Q2

January 14, 2023

1 Answer of 8.1

$$
\begin{aligned}
E' &= -k_1 \times E \times S + k_2 \times SE + k_3 \times SE \\
S' &= -k_1 \times E \times S + k_2 \times SE \\
SE' &= k_1 \times E \times S - k_2 \times SE - k_3 \times SE \\
P' &= k_3 \times SE
\end{aligned}
$$

2 Answer of 8.2&8.3

```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def func(x, alpha, beta, gama):
    return np.array([-alpha*x[0]*x[1]+beta*x[2]+gama*x[2], -alpha*x[0]*x[1]+beta*x[2], alpha

def runge_kutta(y, f, alpha, beta, gama, h=0.1):
    k1 = f(y, alpha, beta, gama)
    k2 = f(y + (h/2) * k1, alpha, beta, gama)
    k3 = f(y + (h/2) * k2, alpha, beta, gama)
    k4 = f(y + h * k3, alpha, beta, gama)
    return y + (k1 + 2 * k2 + 2 * k3 + k4) / 6


ESMP = np.array([1, 10, 0.0, 0.0])
t = 0.
dt = 0.01
ys, ts = [], []
#alpha, beta, gama= 100/60/1000, 600/60/1000, 150/60/1000
alpha, beta, gama= 100/60/100, 600/60/100, 150/60/100
```

```
E, S, M, P = [], [], [], []
E_, S_, M_, P_ = [], [], [], []
while t <= 1:
    tmp= runge_kutta(ESMP, func, alpha, beta, gama)
    t += dt
    e,s,m,p = tmp[0],tmp[1],tmp[2],tmp[3]

    E.append(tmp[0])
    S.append(tmp[1])
    M.append(tmp[2])
    P.append(tmp[3])

    E_.append(-alpha*e*s+beta*m)
    S_.append(-alpha*e*s+(beta+gama)*m)
    M_.append(gama*m)
    P_.append(alpha*e*s-(beta+gama)*m)

    ESMP = tmp
    ts.append(t)


plt.plot(ts, E, c='r', label='E')
plt.plot(ts, S, c='g', label='S')
plt.plot(ts, M, c='y',label='M')
plt.plot(ts, P, c='b', label='P')
plt.plot(ts, E_, c='lightcoral', label='E_')
plt.plot(ts, S_, c='lightgreen', label='S_')
plt.plot(ts, M_, c='lightyellow',label='M_')
plt.plot(ts, P_, c='lightblue', label='P_')
plt.legend()
plt.show()

def v(s, v_max, k_m):
    return (v_max * s) / (k_m + s)

data = np.array([S,P_]).T

v_real = data[:, 1]
s_real = data[:, 0]

def loss(theta):
    v_max, k_m = theta
    v_pred = v(s_real, v_max, k_m)
    temp = np.sum((v_real - v_pred)**2)
    return np.sum((v_real - v_pred)**2)
```

```
res = minimize(loss, [1, 1])

from scipy.optimize import curve_fit
def fitfun(x, a, b, c):

    return a*(x-b)**2 + c

popt, pcov = curve_fit(fitfun, s_real, v_real, p0=[3,2,-16])
a, b, c = popt

x_model = np.linspace(min(s_real), max(s_real), 100)

y_model = fitfun(x_model, a, b, c)

plt.plot(x_model,y_model, color='r')


plt.scatter(s_real, v_real)

# s_plot = np.linspace(8, 10, 100)
# plt.plot(s_plot, v(s_plot, res.x[0], res.x[1]))
# plt.xlim([0, 10])
# plt.ylim([0, 1.5])
plt.xlabel('$[S]$')
plt.ylabel('$v$')
plt.show()
```

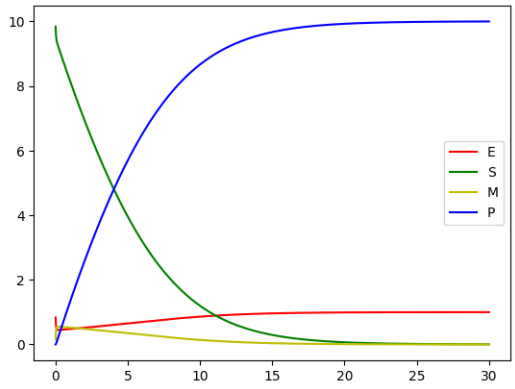

Figure 1: concentration change with time

It is hard for me to find the V_{max}. Maybe there exists some problems with the data that I produced...

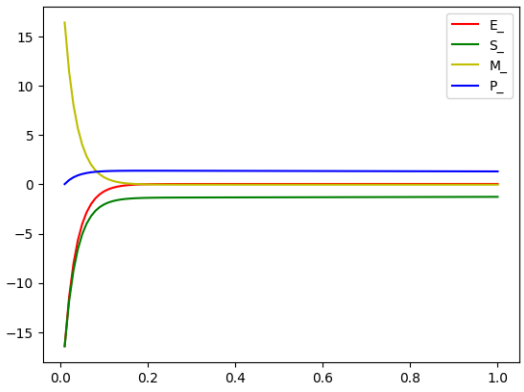

Figure 2: rate change with time

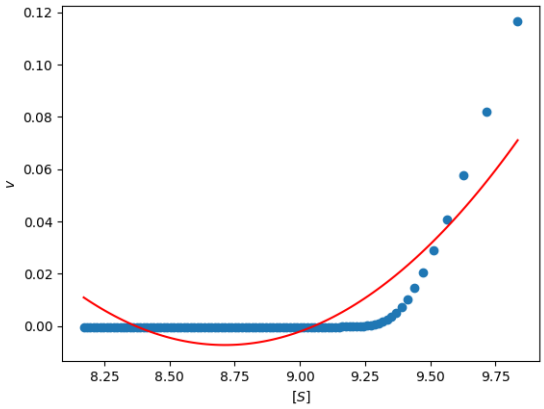

Figure 3: velocity change with the concentration of the substrate S