Assignment4

Fernando Freire

April 27, 2019

Contents

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*

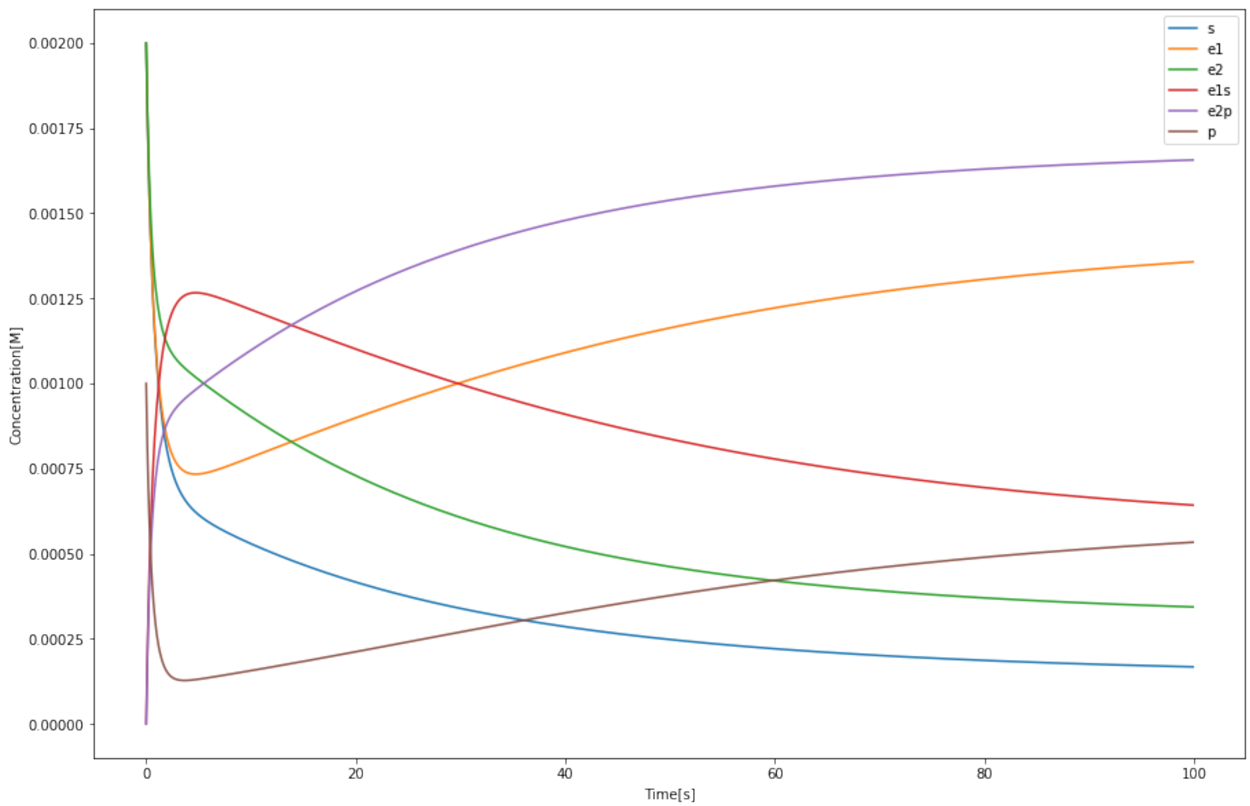
Script 1.2.1 (python)

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Initial conditions
k_1 = 500
k_2 = 0.15
k_3 = 0.03
k_4 = 1000
k_5 = 0.1
k_6 = 0.01
e1_0 = 0.002
```

```python
e2_0 = 0.002
s_0 = 0.002
p_0 = 0.001
e2p_0 = 0
e1s_0 = 0
total_time = 100
dt = 0.1
```

Script 1.2.2 (python)

```python
def modelkp_6(y, t, k_1, k_2, k_3, k_4, k_5, k_6):
    [s, e1, e2, e1s, e2p, p] = y
    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

    dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
    return dydt

# initial condition
y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]

# time points
t = np.arange(0, total_time, dt)

# solve ODE
y = odeint(modelkp_6, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6))

plt.figure(figsize=(15,10))

# plot results
plt.plot(t, y[:,0], label = "s")
plt.plot(t, y[:,1], label = "e1")
plt.plot(t, y[:,2], label = "e2")
plt.plot(t, y[:,3], label = "e1s")
plt.plot(t, y[:,4], label = "e2p")
plt.plot(t, y[:,5], label = "p")

plt.xlabel("Time[s]")
plt.ylabel("Concentration[M]")
plt.legend(loc = "best")
plt.show()
```



1.3 Conservation of mass

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

$$e1 = e1_0 - e1s$$
$$e2 = e2_0 - e2p$$

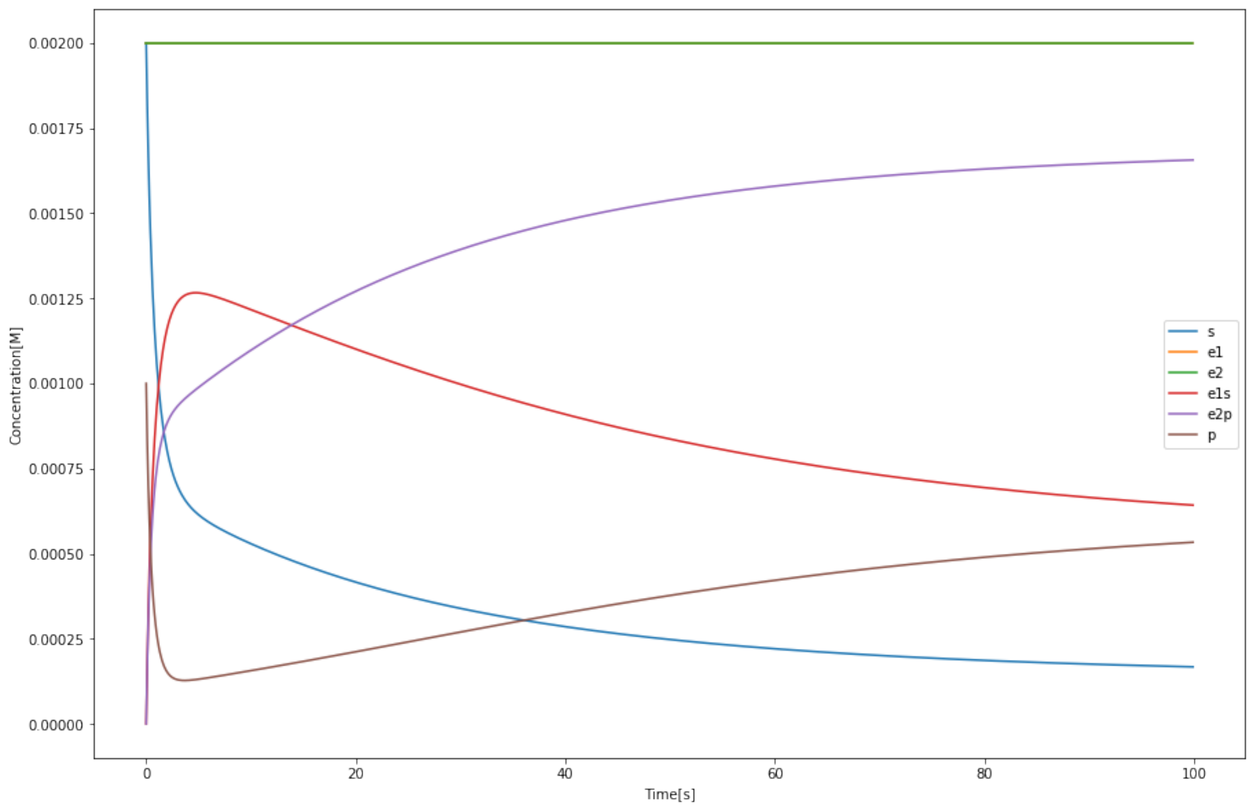
We write directly on the model method *modelkp_4* both constraints, equating to zero the derivatives.

1.4 Numerical simulation *modelkp_4*

Script 1.4.1 (python)

```python
def modelkp_4(y, t, k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0):
    [s, _, _, e1s, e2p, p] = y
    # Constraints
    e1 = e1_0 - e1s
    e2 = e2_0 - e2p
    dsdt = -k_1*s*e1 + k_2 * e1s  + k_6 * e2p
    de1dt = 0
    de2dt = 0
    de1sdt = -k_3 * e1s + k_1*s*e1 - k_2 * e1s
    de2pdt = -k_5 * e2p + k_4*p*e2 - k_6 * e2p
```

```python
    dpdt = k_3 * e1s - k_4*p*e2 + k_5 * e2p

    dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
    return dydt

# initial condition
y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]

# time points
t = np.arange(0, total_time, dt)

# solve ODE
y = odeint(modelkp_4, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))

plt.figure(figsize=(15,10))

# plot results
plt.plot(t, y[:,0], label = "s")
plt.plot(t, y[:,1], label = "e1")
plt.plot(t, y[:,2], label = "e2")
plt.plot(t, y[:,3], label = "e1s")
plt.plot(t, y[:,4], label = "e2p")
plt.plot(t, y[:,5], label = "p")

plt.xlabel("Time[s]")
plt.ylabel("Concentration[M]")
plt.legend(loc = "best")
plt.show()
```



1.5 Second constraint: Michaelis-Menten

$$de1s/dt = 0$$
$$de2p/dt = 0$$

By imposing:

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

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

We introduce directly the new constraint over *modelkp_2* method.

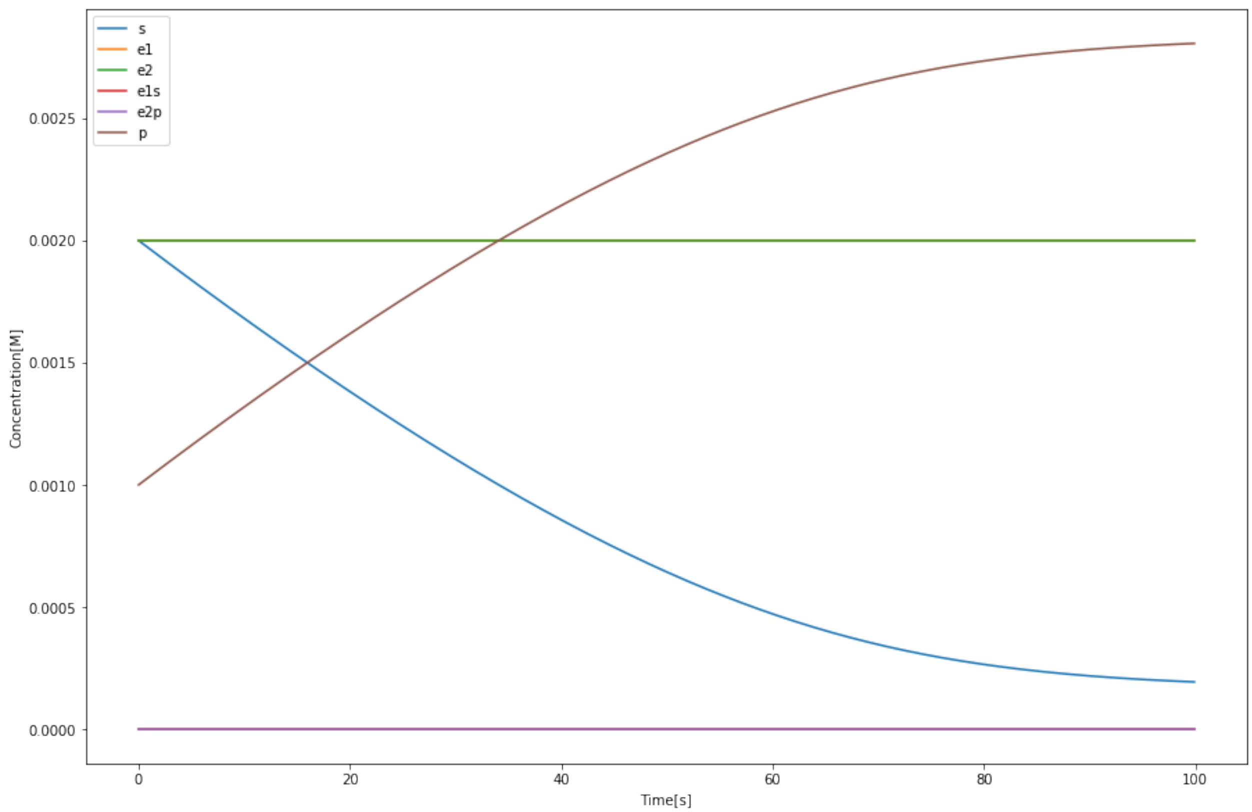
1.6 Numerical simulation modelkp_2

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

Script 1.6.1 (python)

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

```
    # from de2p/dt = 0 and e2 = e2_0 - e2p
    e2p = (k_4 * p * e2_0) / (k_6 + k_5 + k_4*p)
    e1 = e1_0 - e1s
    e2 = e2_0 - e2p

    dsdt = -k_1*s*e1 + k_2 * e1s  + k_6 * e2p
    de1dt = 0
    de2dt = 0
    de1sdt = 0
    de2pdt = 0
    dpdt = k_3 * e1s - k_4*p*e2 + k_5 * e2p

    dydt = [dsdt, de1dt, de2dt, de1sdt, de2pdt, dpdt]
    return dydt

# initial condition
y0 = [s_0, e1_0, e2_0, e1s_0, e2p_0, p_0]

# time points
t = np.arange(0, total_time, dt)

# solve ODE
y = odeint(modelkp_2, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6,e1_0, e2_0))

plt.figure(figsize=(15,10))

# plot results
plt.plot(t, y[:,0], label = "s")
plt.plot(t, y[:,1], label = "e1")
plt.plot(t, y[:,2], label = "e2")
plt.plot(t, y[:,3], label = "e1s")
plt.plot(t, y[:,4], label = "e2p")
plt.plot(t, y[:,5], label = "p")

plt.xlabel("Time[s]")
plt.ylabel("Concentration[M]")
plt.legend(loc = "best")
plt.show()
```



Script 1.6.2 (python)

```

```

1.7 Find parameters compatible with Michaelis Merten

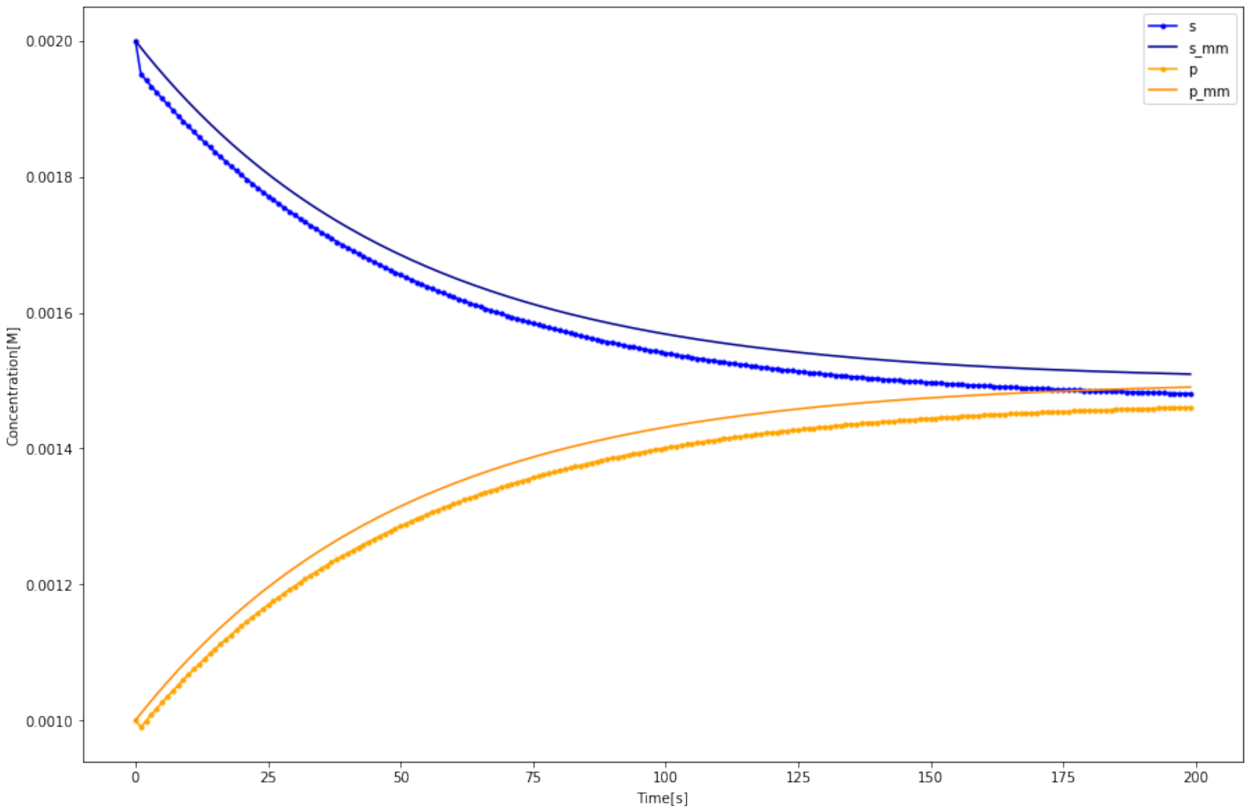
We use this MM conditions to establish relations between parameters:

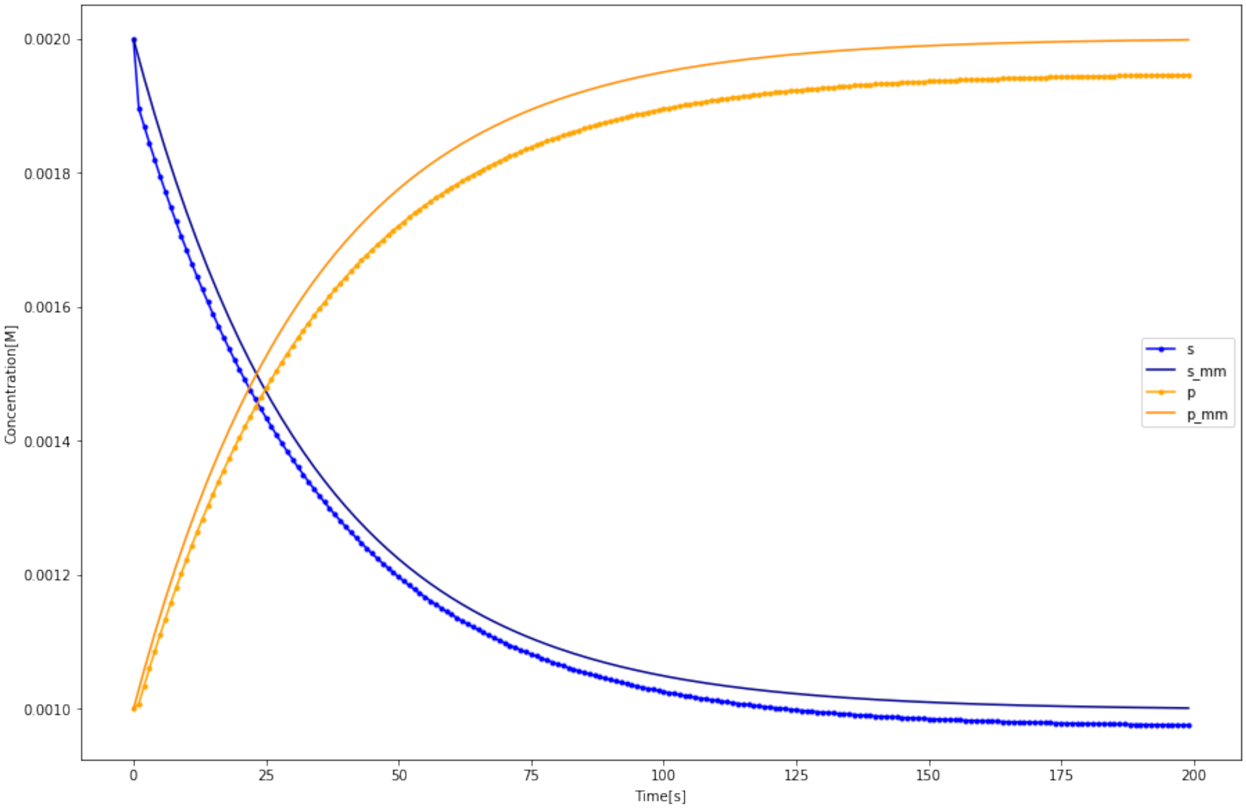
$$s * k_1 \approx k_2 \gg k_3$$
$$p * k_4 \approx k_5 \gg k_6$$

Script 1.7.1 (python)

```python
def solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0):
    # Solve model2 and model3
    y_2 = odeint(modelkp_2, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6,e1_0, e2_0))
    y_4 = odeint(modelkp_4, y0, t, args=(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0))

    # Plot models
    plt.figure(figsize=(15,10))
    plt.plot(t, y_4[:,0], label = "s", color="blue", marker = ".")
    plt.plot(t, y_2[:,0], label = "s_mm", color="darkblue")
```

```
    plt.plot(t, y_4[:,5], label = "p", color="orange", marker = ".")
    plt.plot(t, y_2[:,5], label = "p_mm", color="darkorange")
    plt.xlabel("Time[s]")
    plt.ylabel("Concentration[M]")
    plt.legend(loc = "best")
    plt.show()

dt = 1
total_time = 200
t = np.arange(0, total_time, dt)

# Initial conditions
s_0 = 0.5
p_0 = 0.5

e1_0 = 0.01
e2_0 = 0.01
e2p_0 = 0
e1s_0 = 0

k_1 = 10000
k_2 = k_1 * s_0
k_3 = k_2/10000.0
k_4 = 10000
k_5 = k_4 * p_0
k_6 = k_5/10000.0

solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0)

# Initial conditions
s_0 = 0.5
p_0 = 0.5

e1_0 = 0.02
e2_0 = 0.01
e2p_0 = 0
e1s_0 = 0

k_1 = 10000
k_2 = k_1 * s_0
k_3 = k_2/10000.0
k_4 = 10000
k_5 = k_4 * p_0
k_6 = k_5/10000.0

solve_plot_models(k_1, k_2, k_3, k_4, k_5, k_6, e1_0, e2_0)
```


s
s_mm
p
p_mm
0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
Concentration[M]
0
25
50
75
100
125
150
175
200
Time[s]



0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
Concentration[M]
0
25
50
75
100
125
150
175
200
Time[s]
s
s_mm
p
p_mm