ASSIGNMENT 1

Q: Here in this assignment, we deal with **Stokes Law**.

We have to solve the given ODE numerically in Python and verify it with the value calculated after approximation of terminal velocity.

Stokes Law: $$\frac{dv}{dt} = \frac{(\rho_s - \rho_l)}{\rho_s} g - \frac{9}{2} \frac{\eta v}{\rho_s r^2}$$

And Terminal velocity: $$v_t = \frac{2}{9} r^2 g (\frac{\rho_s - \rho_l}{\eta})$$

Now here comes the Procedure to solve the problem

1. Take density of object, density of liquid, viscosity coeffcient and radius of object from user as input.

```
ps = float(input("Enter the density of object: ")) // 1080
pl = float(input("Enter the density of liquid: "))  // 1000
n = float(input("Enter the viscosity coefficient: ")) // 1.0016
r = float(input("Enter the radius of object: "))   // 0.08
g = 9.8
```

2. Take Δ t as 0.01
3. Also the tolerance is 10^{-9}
4. first take v_{prev} as 0 and error as 1

```
delt = 0.01
tol = 1e-9
error = 1
v_prev = 0
```

5. Then keep iteration till error>tol:
   - Calculate $\frac{dv}{dt} = \frac{(\rho_s - \rho_l)}{\rho_s} g - \frac{9}{2} \frac{\eta v}{\rho_s r^2}$
   - then update $v_{next} = v_{prev} + \frac{dv}{dt} * \Delta t$
   - update error to : $(v_{next} - v_{prev})/v_{next}$
   - Also now $v_{prev} = v_{next}$ and return to the loop

```
while (error > tol) :
    dv_dt = ((ps-pl)/ps)*g - (9/2)*((n*v_prev)/(ps*r*r))
    v_next = v_prev + (dv_dt * delt)
    error = (v_next - v_prev)/v_next
    v_prev = v_next
```

6. After the loop the final velocity is v_{next} (This is the estimated terminal velocity)

7. Calculate terminal velocity from the formula $v_t = \frac{2}{9} r^2 g (\frac{\rho_s - \rho_l}{\eta})$

8. The percentage error would be : $[(v_t - v_{next})/v_t] * 100$

```
print("The velocity is: ",v_next)

v_t = (2*r*r*g*(ps-pl))/(9*n)
print("The terminal velocity is: ",v_t)

err = ((v_t - v_next)/v_t)*100
print("The error is: ",err,"%")
```

Result :

The velocity is: 1.1132408676660517

The terminal velocity is: 1.1132410365637204

The error is: 1.5171707042086206e-05 %

Inference

The velocity calculated from differential equation converges to terminal velcocity at some tolerance level

ASSIGNMENT 2

Q: Heat conduction Equation (Partial Differential Equation)

$$\rho C_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(K\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(K\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(K\frac{\partial T}{\partial z}\right) + \dot{e}_{gen}$$

-- Heat conduction equation in cartesian form

Note : If we assume K =constant , the conduction equation reduces to

$$\nabla^2 T + \frac{\dot{e}_{gen}}{K} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \qquad \text{where } \alpha = \frac{k}{\rho Cp} = \text{Thermal Diffusivity}$$

Now we solve T using Partial Differential equation solve in ESO208 by making rectangular grid.

$$T_{x,y}^{k+1} = \gamma\,(T_{x+1,y}^{k} + T_{x-1,y}^{k} + T_{x,y+1}^{k} + T_{x,y-1}^{k} - 4T_{x,y}^{k}) + T_{x,y}^{k}$$

Where $\gamma = \frac{\alpha \Delta t}{\Delta x^2}$ and we assume Δx = Δy

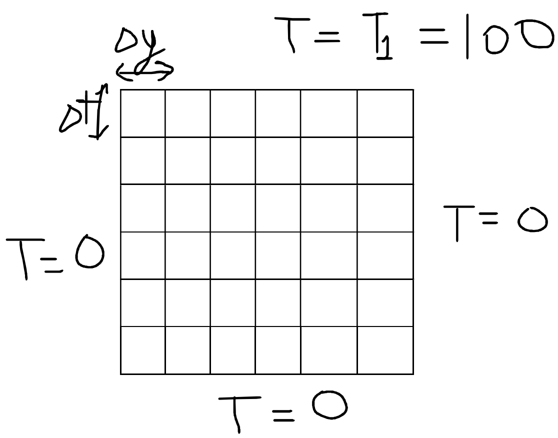
Δx = 1 , $\alpha = 2$, Δt $= 0.1$, Δy $= 1$, x $=$ 50m , y $=$ 50m

Get the evaluation of temperature for t_{iter} = 500 steps

Initially(at t=0) , Temperature of the whole plate = 0

Take:

a. Right edge Boundary condition : T = 0
b. Right edge Boundary condition : T = 50



Procedure

1. First we import all the important libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
```

2. Then we define all the boundary conditions given in the questions

```
plate_length = 50
t_iter = 500

alpha = 2
delta_x = 1

delta_t = (delta_x ** 2)/(4 * alpha)
gamma = (alpha * delta_t) / (delta_x ** 2)
t = np.empty((t_iter, plate_length, plate_length))

t_initial = 0

t_top = 100.0
t_left = 0.0
t_bottom = 0.0
t_right = 0.0
# t_right=50.0

t.fill(t_initial)

t[:, (plate_length-1):, :] = t_top
t[:, :, :1] = t_left
t[:, :1, 1:] = t_bottom
t[:, :, (plate_length-1):] = t_right
```

3. Then we make a function that calculate from $T_{x,y}^{k+1}$

$$T_{x,y}^{k+1} = \gamma\,(T_{x+1,y}^{k} + T_{x-1,y}^{k} + T_{x,y+1}^{k} + T_{x,y-1}^{k} - 4T_{x,y}^{k}) + T_{x,y}^{k}$$

at each iterations

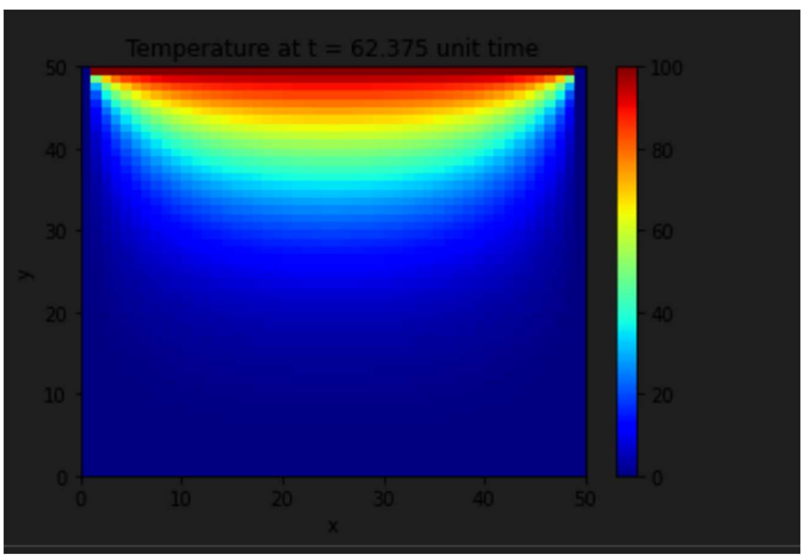
```
def calculate(t):
    for k in range(0, t_iter-1, 1):
        for i in range(1, plate_length-1, delta_x):
            for j in range(1, plate_length-1, delta_x):
                t[k + 1, i, j] = gamma * (t[k][i+1][j] +t[k][i-1][j] +
t[k][i][j+1] + t[k][i][j-1] - 4*t[k][i][j]) + t[k][i][j]
    return t
```

4. Lastly calculate the final temperature and plot t v/s x,y

```
def plotheatmap(t_k, k):
    plt.clf()

    plt.title(f"Temperature at t = {k*delta_t:.3f} unit time")
    plt.xlabel("x")
    plt.ylabel("y")

    plt.pcolormesh(t_k, cmap=plt.cm.jet, vmin=0, vmax=100)
    plt.colorbar()

    return plt

t = calculate(t)
```

5. Lastly we can animate our plot at each iteration using animation function in python

```
def animate(k):
    plotheatmap(t[k], k)
anim = animation.FuncAnimation(plt.figure(), animate, interval=1,
frames=t_iter, repeat=False)
anim.save("heat_equation.gif")
```



Note : The animation is in the link given below –

https://drive.google.com/file/d/1iuZrypxczg9B47jcZoHGC7RCAzPu1AXz/view?usp=sharing

ASSIGNMENT 3

Co-current Flow :

Energy balance: Accumulation = In - Out + Generation

- For inner Cylinder $\frac{dT_1}{dt} = \frac{m_1 * C_{p1} * (T_1(i-1) - T_1(i)) + U * 2\pi r_1 * dx * (T_2(i) - T_1(i))}{\rho_1 * C_{p1} * A_1 * dx}$
- For Outer cylinder $\frac{dT_2}{dt} = \frac{m_2 * C_{p2} * (T_2(i-1) - T_2(i)) + U * 2\pi r_1 * dx * (T_2(i) - T_1(i))}{\rho_2 * C_{p2} * A_2 * dx}$

Q: Solve, and obtain the transient response of Temperature with time for the concentric cylinder double pipe heat exchanger, as shown above.

Details:

1) Length of pipe = L = 60 m
2) Inner radius = r1 = 0.1 m
3) Outer radius = r2 = 0.15 m
4) Number of internal points = n = 100 (Can increase this for better accuracy)
5) For fluid 1 (Water here):
    1) m1 = Mass flow rate = 3 kg/s
    2) Cp1 = Heat capacity of fluid (water) = 4180 J/kg.K
    3) rho1 = Density of fluid (water) = 1000 kg/m^3
6) For fluid 2 (Water here again):
    1) m2 = Mass flow rate = 5 kg/s
    2) Cp2 = Heat capacity of fluid (water) = 4180 J/kg.K
    3) rho2 = Density of fluid (water) = 1000 kg/m^3
7) Initial temperature of fluid throughout the pipe = T0 = 300K
8) Inlet temperature of fluid 1 = T1i = 400 K
9) Inlet temperature of fluid 2 = T2i = 800 K
10) Overall heat transfer coefficient = U = 340 W/m^2

Simulate for t_final = 1000 seconds, with a time step (Δt) of 1 sec for each step.

For each time step, get the temperature profile (T1 and T2 for the whole pipe) and plot them in a single figure. Clear the figure, and update that plot with the next figure (next time step).

Procedure

1. First we import all the important libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
```

2. We take all inputs given in the question

```
l = 60 #length of pipe
r1 = 0.1 # inner radius
r2 = 0.15 # outer radius
n = 100 # number of internal points (take n = 500 also)
pi = 3.14
a1 = pi*np.square(r1) # area of cylinder 1
a2 = pi*(np.square(r2) - np.square(r1)) # area of cylinder 2
m1 = 3 # mass flow rate (fluid 1)
cp1 = 4180 # heat capacity of water (fluid 1)
d1 = 1000 # density of water (fluid 2)
m2 = 5 # mass flow rate (fluid 2)
cp2 = 4180 # heat capacity of water (fluid 2)
d2 = 1000 # density of water (fluid 2)
```

3. Then take segment of dx = 1/n and also make an array of value = T_0 for T_1 and T_2

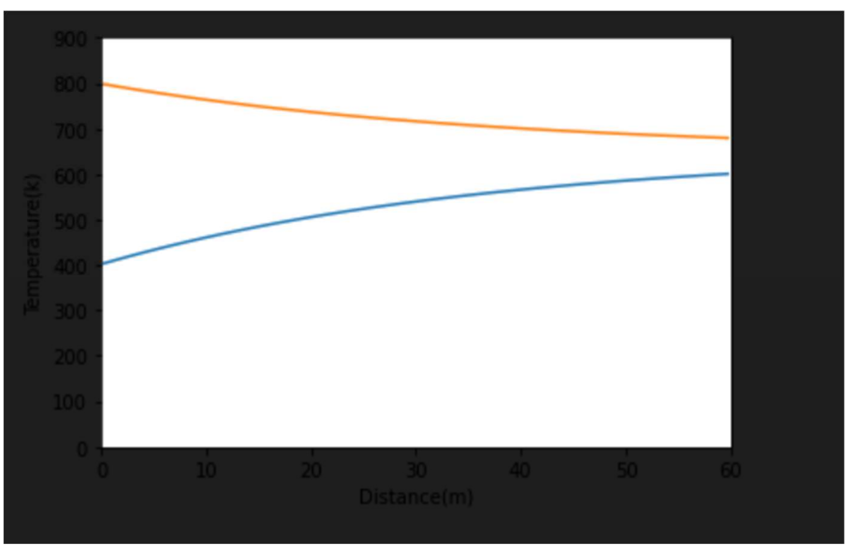
```
dx = l/n
t_final = 1000
dt = 1

t0 = 300 # initial temperature of fluid
t1i = 400 # inlet temperature of fluid 1
t2i = 800 # inlet temperature of fluid 2
U = 340 # overall heat transfer coefficient

x = np.linspace(dx/2,l-dx/2,n) # take evenly spaced numbers array from dx/2 to
l-dx/2 of size n
T1 = np.ones(n)*t0 # create array of size n with all value = t0
T2 = np.ones(n)*t0
dT1_dt = np.zeros(n) # create array of zeros of size n
dT2_dt = np.zeros(n)
Tin = np.zeros((t_final,n))
Tout = np.zeros((t_final,n))
t = np.arange(0,t_final,dt)
```

4. Now transverse the formula for $\frac{dT_1}{dt}$ and $\frac{dT_2}{dt}$ in for loop iterated up to given no. of iterations.

```
for i in range(1,len(t)):
    dT1_dt[1:n] = ( m1*cp1*(T1[0:n-1]-T1[1:n])+ U*2*pi*r1*dx*(T2[1:n]-T1[1:n])
) / ( d1*cp1*dx*a1 )
    dT1_dt[0] = ( m1*cp1*(t1i-T1[0])+ U*2*pi*r1*dx*(T2[0]-T1[0]) ) / (
d1*cp1*dx*a1 )
    dT2_dt[1:n] = ( m2*cp2*(T2[0:n-1]-T2[1:n])- U*2*pi*r1*dx*(T2[1:n]-T1[1:n])
) / ( d2*cp2*dx*a2 )
    dT2_dt[0] = ( m2*cp2*(t2i-T2[0])- U*2*pi*r1*dx*(T2[0]-T1[0]) ) / (
d2*cp2*dx*a2 )
    T1 = T1+dT1_dt*dt
    T2 = T2+dT2_dt*dt
    Tin[i,:] = T1
    Tout[i,:] = T2
```

5. Finally, we get the T_{in} and T_{out}. Now we plot a graph between distance(x) [x-axis] and T_{in} and T_{out} [y-axis].

```
def plotheatmap(Tin,Tout):
    plt.clf()
    plt.plot(x,Tin)
    plt.plot(x,Tout)
    plt.xlabel('Distance(m)')
    plt.ylabel('Temperature(k)')
    plt.axis([0,l,0,900])
    return plt
```



6. Also we can animate the plot at each iteration by using Function animation that is inbuilt in python.

```
def animate(j):
    plotheatmap(Tin[j,:],Tout[j,:])
anim = animation.FuncAnimation(plt.figure(), animate, interval=dt,
frames=t_final, repeat=False)
anim.save("answer.gif")
```

Note : The animation is in the link given below -
https://drive.google.com/file/d/1BO7QLtuFwSaAyiqMOTZWl4rLde6Mq0HL/view