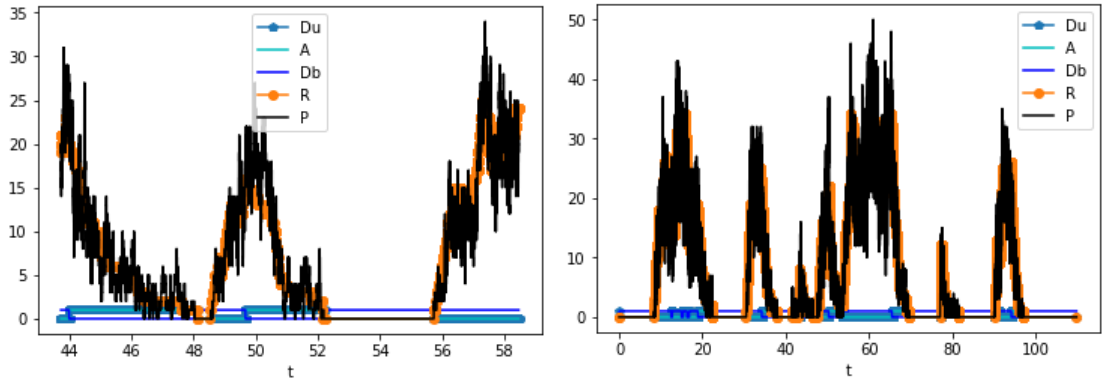
(a) From 4/10 to 5/10 of the time steps

(b) With all time steps

Figure 5: Evolution of time for the 5 species with $\{c_i\}_{i=1}^{4} = c_6 = 25$, and $c_5 = 1$

Now let us change the values above to $c_1 = c_2 = 0.25$ (rest the same) and simulate the trajectory up to $t = 100$ then display the evolution over time for the five species. What can we observe? Do you (the reader) expect there to be any discrepancy?

Using these changes and the same code, we plot the figures again with the same percentages of data to compare their behaviour.



(a) From 4/10 to 5/10 of the time steps

(b) With all time steps

Figure 6: Evolution of time for the 5 species with $c_1 = c_2 = 0.25$, $c_3 = c_4 = c_6 = 25$, and $c_5 = 1$

Observe if the activation of bounded DNA is less consistent, then the reaction will produce proteins less consistently. Notice there is a dry spell of unbounded DNA, which meant that no RNA (and hence protein) was being formed because it died off too fast. Although P's decay rate is large, R's decay rate is small, so we expect to not see a significant difference in the amount of RNA and P.

Notice the lack of difference in scale in the amount being produced. Despite the binding and unbinding reactions occurring at slower rates, changing the propensity constants seems to significantly affect the amount of binding DNA being produced. Sub-Figures 5.a has a much larger amplitude of binding and unbinding DNA, whereas Sub-Figure 5.a has a much less frequent production cycle. Notice the rest of the molecules follow the same exponential growth trend (zoomed in) with minimal chaos. In both situations, binded DNA, RNA, and P are the final products. However, unbinded DNA only appears as a final product in part (a).

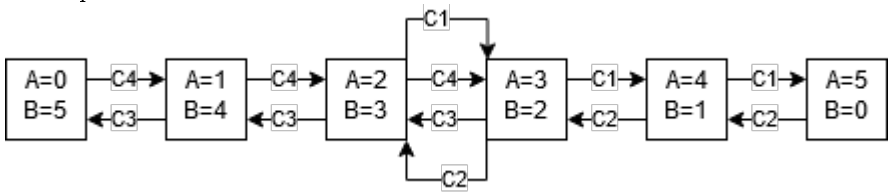
Problem 3. Consider the reaction network

$$R1 : 2A + B \underset{c_2=\epsilon}{\overset{c_1=1}{\rightleftarrows}} 3A$$

$$R2 : A + 2B \underset{c_4=\epsilon}{\overset{c_3=1}{\rightleftarrows}} 3B$$

where $\epsilon > 0$.

Here is a sketch the state space of the Markov Chain for A + B = 5 with all of the accessible states and all of the possible transitions.



Notice state $A = 3, B = 2$ and $A = 2, B = 3$ appears in at least 4 different situations for reactions 1 and 2.

Next, we will use the stochastic simulation algorithm to simulate the system for the values $A(0) = 50$, $B(0) = 50$, and (i) $\epsilon = 2$; and (ii) $\epsilon = 0.5$ to produce plots for up to $t = 100$. What can we observe?

We used the following code to simulate the reaction.

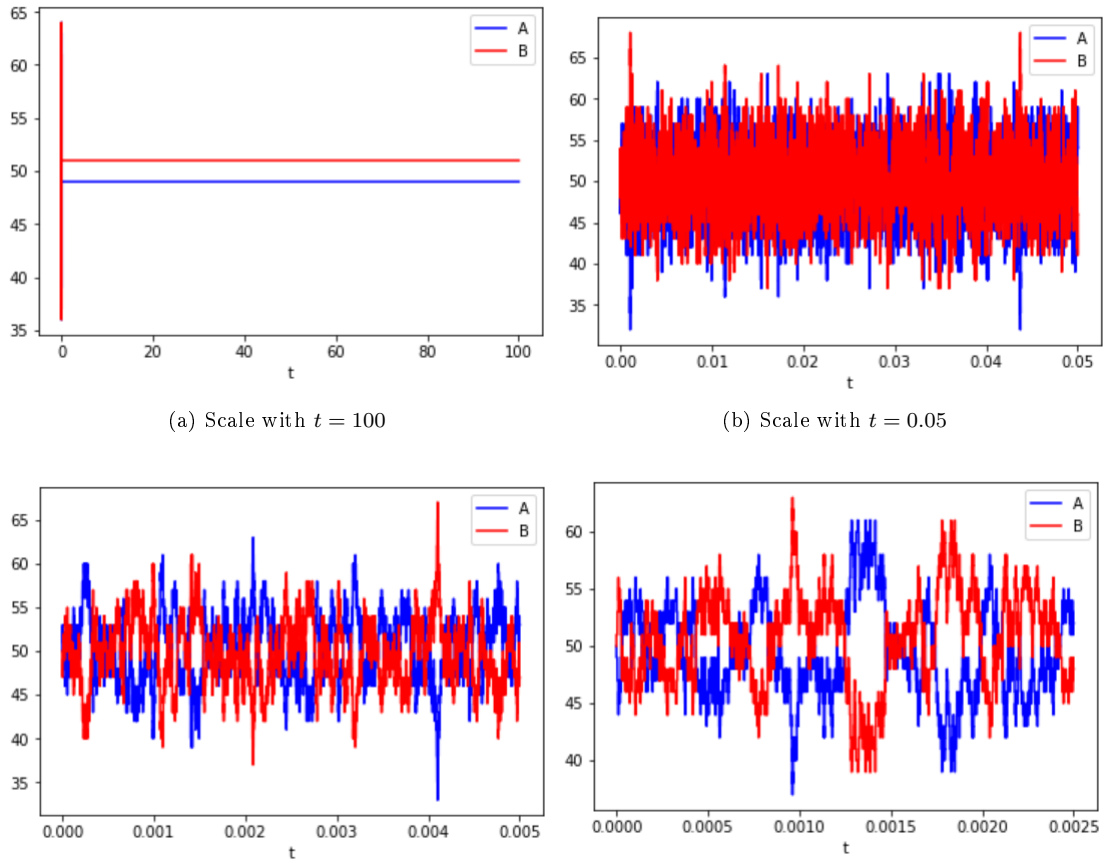
```
###PROBLEM 3 part b###
# Import packages
import numpy as np
import matplotlib.pyplot as plt
import random as rand
# Parameters
c1 = 1.0
c2 = 1.0
#c3 and c4 are epsilon
c3 = c4 = 2.0
#c3 = c4 = 0.5

# Initial data
A = [50.0]
B = [50.0]
t = [0.0]
ts = [0.00025,0.005,0.05,100]
```

```
t_end = ts[0]
#dwell values for each state of A because curiosity
Dwell = [0]*100
# Simulate trajectory
while t[-1] < t_end:
        rand_wait = rand.random()
        rand_reaction = rand.random()
        #propensities
        prop = [0,0,0,0]
        #reaction 1 (f)
        prop[0] = c1*(A[-1]**2)*B[-1]
        #reaction 1 (b)
        prop[1] = c3*A[-1]**3
        #reaction 2 (f)
        prop[2] = c2*A[-1]*(B[-1]**2)
        #reaction 2 (b)
        prop[3] = c4*B[-1]**3
        #total propensities
        total_prop = sum(prop)
        if total_prop != 0 and rand_wait > 0.00001:
                t_temp = -1/total_prop*np.log(rand_wait)
                #if reaction wait is from (0,prop1)
                if rand_reaction <= prop[0]/total_prop:
                        #A loses 2 and gains 3
                        Atemp = A[-1]+1
                        Btemp = B[-1]-1
                #if reaction is from (prop1,prop2)
                elif rand_reaction <= sum(prop[0:2])total_prop:
                        #A loses 3 and gains 2
                        Atemp = A[-1]-1
                        Btemp = B[-1]+1
                #if reaction is from (prop2,prop3)
                elif rand_reaction <= sum(prop[0:3])total_prop:
                        #B loses 2 and gains 3
                        Atemp = A[-1]-1
                        Btemp = B[-1]+1
                elif rand_reaction <= 1:
                        #B loses 3 and gains 2
                        Atemp = A[-1]+1
                        Btemp = B[-1]-1
                t.append(t[-1]+t_temp)
                A.append(Atemp)
                B.append(Btemp)
                #part c(ii) record the time value
                Dwell[int(A[-1])] = Dwell[int(A[-1])] + t_temp
        else:
                A.append(A[-1])
                B.append(B[-1])
                t.append(t[-1]+t_end)
plt.plot(t,A,'b',linestyle='steps',label='A')
plt.plot(t,B,'r',linestyle='steps',label='B')
plt.xlabel('t')
plt.legend()
plt.show()
```

Observations:

In Figures 8 and 9, we show four plots with $\epsilon = 2$ and $\epsilon = 0.5$. Each has a plot of the reaction on the whole interval of $t = 100$ and the others are with smaller t's to scale the image better (see Issues). Since the propensity is much higher for (i) than in (ii), we expect the reactions to occur more frequently with higher density in the plot. Observe the gap in the middle grows larger as the propensity for ϵ reactions is smaller. One can see this difference in Sub-Figures 7d and 8d. We still have a symmetric plot because the reactions are symmetric, with growing and increasing by A or B respectively.



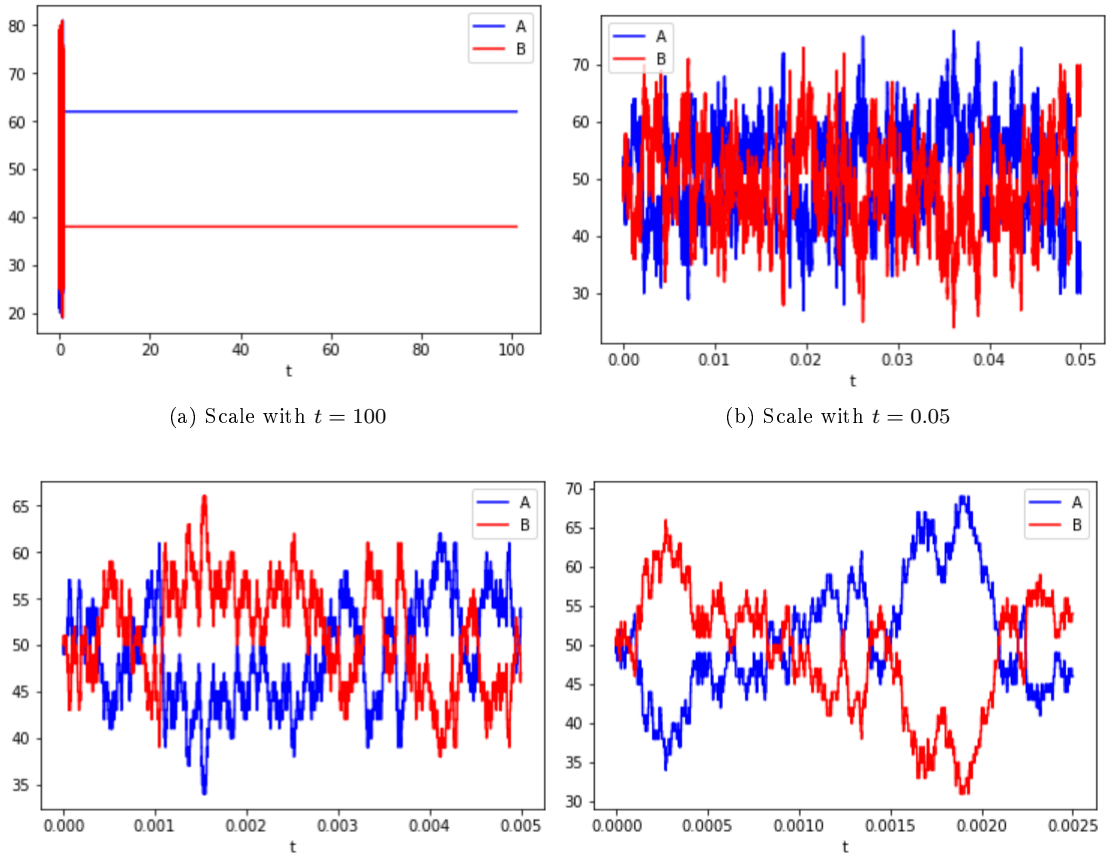
(a) Scale with $t = 100$

(b) Scale with $t = 0.05$

(c) Scale with $t = 0.005$

(d) Scale with $t = 0.0025$

Figure 7: Plots of reaction 1 and 2 with $\epsilon = 2$ with different scales



(a) Scale with $t = 100$

(b) Scale with $t = 0.05$

(c) Scale with $t = 0.005$

(d) Scale with $t = 0.0025$

Figure 8: Plots of reaction 1 and 2 with $\epsilon = 0.5$ with different scales

Issues running original code:

We had to convert most of the floating point operations to list operations to increase the code's speed.

The line

t_temp = -1/total_prop*np.log(rand_wait)

will pose some potential problems as it generates extremely small t_temps as the total propensity grows.

Example. Small time steps generated by the t_temp

```
>>min(waits)
1.703909738859366e-07
>>max(waits)
0.9999989949542213

>>import numpy as np
>>np.mean(waits)
0.49991258907949276
>>np.var(waits)
0.08340218140899762
```

Notice there is a small variance between the wait times, which means that the amount of extremely small t_steps will not be an isolated incident and greatly affect the performance. Notice how many times it would take to run for just one time step $t = 1$ (at the extreme case where all t_steps are minimal)

```
#smallest t_temp at the first total propensity
>>-1/25**5*np.log(min(waits))
1.595921427876569e-06
#number of while-loops to run for t=1
>>1/(-1/25**6*np.log(min(waits)))
626597.2638330548
```

My computer cannot run that. To compensate for this, we added an additional constraint to avoid extremely small time steps:

```
if total_prop != 0 and rand_wait > 0.00001:
t_temp = -1/total_prop*np.log(rand_wait)
```

Any smaller and the computer stalls for a very long time. The drawback is that we will only get see a small percentage of the actual reactions, but we should be able to get the full picture by adjusting time scales.