EE430 – Assignment 2
Behiye Erdemir
240206013

1. Euler Method with Taylor Series

Euler method is a numerical solution to find solutions of ordinary differential equations. For computer operations, it is possible to perform the difference process instead of differentiate. Euler's method, which is an approximate solution, is used for this [1].

Taylor Series:

$$x(t+\Delta t) = x(t) + \Delta t x'(t) + \frac{1}{2}\Delta t^2 x''(t) + \frac{1}{3!}\Delta t^3 x'''(t) + \cdots$$

In this method , the Taylor series is used to estimate solutions. The Euler Method can be considered a Taylor Method of order one. The approximation used for Euler's method is to take the first two terms of the Taylor Seris:

$$x(t+\Delta t) = x(t) + \Delta t x'(t)$$

If $\Delta t x'(t)$ expression replaced with the below expression, a discrete process can be achieved.

$$\frac{dx}{dt} = f(t,x)$$

$$x(t+\Delta t) = x(t) + \Delta t f(t, x(t))$$

For $t > 0$ given the initial condition $x(0) = x_0$.

$$x'(0) = f(0, x_o)$$

$$x(\Delta t) = x_0 + \Delta t * f(0, x_0)$$

The result approaches its real value as Δt gets smaller.

2. Calculation For The Concentrations

Firstly, a function was wroten to calculate the transient responses of the concentrations of Y(t) and Y_{mRNA} by using Euler method. With the function, the concentration of inputs is dynamically calculated for system parameters such as time, gene number, time parameters.

```
# to calculate the concentration of genes
def conc_act(t0, t_end, dt,betas, alphas,nodes, kappa, n_2gene, t_ext, t_act):
    # number of interval according to time variables
    N = int((t_end - t0) / dt + 1)
    # a node matrice, initially zero
    node_matrice = np.zeros((N, len(nodes)))
    # a zero array for initial concentration
    c_nodes = [0] * len(nodes)
    # initialization the array that indicates the change of nodes over time
    dN_dt = [0] * len(nodes)
    # time array
    t_ax = np.arange(t0, t_end + dt, dt)
    # calculates the new concentration according to time
    for index, t in enumerate(t_ax):
        # change of transcription factor over time
        dN_dt[0] = betas[0] * ((t >= t_act[0]) & (t <= t_ext[0])).astype(float) - alphas[0] *
c_nodes[0]
        #for more than one node
        if len(nodes) > 1:
            # calculates concentrations of other nodes
            for i in range(len(nodes) - 1):
                # initial variable for next genes
                new_n=[0] * (len(nodes) - 1)
                # formula of the instant concentration of gene according to activator
                dN_dt[i + 1] = betas[i + 1] * (((t >= t_act[i + 1]) &
(t<=t_ext[i+1])).astype(float)
                                * c_nodes[i] ** n_2gene) / (kappa ** n_2gene + ((t >= t_act[i
+ 1])
                                & (t <= t_ext[i + 1])).astype(float) * c_nodes[i] ** n_2gene)
- alphas[i + 1] * c_nodes[i + 1]
                # total concentration
                new_n[i] = c_nodes[i + 1] + dN_dt[i + 1] * dt
                # assign the new value to the output matrice
                node_matrice[:,i + 1][index] = new_n[i]
                # to calculate cumulatively
                c_nodes[i + 1] = new_n[i]
        # new concentration valur for activator
        new_act = c_nodes[0] + dN_dt[0] * dt
        # assign the new value to the output matrice
        node_matrice[:,0][index] = new_act
        # to calculate cumulatively
        c_nodes[0] = new_act
    #gives neew concentration values as outputs
    return node_matrice
```

After the necessary libraries are added, values for alpha and beta given within the scope of homework have been initialized.

```
# necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# for 1 < alpha, beta_prime, beta_mRNA < 2
alpha = np.random.rand() + 1.0
beta_prime = np.random.rand() + 1.0
beta_mRNA = np.random.rand() + 1.0
# for alpha_mRNA >> alpha, 20 is selected randomly
alpha_mRNA = 20.0 * np.random.rand()
beta=beta_prime * beta_mRNA / alpha_mRNA
```

a) Calculations are made for the transient responses of the concentrations Y(t) and Y_{mRNA} by using the given parameters.

```
nodes = ['YmRNA', 'Y']                  # e.g. YmRNA->Y
betas = [beta_mRNA, beta_prime]         # Values of beta in node order
alphas = [alpha_mRNA, alpha]            # Values of alpha in node order
kappa=0.5                               # kappa value
n_2gene=10                              # n order

t0 = -1                                 # time before activation
t_end = 5                               # end time
dt = 0.01                               # time interval
t_act=[0,0]                             # activation time
t_ext=[t_end,t_end]                     # for extra time interventions

# to calculate the concentrations
outs= conc_act(t0,t_end,dt,betas,alphas,nodes,kappa,n_2gene,t_ext,t_act)
```

The plots of the transient response curves can be seen in Figure 1.

```
# number of interval according to time variables
N = int((t_end - t0) / dt +1)
# figure size according to number of figures
fig = plt.figure(figsize=(20,len(outs[1])*6))

# loop for creating figures dinamically
for j in range(len(outs[1])):
    # create figures according to number of nodes
    ax = fig.add_subplot(len(outs[1]),1,j+1)
    # plot(t,nodes)
    ax.plot(np.linspace(t0, t_end, N),outs[:,j],'b',label=nodes[j])
    # for steady state
    if j==0:
        ax.axhline(y=beta_mRNA/alpha_mRNA, color='r', linestyle='--', label='Steady State for
Y_mRNA')
    #legend position
    ax.legend(loc="lower right")
    # add x label
    ax.set_xlabel('t')
    # add y label
    ax.set_ylabel('Concentration of {}'.format(nodes[j]))
    # grid on
    ax.grid()
```

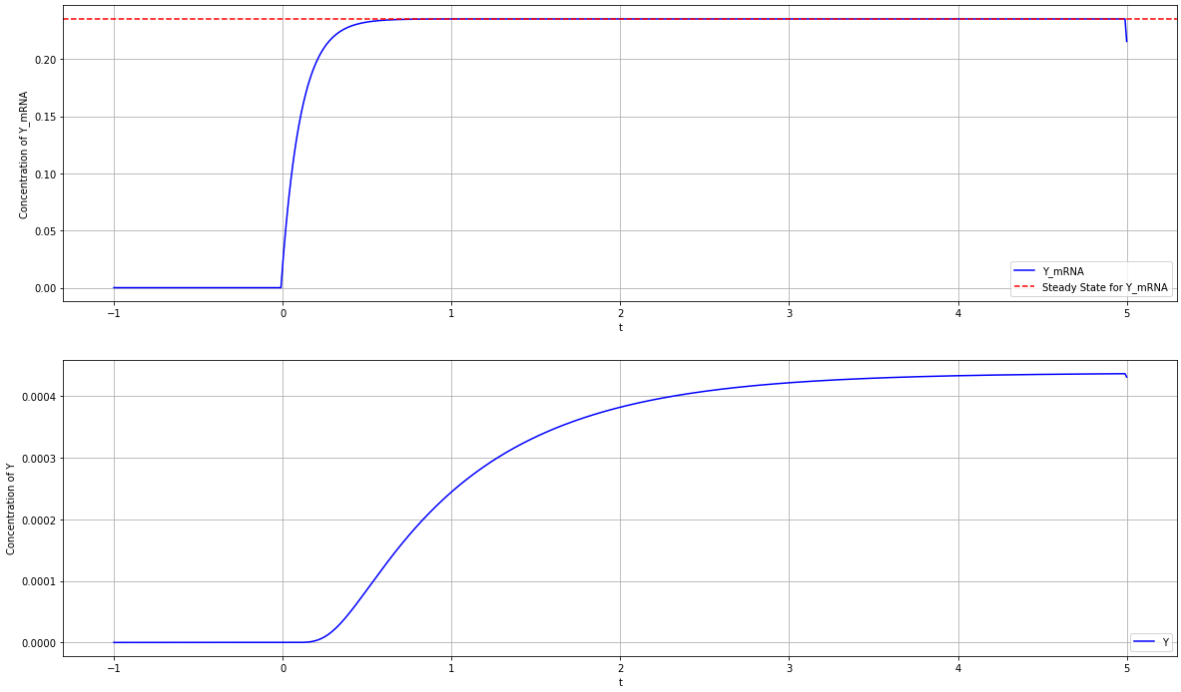

Figure 1: Curves of transient responses of Y(t) and YmRNA(t)

At the steady state of Y_{mRNA} ;

$$\frac{d([Y_{mRNA}])(t)}{dt} = 0 \Rightarrow [Y_{mRNA}]_{st} = \frac{\beta_{mRNA}}{\alpha_{mRNA}}$$

As it can be seen in Figure 1, the point where YmRNA reaches the steady-state has been checked and it has been seen that it has provided the formula. Besides, because the alpha value of the mRNA is much greater than the alpha value of Y, YmRNA reaches a steady-state much faster than Y.

b) For the maximal production rate β of the protein Y is determined as;

$$\beta = \frac{\beta' \beta_{mRNA}}{\alpha_{mRNA}}$$

The curve of the Y protein concentration calculated ignoring the mRNA interactions is as shown in Figure 2.

```
nodes = ['Y']
betas = [beta]          # Values of beta in node order
alphas = [alpha]        # Values of alpha in node order

t0 = -1                 # time before activation
t_end = 5               # end time
dt = 0.01               # time interval
t_act=[0]               # activation time
t_ext=[t_end]           # for extra time interventions

# to calculate the concentrations
outs2= conc_act(t0,t_end,dt,betas,alphas,nodes,kappa,n_2gene,t_ext,t_act)

# number of interval according to time variables
N = int((t_end - t0) / dt +1)
# figure size according to number of figures
fig = plt.figure(figsize=(20,len(outs2[1])*6))

# loop for creating figures dinamically
for j in range(len(outs2[1])):
    # create figures according to number of nodes
    ax = fig.add_subplot(len(outs2[1]),1,j+1)
    # plot(t,nodes)
    ax.plot(np.linspace(t0, t_end, N),outs2[:,j],'b',label=nodes[j])
    ax.legend(loc="upper left")
    # add x label
    ax.set_xlabel('t')
    # add y label
    ax.set_ylabel('Concentration of {}'.format(nodes[j]))
    # grid on
    ax.grid()
```

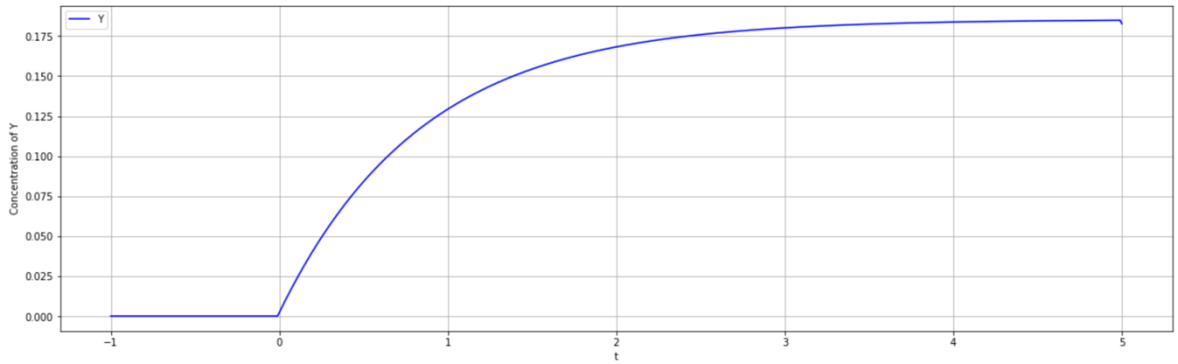

Figure 2: Concentration of Y protein

c) The curves of the two different Y are as shown in Figure 3. On a linear scale, because of the huge difference, it's difficult to compare. On a logarithmic scale, a concentration comparison of Y found in different situations can be made. With the maximal production rate β, In the second case, more concentration of Y protein is produced.

```
# number of interval according to time variables
N = int((t_end - t0) / dt +1)
# figure size
fig = plt.figure(figsize=(20,6))

# create figures according to number of nodes
ax = fig.add_subplot(len(outs2[1]),1,1)
# plot(t,nodes)
ax.plot(np.linspace(t0, t_end, N),outs[:,1],'--g', label='Y')
ax.plot(np.linspace(t0, t_end, N),outs2[:,0],'r',label='Maximal Y')
ax.legend(loc="upper left")
# add x label
ax.set_xlabel('t')
#log scale
ax.set_yscale('log')
# add y label
ax.set_ylabel('Concentration of {}'.format(nodes[j]))
# grid on
ax.grid()
```

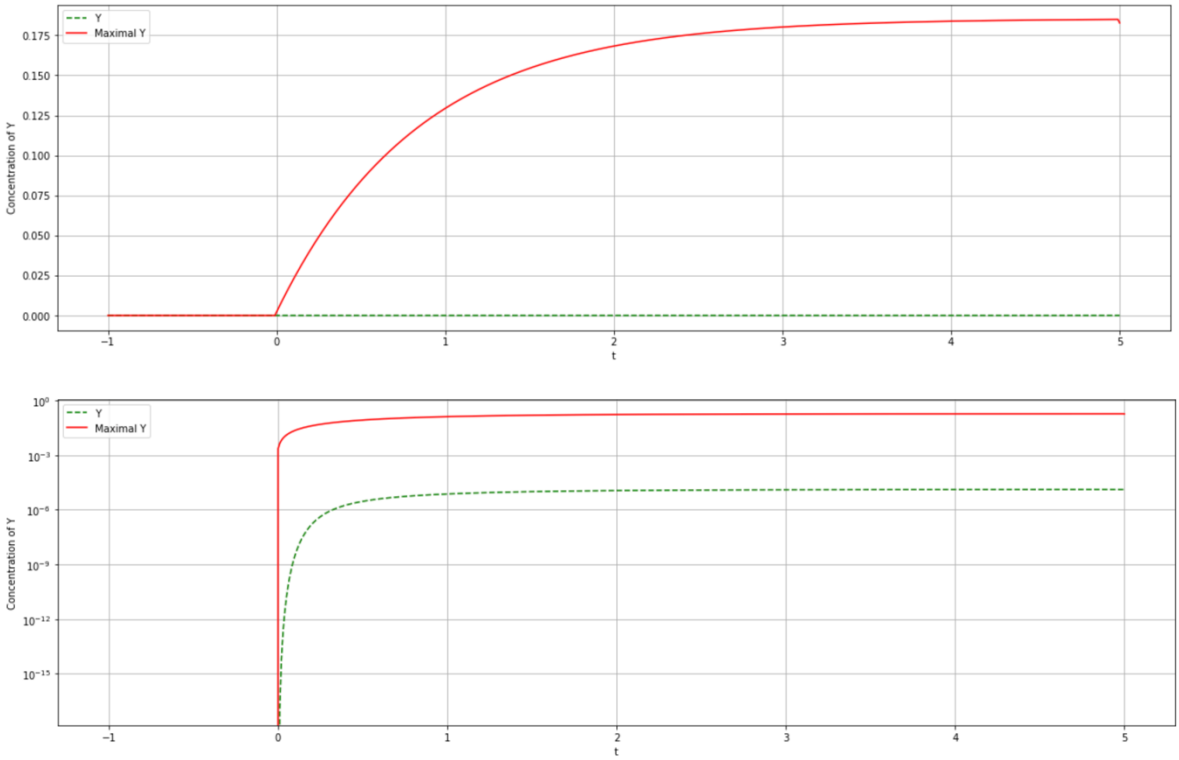

Figure 3: Comparision of Y in linear and logarithmic scale

The whole code can be accessed in Colab Notebook[1].

[1] https://colab.research.google.com/drive/1nofYV6D8Tw3DJayc8-BRetnSNFmJZdBI?usp=sharing

References

[1] S. Gottlieb, "Euler's Method, Taylor Series Method, Runge Kutta Methods, Multi-Step Methods and Stability.".