భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Project Report

Basics & Applications of AI/ML for Process Systems Engineering

Submitted by: -

ASHISH KUMAR SINGH

PB22MTECH11003

Question:-1

$$A \xrightarrow{k_1} B$$

$$A \xrightarrow{k_2} C$$

Rate equations:

$$\frac{dC_A}{dt} = -(k_1 C_A + k_2 C_A)$$

$$\frac{dC_B}{dt} = k_1 C_A$$

$$\frac{dC_C}{dt} = k_2 C_A$$

Steps:

- Importing the numpy and pandas library
    - import numpy as np
    - import pandas as pd

- Loading the Data excel file and printing the data
    - data = pd.read_excel("C:\\Users\\ASUD\\Downloads\\data.xlsx")
    - data.shape

- Loading the t_span, concentration of A,B,C from the loaded data and ptinting
    - t_span = np.array(data.iloc[:,0])
    - Ca=np.array(data.iloc[:,1])
    - Cb=np.array(data.iloc[:,2])
    - Cc=np.array(data.iloc[:,3])

- Defining the ODEs function
  - def odes(t,y,*K):

    dydt=np.zeros(3)

    Ca=y[0]

    Cb=y[1]

    Cc=y[2]

    k1=K[0]

    k2=K[1]

    dydt[0]=-(k1*Ca + k2*Ca)

    dydt[1]=k1*Ca

    dydt[2]=k2*Ca

    return (dydt)
- Importing the library
  - from scipy.optimize import minimize
  - import math
  - from scipy.integrate import solve_ivp
  - import matplotlib.pyplot as plt

- Defining the objective function for finding the RMSE
  - def object_func(K):
  - y0 = [1,0,0]
  - t_span = np.array([0,1])
  - t_point= np.linspace(0,1,21)
  -
  - soln = solve_ivp(odes,t_span,y0,t_eval=t_point, args=(K))
  - y= soln.y.T
  -
  - sum1=0
  - sum2=0
  - sum3=0

- for i in np.linspace(0,1,21):
-     i=int(i)
-     sum1=sum1+((Ca[i])-(y[i,0]))**2
-     sum2=sum2+((Cb[i])-(y[i,1]))**2
-     sum3=sum3+((Cc[i])-(y[i,2]))**2
-
- rmse1=((math.sqrt((sum1)))/20)
- rmse2=((math.sqrt((sum2)))/20)
- rmse3=((math.sqrt((sum3)))/20)
- rmse=(rmse1+rmse2+rmse3)/3
- return (rmse)

- Minimizing the objective function by estimating K1 & K2 by using BFGS method **Initial guess of K1 & K2 K0=[0.1,2]**
  - soln1= minimize(object_func, K0, method='TNC')

- Solving the ODEs function by the solve_ivp solver
  - K=soln1.x

  Initial guess of Y1, Y2,Y3 as y0 = [1,0,0]

  t_span = np.array([0,1])

  t_point= np.linspace(0,1,21)

  soln = solve_ivp(odes,t_span,y0,t_eval=t_point,args=(K))

  y= soln.y.T

- Plotting the graph between time and concentration of the A, B,C
  - t=soln.t

  A=soln.y[0]

  B=soln.y[1]

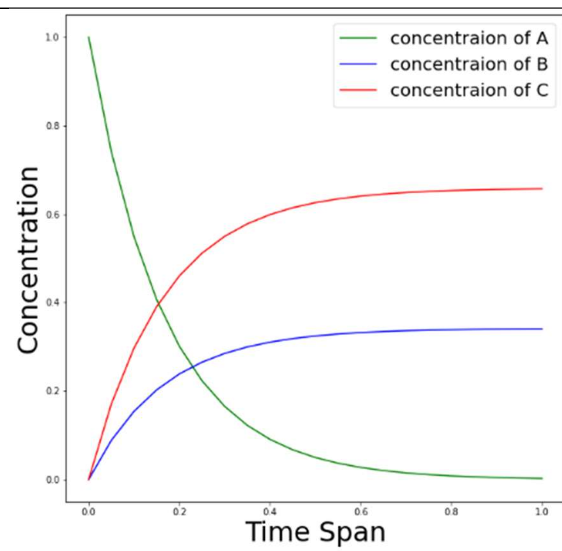
  C=soln.y[2]

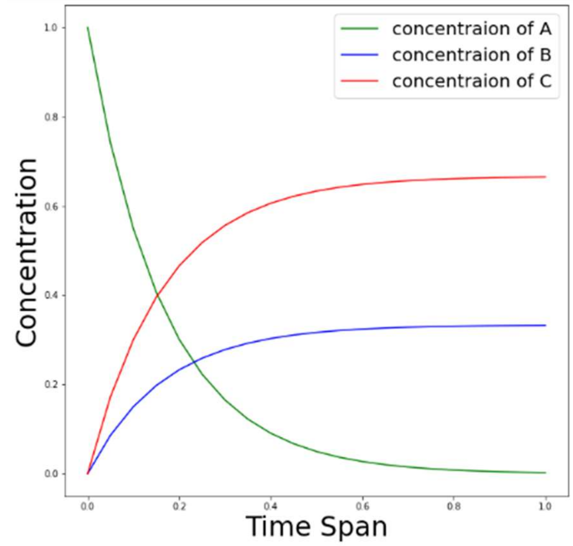
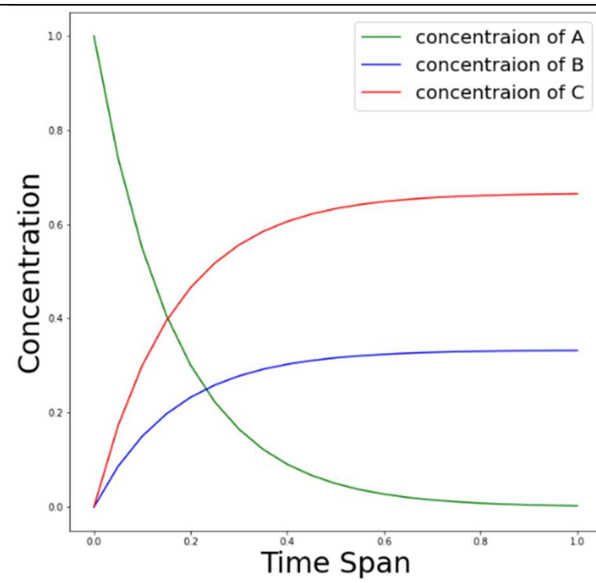
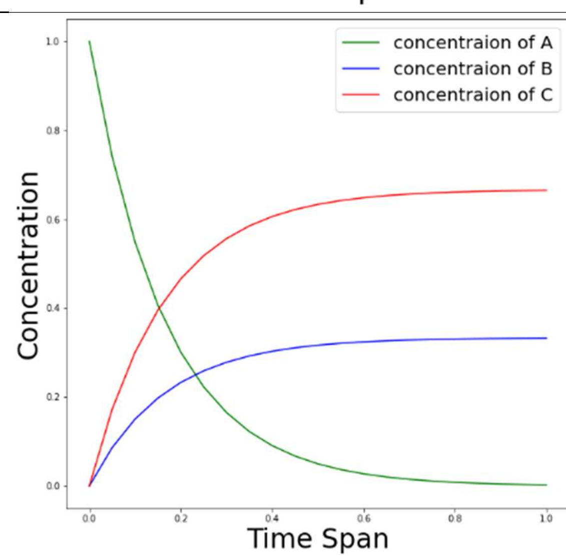
  plt.figure(figsize=(10,10))

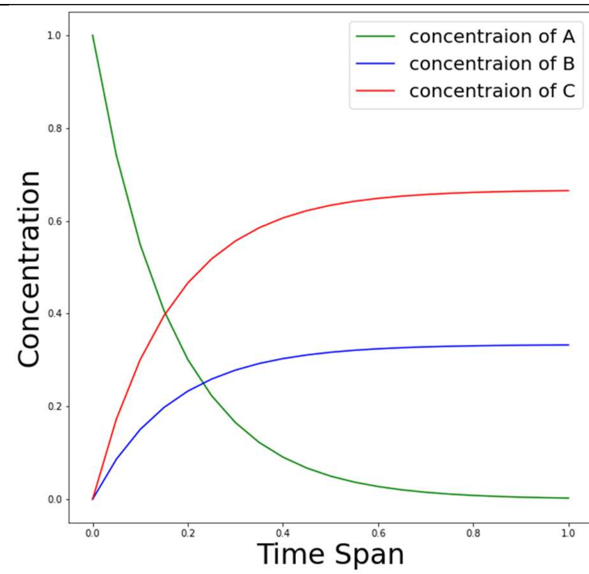
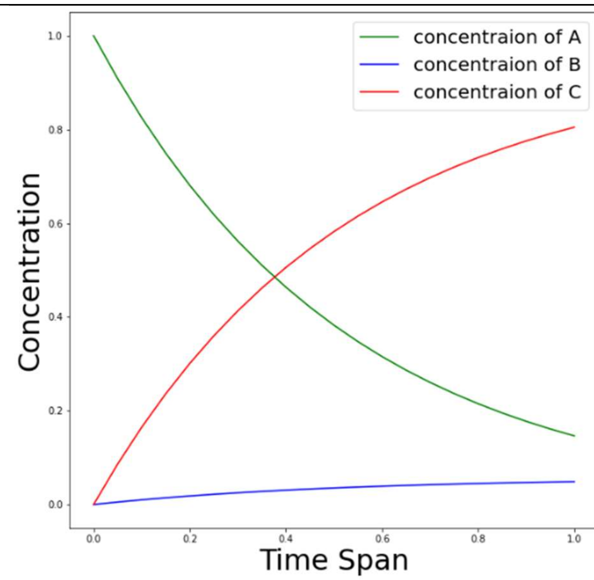
```
plt.plot(t,A,label='concentraion of A',color='green')
plt.plot(t,B,label='concentraion of B',color='blue')
plt.plot(t,C,label='concentraion of C',color='red')
plt.xlabel('Time Span',fontsize=30)
plt.ylabel('Concentration',fontsize=30)
plt.legend(fontsize=20)
plt.show()
```

Observation for the Different optimization Algorithms (Methods)

- After applying different method, it is observed that there is not much change in the value of the X1 and X2

SLSQP (X1=2.04860817) (X2=3.9505 7179)	

Nelder-Mead (X1=1.99970889) (X2=3.99952223)	
Powell (X1=1.99974275) (X2=3.9994855)	
Cobyla (X1=1.99977397) (X2=3.99934162)	

CG (X1=1.99974274) (X2=3.9994855)	
BFGS (X1=1.99974471) (X2=3.9994865)	

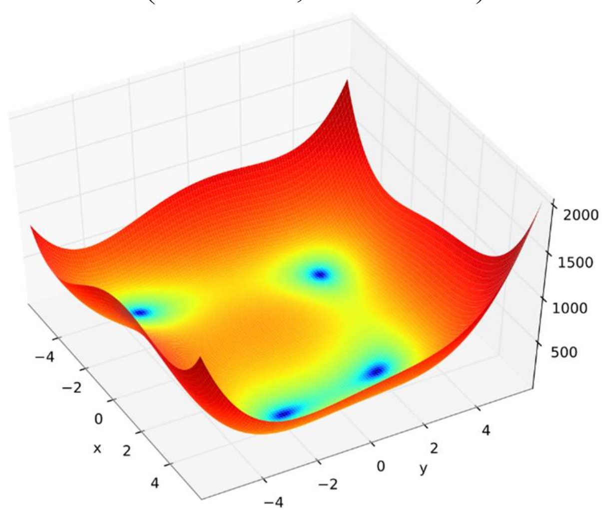
Question:-2

The given test function is Himmelblau

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

The Himmelblau Function has four identical local minimum at:

- $f(x*)=0$ at $x*=(3, 2)$
- $f(x*)=0$ at $x*=(-2.805118, 3.283186)$
- $f(x*)=0$ at $x*=(-3.779310, -3.283186)$
- $f(x*)=0$ at $x*=(3.584458, -1.848126)$



The given Constraint is $x1**2 + x2**2 \geq 25$

$$-5 \leq x1\ ,\ \ x2 \leq 5$$

Python Implementation of Himmelblau Function

((x[0]**2+x[1]-11)**2+(x[0]+x[1]**2-7)**2

SLSQP Optimization Algorithm

Steps:-

- ❖ Defining the Himmelblau function
  - ➢ def fun(x):

    return((x[0]**2+x[1]-11)**2+(x[0]+x[1]**2-7)**2)

- ❖ Importing scipy.optimize
  - ➢ import scipy.optimize as optimize

- ❖ Giving the constraint of the function
  - ➢ constraint=({'type':'ineq','fun':lambda x: x[0]**2+x[1]**2-25})

- ❖ Providing the Upper and Lower bounds of the X1 & X2
  - ➢ bounds=((-5,10),(-10,5))

- ❖ Minimizing fun with initial conditions & using SLSQP Algorithm
  - ➢ res=optimize.minimize(fun,(4,3),method='SLSQP',bounds=bounds,constraints=constraint)

- ❖ Optimize X1 & X2 value
  - ➢ res.x

    X1=-3.77931004 **X2=-3.28318593**

- ❖ Optimize Fun value
  - ➢ Res.fun
  - ➢ **fun value= 2.5289416192976698e-12**

➢ ***Genetic Algorithm***

Steps:-

❖ Importing the library

- ➢ from pymoo.core.problem import Problem
- ➢ import numpy as np

❖ Defining the function and constraint and upper ,lower bound of the X1 and X2

➢ class G1(Problem):

def __init__(self):

n_var1 = 2 **# Number of variable is 2 i.e. X1 & X2**

xlo = np.array([-5, -10], dtype=float) **# lower bound of x1=-5 ,X2=-10**

xu = np.array([10, 5], dtype=float) **# Upper bound of X1=10,X2=5**

super().__init__(n_var=n_var1,n_obj=1, n_ieq_constr=1, xl=xlo, xu=xu, vtype=float)

def _evaluate(self, x, out, *args, **kwargs):

x1 = x[:, 0]

x2 = x[:, 1]

f =(x1**2+x2-11)**2+(x1+x2**2-7)**2

Constraints

g1 =-x1**2-x2**2+25

out["F"] = f

out["G"] = [g1]

❖ Importing the Library

➢ from pymoo.algorithms.soo.nonconvex.ga import GA
➢ from pymoo.optimize import minimize

➢ problem =G1()

algorithm = GA(

pop_size=200,

eliminate_duplicates=True)

res = minimize(problem,

algorithm,

seed=1,

verbose=False)

❖ Printing the value of X1 & X2

➢ res.X

X1=-3.77930249 X2=-3.28315084

❖ Printing the function value
➢ res.fun

function value=5.03080946e-08

Conclusion:- Both the algorithm gives the approximately same value

SLSQP	***X1=*-3.77931004**	***X2=*=-3.28318593**	***Fun=*2.5289416192976698e-12**
GA	***X1=*-3.77930249**	***X2=* -3.28315084**	***Fun=*5.03080946e-08**