

# Assignment4

Fernando Freire

April 29, 2019

## Contents

<b>1</b>	<b>Two enzyme kinetics: kinase-phosphatase</b>	<b>2</b>
1.1	Equations . . . . .	2
1.2	Numerical simulation <i>modelkp_6</i> . . . . .	2
1.3	Conservation of mass . . . . .	4
1.4	Numerical simulation <i>modelkp_4</i> . . . . .	4
1.5	Second constraint: Michaelis-Menten . . . . .	6
1.6	Numerical simulation <i>modelkp_2</i> . . . . .	6
1.7	Find parameters compatible with Michaelis Merten . . . . .	8

# 1 Two enzyme kinetics: kinase-phosphatase

## 1.1 Equations

We obtain the equations as follows.

1. Calculate the six velocities of reaction:

$$\begin{aligned} & k_1 * s * e1 \\ & k_2 * e1s \\ & k_3 * e1s \\ & k_4 * p * e2 \\ & k_5 * e2p \\ & k_6 * e2p \end{aligned}$$

2. Assign derivatives with his sign: minus if arrow pointing from the molecule, plus if arrow pointing to molecule

$$\begin{aligned} k_1 * s * e1 &\rightarrow \{-ds/dt, -de1/dt, de1s/dt\} \\ k_2 * e1s &\rightarrow \{ds/dt, de1/dt, -de1s/dt\} \\ k_3 * e1s &\rightarrow \{-de1s/dt, de1/dt, dp/dt\} \\ k_4 * p * e2 &\rightarrow \{-dp/dt, -de2/dt, de2p/dt\} \\ k_5 * e2p &\rightarrow \{dp/dt, de2/dt, -de2p/dt\} \\ k_6 * e2p &\rightarrow \{dp/dt, de2/dt, -de2p/dt\} \end{aligned}$$

3. Reverse the assignments, from derivatives to reaction rates, adding in each diferencial equation, the rates with their sign. With this approach we construct the equations writing directly on method *modelkp\_6*.

```
dsdt = -k_1*s*e1 + k_2 * e1s + k_6 * e2p
de1dt = -k_1*s*e1 + k_2 * e1s + k_3 * e1s
de2dt = -k_4 * p * e2 + k_5 * e2p + k_6 * e2p
de1sdt = -k_3 * e1s + k_1*s*e1 - k_2 * e1s
de2pdt = -k_5 * e2p + k_4*p*e2 - k_6 * e2p
dpdt = k_3 * e1s - k_4*p*e2 + k_5 * e2p
```

## 1.2 Numerical simulation *modelkp\_6*

### 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 k_1 = 500
7 k_2 = 0.15
8 k_3 = 0.03
9 k_4 = 1000
10 k_5 = 0.1
11 k_6 = 0.01
12 e1_0 = 0.002
```

```

13 e2_0 = 0.002
14 s_0 = 0.002
15 p_0 = 0.001
16 e2p_0 = 0
17 e1s_0 = 0
18 total_time = 100
19 dt = 0.1

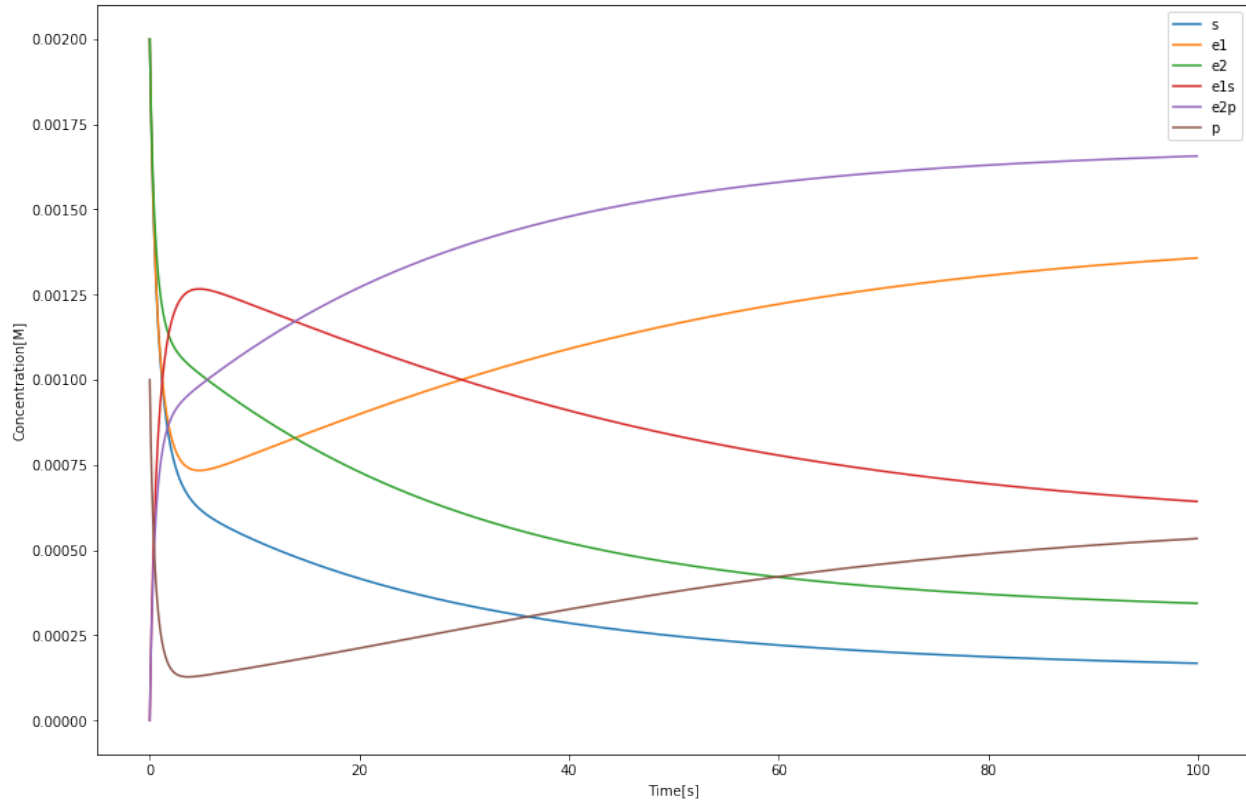
```

### Script 1.2.2 (python)

```

1 def modelkp_6(y, t, k_1, k_2, k_3, k_4, k_5, k_6):
2     [s, e1, e2, e1s, e2p, p] = y
3     dsdt = -k_1*s*e1 + k_2 * e1s + k_6 * e2p
4     de1dt = -k_1*s*e1 + k_2 * e1s + k_3 * e1s
5     de2dt = -k_4 * p * e2 + k_5 * e2p + k_6 * e2p
6     de1sdt = -k_3 * e1s + k_1*s*e1 - k_2 * e1s
7     de2pdt = -k_5 * e2p + k_4*p*e2 - k_6 * e2p
8     dpdt = k_3 * e1s - k_4*p*e2 + k_5 * e2p
9
10    dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
11    return dydt
12
13    # initial condition
14    y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]
15
16    # time points
17    t = np.arange(0, total_time, dt)
18
19    # solve ODE
20    y = odeint(modelkp_6, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6))
21
22    plt.figure(figsize=(15,10))
23
24    # plot results
25    plt.plot(t, y[:,0], label = "s")
26    plt.plot(t, y[:,1], label = "e1")
27    plt.plot(t, y[:,2], label = "e2")
28    plt.plot(t, y[:,3], label = "e1s")
29    plt.plot(t, y[:,4], label = "e2p")
30    plt.plot(t, y[:,5], label = "p")
31
32    plt.xlabel("Time[s] ")
33    plt.ylabel("Concentration[M] ")
34    plt.legend(loc = "best")
35    plt.show()

```



### 1.3 Conservation of mass

Concentrations of enzymes  $e1$  and  $e2$  remain constant:

$$e1 = e1_0 - e1s$$

$$e2 = e2_0 - e2p$$

We write directly on the model method *modelkp\_4* both constraints, equating to zero the derivatives.

### 1.4 Numerical simulation *modelkp\_4*

Script 1.4.1 (python)

```

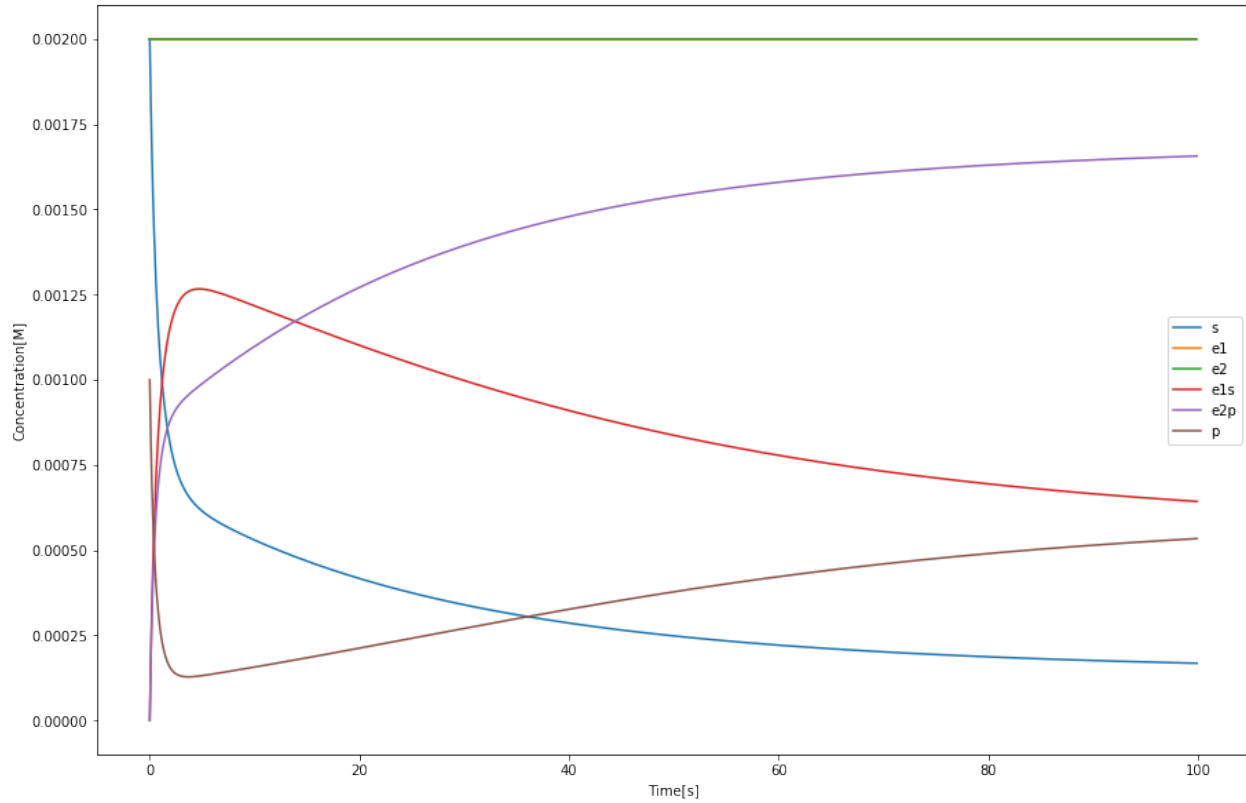
1 def modelkp_4(y, t, k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0):
2     [s, _, _, e1s, e2p, p] = y
3     # Constraints
4     e1 = e1_0 - e1s
5     e2 = e2_0 - e2p
6     dsdt = -k_1*s*e1 + k_2 * e1s + k_6 * e2p
7     de1dt = 0
8     de2dt = 0
9     de1sdt = -k_3 * e1s + k_1*s*e1 - k_2 * e1s
10    de2pdt = -k_5 * e2p + k_4*p*e2 - k_6 * e2p

```

```

11     dpdt = k_3 * e1s - k_4*p*e2 + k_5 * e2p
12
13     dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
14     return dydt
15
16 # initial condition
17 y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]
18
19 # time points
20 t = np.arange(0, total_time, dt)
21
22 # solve ODE
23 y = odeint(modelkp_4, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))
24
25 plt.figure(figsize=(15,10))
26
27 # plot results
28 plt.plot(t, y[:,0], label = "s")
29 plt.plot(t, y[:,1], label = "e1")
30 plt.plot(t, y[:,2], label = "e2")
31 plt.plot(t, y[:,3], label = "e1s")
32 plt.plot(t, y[:,4], label = "e2p")
33 plt.plot(t, y[:,5], label = "p")
34
35 plt.xlabel("Time [s] ")
36 plt.ylabel("Concentration [M] ")
37 plt.legend(loc = "best")
38 plt.show()

```



## 1.5 Second constraint: Michaelis-Menten

$$\begin{aligned} de1s/dt &= 0 \\ de2p/dt &= 0 \end{aligned}$$

By imposing:

$$\begin{aligned} s * k_1 &\approx k_2 \gg k_3 \\ p * k_4 &\approx k_5 \gg k_6 \end{aligned}$$

We introduce directly the new constraint over *modelkp\_2* method.

## 1.6 Numerical simulation modelkp\_2

We saw that the plots pf product and substrate are not similar to previous ones.

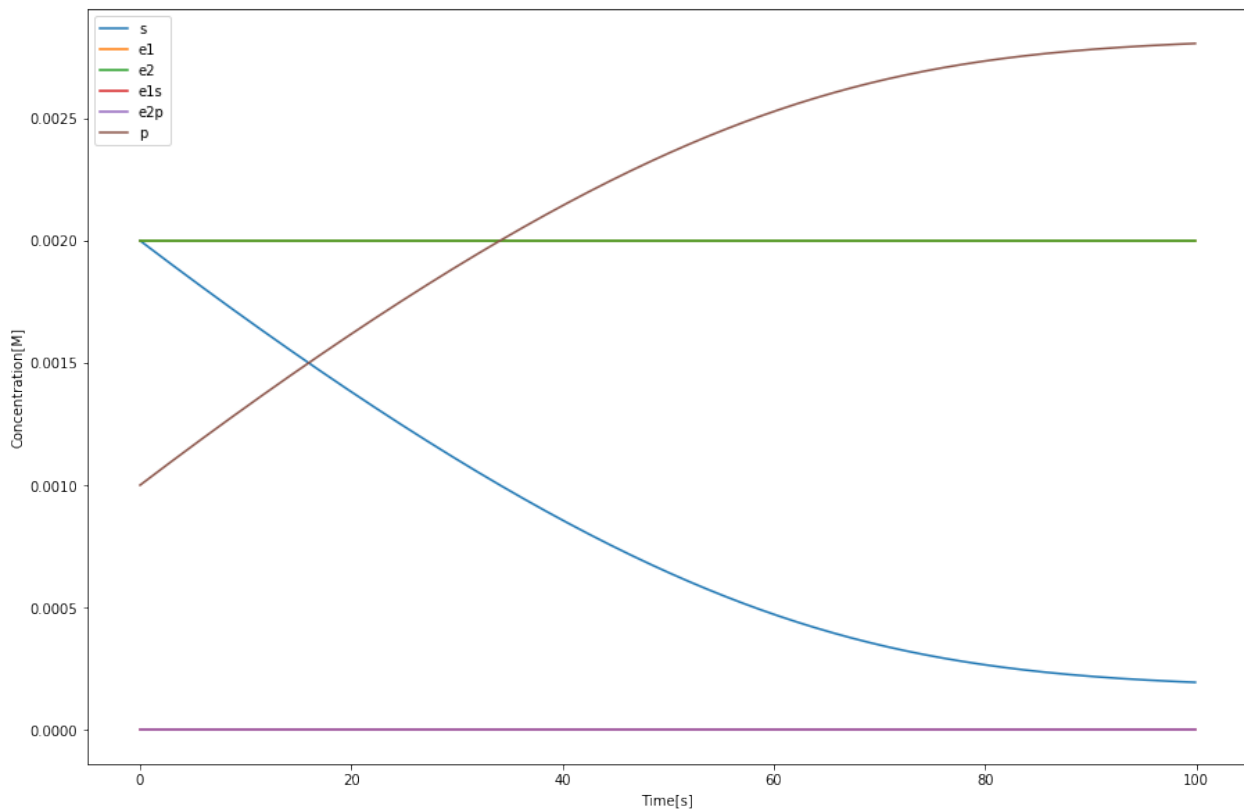
Script 1.6.1 (python)

```
1 def modelkp_2(y, t, k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0):
2     [s, _, _, _, _, p] = y
3     # Constraints
4     # from de1s/dt = 0 and e1 = e1_0 - e1s
5     e1s = (k_1 * s * e1_0) / (k_3 + k_2 + k_1*s)
```

```

6      # from  $de_{2p}/dt = 0$  and  $e_2 = e_{2\_0} - e_{2p}$ 
7       $e_{2p} = (k_4 * p * e_{2\_0}) / (k_6 + k_5 + k_4 * p)$ 
8       $e_1 = e_{1\_0} - e_{1s}$ 
9       $e_2 = e_{2\_0} - e_{2p}$ 
10
11      $dsdt = -k_1 * s * e_1 + k_2 * e_{1s} + k_6 * e_{2p}$ 
12      $de_{1dt} = 0$ 
13      $de_{2dt} = 0$ 
14      $de_{1sdt} = 0$ 
15      $de_{2pdt} = 0$ 
16      $dpdt = k_3 * e_{1s} - k_4 * p * e_2 + k_5 * e_{2p}$ 
17
18     dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
19     return dydt
20
21 # initial condition
22 y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]
23
24 # time points
25 t = np.arange(0, total_time, dt)
26
27 # solve ODE
28 y = odeint(modelkp_2, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))
29
30 plt.figure(figsize=(15,10))
31
32 # plot results
33 plt.plot(t, y[:,0], label = "s")
34 plt.plot(t, y[:,1], label = "e1")
35 plt.plot(t, y[:,2], label = "e2")
36 plt.plot(t, y[:,3], label = "e1s")
37 plt.plot(t, y[:,4], label = "e2p")
38 plt.plot(t, y[:,5], label = "p")
39
40 plt.xlabel("Time[s] ")
41 plt.ylabel("Concentration[M] ")
42 plt.legend(loc = "best")
43 plt.show()

```



## 1.7 Find parameters compatible with Michaelis Menten

We use this MM conditions to establish relations between parameters:

$$s * k_1 \approx k_2 \gg k_3$$

$$p * k_4 \approx k_5 \gg k_6$$

### Script 1.7.1 (python)

```

1 def solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0):
2     # Solve model2 and model3
3     y_2 = odeint(modelkp_2, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))
4     y_4 = odeint(modelkp_4, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))
5     #y_6 = odeint(modelkp_6, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6))
6
7     # Plot models
8     plt.figure(figsize=(15,10))
9     #plt.plot(t, y_6[:,0], label = "s_full", color="blue", marker = "x")
10    plt.plot(t, y_4[:,0], label = "s", color="blue", marker = ".")
11    plt.plot(t, y_2[:,0], label = "s_mm", color="darkblue")
12    #plt.plot(t, y_6[:,5], label = "p_full", color="orange", marker = "x")
13    plt.plot(t, y_4[:,5], label = "p", color="orange", marker = ".")

```



```

14     plt.plot(t, y_2[:,5], label = "p_mm", color="darkorange")
15     plt.xlabel("Time[s]")
16     plt.ylabel("Concentration[M]")
17     plt.legend(loc = "best")
18     plt.show()
19
20     dt = 1
21     total_time = 200
22     t = np.arange(0, total_time, dt)
23
24     # Initial conditions
25     s_0 = 0.5
26     p_0 = 0.5
27
28     e1_0 = 0.01
29     e2_0 = 0.01
30     e2p_0 = 0
31     e1s_0 = 0
32
33     k_1 = 10000
34     k_2 = k_1 * s_0
35     k_3 = k_2/10000.0
36     k_4 = 10000
37     k_5 = k_4 * p_0
38     k_6 = k_5/10000.0
39
40     solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0)
41
42     # Initial conditions
43     s_0 = 0.5
44     p_0 = 0.5
45
46     e1_0 = 0.05
47     e2_0 = 0.01
48     e2p_0 = 0
49     e1s_0 = 0
50
51     k_1 = 10000
52     k_2 = k_1 * s_0
53     k_3 = k_2/10000.0
54     k_4 = 10000
55     k_5 = k_4 * p_0
56     k_6 = k_5/10000.0
57
58     solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0)

```

