PHYS280: Physical Models of Biological Systems

Final Project

Author:
Kavish Senthilkumar

December 15, 2021

1 Repressilator Dynamical System Analysis

Introduction

Oscillations occur in many dynamical biological systems.[1] For one, they appear in biological negative feedback loops—several examples have been experimentally observed and analyzed.[2]

Initially, experimenters had difficulty experimentally tuning the delay in these feedback loop systems in a consistent fashion. But in 2000, Elowitz and Leibler[3] created the repressilator: a system of three repressive transcription factors (TetR, LacI, and cI) wherein each factor represses the the production of the next factor in the system in a ring.

These system was synthetically constructed, and allowed tuning of the growth and decay parameters of each factor, and, as a result, the overall negative feedback.

We analyze a simplified form of the repressilator network, using a continuous deterministic approximation. In the figure below, a diagram of the repressilator is shown. Transcription and translation are summarized with a single arrow.

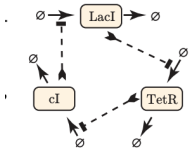

Fig. 13.2

We consider three mRNA species m_A, m_B, m_C the are degraded with the first-order sink parameter set to 1 (via rescaled time units). The species are created with the same maximal rate but production rate is regulated by the product of the other species. We use the same half-maximal concentration ($K_M = 1$).

With this, we derive three rate laws $\frac{d\bar{c}_{m_A}}{d\bar{t}}$, $\frac{d\bar{c}_{m_B}}{d\bar{t}}$, and $\frac{d\bar{c}_{m_C}}{d\bar{t}}$. The gene products are assumed to follow the mass-action rate laws (production and dilution are both first order in this system), and we similarly derive three rates of change for the concentration of product $\frac{d\bar{c}_{p_A}}{d\bar{t}}$, $\frac{d\bar{c}_{p_B}}{d\bar{t}}$, and $\frac{d\bar{c}_{p_C}}{d\bar{t}}$. We have a system of six ordinary differential equations that determine a dynamic system. The equations are shown below.

$$\frac{d\bar{c}_{m_A}}{d\bar{t}} = -\bar{c}_{m_A} + \frac{50}{(1+(\bar{c}_{p_c})^2} \qquad \frac{d\bar{c}_{p_A}}{d\bar{t}} = 0.2(-c_{p_A} + c_{m_A})$$

$$\frac{d\bar{c}_{m_B}}{d\bar{t}} = -\bar{c}_{m_B} + \frac{50}{(1+(\bar{c}_{p_a})^2} \qquad \frac{d\bar{c}_{p_B}}{d\bar{t}} = 0.2(-c_{p_B} + c_{m_B})$$

$$\frac{d\bar{c}_{m_C}}{d\bar{t}} = -\bar{c}_{m_C} + \frac{50}{(1+(\bar{c}_{p_b})^2} \qquad \frac{d\bar{c}_{p_C}}{d\bar{t}} = 0.2(-c_{p_C} + c_{m_C})$$

We will solve this system for the six initial conditions $\bar{c}_{m_A}(0) = c_{p_A}(0) = 1.5, \bar{c}_{m_B}(0) = 0.5, \bar{c}_{p_B}(0) = 2, \bar{c}_{m_C}(0) = 1, c_{p_C}(0) = 2$.

[1] Keith's lecture notes

[2] Pigolotti, S., Krishna, S., amp; Jensen, M. H. (2007). Oscillation patterns in negative feedback loops. Proceedings of the National Academy of Sciences, 104(16), 6533–6537. https://doi.org/10.1073/pnas.0610759104

[3] Elowitz, M., Leibler, S. A synthetic oscillatory network of transcriptional regulators. Nature 403, 335–338 (2000). https://doi.org/10.1038/35002125

Rewriting with a $\bar{c}$ vector, given $\bar{c} = \begin{bmatrix} \bar{c}_{m_A} \\ \bar{c}_{m_B} \\ \bar{c}_{m_C} \\ \bar{c}_{p_A} \\ \bar{c}_{p_B} \\ \bar{c}_{p_C} \end{bmatrix}$, $\frac{d\bar{c}}{d\bar{t}} = F(\bar{c}, \bar{t}) = \begin{bmatrix} -\bar{c}_{m_A} + \frac{50}{(1+(\bar{c}_{p_C})^2} \\ -\bar{c}_{m_B} + \frac{50}{(1+(\bar{c}_{p_a})^2} \\ -\bar{c}_{m_C} + \frac{50}{(1+(\bar{c}_{p_b})^2} \\ 0.2(-c_{p_A} + c_{m_A}) \\ 0.2(-c_{p_B} + c_{m_B}) \\ 0.2(-c_{p_C} + c_{m_C} \end{bmatrix}$

We solve this with the ODE package in `odeint`. Code provided in Appendix.

1a and 1b: 1D and 3D Projections of Solution

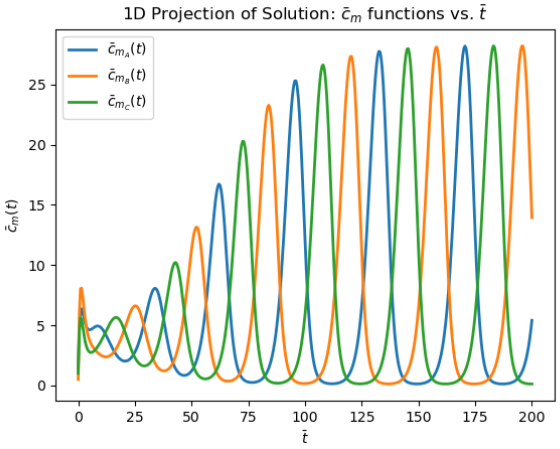

Fig. 1.1

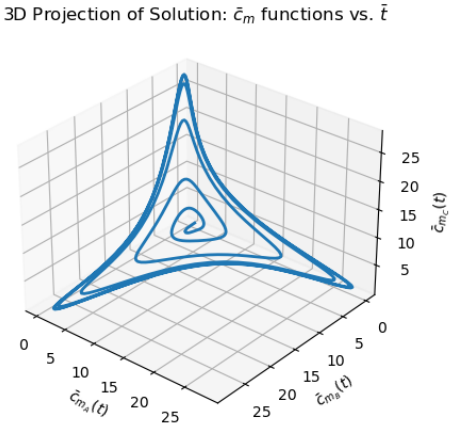

Fig. 1.2

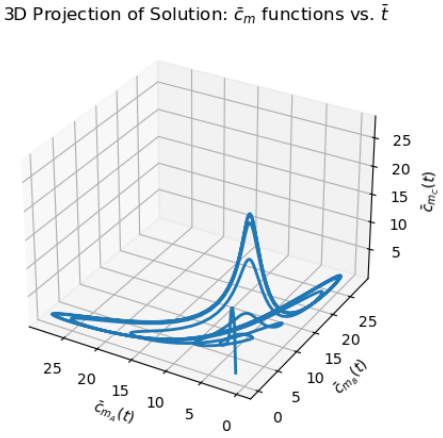

Fig. 1.3

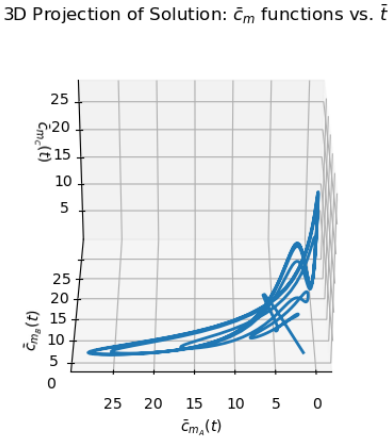

Fig. 1.4

Projections of solutions. **Fig. 1.1:** 1D projection of $\bar{c}_m(\bar{t})$ vs. $\bar{t}$. **Fig 1.1-1.3** Three views of 3D projection of $\bar{c}_m$ functions vs. $\bar{t}$

Figure 1.1 shows a 1D projection of $\bar{c}_{m_A}(\bar{t})$ vs. $\bar{t}$, as well as the other functions $\bar{c}_{m_B}(\bar{t})$ and $\bar{c}_{m_C}(\bar{t})$ vs. $\bar{t}$. It clearly shows oscillatory behavior: these $\bar{c}_m$ values oscillate (with increasing amplitude at the start until they stabilize), and each $\bar{c}_m$ has an individual peak: at this peak, the other two concentrations are at the bottom of an oscillation. This aligns with our understanding of repressilator behavior.

1c: Animation attached

Code in Appendix.

1d: Stable fixed point

A fixed point will be found at the intersection of the nullclines of the system. We see $\frac{d\bar{c}_{p_A}}{d\bar{t}} = 0 \implies 0.2(-c_{p_A} + c_{m_A}) = 0 \implies c_{p_A} = c_{m_A}$ is a nullcline. Similarly we see, $c_{p_B} = c_{m_B}$ and $c_{p_C} = c_{m_C}$ as nullclines. After trying out several initial conditions that fit this criteria, we attempt $c_{m_A} = c_{m_B} = c_{m_C}$ and $c_{p_A} = c_{p_B} = c_{p_C}$. We test various values of this initial condition below, calling $c_{init_1} = c_{m_A} = c_{m_B} = c_{m_C}$ and $c_{init_2} = c_{p_A} = c_{p_B} = c_{p_C}$.

We note that with this initial condition, all the equations for the rate laws ($\frac{d\bar{c}_{m_A}}{d\bar{t}}, \frac{d\bar{c}_{m_B}}{d\bar{t}}, \frac{d\bar{c}_{m_C}}{d\bar{t}}$) become identical to each other. We call the equivalent rate $\frac{d\bar{c}_m}{d\bar{t}}$. The rates for change of product ($\frac{d\bar{c}_{p_A}}{d\bar{t}}, \frac{d\bar{c}_{p_B}}{d\bar{t}}$, and $\frac{d\bar{c}_{p_C}}{d\bar{t}}$) become identical as well. We call this equivalent rate $\frac{d\bar{c}_p}{d\bar{t}}$. We plot 1D projections for this initial condition below.

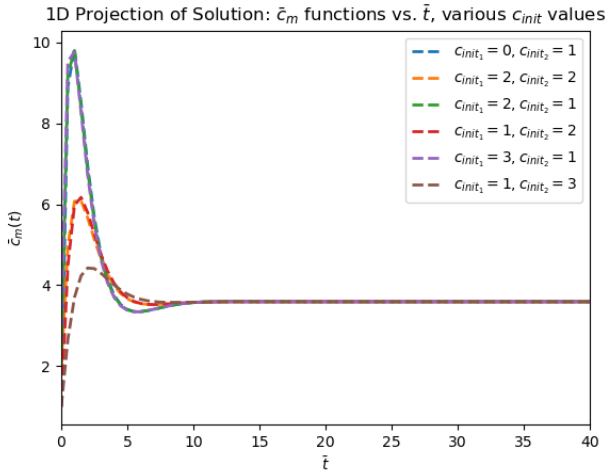

Fig. 2.1

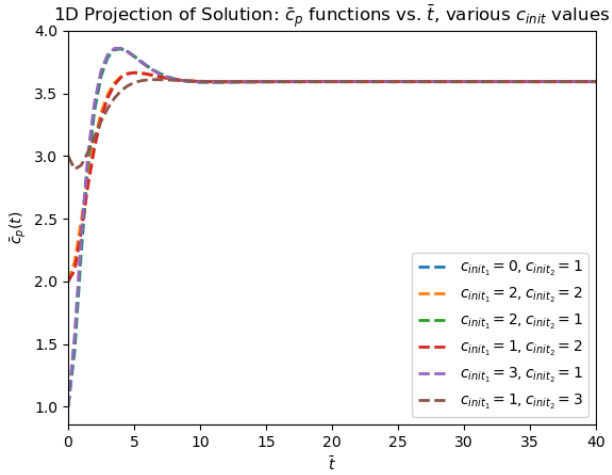

Fig. 2.2

Fig. 2.1: 1D projection of $\bar{c}_m(\bar{t})$ vs. $\bar{t}$ for c_{init} values. **Fig. 2.2** 1D projection of $\bar{c}_m(\bar{t})$ vs. $\bar{t}$ for c_{init} values.

We see the concentrations all stabilize to a state when we have initial conditions c_{init_1} and c_{init_2}. Regardless of the tested value of c_{init_1} and c_{init_2}, we reach a steady state fixed point at $\bar{c} \approx 3.594$ (calculated by setting all the concentration functions equal). We use the

symbolic mathematics package `sympy` to calculate all other fixed points[4], and find all other fixed points are imaginary. Their real components are all also negative, indicating they are nonphysical phenomena.

For a better visualization of our real fixed point, we make a phase portrait for the c_m functions, and draw a line along $c_{mA} = c_{mB} = c_{mC}$. We see a trajectory starting at any point on this line will move along the line until it reaches the fixed point.

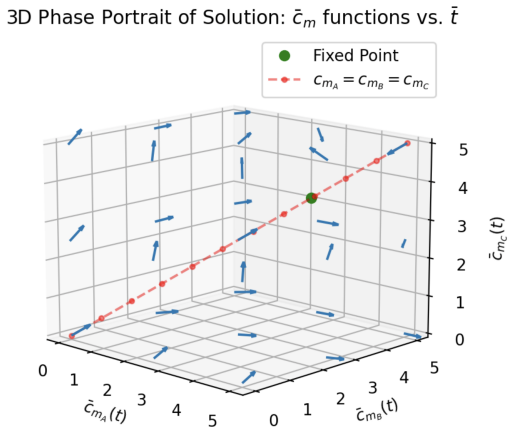

Fig. 2.3

Fig. 2.3: 3D phase portrait with fixed point and the identity line for starting points of trajectories

2 Kinesin Stepping

Introduction

For the four-state, eight-reaction model for kinesin stepping shown on the right, Fisher and Kolomeisky[5] proposed a set of values for parameters. They also proposed a force dependent relation for each reaction, that, combined with these initial values, offers rate constants for the model.

$$\begin{array}{ccc} 0 & \underset{\beta_1^-}{\overset{\beta_0^+}{\rightleftharpoons}} & 1 \\ \beta_3^+ \Big\| \beta_0^- & & \beta_2^- \Big\| \beta_1^+ \\ 3 & \underset{\beta_2^+}{\overset{\beta_3^-}{\rightleftharpoons}} & 2 \end{array}$$

We use Gillespie's direct algorithm to simulate motor stepping in this model for runs of 5000 steps at several force/ATP combinations. The motor steps 8.2 every time it transitions $0 \to 1$, and -8.2 nm every time it makes the opposite transition. For each simulated trial, we find the overall velocity by computing total distance traveled divided by total elapsed time. We plot overall velocity versus ATP for $f = 1.05pN$ and $f = 3.59pN$ for $1\mu M \leq c_{ATP} \leq 5\mu M$ and overall velocity versus force for $c_{ATP} = 5\mu M$ and $1pN \leq f \leq 4.5pN$. Code in Appendix.

[4]Stability analysis with Python, http://systems-sciences.uni-graz.at/etextbook/sw2/phpl$_p$*ython.html*
[5]Fisher and Kolomeisky model from textbook

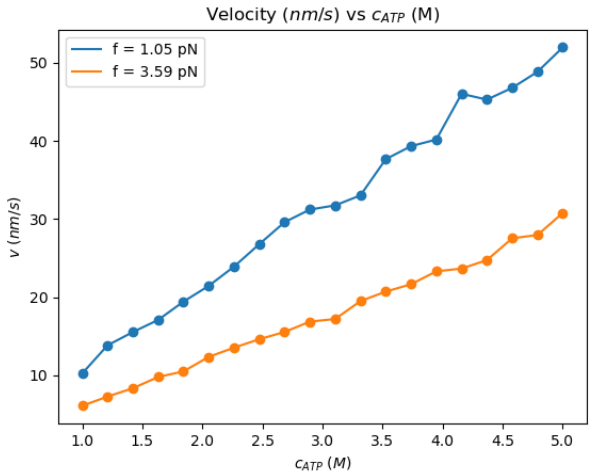

Fig. 3.1

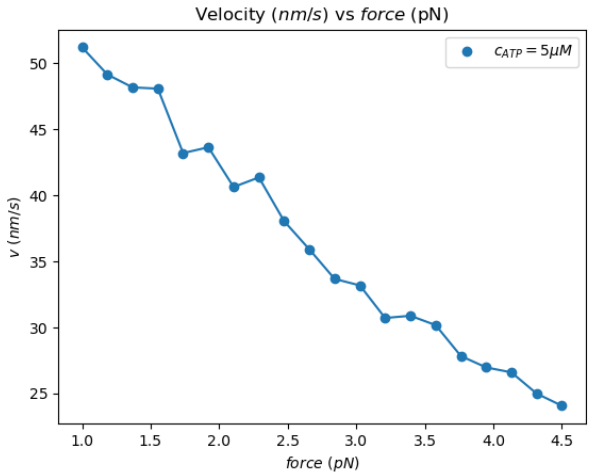

Fig. 3.2

Fig. 3.1: Overall velocity versus ATP for $f = 1.05$ pN and $f = 3.59pN$ for $1 \leq c_{ATP} \leq 5\mu M$
Fig. 3.2 Overall velocity versus force for $c_{ATP} = 5\mu M$ and $1pN \leq f \leq 4.5pN$

In Fig. 3.1, we observe the overall velocity increases with concentration of ATP (potentially because more ATP increases the likelihood of a step occurring) when we restrict concentration to small values. In addition, we see that for all ATP concentrations, a greater external force leads to a lower overall velocity (potentially, larger external force leads to lower likelihood of a forward turn, i.e. because a larger force prevents the motor from stepping forward).

In Fig. 3.2, we see there is an apparently linear decrease in velocity with increasing external force applied. This could support the claim above that a larger external force leads to a lower likelihood of a forward turn and thus a lower overall velocity.

Credit

Compared graphs with Noah and Jas

Appendix: Code

Figure 1.1

```
#create system
def F(c, t):
    dc1 = -c[0] + 50/(1+c[5] ** 2)
    dc2 = -c[1] + 50/(1+c[3]** 2)
    dc3 = -c[2] + 50/(1+c[4] ** 2)
    dc4 = 0.2*(-c[3]+c[0])
    dc5 = 0.2*(-c[4]+c[1])
    dc6 = 0.2*(-c[5]+c[2])
    return [dc1, dc2, dc3, dc4, dc5, dc6]

t_min = 0; t_max = 200; dt = .5
t = np.arange(t_min, t_max+dt, dt)

#solve system with initial conditions
y0 = (1.5,0.5,1,1.5,2,2)
y = odeint(F, y0, t)

#plot
plt.title(r"1D Projection of Solution: $\bar c_m$ functions vs. $\bar t$")
plt.plot(t, y[:, 0], linewidth=2, label = r"$\bar c_{m_A}(t)$")
plt.plot(t, y[:, 1], linewidth=2, label = r"$\bar c_{m_B}(t)$")
plt.plot(t, y[:, 2], linewidth=2, label = r"$\bar c_{m_C}(t)$")

plt.ylabel(r"$\bar c_{m}(t)$")
plt.xlabel(r"$\bar t$")
plt.legend()

plt.show()
plt.figure()
```

Figure 1.2-1.4

```
#plot details
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.invert_xaxis()
plt.title(r"3D Projection of Solution: $\bar c_m$ functions vs. $\bar t$")
ax.set_xlabel(r"$\bar c_{m_A}(t)$")
ax.set_ylabel(r"$\bar c_{m_B}(t)$")
ax.set_zlabel(r"$\bar c_{m_C}(t)$")
```

```
#plot solution
ax.plot3D(y[:, 0], y[:, 1], y[:, 2], linewidth=2)
ax.view_init(30, 270)
fig2 = plt.figure()
```

Attached animation

```
#stepping for animation
def get_step(n):
    ax = plt.axes(projection='3d')
    plt.title(r"3D Projection of solution with animated trajectory: $\bar c_m$
        functions vs. $\bar t$")
    ax.set_xlim(0,30)
    ax.set_ylim(0,30)
    ax.set_zlim(0,30)

    #plot same as 3D graph, with up to step index
    ax.plot3D(y[0:n, 0], y[0:n, 1], y[0:n, 2], linewidth=2);
    ax.set_xlabel(r"$\bar c_{m_A}(t)$")
    ax.set_ylabel(r"$\bar c_{m_B}(t)$")
    ax.set_zlabel(r"$\bar c_{m_C}(t)$")
    plt.show()


# Call the animator and create the movie.
my_movie = animation.FuncAnimation(fig2, get_step, frames=y.shape[0], interval =
    4, blit=True)
plt.show()
```

Figure 2.1-2.2

```
#inital conditions
initial_conds = [[1,1,1,1,1,1],
                 [2,2,2,2,2,2],
                 [2,2,2,1,1,1],
                 [1,1,1,2,2,2],
                 [3,3,3,1,1,1],
                 [1,1,1,3,3,3],
                ]

for y0 in initial_conds:
    y = odeint(F, y0, t)
    #plot cm's
    plt.plot(t, y[:, 0], "--", linewidth=2, label = r"$c_{init_1}="+str(y0[0])+",
        c_{init_2}=$" +str(y0[4]))

plt.ylabel(r"$\bar c_m(t)$")
plt.xlabel(r"$\bar t$")
plt.legend()
```

```
plt.figure()
plt.title(r"1D Projection of Solution: $\bar c_{p}$ functions vs. $\bar t$,
    various $c_{init}$ values")

plt.xlim(0, 40)
for y0 in initial_conds:
    y = odeint(F, y0, t)
    #plot cp's
    plt.plot(t, y[:, 4], "--", linewidth=2, label = r"$c_{init_1}="+str(y0[0])+",
        c_{init_2}=$" +str(y0[4]))

plt.ylabel(r"$\bar c_p(t)$")
plt.xlabel(r"$\bar t$")
plt.legend()
```

Solving for fixed points with symbolic mathematics package

```
import sympy as sm

c1, c2, c3, c4, c5, c6 = sm.symbols('c1, c2, c3, c4, c5, c6')

C1 = -c1 + 50/(1+c6 ** 2)
C2 = -c2 + 50/(1+c4**2)
C3 = -c3 + 50/(1+c5 ** 2)
C4 = 0.2*(-c4+c1)
C5 = 0.2*(-c5+c2)
C6 =  0.2*(-c6+c3)

C1E = sm.Eq(C1, 0)
C2E = sm.Eq(C2, 0)
C3E = sm.Eq(C3, 0)
C4E = sm.Eq(C4, 0)
C5E = sm.Eq(C5, 0)
C6E = sm.Eq(C6, 0)

# compute fixed points
equilibria = sm.solve( (C1E, C2E, C3E, C4E, C5E, C6E), c1, c2, c3, c4, c5, c6 )
print(equilibria)
```

Figure 2.3

```
#make quiver plot
grid_c1 = np.linspace(0, 5, 3)
grid_c2 = np.linspace(0, 5, 3)
grid_c3 = np.linspace(0, 5, 3)

c1s, c2s, c3s = np.meshgrid(grid_c1, grid_c2, grid_c3)
c1dot, c2dot, c3dot = F_edit([c1s, c2s, c3s])
```

```
fig3 = plt.figure()
ax3 = plt.axes(projection='3d')
plt.quiver(c1s, c2s, c3s, c1dot, c2dot, c3dot, length=0.5, normalize=True)

#add FP
plt.plot(3.59, 3.59, 3.59, "go", label = "Fixed Point")

#add line
grid_c1 = np.linspace(0, 5, 12)
grid_c2 = np.linspace(0, 5, 12)
grid_c3 = np.linspace(0, 5, 12)
ax.view_init(30, 200)

#plotting
ax3.plot3D(grid_c1, grid_c1, grid_c1, 'r--.', alpha=0.5, label = r"$c_{m_A} =
    c_{m_B} = c_{m_C}$")
plt.title(r"3D Phase Portrait of Solution: $\bar c_m$ functions vs. $\bar t$")
ax3.set_xlabel(r"$\bar c_{m_A}(t)$")
ax3.set_ylabel(r"$\bar c_{m_B}(t)$")
ax3.set_zlabel(r"$\bar c_{m_C}(t)$")
plt.legend()
```

Figure 3.1-3.2

```
%matplotlib notebook
import pandas as pd
import numpy as np; import matplotlib.pyplot as plt;
import joypy
import scipy.stats as scp
from numpy.random import random as rng
from matplotlib import animation
from matplotlib import cm
from matplotlib.animation import FuncAnimation
from IPython import display
import math as math

"""
Four states:
     0:
     1:
     2:
     3:
8 reactions, rates all in rate_calc

0: B0-: 0 -> 3
1: B0+: 0 -> 1
2: B1-: 1 -> 0
3: B1+: 1 -> 2
4: B2-: 2 -> 1
```

```
5: B2+: 2 -> 3
6: B3-: 3 -> 2
7: B3+: 3 -> 0
"""
def rate_calc(c_ATP ,f):
        kbt = 4.07
        b_inits = np.array([0.225*c_ATP, 1.8*c_ATP, 40, 580, 1.6, 290, 40, 290])
        ds = np.array([1.07, 0.98, 1.07, 0.16, 1.07, 0.16, 3.53, 0.16])
        return b_inits*np.exp(-(f*ds/kbt))

def GDP(F, c_ATP, numtrials):
    movement = list()
    waiting_times = list()
    stoich = [[1,0,0,0],
                [1,0,0,0],
                [0,1,0,0],
                [0,1,0,0],
                [0,0,1,0],
                [0,0,1,0],
                [0,0,0,1],
                [0,0,0,1]]
    delta = [[-1,0,0,1],
                [-1,1,0,0],
                [1,-1,0,0],
                [0,-1,1,0],
                [0,1,-1,0],
                [0,0,-1,1],
                [0,0,1,-1],
                [1,0,0,-1]]

    rates = rate_calc(c_ATP, F)
    rates = rates.reshape(8,1)
    pops = np.zeros(4)
    pops[0] = 1
    # starting position
    for _ in range(numtrials):
        propens = (rates*(pops*stoich)).sum(axis=1) #propensities for each rxn
        norm = propens.sum()
        timechooser = np.random.random()
        which_event = np.random.choice(8,p=propens / norm) # determine which
            reaction will occur
        #make sure can move from state
        while (-1 in pops+delta[which_event]):
            which_event = np.random.choice(8,p=propens / norm)
        #stepping condition
        if which_event == 1:
            movement.append(8.2)
```

```
        elif which_event == 2:
            movement.append(-8.2)
        waiting_times.append(-np.log(timechooser)/norm)
        pops = pops+delta[which_event]
    return sum(movement)/sum(waiting_times) #return velocity


#plotting for ATPs
num_cs = 20
atps = np.linspace(1,5, num_cs)
GDPs = np.empty((2,num_cs))
for ind in range(0, num_cs):
    GDPs[0][ind] = GDP(1.05, atps[ind], 5000)
    GDPs[1][ind] = GDP(3.59, atps[ind], 5000)
ax = plt.gca()
ax.set_title(r"Velocity $(nm/s)$ vs $c_{ATP}$ (M)")
ax.set_xlabel(r"$c_{ATP}\,\, (M)$")
ax.set_ylabel(r"$v \,\, (nm/s)$")
plt.plot(atps, GDPs[0], label = "f = 1.05 pN")
plt.scatter(atps, GDPs[0])
plt.plot(atps, GDPs[1], label = "f = 3.59 pN")
plt.scatter(atps, GDPs[1])
plt.legend()

#plotting for forces
num_fs = 20
fs = np.linspace(1, 4.5, num_fs)
GDPs2 = np.empty(num_fs)

for x in range(0, num_fs):
    GDPs2[x] = GDP(fs[x], 5, 5000)
plt.figure()
plt.title(r"Velocity $(nm/s)$ vs $force$ (pN)")
plt.plot(fs, GDPs2)
plt.scatter(fs, GDPs2, label = r"$c_{ATP} = 5 \mu M$")
plt.xlabel(r"$force\,\, (pN)$")
plt.ylabel(r"$v \,\, (nm/s)$")
plt.legend()
```