

Kinetic Study and Specificity of the product in Organic Reaction

Group-12

Bikram Ghosh, Roll No: 21CY40010

Khushi Garg, Roll No:21CY40020

Md Saimuddin Sk, Roll No:21CY40026

Rahul Kumar, Roll No:21CY40032

▼ Objective:

The study of kinetics of a chemical reaction is one of the important parts of the chemistry. After doing any experiment in the laboratory, one has to determine the reaction rate constant, order, half-life and plot various graphs according to their requirement. If one's specific interest is some organic reactions like aromatic electrophilic, nucleophilic reaction and pericyclic reaction, then it is also a concern that why the particular product is obtained as a major product (sometimes 100%) or minor.

Our group developed a program which requires Hückle matrix of the substrate and experimental data of reactant concentration vs time as input(csv file). Then the program calculates the electron density on each atom and tells that which carbon substituted product will be the major. Then the program plots the eigen vectors of HOMO followed by the electron density curve.

▼ Background Theory

Huckle Theory:: The delocalization energy, π -bond orders, and π -electron population are chemically significant parameters that can be gleaned from the orbital energies and coefficients that are the direct outputs of Hückel theory. In our project we are calculating the π -electron population of each atom to predict the reactivity of the molecule. The π -electron population is calculated by using the orbital coefficients of the Hückel MOs. The π -electron population on atom j is defined as,

$$n_{\pi}(j) = \sum_i n_i [c_j^{(i)}]^2$$

Theory of Kinetics:: If a reaction has an order n with respect to the particular reactant, then rate,

$$v = -\frac{dc}{dt} = kc^n \dots (i)$$

$$\text{or, } \log(v) = \log(k) + n\log(c)$$

A double logarithmic plot of $\log(v)$ vs $\log(c)$ gives a straight line with slope n and intercept $\log(k)$. If a straight line plot is not obtained, the rate can not be obtained by equation-(i).

The procedure is carried out by taking different initial concentration and initial rates are determined by measuring initial slopes. A double logarithmic plot then gives the order of the reaction. This procedure, dealing with initial rates, avoids possible complication due to interference by product.

▼ Section-I:: Data Input Section

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import csv
5 from scipy.optimize import curve_fit
6 #Data Input for the Huckle matrix
7 #substrate_name = input("Give the name of the Substrate: ")
8 substrate_name = "Phenol"          # Give the name of your substrate
9
10 file = open("Phenol.csv")          # Give the name of Input Huckle matrix name as csv file
11 A = np.loadtxt(file, delimiter=",") # The program reads the data fi
12
13 import matplotlib.pyplot as plt
14 import csv
15 import numpy as np
16 from scipy.optimize import curve_fit
17
18
19 x1 = []
20 y1 = []
21 x2 = []
22 y2 = []          # x1, x2, x3, x4 = Conc of reactant
23 x3 = []          # y1, y2, y3, y4 = time data
24 y3 = []
25 x4 = []
26 y4 = []
27
28
29
30 with open('reac_of_Phenol1.csv','r') as csvfile:          # Give first data file name here
31     lines = csv.reader(csvfile, delimiter=',')
32     for row in lines:
33         x1.append(float(row[0]))
34         y1.append(float(row[1]))
35
36
37 with open('reac_of_Phenol2.csv','r') as csvfile:          # Give Second data file name here
38     lines = csv.reader(csvfile, delimiter=',')
39     for row in lines:
40         x2.append(float(row[0]))
41         y2.append(float(row[1]))
42
43
44 with open('reac_of_Phenol3.csv','r') as csvfile:          # Give Third data file name here
45     lines = csv.reader(csvfile, delimiter=',')
46     for row in lines:
47         x3.append(float(row[0]))
48         y3.append(float(row[1]))
49
50
51 with open('reac_of_Phenol4.csv','r') as csvfile:          # Give Forth data file name here
52     lines = csv.reader(csvfile, delimiter=',')
53     for row in lines:
54         x4.append(float(row[0]))
55         y4.append(float(row[1]))

```

▼ Section-II:: Huckle Matrix Calculation and Results

```

1 print("Hucle Matrix for the substrate",substrate_name,"=\n",A)
2
3

```

```

4 # Solving the Huckle Matrix and Indexing it
5
6 eig_val, eig_vec = np.linalg.eig(A)
7 idx = eig_val.argsort()
8 eig_val = eig_val[idx]
9 eig_vec = eig_vec[:,idx]
10
11 # Use Data Frame To Print All The Values Sequentially
12 df = pd.DataFrame(data= eig_val) # this data frame prints the eigen value and eiger
13 print("\nEigenvalues sorted:\n",df)
14 df = pd.DataFrame(data= eig_vec)
15 with pd.option_context('display.max_rows', None,
16                        'display.max_columns', None,
17                        'display.precision',3):
18     print("\nEigenvectors sorted:\n",df)

```

Hucle Matrix for the substrate Phenol =

```

[[-80. -20.  0.  0.  0. -20.]
 [-20. -80. -20.  0.  0.  0.]
 [ 0. -20. -80. -20.  0.  0.]
 [ 0.  0. -20. -80. -20.  0.]
 [ 0.  0.  0. -20. -80. -20.]
 [-20.  0.  0.  0. -20. -80.]]

```

Eigenvalues sorted:

```

0
0 -120.0
1 -100.0
2 -100.0
3 -60.0
4 -60.0
5 -40.0

```

Eigenvectors sorted:

```

      0      1      2      3      4      5
0  0.408 -0.075  0.577  0.089 -0.577  0.408
1  0.408 -0.533  0.289 -0.539  0.289 -0.408
2  0.408 -0.459 -0.289  0.449  0.289  0.408
3  0.408  0.075 -0.577  0.089 -0.577 -0.408
4  0.408  0.533 -0.289 -0.539  0.289  0.408
5  0.408  0.459  0.289  0.449  0.289 -0.408

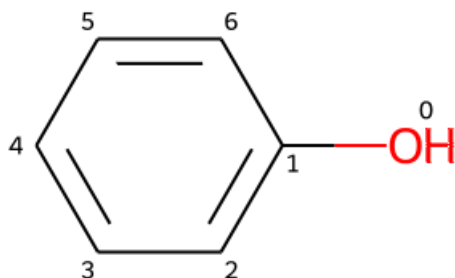
```

```

1 ! pip install rdkit-pypi
2 ! pip install py3Dmol
3
4 from rdkit import Chem
5 from rdkit.Chem.Draw import IPythonConsole
6 from rdkit.Chem import Draw
7 IPythonConsole.drawOptions.addAtomIndices = True
8 IPythonConsole.molSize = 300,300
9
10 mol = Chem.MolFromSmiles("Oc1ccccc1")
11 mol

```

Requirement already satisfied: rdkit-pypi in /usr/local/lib/python3.7/dist-packages (2021.9.2.1)
 Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.7/dist-packages (from rdkit-pypi) (1.19)
 Requirement already satisfied: py3Dmol in /usr/local/lib/python3.7/dist-packages (1.7.0)



```

1 #Calculation of pi electron population
2 n=2                                     # n represents no of electrons
3 N = 8                                   # N represents total no of pi-e
4 def pi_population(j):
5     pi_population = []
6     for i in range(0,(int(N/2))):
7         pi_population.append(n*(eig_vec[j,i])**2)
8     x = print("\nThe pi electron population of",j+1,"th atom is=",np.sum(pi_population))
9     return x,np.sum(pi_population)
10
11
12 d1, e1 = pi_population(0)
13 d2, e2 = pi_population(1)
14 d3, e3 = pi_population(2)
15 d4, e4 = pi_population(3)
16 d5, e5 = pi_population(4)
17 d6, e6 = pi_population(5)             # If user use a 4 pi electron
18 #d7, e7 = pi_population(6)           # If user use a 10 pi electron
19 #d8, e8 = pi_population(7)
20 #d9, e9 = pi_population(8)
21 #d10, e10 = pi_population(9)
22 #d11, e11 = pi_population(10)
23
24 from termcolor import colored
25
26
27 #list = [e1,e2,e3,e4]                  # If user use a 4-pi electron
28 list = [e1,e2,e3,e4,e5,e6]           # If user use a 6-pi electron
29 #list = [e1,e2,e3,e4,e5,e6,e7,e8,e9,e10] # If user use a 10-pi electron
30 #print(list)
31 #max = np.max(list)
32
33 #max = max(list)
34 #print(max)
35 if np.max(list) ==e1:
36     print(colored('\n The major product will be C1 substituted...', 'blue', attrs=['bold']))
37 elif np.max(list)==e2:
38     print(colored('\n The major product will be C2 substituted...', 'blue', attrs=['bold']))
39 elif np.max(list)==e3:
40     print(colored('\n The major product will be C3 substituted...', 'blue', attrs=['bold']))
41 elif np.max(list)==e4:
42     print(colored('\n The major product will be C4 substituted...', 'blue', attrs=['bold']))
43 elif np.max(list)==e5:
44     print(colored('\n The major product will be C5 substituted...', 'blue', attrs=['bold']))
45 elif np.max(list)==e6:
46     print(colored('\n The major product will be C6 substituted...', 'blue', attrs=['bold']))
47 elif np.max(list)==e7:

```

```

48 print(colored('\n The major product will be C7 substituted...', 'blue', attrs=['bold']))
49 elif np.max(list)==e8:
50 print(colored('\n The major product will be C8 substituted...', 'blue', attrs=['bold']))
51 elif np.max(list)==e9:
52 print(colored('\n The major product will be C9 substituted...', 'blue', attrs=['bold']))
53 elif np.max(list)==e10:
54 print(colored('\n The major product will be C10 substituted...', 'blue', attrs=['bold']))

```

The pi electron population of 1 th atom is= 1.0270109240150165

The pi electron population of 2 th atom is= 1.648506156040312

The pi electron population of 3 th atom is= 1.3244829199446722

The pi electron population of 4 th atom is= 1.0270109240150158

The pi electron population of 5 th atom is= 1.6485061560403116

The pi electron population of 6 th atom is= 1.3244829199446724

The major product will be C2 substituted...

▼ Section-III:: Plot of Electron Density and Eigen vector of HOMO

Section-III-A::

In this section the eigen vectors are saved as csv file and from that file it reads the data points for further plotting of HOMO and all eigen vectors...

```

1 import pandas as pd
2 df.to_csv('raw_data1.csv', index=False) # This code saves the previous data frame
3 df = pd.read_csv("raw_data1.csv", na_values = ['no info','.']) # This code reads the saved data file
4 #print(df.to_string()) # This code can prints the readed csv file
5
6 import csv
7 N_number = 6 # Give the no of pi-electronic system
8 N = int(N_number/2)
9 u1 = []
10 u2 = []
11 u3 = []
12 u4 = []
13 u5 = []
14 u6 = []
15 #u7 = [] # Uncomand this according to pi- electronic system
16 #u8 = []
17 #u9 = []
18 #u10 = []
19 with open('raw_data1.csv','r') as csvfile:
20     lines = csv.reader(csvfile, delimiter=',')
21     for row in lines:
22         u1.append(float(row[0]))
23         u2.append(float(row[1]))
24         u3.append(float(row[2]))
25         u4.append(float(row[3]))
26         u5.append(float(row[4]))
27         u6.append(float(row[5]))
28 # u7.append(float(row[6])) # Uncomand this according to pi- electronic system
29 # u7.append(float(row[7]))
30 # u7.append(float(row[8]))
31 # u7.append(float(row[9]))
32
33
34 # For acyclic butadiene system uncomand ths
35 #u = np.array([[np.abs(u1[N]),np.abs(u2[N]),np.abs(u3[N]),np.abs(u4[N])]])
36
37 # for benzine like system uncomand this

```

```

38 u = np.array([[np.abs(u1[N]),np.abs(u2[N]),np.abs(u3[N]),np.abs(u4[N]),np.abs(u5[N])
39 ,np.abs(u6[N])]])
40
41 # for 10-pi electronic system uncomand this
42 #u = np.array([[np.abs(u1[N]),np.abs(u2[N]),np.abs(u3[N]),np.abs(u4[N]),np.abs(u5[N])
43 #,np.abs(u6[N]),np.abs(u7[N]),np.abs(u8[N]),np.abs(u9[N]),np.abs(u10[N])]])
44
45
46
47 up = [1000*i for i in u]

```

```

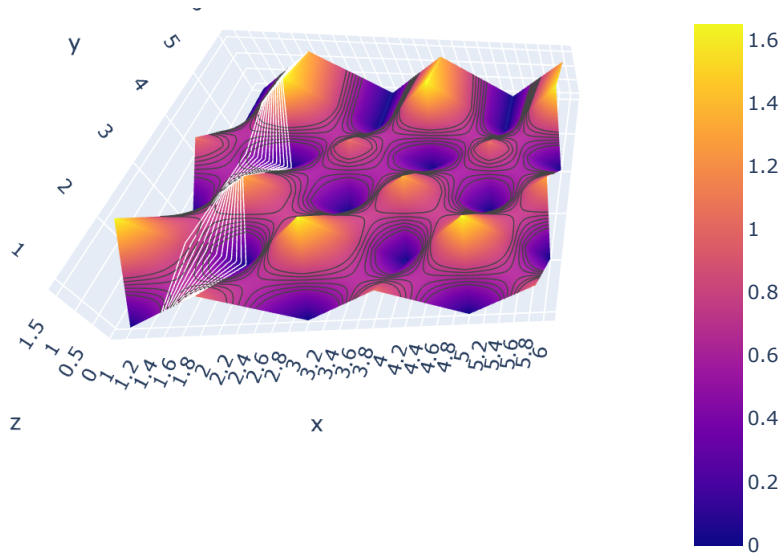
1 import plotly.graph_objects as go
2 n = 6
3 if n == 6:
4     fig = go.Figure(go.Surface(
5         contours = {
6             "x": {"show": True, "start": 1.5, "end": 2, "size": 0.04, "color": "white"},
7             "z": {"show": True, "start": 0.5, "end": 0.8, "size": 0.05}
8         },
9         x = [1,2,3,4,5,6],
10        y = [1,2,3,4,5,6],
11        z = [
12            [0, e1 , 0, e1 , 0, e1],
13            [e2, 0,e2, 0, e2, 0],
14            [0, e3, 0, e3, 0, e3],
15            [e4, 0, e4, 0, e4, 0],
16            [0, e5, 0, e5, 0, e5],
17            [0, e6, 0, e6, 0, e6]
18        ])
19 if n == 4:
20     fig = go.Figure(go.Surface(
21         contours = {
22             "x": {"show": True, "start": 1.5, "end": 2, "size": 0.04, "color": "white"},
23             "z": {"show": True, "start": 0.5, "end": 0.8, "size": 0.05}
24         },
25         x = [1,2,3,4],
26         y = [1,2,3,4],
27         z = [
28             [0, e1 , 0, e1],
29             [e2, 0,e2, 0],
30             [0, e3, 0, e3],
31             [e4, 0, e4, 0],
32             [0, e5, 0, e5],
33             [0, e6, 0, e6]
34         ])
35 if n == 10:
36     fig = go.Figure(go.Surface(
37         contours = {
38             "x": {"show": True, "start": 1.5, "end": 2, "size": 0.04, "color": "white"},
39             "z": {"show": True, "start": 0.5, "end": 0.8, "size": 0.05}
40         },
41         x = [1,2,3,4,5,6,7,8,9,10],
42         y = [1,2,3,4,5,6,7,8,9,10],
43         z = [
44             [0, e1 , 0, e1 , 0, e1, 0, e1, 0, e1],
45             [e2, 0, e2, 0, e2, 0, e2, 0, e2, 0],
46             [0, e3, 0, e3, 0, e3, 0, e3, 0, e3],
47             [e4, 0, e4, 0, e4, 0, e4, 0, e4, 0],
48             [0, e5, 0, e5, 0, e5, 0, e5 ,0, e5],
49             [0, e6, 0, e6, 0, e6, 0, e6, 0, e6]
50         ])
51
52 fig.update_layout(
53     scene = {

```

```

54     "xaxis": {"nticks": 40},
55     "zaxis": {"nticks": 8},
56     'camera_eye': {"x": 0, "y": -1, "z": 0.5},
57     "aspectratio": {"x": 1, "y": 1, "z": 0.2}
58     })
59 fig.show()

```



```

1 import plotly.graph_objects as go
2 import pandas as pd
3
4
5 print(colored('Eigen Vectors of HOMO::', 'blue', attrs=['bold']))
6 import matplotlib.pyplot as plt
7 #import numpy as np
8 rng = np.random.RandomState(0)
9 # if Acyclic 4-pi system is used uncommand this
10 #w = [ -9.26589 , -7.83580 , -3.07318 , -1.44846]
11 #q = [ 1.08457, 3.69993, 3.81303 , 1.29319]
12
13 # For Cyclic Six Member System
14 w = [ 1.56867,0.17165,2.33098, 1.69626, 0.29923, -0.46307]
15 q = [1.19053,1.11698, 0.01740, -1.22927, -1.30282, -0.12969]
16
17 # For 10-pi system System
18 #w = [ -5.94808,-7.16528,-7.18437, -5.98641, -4.75718, -4.73781,-3.50858,-2.31062 , -2.32971, -3.54691 ]
19 #q = [ 2.61584,1.92835, 0.53090, -0.18959, 0.48784, 1.90508,2.58251, 1.86202, 0.46457,-0.22292]
20
21

```

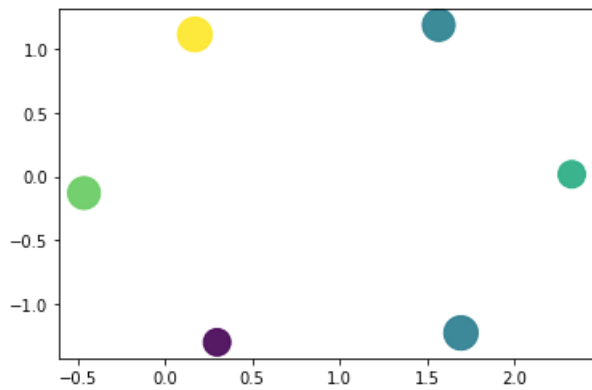
```

22 colors = rng.rand(6)    # Change the bracketed number according to pi-system electron
23 sizes = up
24 #plt.scatter(w, q, s=sizes, alpha=0.9,
25 #cmap='viridis')
26
27 plt.scatter(w, q, c=colors, s=sizes, alpha=0.9,
28 cmap='viridis')
29 #plt.colorbar()
30 #print(1000 * rng.rand(6))

```

Eigen Vectors of HOMO::

<matplotlib.collections.PathCollection at 0x7fdd6a7e8690>



▼ Section-IV:: Kinetic Part: Order, half-life, rate constant determination

```

1
2 import matplotlib.gridspec as gridspec
3 fig1 = plt.figure(constrained_layout=True,figsize=(8,8))
4 spec1 = gridspec.GridSpec(ncols=2, nrows=2, figure=fig1)
5 f1_ax1 = fig1.add_subplot(spec1[0, 0])
6
7
8 #Plotting of the Experimental data set
9 plt.plot(x1,y1,"o-")
10 plt.plot(x2,y2,"o-")
11 plt.plot(x3,y3,"o-")
12 plt.plot(x4,y4,"o-")
13 plt.xlabel("time",fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
14 plt.ylabel("Reactant concentration",fontweight='bold', color = 'blue', fontsize='10'
15 , horizontalalignment='center')
16
17 # User Defined forward difference function
18 def fw_diff(y,t,h):
19     f_x= []
20     x= []
21     for i in range(len(y)-1):
22         f_x.append((y[i+1]-y[i])/h)
23         x.append(t[i])
24     return f_x, x
25
26 # Rate determination using the forward difference
27 r1, t1= fw_diff(y1,x1,5)
28 r2, t2= fw_diff(y2,x2,5)
29 r3, t3= fw_diff(y3,x3,5)
30 r4, t4= fw_diff(y4,x4,5)
31
32
33 # Conversion the rate and time in log scale for the second plot mention in theory
34
35 r1 = np.log(abs(r1[0]))
36 r2 = np.log(abs(r2[0]))
37 r3 = np.log(abs(r3[0]))
38 r4 = np.log(abs(r4[0]))

```



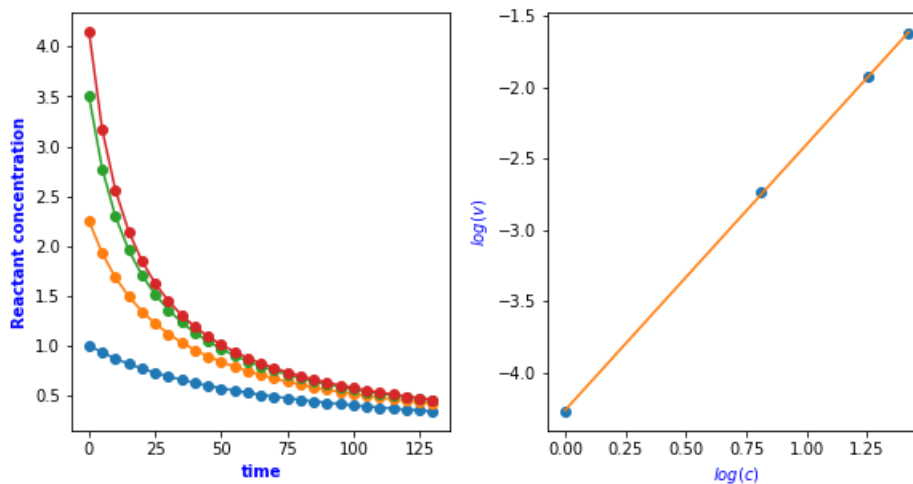
```

38 r1 = np.log(400/(1+0.1))
39
40
41
42 y1_log = [np.log(i) for i in y1]
43 y2_log = [np.log(i) for i in y2]
44 y3_log = [np.log(i) for i in y3]
45 y4_log = [np.log(i) for i in y4]
46
47
48 rate = np.array([r1,r2,r3,r4])
49 conc = np.array([y1_log[0],y2_log[0],y3_log[0],y4_log[0]])
50
51
52 f1_ax2 = fig1.add_subplot(spec1[0, 1])
53 plt.plot(conc, rate, "o")
54 plt.xlabel("$\log(c)$",fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
55 plt.ylabel("$\log(v)$",fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
56
57 def func(x,m,c):
58     return (x*m) + c
59
60
61 # Use of Curve Fitting to fit the log(v) vs log(c) plot and determination of slope as order
62 popt, pcov = curve_fit(func,conc,rate)
63 if popt[0] <= 0.001:
64     plt.plot(conc,rate)
65     print(colored('\n Order of the reaction =', 'blue', attrs=['bold']),popt[0].round(6))
66
67 else:
68     plt.plot(conc, func(conc,*popt))
69     print(colored('\n Order of the reaction =', 'blue', attrs=['bold']),popt[0])
70
71 n = popt[0] # n= Order of the reaction
72 k = np.exp(popt[1])
73 print(colored('\n Rate Constant of the Reaction =', 'blue', attrs=['bold']),k)

```

Order of the reaction = 1.8633539502132392

Rate Constant of the Reaction = 0.014070523235543338



```

1 # Code for Half-life Determination
2 if n==1:
3     t_half = 0.693/k
4 else:
5     t_half = (((2**(n-1)) - 1)/(k*(n-1))) * y1[0]**(1-n)
6 print(colored('Half-life of the Reaction =', 'blue', attrs=['bold']), t_half, "sec")
7
8
9 t = np.linspace(0,400,400)
10 x =y1[0]- k*t

```

```

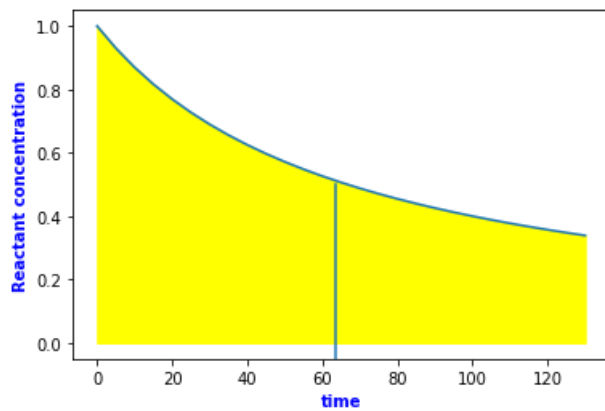
11 plt.plot(x1,y1)
12 plt.xlabel("time",fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
13 plt.ylabel("Reactant concentration",fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
14
15 plt.fill_between(x1,y1,color="yellow") #plt.fill_between(r,R2,where=(0>R2),color
16
17 plt.axvline(x= t_half,ymin = 0,ymax=y1[0]/2)
18 plt.axis('tight')
19
20 # Code for Fractional life determination
21
22 #f = float(input("Give the percentage of completion of the reaction "))
23 f = 45 # Give the percentage of completion
24 f = f/100
25 if n==1:
26     t_f = (np.log(1/(1-f)))/k
27 else:
28     t_f = ((1/(1-f)**(n-1))-1)/((n-1)*k*(y1[0]**(n-1)))
29
30 print(colored('To Complete', 'blue', attrs=['bold']),f*100,colored('% of Reaction', 'blue', attrs=['bold']),
31 colored('time required =', 'blue', attrs=['bold']), t_f," sec")

```

Rate Constant of the reaction, $k = 0.014925237204012802$

Half-life of the Reaction = 63.57891402811933 sec

To Complete 45.0 % of Reaction time required = 52.426529557349006 sec



▼ Section-V:: Kinetic Plots of the Studied Reaction

In this Section some functions are defined which can return reactant concentration and product concentration and then the functions are called to generate data points according to the order of the reaction. This data is used for plotting various kinetic graphs.

```

1 def first(t, a, k):                                # a=Initial_concentration
2     reactant_concentration = []                     # k=rate_constant
3     product_concentration = []                      # Function for data generation of first order reaction
4     for i in range(len(t)):
5         product_concentration.append(a* np.exp(-k*t[i]))
6         reactant_concentration.append(a* np.exp(-k*t[i]))    # t = time array
7     return reactant_concentration, product_concentration
8
9
10
11 def n_th(t, a, k):                                # Function for data generation of n_th order reaction
12     reactant_conc = []
13     product_concentration = []
14     for i in range(len(t)):                          # a* np.exp(-k*t[i]
15         reactant_conc.append(( a**(1-n) + ((n-1)*k*t[i]) )**(1/(1-n)))
16         product_concentration.append(a-(((n-1)*k*t[i])+a**(1-n))**(1/(1-n))))
17     return reactant_conc, product_concentration

1 # Calling the functions
2 t = x1
3 a = y1[0]
4 if n != 1:
5     x, pdt = n_th(t, a, k)
6 if n == 1:
7     x,pdt = first(t, a, k)
8
9 #.....
10 #.....
11 # The portion in this dotted line is only for data generation of some specific plot
12 s =[]
13 for i in range(len(t)):
14     s.append(1*a)
15 a = s
16
17 # Rate calculation
18 r1, t1= fw_diff(x,t,0.1)
19
20
21 v=[]
22 for i in range(len(t)):
23     v.append(np.log(a[i]/x[i]))
24
25
26 t_1 = []
27 x_1 = []
28 for i in range(1,len(t)):
29     t_1.append(1/t[i])
30     x_1.append(1/x[i])
31
32 kt = []
33 for i in range(len(t)):
34     kt.append(k*t[i])
35 #.....
36 #.....
37
38 import matplotlib.gridspec as gridspec
39 fig2 = plt.figure(constrained_layout=True,figsize=(8,8))

```

```

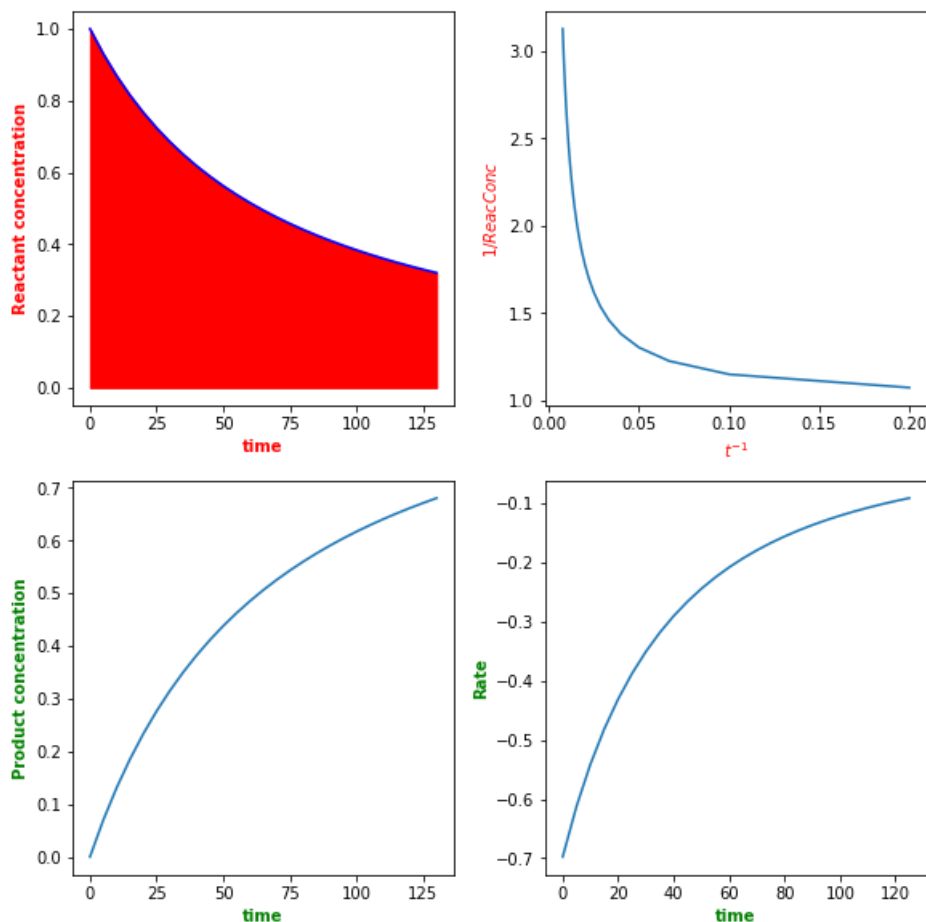
40 spec2 = gridspec.GridSpec(ncols=2, nrows=2, figure=fig2)
41
42
43 if n != 1:
44     f2_ax1 = fig2.add_subplot(spec2[0, 0])
45     plt.plot(t, x, "b") # reactant concentration vs time
46     plt.fill_between(t, x, color="red")
47     plt.xlabel("time", fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
48     plt.ylabel("Reactant concentration", fontweight='bold', color = 'red', fontsize='10',
49               , horizontalalignment='center')
50     #.....
51     f2_ax2 = fig2.add_subplot(spec2[0, 1])
52     plt.plot((t_1), (x_1)) # (1/x) vs (1/t) # (1/x) vs (1/t)
53     plt.xlabel("$t^{-1}$", fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
54     plt.ylabel("$1/ReacConc$", fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
55     #.....
56     f2_ax3 = fig2.add_subplot(spec2[1, 0])
57     plt.plot(t, pdt) # Product concentration vs time
58     plt.xlabel("time", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
59     plt.ylabel("Product concentration", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
60     #.....
61     f2_ax4 = fig2.add_subplot(spec2[1, 1])
62     plt.plot(t1, r1) # rate vs time
63     plt.xlabel("time", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
64     plt.ylabel("Rate", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
65 if n == 1:
66     fig2 = plt.figure(constrained_layout=True, figsize=(10,10))
67     spec2 = gridspec.GridSpec(ncols=2, nrows=4, figure=fig2)
68     f2_ax1 = fig2.add_subplot(spec2[0, 0])
69     plt.plot(t, x) # reactant concentration vs time
70     plt.xlabel("time", fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
71     plt.ylabel("Reactant concentration", fontweight='bold', color = 'blue', fontsize='10',
72               , horizontalalignment='center')
73     #.....
74     if n == 1:
75         a = np.array([y1[0], y2[0], y3[0], y4[0]])
76         t_half_plt = []
77         for i in range(len(a)):
78             t_half_plt.append(1*t_half)
79         f2_ax2 = fig2.add_subplot(spec2[0, 1])
80         plt.plot(a, t_half_plt) # half life vs initial concentration of the reactant
81         plt.xlabel("Initial Concentration of the Reactant", fontweight='bold', color = 'blue', fontsize='10',
82                   , horizontalalignment='center')
83         plt.ylabel("Half life", fontweight='bold', color = 'blue', fontsize='10', horizontalalignment='center')
84         #.....
85         f2_ax3 = fig2.add_subplot(spec2[1, 0])
86         plt.plot(t, pdt) # Product concentration vs time
87         plt.xlabel("time", fontweight='bold', color = 'orange', fontsize='10', horizontalalignment='center')
88         plt.ylabel("Product concentration", fontweight='bold', color = 'orange', fontsize='10',
89                   , horizontalalignment='center')
90         #.....
91         f2_ax4 = fig2.add_subplot(spec2[1, 1])
92         plt.plot(t1, r1) # rate vs time
93         plt.xlabel("time", fontweight='bold', color = 'orange', fontsize='10', horizontalalignment='center')
94         plt.ylabel("Rate", fontweight='bold', color = 'orange', fontsize='10', horizontalalignment='center')
95         #.....
96         f2_ax5 = fig2.add_subplot(spec2[2, 0])
97         plt.plot(t, v) # log(a/(a-x)) vs time
98         plt.xlabel("time", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
99         plt.ylabel("log(a/(a-x))", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
100        #.....
101        f2_ax6 = fig2.add_subplot(spec2[2, 1])
102        plt.plot((kt), v) # log(a/(a-x)) vs k*t
103        plt.xlabel("k*t", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
104        plt.ylabel("log(a/(a-x))", fontweight='bold', color = 'green', fontsize='10', horizontalalignment='center')
105        #.....
106        f2_ax7 = fig2.add_subplot(spec2[3, 0])
107        plt.plot(t, np.log(x)) # log(a-x) vs time
108        plt.xlabel("time", fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')

```

```

109 plt.ylabel("log(a-x)",fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
110 #.....
111 f2_ax8 = fig2.add_subplot(spec2[3,1])
112 plt.plot((t_1), (x_1)) # (1/x) vs (1/t)
113 plt.xlabel("$t^{-1}$",fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
114 plt.ylabel("$1/ReacConc$",fontweight='bold', color = 'red', fontsize='10', horizontalalignment='center')
115 #plt.tight_layout(pad=2.0)

```



▼ Future Plan:

In this project we have shown the reactivity and kinetic part of a reaction. In future we want to calculate the energy of the substrate, intermediate, and product and then prepare the energy profile diagram of the reaction. We also want to draw the 3-D molecular orbitals of the substrate, which helps the user to visualise the reaction in more detail.

▼ Acknowledgement

In the starting of the M.Sc, we have no knowledge about the world of computational chemistry. It is our teacher and guide Prof.Sabyasachi Misra's inspiration and teaching that we are learn so many interesting thing and able to submit the project. We, here by sincerely acknowledge our teacher Prof.Sabyasachi Misra to introduce us in the word of computational chemistry. We also acknowledge our friend Suryadeep Sur to help us in the project.

▼ Sources Used ::


Plotly graph : <https://plotly.com/python/3d-surface-plots/>

Matplotlib Python : <https://matplotlib.org/>

RDKit Cookbook: <https://www.rdkit.org/docs/Cookbook.html>

Theory : https://en.wikipedia.org/wiki/H%C3%BCckel_method (Wikipedia)

K J laidler chemical kinetics

▼  **Link to the colab page**

<https://colab.research.google.com/drive/13bNj8nAqYKCbhWJa8R8-VeOIJso0S3Ac#scrollTo=GKzO4Pbzc3Sz>

✓ 0s completed at 22:15

● ×