Question2

Name: **WU QILONG**

Application No: **C2345311**

8.1

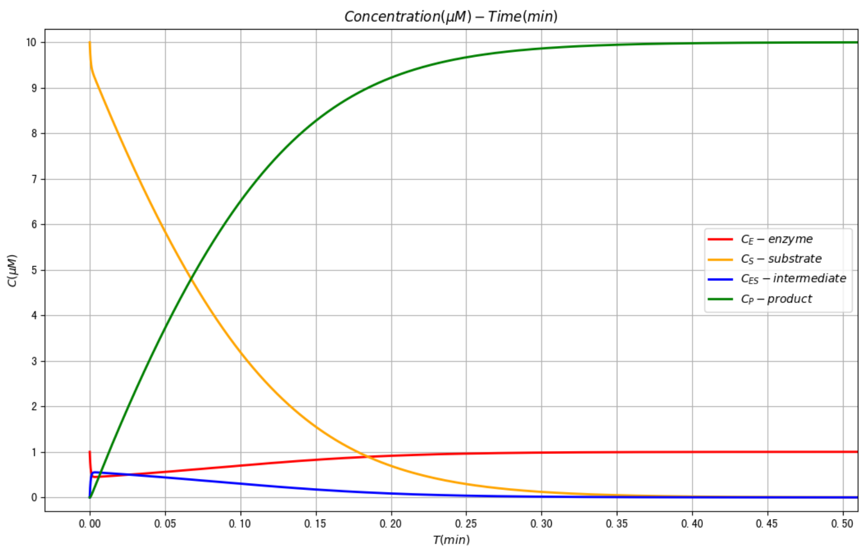
$$\begin{cases} \dfrac{d[E]}{dt} = -k1[E][S] + k2[ES] + k3[ES] \\ \dfrac{d[S]}{dt} = -k1[E][S] + k2[ES] \\ \dfrac{d[ES]}{dt} = k1[E][S] - k2[ES] - k3[ES] \\ \dfrac{d[P]}{dt} = k3[ES] \end{cases}$$

[*E*], also known as $\boldsymbol{C_E}$, accounts for the concentration of E, and **[*S*]**, **[*ES*]**, **[*P*]** are the same meaning as that. ***k*1**, ***k*2** and ***k*3** are the rates of constants, and ***t*** stands for reaction time.

8.2

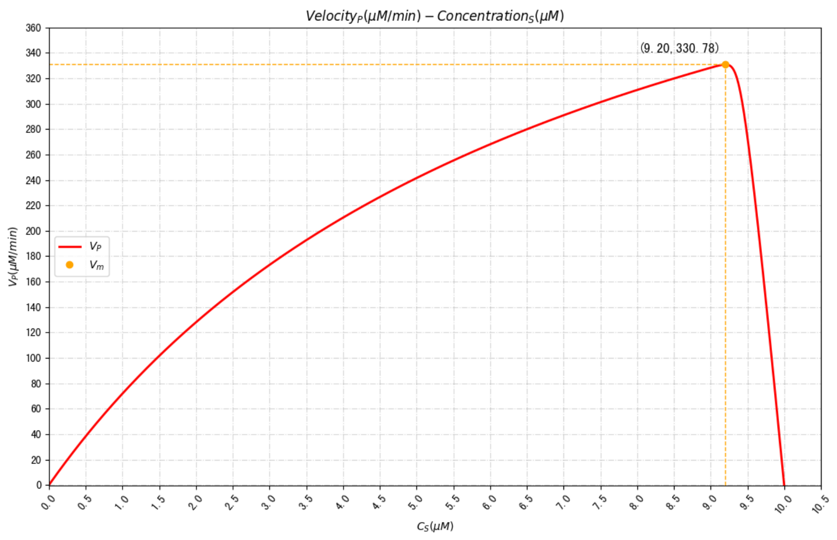
Choose the steps ***h*** as **0.00001(*min*)**, the end time ***t_end*** as **0.51(*min*)**, and other parameters are exactly as what are given in the question.

As what is shown in Runge_Kutta.py, these four equations are solved numerically by using the fourth-order Runge-Kutta method and I drew the figure (saved as f1.png) of Concentrations-Time diagram:



8.3

As what is shown in Runge_Kutta.py, after solving those four equations numeriaclly I drew the figure (saved as f2.png) of Velocity of P-Concentration of S diagram:



In my numerical simulation experiment, I figured out that the $\boldsymbol{V_m}$ (maximum value of $\boldsymbol{V_P}$) is **330.78**$\boldsymbol{(\mu M/min)}$ and found that $\boldsymbol{V_P}$ increases approxiamately linearly before $\boldsymbol{C_S}$ rises to about **0.5**$\boldsymbol{(\mu M)}$ and that $\boldsymbol{V_P}$ saturates to a maximum value, namely $\boldsymbol{V_m}$, at large $\boldsymbol{C_S}$.

Appendix

```
#!/usr/bin/python3/Runge_Kutta.py
# -*- coding:utf-8 -*-

import math
import matplotlib.pyplot as plt
import numpy as np



# Using the fourth-order Runge-Kutta method to solve the equations:

# 1.define function
def RK4(y, y1, y2, y3, h, f, t=0):
    """
    :param y1: introduce other species' concentrasions when necessary
    :param y2: same as before
    :param y3: same as before
    :param y: initial func value
    :param t: initial time, and the default value is 0 second
    :param h: time step
    :param f: func of dy/dt
    :return: next y with time step h
    """
    k_1 = f(t, y, y1, y2, y3)
    k_2 = f(t + h / 2, y + h / 2 * k_1, y1, y2, y3)
    k_3 = f(t + h / 2, y + h / 2 * k_2, y1, y2, y3)
    k_4 = f(t + h, y + h * k_3, y1, y2, y3)
    return y + h / 6 * (k_1 + 2 * k_2 + 2 * k_3 + k_4)
```

```
def reaction(C0, h, parameters):
    """
    :param C0: initial concentrations
    :param h: time step-accuracy
    :param parameters: rate constants, k1, k2, k3
    :return: next concentrations with time step h by RK4 method
    """
    E, S, ES, P = C0
    k1, k2, k3 = parameters

    def dE(t, E, S, ES, P):
        return -k1 * E * S + k2 * ES + k3 * ES

    def dS(t, S, E, ES, P):
        return -k1 * E * S + k2 * ES

    def dES(t, ES, S, E, P):
        return k1 * E * S - k2 * ES - k3 * ES

    def dP(t, P, S, ES, E):
        return k3 * ES

    E_next = RK4(y=E, y1=S, y2=ES, y3=P, h=h, f=dE)
    S_next = RK4(y=S, y1=E, y2=ES, y3=P, h=h, f=dS)
    ES_next = RK4(y=ES, y1=S, y2=E, y3=P, h=h, f=dES)
    P_next = RK4(y=P, y1=S, y2=ES, y3=E, h=h, f=dP)
    return [E_next, S_next, ES_next, P_next]
```

```
# 2.set initial conditions
h = 0.00001
## time step
parameters = [100, 600, 150]
## rate constants(/min)
C0 = [1, 10, 0, 0]
## concentrations(miuM), E, S, ES, P, respectively
t_end = 0.51
## end time(min)
time = np.arange(0, t_end, h)
## reaction time(min)
C = C0
## set C as concentrations when the reaction is processing
data = [C]
## save concentrations throughout the reaction

# 3.solve--simulation and iteration
for t in time:
    C = reaction(C0=C, h=h, parameters=parameters)
    data.append(C)
data_np = np.array(data)
C_E = data_np[:, 0]
C_S = data_np[:, 1]
C_ES = data_np[:, 2]
C_P = data_np[:, 3]
k3 = parameters[1]
V_P = k3 * C_ES
Max_arg = np.argmax(V_P)
total_time = np.arange(0, h * data_np.shape[0], h)
```

```
# 4.draw
from matplotlib.pyplot import MultipleLocator
# figure1--Concentrations~Time
plt.figure('f1')
plt.plot(total_time[:], C_E[:], '-', color='red', label="$C_{E}-enzyme$", linewidth=2)
plt.plot(total_time[:], C_S[:], '-', color='orange', label="$C_{S}-substrate$",
linewidth=2)
plt.plot(total_time[:], C_ES[:], '-', color='blue', label="$C_{ES}-intermediate$",
linewidth=2)
plt.plot(total_time[:], C_P[:], '-', color='green', label="$C_{P}-product$", linewidth=2)
plt.legend(loc='best')
plt.grid(True)
plt.title(r"$Concentration({\mu}M)-Time(min)$")
ax = plt.gca()
y_major_locator = MultipleLocator(1)
ax.yaxis.set_major_locator(y_major_locator)
x_major_locator = MultipleLocator(0.05)
ax.xaxis.set_major_locator(x_major_locator)
plt.ylim(-0.3, 10.3)
plt.xlim(-0.03, t_end)
plt.ylabel(r"$C({\mu}M)$")
plt.xlabel("$T(min)$")
# figure2--Velocity~Concentration
plt.figure('f2')
plt.plot(C_S, V_P, '-', color='red', label="$V_{P}$", linewidth=2)
plt.plot(C_S[Max_arg], V_P[Max_arg], 'o', color='orange', label="$V_{m}$",
linewidth=3)
line1 = np.arange(-0.05, C_S[Max_arg], h)
line2 = np.arange(-0.5, V_P[Max_arg], h)
plt.plot(line1, np.ones(len(line1)) * V_P[Max_arg], '--', color='orange', linewidth=1)
plt.plot(np.ones(len(line2)) * C_S[Max_arg], line2, '--', color='orange', linewidth=1)
plt.legend(loc='center left')
plt.grid(True, ls='-.', alpha=0.5)
plt.title('$Velocity_{P}({\mu}M/min)-Concentration_{S}({\mu}M)$')
ax = plt.gca()
plt.annotate("(%1.2f,%1.2f)" % (C_S[Max_arg], V_P[Max_arg]), (8, 340), xytext=(8,
340), fontsize=12)
y_major_locator = MultipleLocator(20)
ax.yaxis.set_major_locator(y_major_locator)
x_major_locator = MultipleLocator(0.5)
ax.xaxis.set_major_locator(x_major_locator)
plt.xlim(0, 10.5)
plt.ylim(-0.5, 360)
plt.xticks(rotation=50)
plt.xlabel(r"$C_{S}({\mu}M)$")
plt.ylabel(r"$V_{P}({\mu}M/min)$")
plt.show()
```