Notebook_example

November 27, 2021

[1]:
```
import os
os.chdir(os.getcwd()+'/..')
os.chdir('..')
print(os.getcwd())
import numpy as np
import pandas as pd
from scipy.integrate import odeint
import functional.ntwidgets as ntw
import matplotlib.pyplot as plt
import functional.rw2file as rw
import functional.helpfunc as hf
from scipy.integrate import odeint
from functional.solution import solve
from diffeqn.numeric.dy_CP_3 import dy as smdywt

import seaborn as sns

# genetic imports
from deap import base
from deap import creator
from deap import tools
import elitism

%matplotlib inline

solve_type = 'numeric'
```

```
J:\    \PyProject\
```

0.0.1 Specify experimantal data files names

[2]:
```
fnames = [
    os.path.join(solve_type, '19_04_21-Pr','CP_320_gC.xlsx'),
    os.path.join(solve_type, '19_04_21-Pr','CP_340_gC.xlsx'),
    os.path.join(solve_type, '19_04_21-Pr','CP_360_gC.xlsx')
]
```

0.0.2 Define the contact times array and the concentration of the products matrix

[3]:
```
ye,te = rw.rdexpdata(fnames)
pd.DataFrame(ye[0])
```

[3]:

	0	1	2	3	4	5	6	\
0	0.004909	0.000000	0.000000	0.000000	0.000100	0.000000	0.000000	
1	0.003375	0.000867	0.000667	0.000153	0.000342	0.000101	0.000025	
2	0.001717	0.001071	0.002121	0.000403	0.000528	0.000199	0.000053	
3	0.001070	0.000960	0.002880	0.000514	0.000574	0.000244	0.000061	
4	0.000278	0.000676	0.003955	0.000638	0.000513	0.000313	0.000077	

	7	8	9	10	11	12	13	\
0	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	
1	0.000061	0.000062	0.000146	4.451938e-07	0.000003	0.000010	0.000011	
2	0.000105	0.000148	0.000336	1.179868e-06	0.000017	0.000044	0.000029	
3	0.000112	0.000179	0.000375	1.650946e-06	0.000027	0.000066	0.000058	
4	0.000113	0.000287	0.000604	2.660132e-06	0.000050	0.000115	0.000095	

	14	15	16	17
0	0.002	0.002	0.002	0.002
1	0.002	0.002	0.002	0.002
2	0.002	0.002	0.002	0.002
3	0.002	0.002	0.002	0.002
4	0.002	0.002	0.002	0.002

[4]:
```
pd.DataFrame(te[0])
```

[4]:

	0
0	0.000000
1	0.852996
2	1.783292
3	2.441233
4	3.607244

0.0.3 Define temperature array

[5]:
```
np.set_printoptions(linewidth=100)
T = np.array([320, 340, 360])+np.array([273]*1)
```

0.0.4 Define intial kinetic parameters

Ak **is Pre-exponential factor**

Bk **is Acrivation Energy**

$alpha$ **is Reaction orders with respect to the component**

[6]:
```
Ak = np.ones(27)*0
Bk = np.zeros(27)
```

[7]:
```
knst = np.zeros((Ak.shape[0]*2,))
knst[::2] = Ak
knst[1::2] = Bk

Aalpha = np.ones(5)
Balpha = np.zeros(5)

alpha = np.zeros((Aalpha.shape[0]*2,))

alpha[::2] = Aalpha
alpha[1::2] = Balpha
k = np.concatenate((knst,alpha))
```

[8]:
```
pd.DataFrame(k, columns=['initial kinetic parameters'])
```

[8]:

	initial kinetic parameters
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
..	…
59	0.0
60	1.0
61	0.0
62	1.0
63	0.0

[64 rows x 1 columns]

Method L-M - Levenberg-Marqurdt alghorithm

__1 - objective function, that is defined by equations (4) and (5) in manuscript

[9]:
```
display(ntw.solfButton)
display(ntw.methodButton)
```

```
RadioButtons(description='Method:', options=('Method L-M', 'Simulated␣
↪annealing'), value='Method L-M')
```

```
RadioButtons(description='Objective function:',␣
↪layout=Layout(width='max-content'), options=(' _1', ' _2', ' _…
```

0.0.5 Kinetic parameters evalustion by L-M algorithm

[10]:
```
import functional.minifunc.khandler as khf
method=ntw.method[ntw.methodButton.value]
solf=ntw.solf[ntw.solfButton.value]
dyargs=[T]
res = solve(smdywt, ye, te, k,
            tnum=5000,
            method=method,
            niter=5,
            solf=solf,
            dyargs=dyargs,
            dyargsf=dyargsf,
            slctY=slctY,
            khandler=khf.resfunc['absk'], #absk - c       K, #tempk -
 ↪0
            khandlerArgs=khandlerArgs,
            yrat=yrat)
```

0.0.6 Define limits for Genetic algorithm

[11]:
```
k = res
#print(k.load())
k_start_min = res*0.8
k_start_max = res*1.2
k_mut_min = res*0.1
k_mut_max = res*10
```

[12]:
```
# problem constants:
DIMENSIONS = k.size  # number of dimensions

# Genetic Algorithm constants:
POPULATION_SIZE = 500
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.9   # (try also 0.5) probability for mutating an individual
MAX_GENERATIONS = 100
HALL_OF_FAME_SIZE = 25
CROWDING_FACTOR = 20.0  # crowding factor for crossover and mutation

# set the random seed:
#RANDOM_SEED = 42
#random.seed(RANDOM_SEED)
```

[13]:
```
toolbox = base.Toolbox()

# define a single objective, minimizing fitness strategy:
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

```
# create the Individual class based on list:
creator.create("Individual", list, fitness=creator.FitnessMin)
```

[14]:
```
# helper function for creating random real numbers uniformly distributed within␣
↪a given range [low, up]
# it assumes that the range is the same for every dimension
def randomFloat(low, up):
    return [random.uniform(l, u) for l, u in zip(low, up)]
```

[15]:
```
# create an operator that randomly returns a float in the desired range and␣
↪dimension:
toolbox.register("attrFloat", randomFloat, k_start_min, k_start_max)

# create the individual operator to fill up an Individual instance:
toolbox.register("individualCreator", tools.initIterate, creator.Individual,␣
↪toolbox.attrFloat)

# create the population operator to generate a list of individuals:
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.
↪individualCreator)
```

[16]:
```
# Eggholder function as the given individual's fitness:
def differential(individual):
    # global alphap
    global y_norm_weights
    dyargs=T

    #        beta,                          ,          -- max 23.01.21
    #konst = set_beta2end(np.array(individual), alphap.size)

    f = solve_dy( smdywt,
                  ye,
                  te,
                  np.array(individual),  # konst
                  resfunc=rmf.basinhoppingres,
                  dyargs=dyargs,
                  dyargsf=af.argsfncGen,
                  yrat=y_norm_weights,
                  method=' 1'
                 )
    if math.isnan(f): f = 10e+100
    return f,  # return a tuple

toolbox.register("evaluate", differential)
```

[17]:
```
# genetic operators:
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("mate", tools.cxSimulatedBinaryBounded, low=k_mut_min,␣
↪up=k_mut_max, eta=CROWDING_FACTOR)
toolbox.register("mutate", tools.mutPolynomialBounded, low=k_mut_min,␣
↪up=k_mut_max, eta=CROWDING_FACTOR, indpb=1.0/DIMENSIONS)
```

0.0.7 Genetic algorithm run

[18]:
```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)
stats.register("avg", np.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

# perform the Genetic Algorithm flow with elitism:
population, logbook = elitism.eaSimpleWithElitism(population, toolbox,␣
↪cxpb=P_CROSSOVER, mutpb=P_MUTATION,
                                                  ngen=MAX_GENERATIONS, stats=stats,␣
↪halloffame=hof, verbose=True)

# print info for best solution found:
best = hof.items[0]
```

The set of constants obtained by the genetic algorithm was optimized by the Levenberg-Marquardt algorithm one more time and the result obtained was final.

[19]:
```
method=ntw.method[ntw.methodButton.value]
solf=ntw.solf[ntw.solfButton.value]
dyargs=[T]
res = solve(smdywt, ye, te, best,
            tnum=5000,
            method=method,
            niter=5,
            solf=solf,
            dyargs=dyargs,
            dyargsf=dyargsf,
            slctY=slctY,
            khandler=khf.resfunc['absk'], #absk - c       K, #tempk -          ␣
↪0
            khandlerArgs=khandlerArgs,
            yrat=yrat)
```

[20]:
```
np.set_printoptions(linewidth=20)
a, b = res[::2],res[1::2]
b = b
print('final result')
pd.DataFrame(np.transpose(np.array([a,b])), columns=['a','b'])

#print(res)
```

```
final result
```

[20]:

	a	b
0	2.015754e+05	9.826177e+04
1	6.218173e+10	3.667129e+08
2	9.191317e+07	1.139626e+05
3	1.092973e+13	1.145831e+05
4	3.897215e+09	7.778796e+04
5	9.841265e+04	8.315703e+04
6	1.959020e+07	4.459044e+04
7	3.007251e+06	4.489916e+04
8	1.895426e+07	3.799571e+04
9	5.666073e+05	3.205785e+04
10	2.609692e+12	1.324323e+05
11	8.095262e+09	7.203290e+04
12	1.894369e+11	1.216633e+05
13	6.345303e+08	1.000740e+05
14	6.343143e+08	1.000728e+05
15	4.723158e+14	1.504320e+05
16	8.480130e+12	1.807091e+05
17	2.559579e+05	1.055490e+05
18	5.321891e+04	9.990311e+04
19	8.553946e+04	1.019374e+05
20	3.359911e+12	1.108680e+05
21	2.557336e+11	9.058068e+04
22	1.527502e+07	1.509290e+05
23	1.829903e+06	7.390672e+04
24	6.240029e+06	9.108139e+04
25	9.151398e+06	1.038059e+05
26	1.903401e+08	8.540581e+04
27	1.975783e+00	0.000000e+00
28	2.351995e-03	0.000000e+00
29	1.392442e+00	0.000000e+00
30	8.600648e-01	0.000000e+00
31	5.429743e-01	0.000000e+00

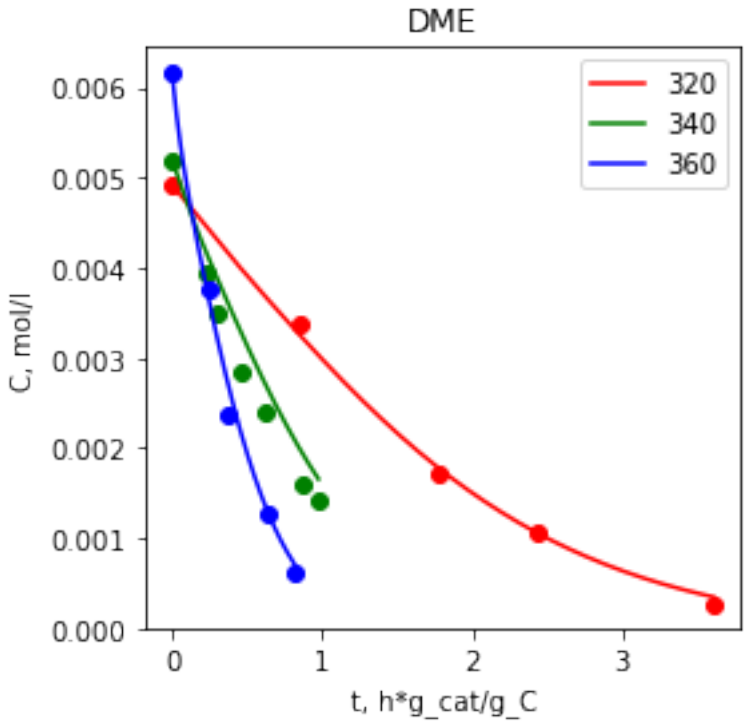
0.0.8 Modeling

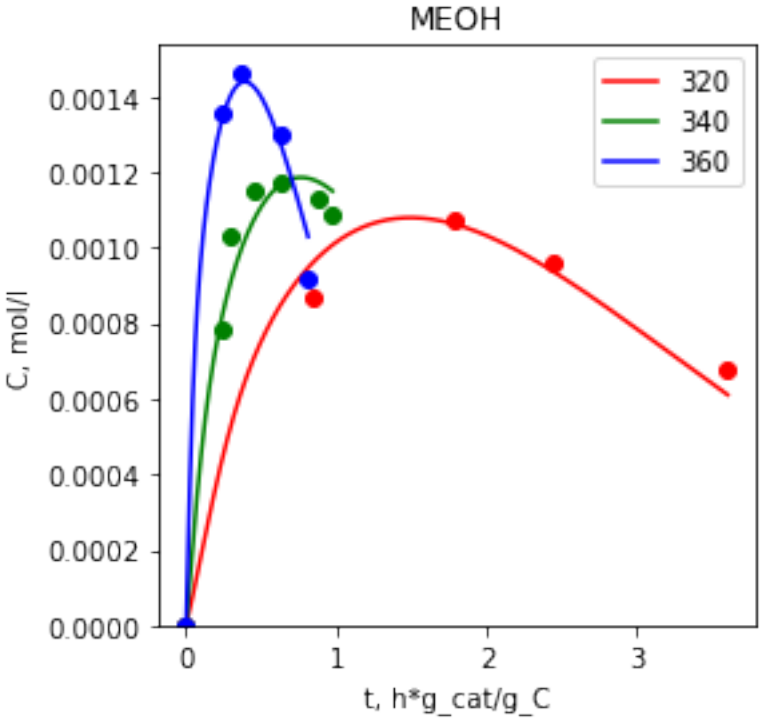
[21]:
```
t = [np.linspace(tm[0,0], tm[-1,0], 500) for tm in te]
tlen = len(fnames)
y = [odeint(smdywt, ye[i][0,:], t[i], (res, T[i]), tfirst=True) for i in␣
 ↪range(tlen)]
yk0 = [odeint(smdywt, ye[i][0,:], t[i], (k, T[i]), tfirst=True) for i in␣
 ↪range(tlen)]
```

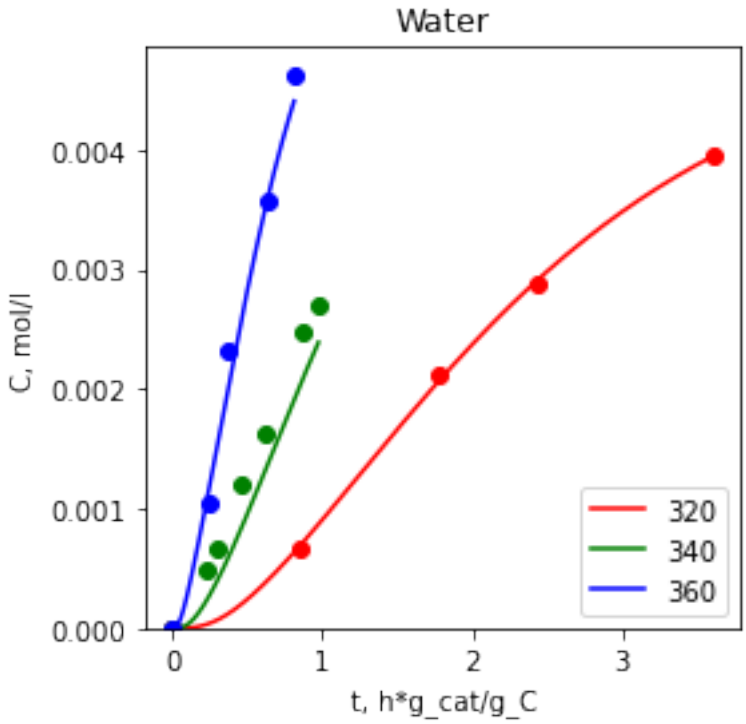
[22]:
```
from IPython.core.display import display, HTML
display(HTML("<style>div.output_scroll { height: 44em; }</style>"))
```

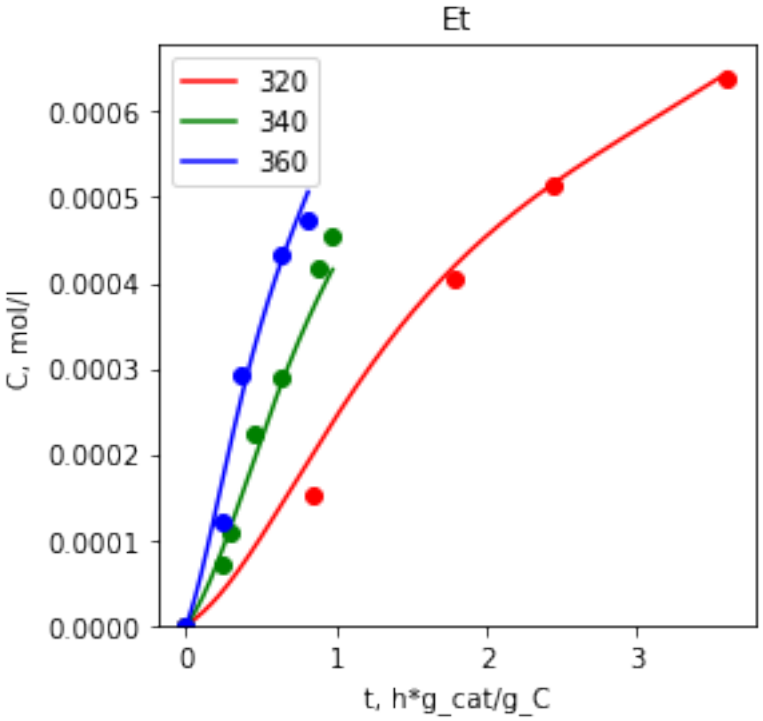
```
<IPython.core.display.HTML object>
```

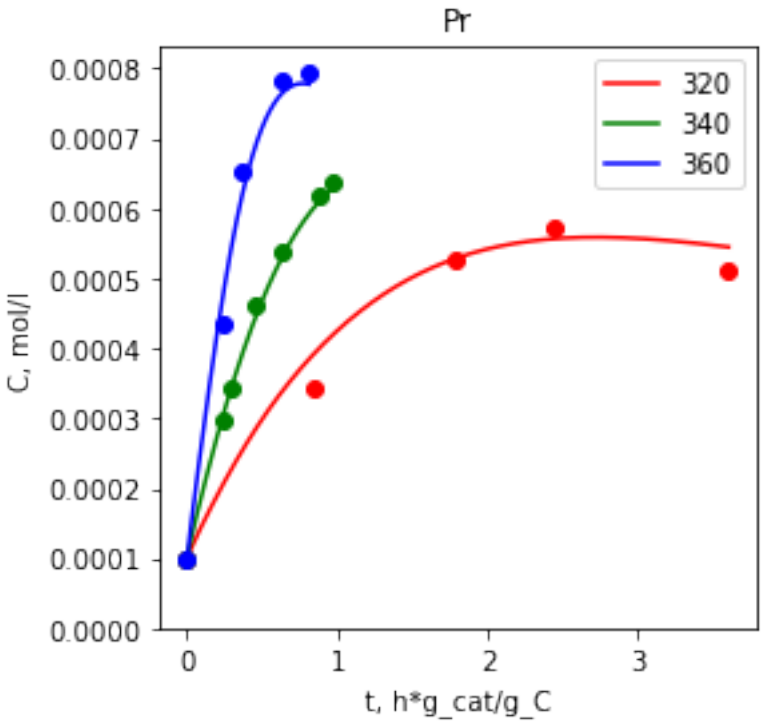
[23]:
```
Names = ['DME', 'MEOH', 'Water', 'Et', 'Pr', 'Bu', 'Pent', 'Hex', 'Arom8',␣
 ↪'CH4', 'Het', ' HPr', 'HBu', 'HPent', 'ZH', 'CP', 'C6', 'C8']

color = ['r', 'g', 'b', 'm']

for i in range(0, 14):
    fig, ax1 = plt.subplots(nrows=1, ncols=1,figsize=(4,4 ))

    for j in range(tlen):
        ax1.plot(t[j], y[j][:,i], color[j], label=T[j]- 273)
        ax1.plot(te[j], ye[j][:,i], 'o'+color[j])

    ax1.set_ylim(bottom=0)
    ax1.set_xlabel('t, h*g_cat/g_C')
    ax1.set_ylabel('C, mol/l')
    ax1.set_title(Names[i])
    ax1.legend()
```

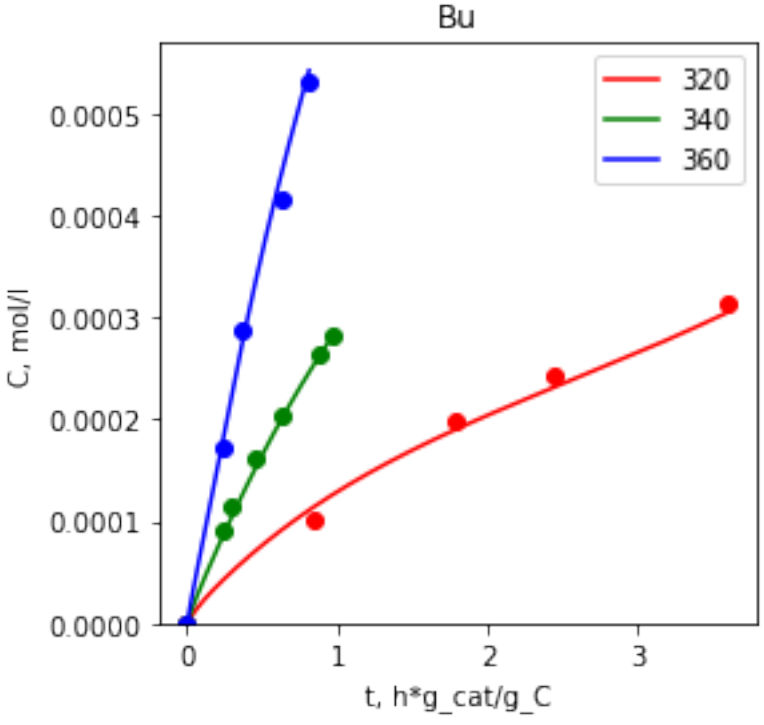

DME
320
340
360
0.006
0.005
0.004
0.003
0.002
0.001
0.000
C, mol/l
0
1
2
3
t, h*g_cat/g_C

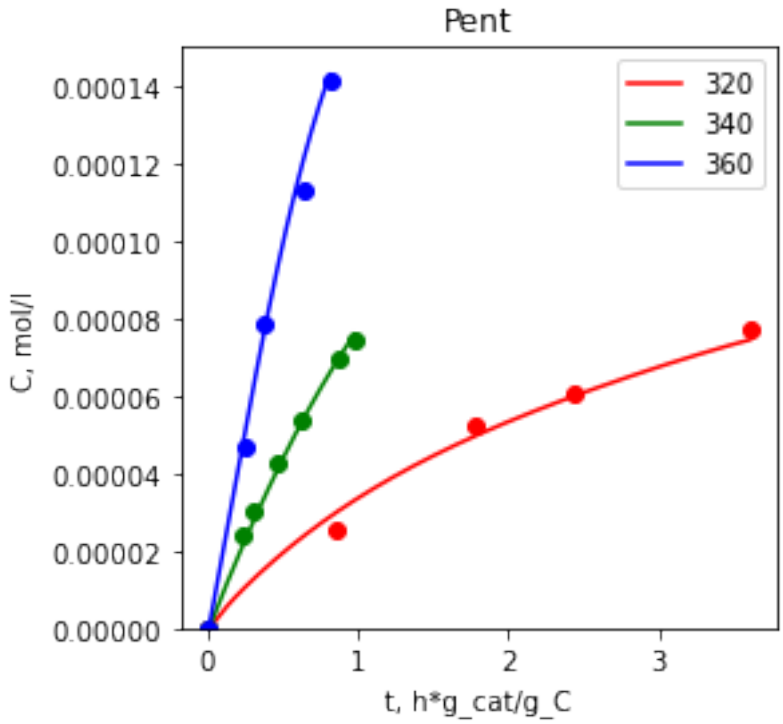

MEOH
320
340
360
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
C, mol/l
0
1
2
3
t, h*g_cat/g_C

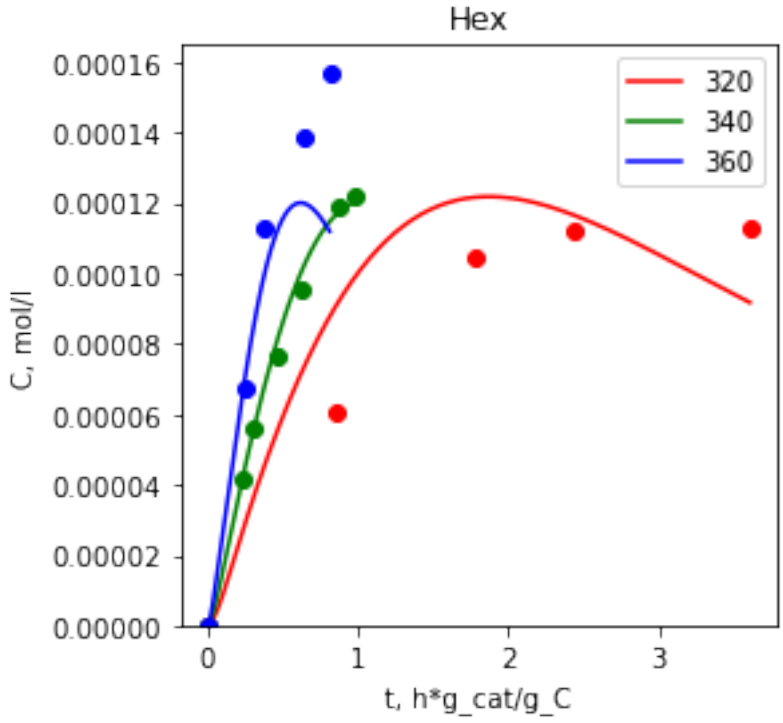

Water
0.004
0.003
0.002
0.001
0.000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360

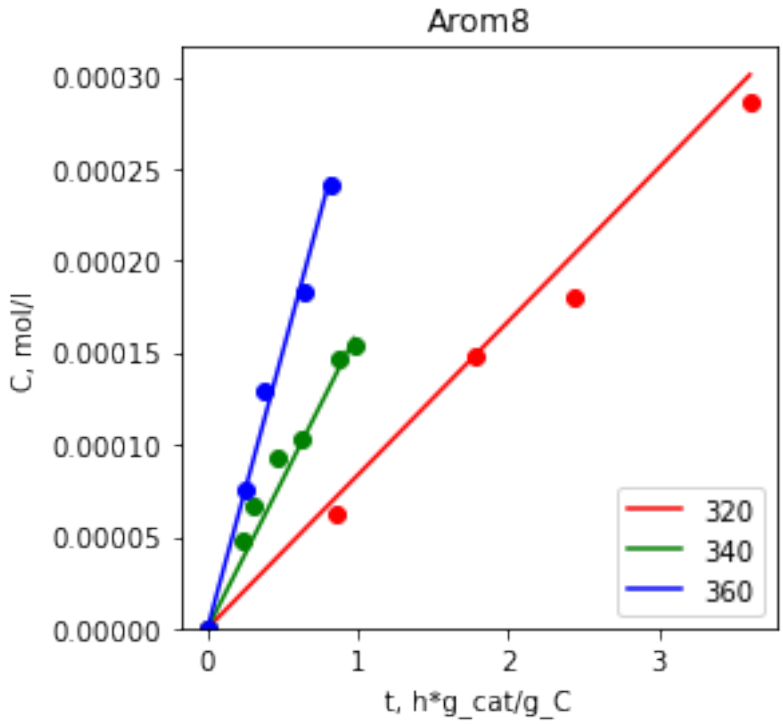

Et
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360

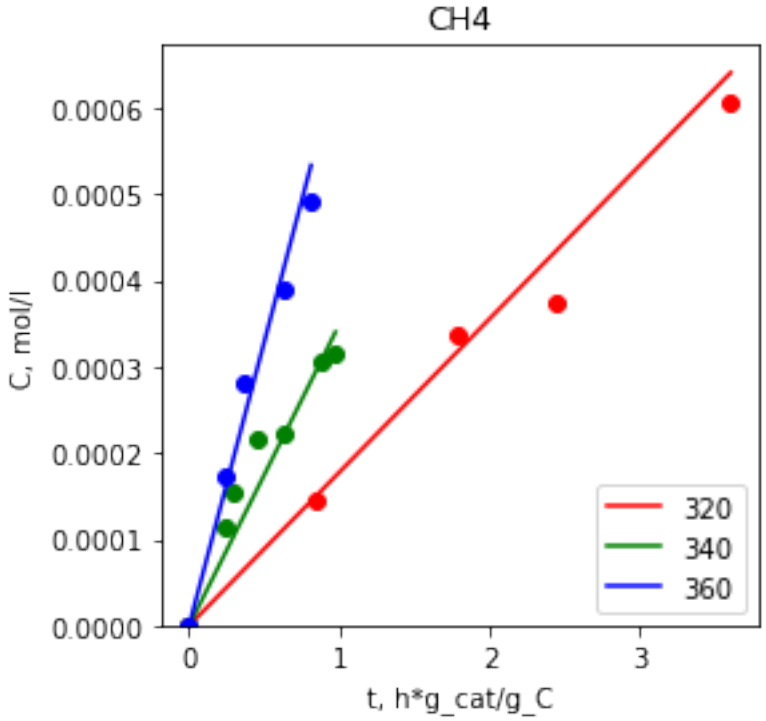

Pr
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360

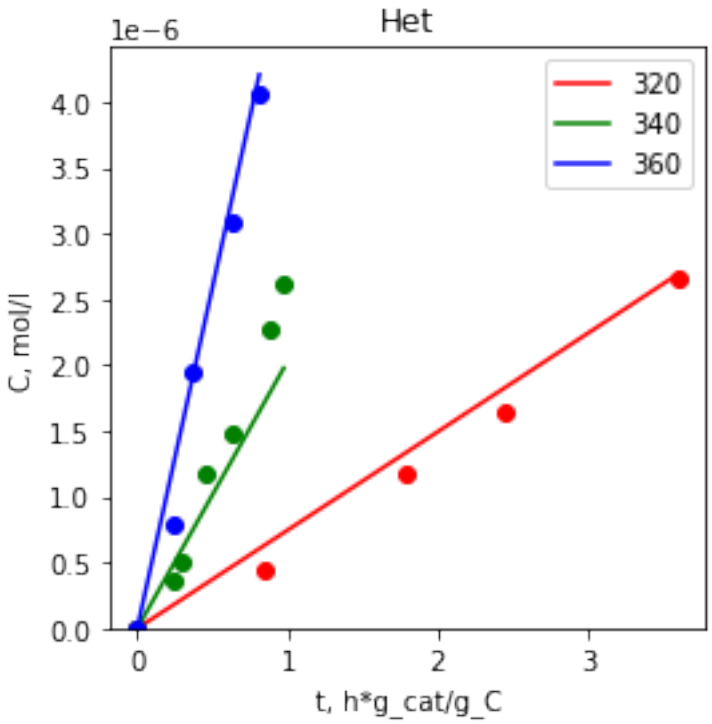

Bu
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360

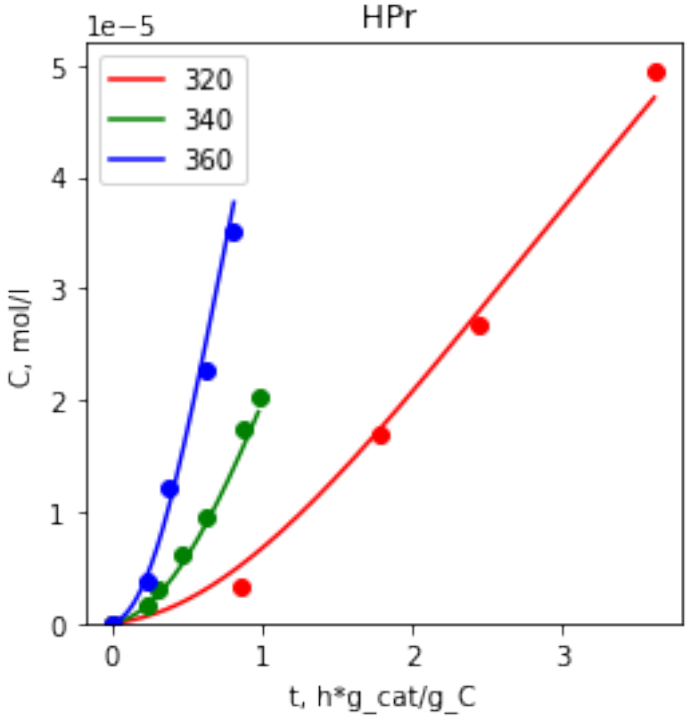

Pent
320
340
360
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
C, mol/l
0
1
2
3
t, h*g_cat/g_C

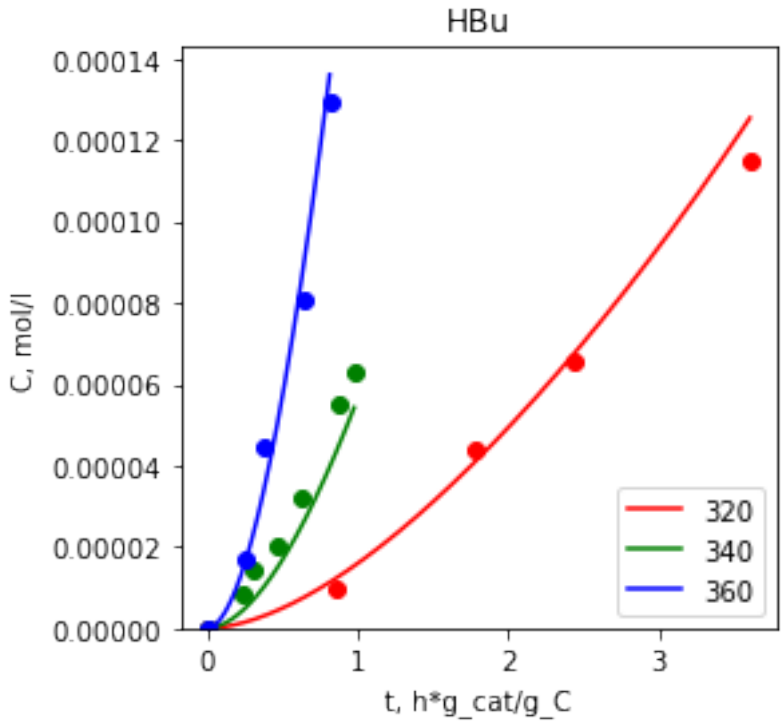

Hex
320
340
360
0.00016
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
C, mol/l
0
1
2
3
t, h*g_cat/g_C

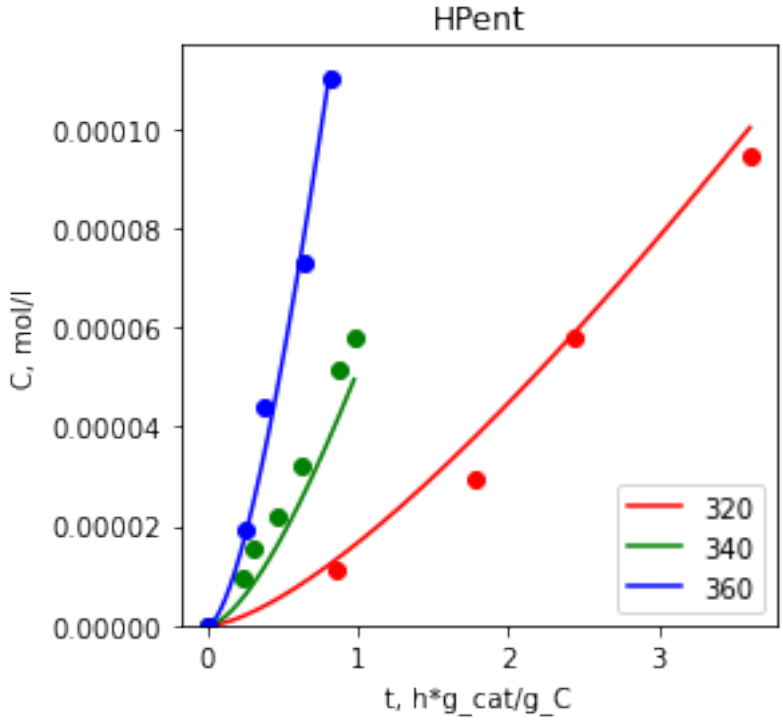

Arom8
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360


CH4
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360


Het
1e−6
320
340
360
C, mol/l
t, h*g_cat/g_C
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
2
3


HPr
1e−5
320
340
360
C, mol/l
t, h*g_cat/g_C
0
1
2
3
4
5
0
1
2
3


HBu
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360


HPent
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
C, mol/l
0
1
2
3
t, h*g_cat/g_C
320
340
360

0.0.9 Result

[24]:
```
import matplotlib.backends.backend_pdf
import re

file = open('diffeqn/numeric/dy_CP_3.py', 'r')
s = "".join(file.readlines())
equations = list(re.findall(r'r.append\((.*)\)', s))

while(len(equations) < len(a)):
    equations.append('    !')

#array = [[a[i]*np.exp(-b[i]/T[j]) for j in range(len(T))]for i in␣
 ↪range(len(a)) ]
array = [[0 for j in range(len(T))]for i in range(len(a))]
df = pd.DataFrame(array, columns= T-273)

for i in range(len(T)):
    df[T[i]-273] = a*np.exp(-b/(T[i]*8.31))
    df[T[i]-273] = [ "{:.2E}".format(el) for el in df[T[i]-273]]


sa = [ "{:.2E}".format(el) for el in a]
sb = [ "{:.2E}".format(el) for el in b]

nums = list(range(1,len(a)+1))

table = pd.DataFrame(np.transpose(np.array([equations,sa,sb])),␣
 ↪columns=['equation','A','Ea'])
table = table.join(df)
table
```

[24]:

	equation	A	Ea	320	340	360
0	1.0	2.02E+05	9.83E+04	4.41E-04	8.45E-04	1.56E-03
1	Pr**2	6.22E+10	3.67E+08	0.00E+00	0.00E+00	0.00E+00
2	Et*MeOMe	9.19E+07	1.14E+05	8.31E-03	1.77E-02	3.58E-02
3	Et*MeOH	1.09E+13	1.15E+05	8.72E+02	1.86E+03	3.79E+03
4	Pr*MeOMe	3.90E+09	7.78E+04	5.44E+02	9.10E+02	1.47E+03
5	Pr*MeOH	9.84E+04	8.32E+04	4.62E-03	8.01E-03	1.34E-02
6	Bu*MeOMe	1.96E+07	4.46E+04	2.30E+03	3.09E+03	4.08E+03
7	Bu*MeOH	3.01E+06	4.49E+04	3.32E+02	4.47E+02	5.90E+02
8	Pent*MeOMe	1.90E+07	3.80E+04	8.49E+03	1.09E+04	1.38E+04
9	Pent*MeOH	5.67E+05	3.21E+04	8.47E+02	1.05E+03	1.28E+03
10	Hex	2.61E+12	1.32E+05	5.56E+00	1.34E+01	3.04E+01
11	Pr**2	8.10E+09	7.20E+04	3.63E+03	5.85E+03	9.14E+03
12	Hex	1.89E+11	1.22E+05	3.59E+00	8.04E+00	1.71E+01
13	1.0	6.35E+08	1.00E+05	9.61E-01	1.86E+00	3.47E+00

```
                      1.0  6.34E+08  1.00E+05  9.61E-01  1.86E+00  3.47E+00
1*(pow(MeOMe, order_Ox))  4.72E+14  1.50E+05  2.61E+01  7.06E+01  1.80E+02
 1*(pow(MeOH, order_Ox))  8.48E+12  1.81E+05  1.01E-03  3.33E-03  1.02E-02
                        1  2.56E+05  1.06E+05  1.28E-04  2.57E-04  4.94E-04
                        1  5.32E+04  9.99E+04  8.35E-05  1.62E-04  3.01E-04
                        1  8.55E+04  1.02E+05  8.88E-05  1.74E-04  3.28E-04
              MeOMe*Water  3.36E+12  1.11E+05  5.69E+02  1.19E+03  2.36E+03
                  MeOH**2  2.56E+11  9.06E+04  2.66E+03  4.84E+03  8.50E+03
           pow(Et,alfa_1)  1.53E+07  1.51E+05  7.63E-07  2.07E-06  5.28E-06
          pow(Pr, alfa_2)  1.83E+06  7.39E+04  5.61E-01  9.15E-01  1.45E+00
          pow(Bu, alfa_3)  6.24E+06  9.11E+04  5.86E-02  1.07E-01  1.89E-01
        pow(Pent, alfa_4)  9.15E+06  1.04E+05  6.50E-03  1.29E-02  2.46E-02
                      Hex  1.90E+08  8.54E+04  5.66E+00  9.96E+00  1.69E+01
                     !  1.98E+00  0.00E+00  1.98E+00  1.98E+00  1.98E+00
                     !  2.35E-03  0.00E+00  2.35E-03  2.35E-03  2.35E-03
                     !  1.39E+00  0.00E+00  1.39E+00  1.39E+00  1.39E+00
                     !  8.60E-01  0.00E+00  8.60E-01  8.60E-01  8.60E-01
                     !  5.43E-01  0.00E+00  5.43E-01  5.43E-01  5.43E-01
```