

Assignment5

Fernando Freire

April 30, 2019

Contents

1	The central dogma	2
1.1	Equations	2
1.2	Numerical simulation first model	2
1.3	Hill function for activators	5
1.4	Hill function for represors	9
1.5	Possitive feedback	12
1.6	Negative feedback	15

1 The central dogma

1.1 Equations

1.2 Numerical simulation first model

Script 1.2.1 (python)

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.integrate import odeint
4
5 # Initial conditions
6 M_0 = 0
7 P_0 = 0
8 Km = 1
9 Kp = 1
10 dm = 1
11 dp = 1
12 a = 0.5
13 T_0 = 1
14 D_0 = 1
15
16 total_time = 10
17 dt = 0.05
```

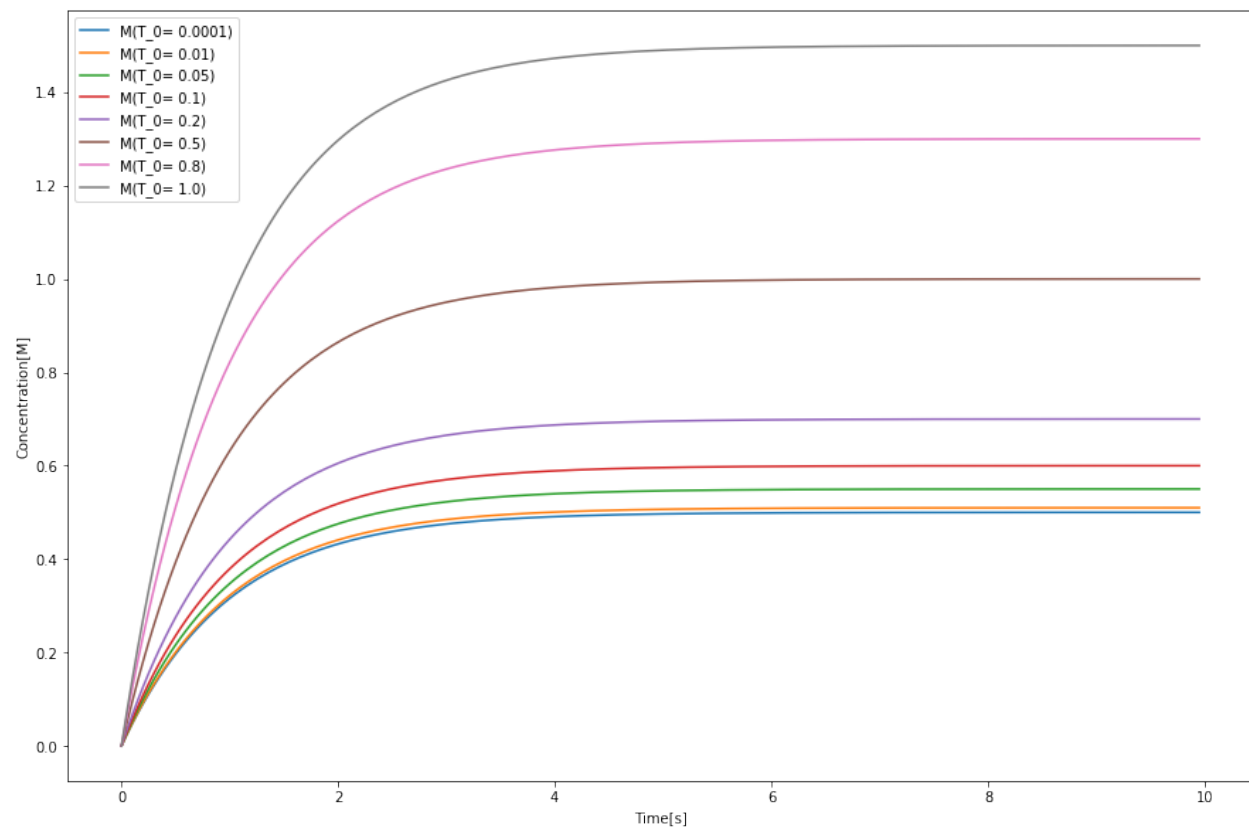
Script 1.2.2 (python)

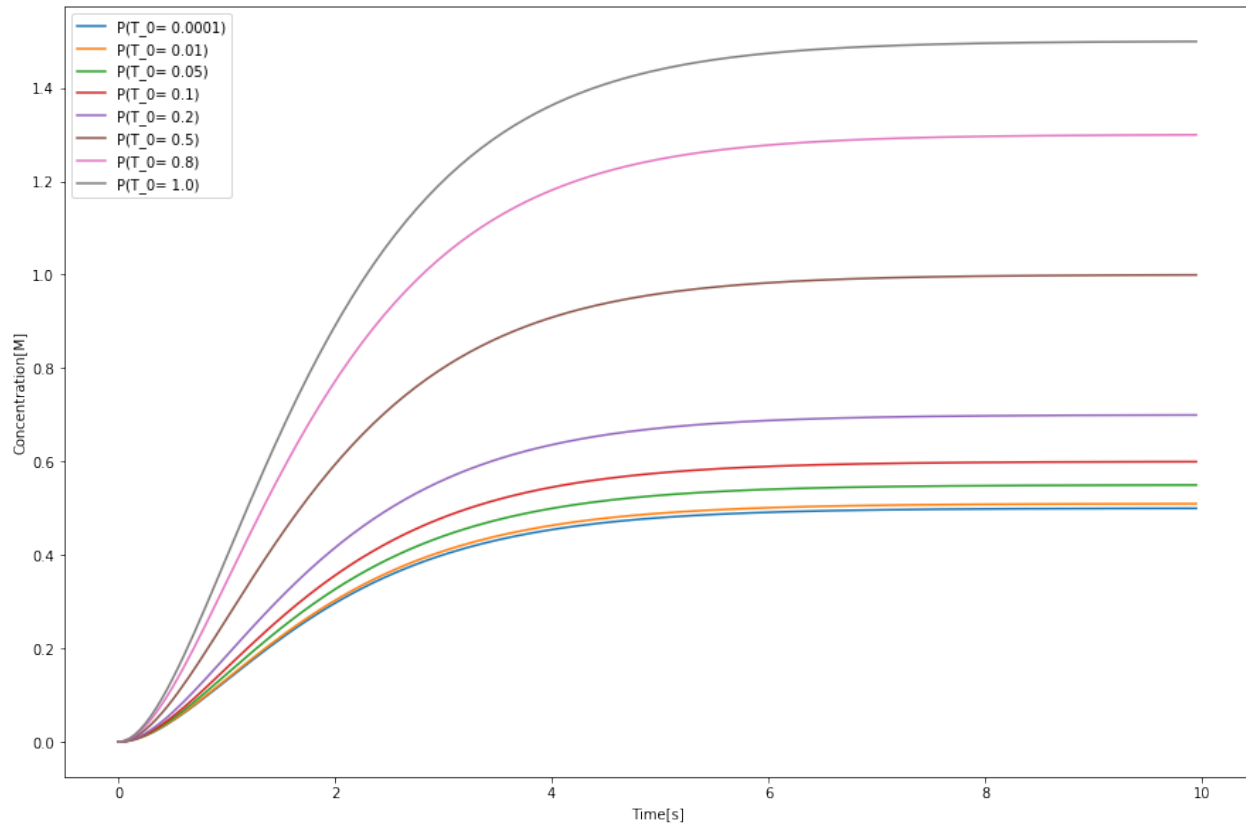
```
1 # M = mRNA
2 # D = DNA
3 # P = protein
4 # T = transcription factor
5 # Km = rate of mRNA production
6 # Kp = rate of protein production
7 # dm = rate of mRNA degradation
8 # dp = rate of protein degradation
9 # a = promoter leaking
10
11 # T + D -Km-> M + T + D, T is not consumed and nor DNA
12 # M -Kp-> P + M, not being consumed
13 # M -dm-> 0
14 # P -dp-> 0
15 # 0 -a-> M
16
17 def modelD(y, t, Km, Kp, dm, dp, a, T_0, D_0):
18     [M, P] = y
19     dMdt = Km*T_0*D_0 - dm*M + a
20     dPdt = Kp*M - dp*P
21
22     dydt = [dMdt, dPdt]
23     return dydt
24
```

```

25 def plot_modelD(y, t, Km, Kp, dm, dp, a, T_0, D_0, n, plot_M=True, plot_P=True):
26     # initial condition
27     y0 = [M_0, P_0]
28
29     # time points
30     t = np.arange(0, total_time, dt)
31
32     # solve ODE
33     y = odeint(modelD, y0, t, args=(Km, Kp, dm, dp, a, T_0, D_0))
34
35
36     # plot results
37     if plot_M: plt.plot(t, y[:,0], label = "M(T_0= " + str(T_0) + ")")
38     if plot_P: plt.plot(t, y[:,1], label = "P(T_0= " + str(T_0) + ")")
39
40     plt.xlabel("Time[s]")
41     plt.ylabel("Concentration[M]")
42     plt.legend(loc = "best")
43
44     plt.figure(figsize=(15,10))
45     for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]:
46         plot_model(y, t, Km, Kp, dm, dp, a, T_0, D_0, n, plot_P=False)
47
48     plt.show()
49
50     plt.figure(figsize=(15,10))
51     for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0]:
52         plot_model(y, t, Km, Kp, dm, dp, a, T_0, D_0, n, plot_M=False)
53
54     plt.show()

```





1.3 Hill function for activators

New activation equation for D_i to D_a (see python comments)

Script 1.3.1 (python)

```

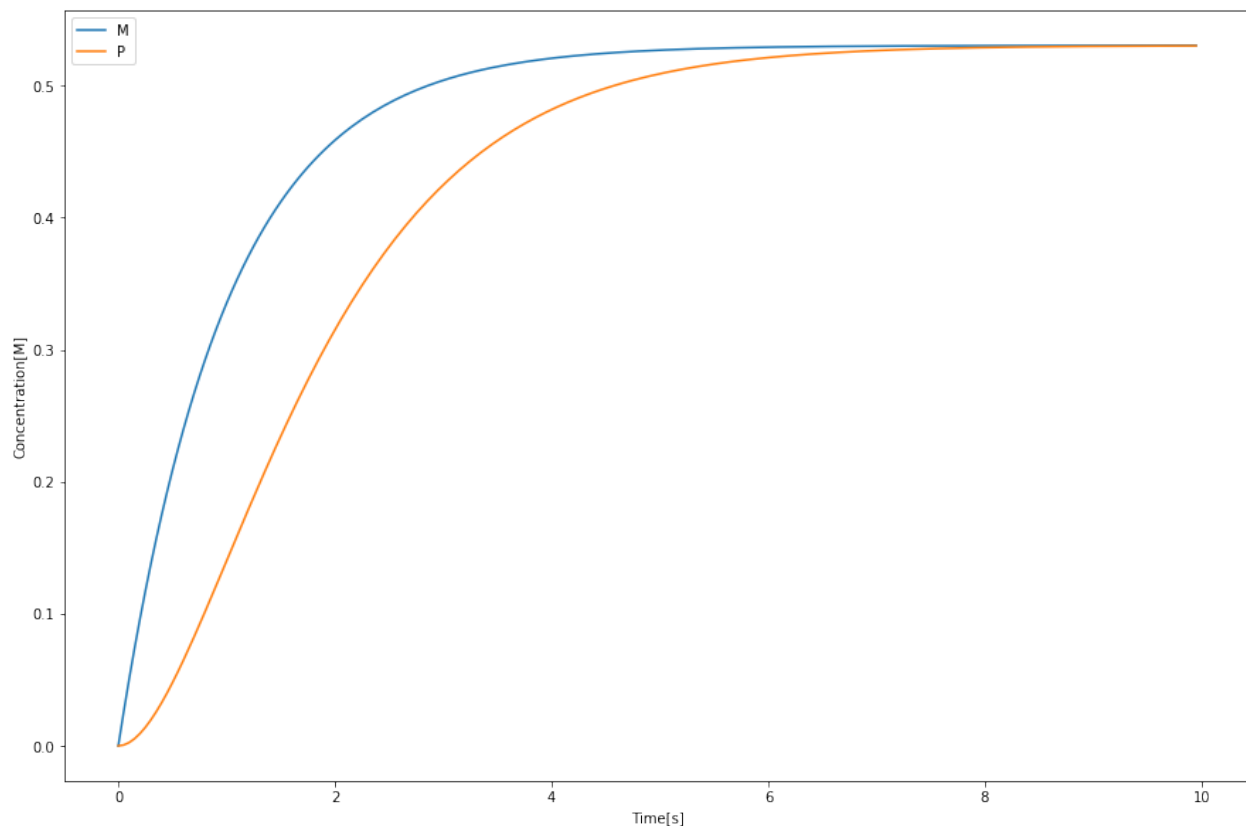
1  # M = mRNA
2  # D = DNA
3  # P = protein
4  # T = transcription factor
5  # Km = rate of mRNA production
6  # Kp = rate of protein production
7  # dm = rate of mRNA degradation
8  # dp = rate of protein degradation
9  # a = promoter leaking
10
11 # T + D -Km-> M + T + D, T is not consumed and nor DNA
12 # M -Kp-> P + M, not being consumed
13 # M -dm-> 0
14 # P -dp-> 0
15 # 0 -a-> M
16
17 # Activation of DNA with n T

```

```

18 #  $Di + nT \xrightarrow{-K1/K2} K1 * Di * T^{**n} = K2 * Da \Rightarrow Di * T^{**n} = Kd * Di$ , where  $Kd = K1/K2$ 
19 #  $Dt = Di + Da$ 
20 # so
21 #  $Da/Dt = T^{**n}/(Kd(1+Da/Di))$ 
22
23 #  $Da/Dt = T^{**n}/(K^{**n} + T^{**n})$ 
24
25 K = 2
26 n = 5
27 def hill(T, K, n):
28     return T**n/(K**n + T**n)
29
30 def modelDH(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n):
31     [M, P] = y
32     dMdt = Km*hill(T_0, K, n)*D_0 - dm*M + a
33     dPdt = Kp*M - dp*P
34
35     dydt = [dMdt, dPdt]
36     return dydt
37
38 # initial condition
39 y0 = [M_0, P_0]
40
41 # time points
42 t = np.arange(0, total_time, dt)
43
44 # solve ODE
45 y = odeint(modelDH, y0, t, args=(Km, Kp, dm, dp, a, T_0, D_0, K, n))
46
47 plt.figure(figsize=(15,10))
48
49 # plot results
50 plt.plot(t, y[:,0], label = "M")
51 plt.plot(t, y[:,1], label = "P")
52
53 plt.xlabel("Time[s] ")
54 plt.ylabel("Concentration[M] ")
55 plt.legend(loc = "best")
56 plt.show()

```



Script 1.3.2 (python)

```

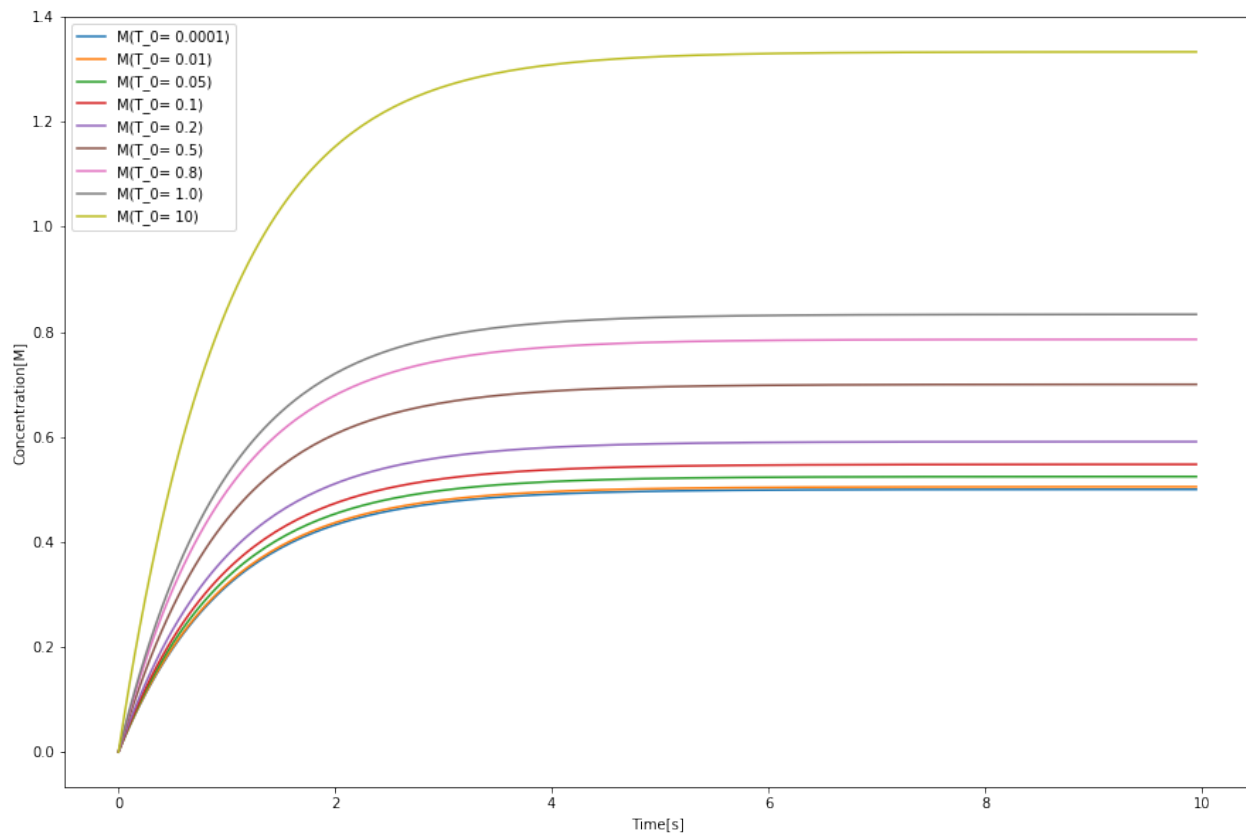
1 n = 1
2 def plot_modelDH(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_M=True, plot_P=True):
3     # initial condition
4     y0 = [M_0, P_0]
5
6     # time points
7     t = np.arange(0, total_time, dt)
8
9     # solve ODE
10    y = odeint(modelDH, y0, t, args=(Km, Kp, dm, dp, a, T_0, D_0, K, n))
11
12
13    # plot results
14    if plot_M: plt.plot(t, y[:,0], label = "M(T_0= " + str(T_0) + ")")
15    if plot_P: plt.plot(t, y[:,1], label = "P(T_0= " + str(T_0) + ")")
16
17    plt.xlabel("Time[s]")
18    plt.ylabel("Concentration[M]")
19    plt.legend(loc = "best")
20
21    plt.figure(figsize=(15,10))

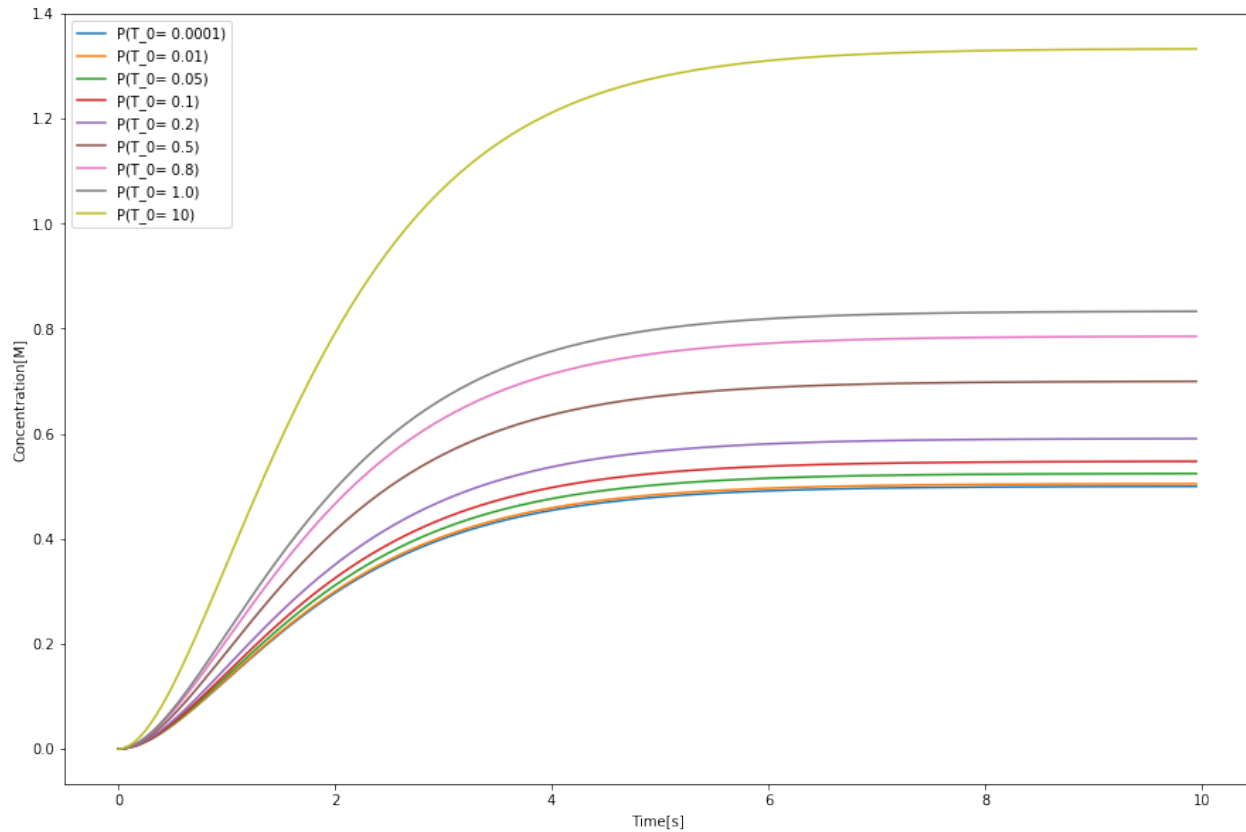
```

```

22 for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0, 10]:
23     plot_modelDH(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_P=False)
24
25 plt.show()
26
27 plt.figure(figsize=(15,10))
28 for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0, 10]:
29     plot_modelDH(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_M=False)
30
31 plt.show()

```





1.4 Hill function for represors

New repression equation for D_a to D_i (see python comments)

Script 1.4.1 (python)

```

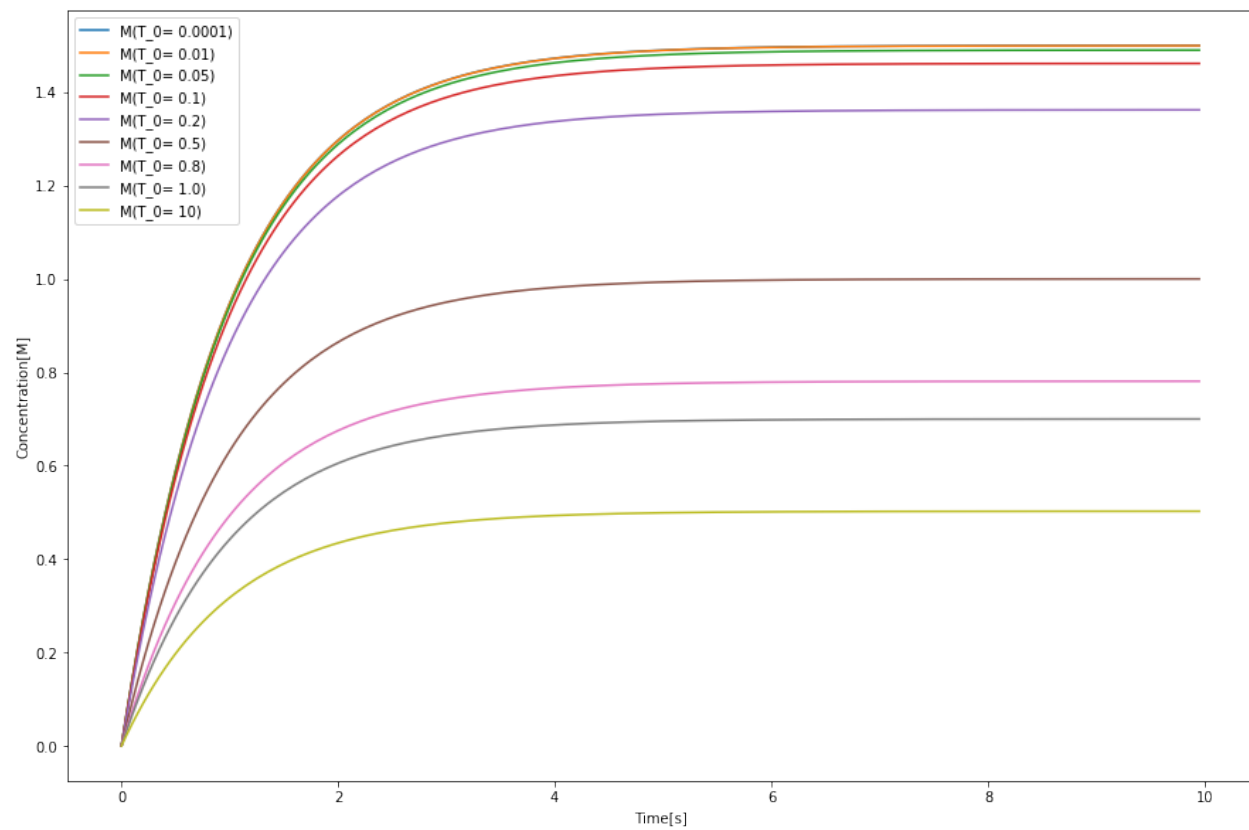
1  # M = mRNA
2  # D = DNA
3  # P = protein
4  # T = transcription factor
5  # Km = rate of mRNA production
6  # Kp = rate of protein production
7  # dm = rate of mRNA degradation
8  # dp = rate of protein degradation
9  # a = promoter leaking
10
11 # T + D -Km-> M + T + D, T is not consumed and nor DNA
12 # M -Kp-> P + M, not being consumed
13 # M -dm-> 0
14 # P -dp-> 0
15 # 0 -a-> M
16
17 # Da + nT <-K1/K2-> Di

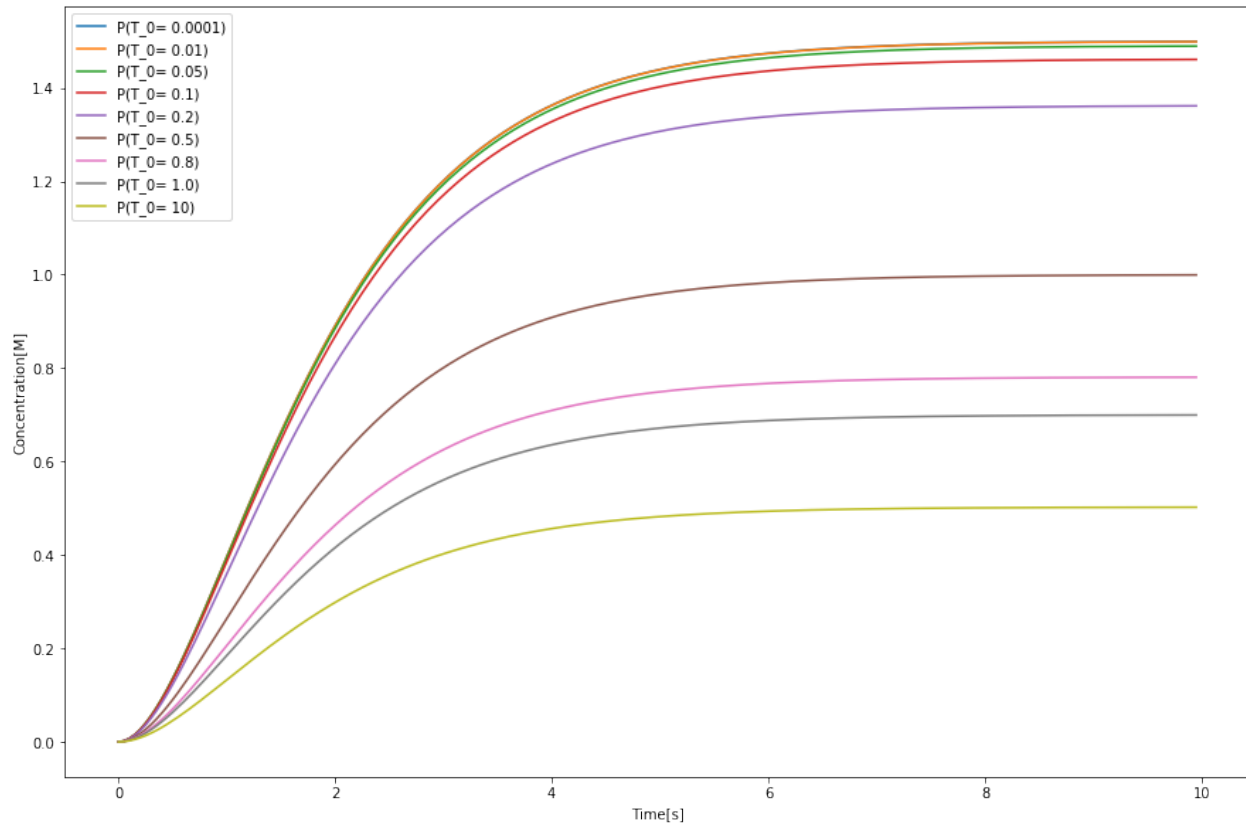
```

```

18 # implies  $Da/Di = K**n/(T**n + K**n)$ 
19
20 def hill_repress(T, K, n):
21     return K**n/(K**n + T**n)
22
23 def modelDHR(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n):
24     [M, P] = y
25     dMdt = Km*hill_repress(T_0, K, n)*D_0 - dm*M + a
26     dPdt = Kp*M - dp*P
27
28     dydt = [dMdt, dPdt]
29     return dydt
30
31 n = 2
32 K = 0.5
33 def plot_modelDHR(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_M=True, plot_P=True):
34     # initial condition
35     y0 = [M_0, P_0]
36
37     # time points
38     t = np.arange(0, total_time, dt)
39
40     # solve ODE
41     y = odeint(modelDHR, y0, t, args=(Km, Kp, dm, dp, a, T_0, D_0, K, n))
42
43
44     # plot results
45     if plot_M: plt.plot(t, y[:,0], label = "M(T_0= " + str(T_0) + ")")
46     if plot_P: plt.plot(t, y[:,1], label = "P(T_0= " + str(T_0) + ")")
47
48     plt.xlabel("Time[s]")
49     plt.ylabel("Concentration[M]")
50     plt.legend(loc = "best")
51
52
53 plt.figure(figsize=(15,10))
54 for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0, 10]:
55     plot_modelDHR(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_P=False)
56
57 plt.show()
58
59 plt.figure(figsize=(15,10))
60 for T_0 in [0.0001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0, 10]:
61     plot_modelDHR(y, t, Km, Kp, dm, dp, a, T_0, D_0, K, n, plot_M=False)
62
63 plt.show()

```





1.5 Positive feedback

Script 1.5.1 (python)

```

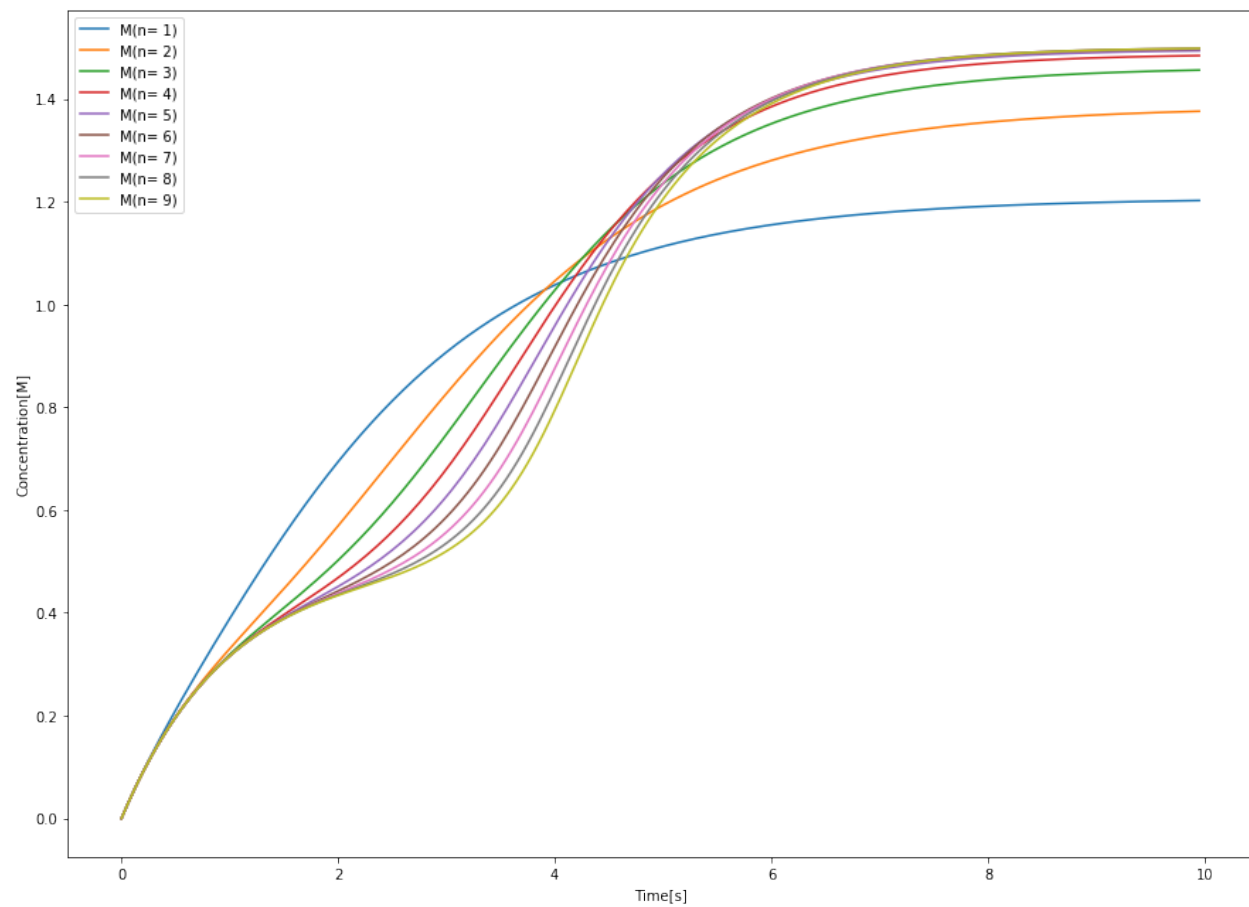
1  # M = mRNA
2  # D = DNA
3  # P = protein
4  # T = transcription factor
5  # Km = rate of mRNA production
6  # Kp = rate of protein production
7  # dm = rate of mRNA degradation
8  # dp = rate of protein degradation
9  # a = promoter leaking
10
11 # P + D -Km-> M + D, D is not consumed and nor DNA
12 # M -Kp-> P + M, not being consumed
13 # M -dm-> 0
14 # P -dp-> 0
15 # 0 -a-> M
16
17 # Da + nT <-K1/K2-> Di
18 # implies Da/Di = K**d/(T**n + K**n)

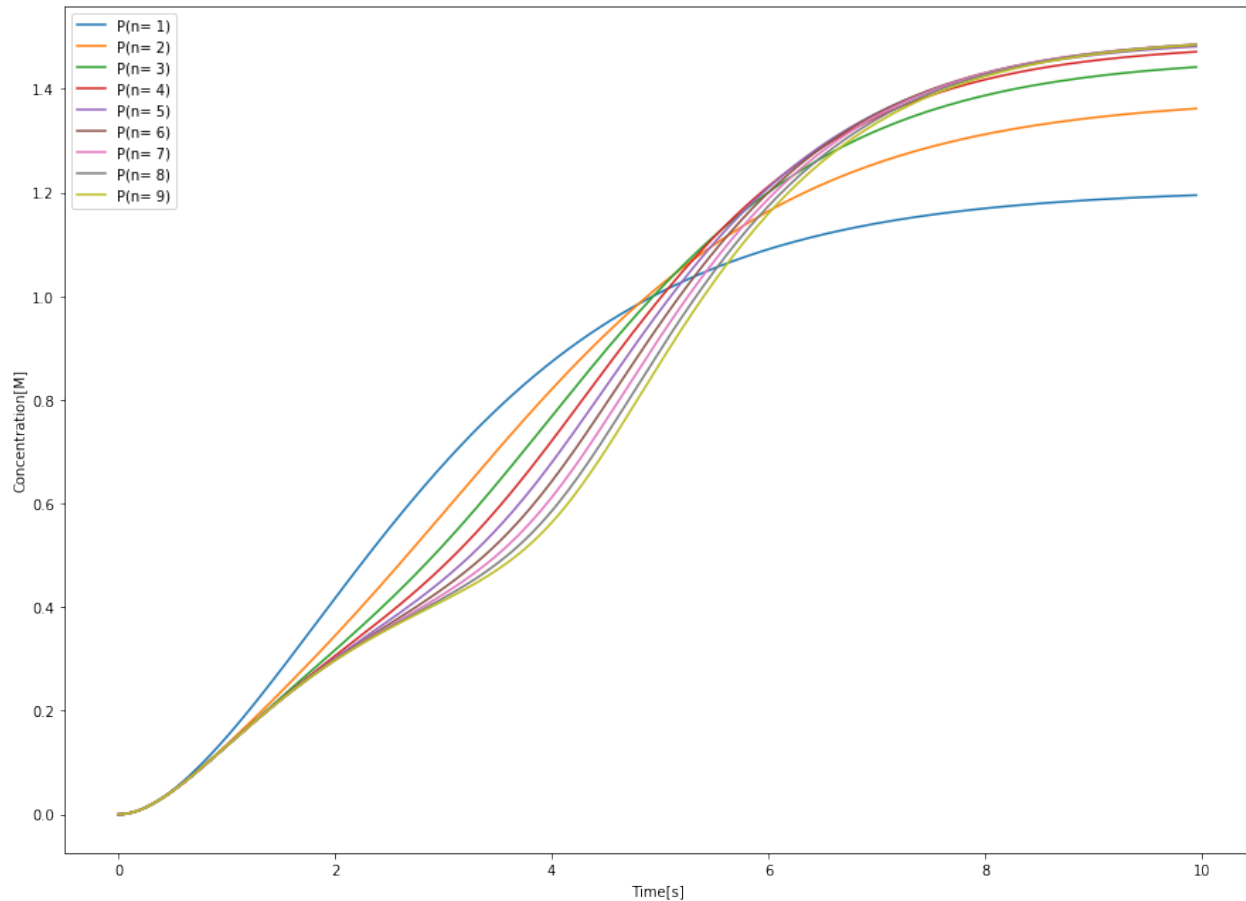
```

```

19
20 n = 2
21 K = 1
22
23 def hill(T, K, n):
24     return T**n/(K**n + T**n)
25
26 def modelP(y, t, Km, Kp, dm, dp, a, D_0, K, n):
27     [M, P] = y
28     dMdt = Km*hill(P, K, n)*D_0 - dm*M + a
29     dPdt = Kp*M - dp*P
30
31     dydt = [dMdt, dPdt]
32     return dydt
33
34 K = 0.5
35 def plot_modelP(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_M=True, plot_P=True):
36     # initial condition
37     y0 = [M_0, P_0]
38
39     # time points
40     t = np.arange(0, total_time, dt)
41
42     # solve ODE
43     y = odeint(modelP, y0, t, args=(Km, Kp, dm, dp, a, D_0, K, n))
44
45
46     # plot results
47     if plot_M: plt.plot(t, y[:,0], label = "M(n= " + str(n) + ")")
48     if plot_P: plt.plot(t, y[:,1], label = "P(n= " + str(n) + ")")
49
50     plt.xlabel("Time[s]")
51     plt.ylabel("Concentration[M]")
52     plt.legend(loc = "best")
53
54 plt.figure(figsize=(15,11))
55 for n in range(1,10):
56     plot_modelP(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_P=False)
57
58 plt.show()
59
60 plt.figure(figsize=(15,11))
61 for n in range(1,10):
62     plot_modelP(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_M=False)
63
64 plt.show()

```





1.6 Negative feedback

Script 1.6.1 (python)

```

1  # M = mRNA
2  # D = DNA
3  # P = protein
4  # T = transcription factor
5  # Km = rate of mRNA production
6  # Kp = rate of protein production
7  # dm = rate of mRNA degradation
8  # dp = rate of protein degradation
9  # a = promoter leaking
10
11 # P + D -Km-> M + D, D is not consumed and nor DNA
12 # M -Kp-> P + M, not being consumed
13 # M -dm-> 0
14 # P -dp-> 0
15 # 0 -a-> M
16

```

```

17 #  $Da + nT \xrightarrow{-K1/K2} Di$ 
18 # implies  $Da/Di = K**d/(T**n + K**n)$ 
19
20 n = 2
21 K = 1
22
23 def hill_repress(T, K, n):
24     return K**n/(K**n + T**n)
25
26 def modelN(y, t, Km, Kp, dm, dp, a, D_0, K, n):
27     [M, P] = y
28     dMdt = Km*hill_repress(P, K, n)*D_0 - dm*M + a
29     dPdt = Kp*M - dp*P
30
31     dydt = [dMdt, dPdt]
32     return dydt
33
34 K = 0.5
35 def plot_modelN(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_M=True, plot_P=True):
36     # initial condition
37     y0 = [M_0, P_0]
38
39     # time points
40     t = np.arange(0, total_time, dt)
41
42     # solve ODE
43     y = odeint(modelN, y0, t, args=(Km, Kp, dm, dp, a, D_0, K, n))
44
45     # plot results
46     if plot_M: plt.plot(t, y[:,0], label = "M(n= " + str(n) + ")")
47     if plot_P: plt.plot(t, y[:,1], label = "P(n= " + str(n) + ")")
48
49     plt.xlabel("Time[s]")
50     plt.ylabel("Concentration[M]")
51     plt.legend(loc = "best")
52
53     plt.figure(figsize=(15,11))
54     for n in range(1,10):
55         plot_modelN(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_P=False)
56
57     plt.show()
58
59     plt.figure(figsize=(15,11))
60     for n in range(1,10):
61         plot_modelN(y, t, Km, Kp, dm, dp, a, D_0, K, n, plot_M=False)
62
63     plt.show()
64

```