WEEK 10 - Python Apllication in Chemical Engineering

Name: Wong Jin Wui

Matric Number: 23005209

Things you can do in python

- Process simulate
- Automation of chemical data analysis
- Chemical reaction and kinetic modelling
- Data mining
- Process control and monitoring

Linear Algebra 1

```
In [6]: # import libraries (https://docs.scipy.org/doc/scipy/reference/linalg.html)
import numpy as np
import scipy as sc
```

$$x + 3y + 5z = 10$$
$$2x + 5y + z = 8$$
$$2x + 3y + 8z = 3$$

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix}$$

$$x + 3y + 5z = 10$$
$$2x + 5y + z = 8$$
$$2x + 3y + 8z = 3$$

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix}$$


```
In [10]: # define the coefficient matrix A
a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
print(a)
```

```
[[1 3 5]
 [2 5 1]
 [2 3 8]]
```

```
In [12]: # Define the right hand side vector to b
b = np.array([[10], [8], [3]])
print (b)
```

```
[[10]
 [ 8]
 [ 3]]
```

```
In [16]: # Calculate the inverse of A and multiply by b to find out the solution further
c = sc.linalg.inv(a).dot(b)
print (c)
```

```
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

```
In [18]: # solve function to find the solution
d = sc.linalg.solve(a, b)
print (d)
```

```
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

Linear Algebra Execution

```
In [23]: a = np.array([[3, 2], [1, 2]])
print (a)
```

```
[[3 2]
 [1 2]]
```

```
In [37]: b = np.array([[1], [0]])
print (b)
```

```
[[1]
 [0]]
```

```
In [39]: # solve function to find the solution
d = sc.linalg.solve(a, b) # longer version but dont need to import for every little funcion
print (d)
```

```
[[ 0.5 ]
 [-0.25]]
```

```
In [47]: solution = solve(a, b)
solution
```

```
Out[47]: array([[ 0.5 ],
       [-0.25]])
```

```
In [43]: # ways to resolve
from scipy.linalg import solve
```

```
In [45]: solution = solve (a, b) # shorter function but need to import for every little function available in library
solution
```

```
Out[45]: array([[ 0.5 ],
       [-0.25]])
```

Solving ODE using Runge-Kutta method

The Runge–Kutta method [edit]

The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method" or simply as "the Runge–Kutta method".

Let an initial value problem be specified as follows:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Here y is an unknown function (scalar or vector) of time t, which we would like to approximate; we are told that $\frac{dy}{dt}$, the rate at which y changes, is a function of t and of y itself. At the initial time t_0 the corresponding y value is y_0. The function f and the initial conditions t_0, y_0 are given.

Now pick a step-size $h > 0$ and define

$$y_{n+1} = y_n + \frac{1}{6} h \left(k_1 + 2k_2 + 2k_3 + k_4\right),$$
$$t_{n+1} = t_n + h$$

for $n = 0, 1, 2, 3, \ldots$, using[2]

$$k_1 = f(t_n, y_n),$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),$$
$$k_4 = f\left(t_n + h, y_n + hk_3\right).$$

The Runge–Kutta method [edit]

The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method" or simply as "the Runge–Kutta method".

Let an initial value problem be specified as follows:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Here y is an unknown function (scalar or vector) of time t, which we would like to approximate; we are told that $\frac{dy}{dt}$, the rate at which y changes, is a function of t and of y itself. At the initial time t_0 the corresponding y value is y_0. The function f and the initial conditions t_0, y_0 are given.

Now pick a step-size $h > 0$ and define

$$y_{n+1} = y_n + \frac{1}{6} h \left(k_1 + 2k_2 + 2k_3 + k_4\right),$$
$$t_{n+1} = t_n + h$$

for $n = 0, 1, 2, 3, \ldots$, using[2]

$$k_1 = f(t_n, y_n),$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),$$
$$k_4 = f\left(t_n + h, y_n + hk_3\right).$$

differential eq: dy/dx = (x-y)/2 given: x0, y0, h(step size)

```
In [81]: # def differential equation
def dydx(x, y):
    return ((x-y)/2)
```

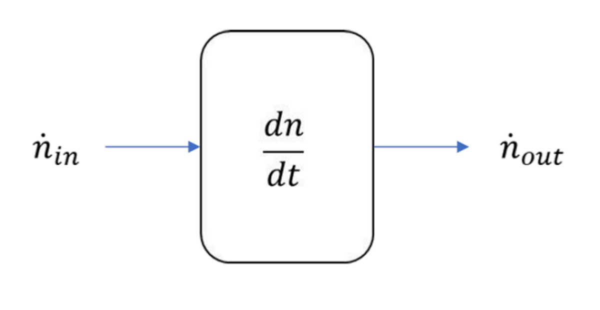
```
In [91]: def rungeKutta(x0, y0, x, h):
    n = int((x-x0)/h)
    y = y0
    for i in range (1, n+1):
        k1 = h * dydx(x0,y)
        k2 = h * dydx(x0 + 0.5*h, y+0.5*k1)
        k3 = h * dydx(x0 + 0.5*h, y+0.5*k2)
        k4 = h* dydx(x0 + h, y + k1)
        # update value of y
        y = y + (1/6)*(k1 + 2*k2 + 2*k3 + k4)
        # update value of x
        x0 = x0 + h

    return y
```

```
In [93]: # driver method
x0 = 0
y = 1
x = 5
h = 0.2
print ("The value of y at x is:", rungeKutta(x0, y, x, h))
```

```
The value of y at x is: 3.246794758464962
```

Pressure profile in vessel (simulation)



Mass balance in the vessel:

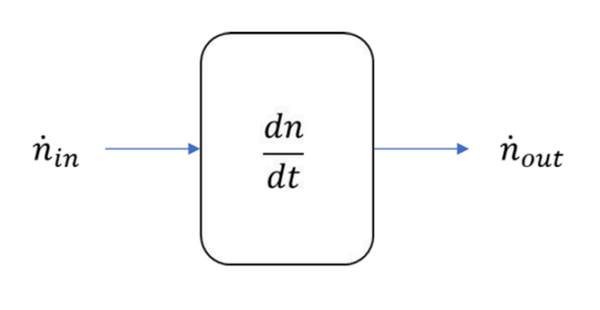
$$\frac{dn}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$PV = nRT$$

$$\frac{d\left(\frac{PV}{RT}\right)}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$\frac{V}{RT}\frac{dP}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$\frac{dP}{dt} = \frac{\dot{n}_{in} - \dot{n}_{out}}{\frac{V}{RT}}$$



Mass balance in the vessel:

$$\frac{dn}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$PV = nRT$$

$$\frac{d\left(\frac{PV}{RT}\right)}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$\frac{V}{RT}\frac{dP}{dt} = \dot{n}_{in} - \dot{n}_{out}$$

$$\frac{dP}{dt} = \frac{\dot{n}_{in} - \dot{n}_{out}}{\frac{V}{RT}}$$


```
In [96]: # import library
import matplotlib.pyplot as plt
```

```
In [100… # parameter values for original pressure
V = 1154 #m3
R = 8.314 # R[J/(mol*K)]
T = 120 # degree celcius

mass_in = 473220 # kg/h
MW_in = 56.6
mass_out = 28742 # kg/h
MW_out = 40.4
```

```
In [105… # conversion and calculation
Tk = T + 273.15 # convert to Kelvin
mol_in = mass_in/MW_in/60 # kmol/min
mol_out = mass_out/MW_out/60 # kmol/min
```

```
In [107… # Initial condition
P0 = 1830 #kPa
```

```
In [109… # differential equation using ideal gas
def dpdt(P, t):
    dpdt = (mol_in - mol_out)/(V/(R*Tk))
    return dpdt
```

```
In [111… # time to response is 40 minutes
start = 0
end = 30
t = np.linspace(start, end, end)
Pinitial = np.linspace(P0, P0, end)
```

```
In [115… # integrate the differential equation over 30 minutes to compute the pressure
p = sc.integrate.odeint(dpdt, P0, t)
```

```
In [125… # Plot the result
plt.figure()
plt.plot(t, p, "r", label = "Pressure Increases")
plt.plot(t, Pinitial, "b--", label = "Original Pressure", linewidth=2)
plt.title(f"System Pressure Profile for total Volume of {V} m^3")
plt.xlabel("Time (min)")
plt.ylabel("Pressure (kPa)")
plt.grid(True, which = "both", linestyle = '--', linewidth = 0.5)
plt.minorticks_on()
plt.legend()
plt.show()
```


```
In [127… # zombie apocalypse modeling
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
plt.ion()
plt.rcParams['figure.figsize'] = 10, 8

P = 0       # birth rate
d = 0.0001  # natural death percent (per day)
B = 0.0095  # transmission percent  (per day)
G = 0.0001  # resurect percent (per day)
A = 0.0001  # destroy percent  (per day)

# solve the system dy/dt = f(y, t)
def f(y, t):
     Si = y[0]
     Zi = y[1]
     Ri = y[2]
     # the model equations (see Munz et al. 2009)
     f0 = P - B*Si*Zi - d*Si
     f1 = B*Si*Zi + G*Ri - A*Si*Zi
     f2 = d*Si + A*Si*Zi - G*Ri
     return [f0, f1, f2]

# initial conditions
S0 = 500.              # initial population
Z0 = 0                 # initial zombie population
R0 = 0                 # initial death population
y0 = [S0, Z0, R0]     # initial condition vector
t  = np.linspace(0, 5., 1000)         # time grid

# solve the DEs
soln = odeint(f, y0, t)
S = soln[:, 0]
Z = soln[:, 1]
R = soln[:, 2]

# plot results
plt.figure()
plt.plot(t, S, label='Living')
plt.plot(t, Z, label='Zombies')
plt.xlabel('Days from outbreak')
plt.ylabel('Population')
plt.title('Zombie Apocalypse - No Init. Dead Pop.; No New Births.')
plt.legend(loc=0)

# change the initial conditions
R0 = 0.01*S0   # 1% of initial pop is dead
y0 = [S0, Z0, R0]

# solve the DEs
soln = odeint(f, y0, t)
S = soln[:, 0]
Z = soln[:, 1]
R = soln[:, 2]

plt.figure()
plt.plot(t, S, label='Living')
plt.plot(t, Z, label='Zombies')
plt.xlabel('Days from outbreak')
plt.ylabel('Population')
plt.title('Zombie Apocalypse - 1% Init. Pop. is Dead; No New Births.')
plt.legend(loc=0)

# change the initial conditions
R0 = 0.01*S0   # 1% of initial pop is dead
P  = 10        # 10 new births daily
y0 = [S0, Z0, R0]

# solve the DEs
soln = odeint(f, y0, t)
S = soln[:, 0]
Z = soln[:, 1]
R = soln[:, 2]

plt.figure()
plt.plot(t, S, label='Living')
plt.plot(t, Z, label='Zombies')
plt.xlabel('Days from outbreak')
plt.ylabel('Population')
plt.title('Zombie Apocalypse - 1% Init. Pop. is Dead; 10 Daily Births')
plt.legend(loc=0)
```

```
Out[127… <matplotlib.legend.Legend at 0x2074b7f9b80>
```