# 1 Methods

We will use the Upwind scheme for the spatial discretization. The conditions for the Upwind scheme are listed below:

$$\phi_{i+1} = \begin{cases} \phi_I & \overline{V} > 0 \\ \phi_{I+1} & \overline{V} < 0 \end{cases}, \tag{1}$$

and

$$\left(\frac{\partial \phi}{\partial x}\right)_{i-1} = \frac{\phi_I - \phi_{I-1}}{\Delta x}. \tag{2}$$

To set up the temporal discretization, we begin with the transient convection-diffusion equation.

$$\int\limits_{\Delta t} \frac{\partial}{\partial t}\left(\int\limits_{CV} \rho\phi dV\right) dt + \int\limits_{\Delta t}\int\limits_{S} \rho u_x \phi \cdot \overline{n}\, dS\, dt = \int\limits_{\Delta t}\int\limits_{S} \Gamma \frac{\partial \phi}{\partial x} \cdot \overline{n}\, dS\, dt \tag{3}$$

We can apply the surface integral approximations to get:

$$\int\limits_{\Delta t} \frac{\partial}{\partial t}\left(\int\limits_{CV} \rho\phi dV\right) dt + \int\limits_{\Delta t} \left[\rho u_x S\left(\phi_{i+1} - \phi_{i-1}\right)\right]\, dt = \int\limits_{\Delta t} \Gamma S \left[\left(\frac{\partial \phi}{\partial x}\right)_{i+1} - \left(\frac{\partial \phi}{\partial x}\right)_{i-1}\right] dt\,.$$

Performing the rest of the integrations, we get:

$$\rho V\left(\phi_I^{n+1} - \phi_I^n\right) + \rho u_x S\left(\phi_{i+1} - \phi_{i-1}\right)\, \Delta t = \Gamma S \left[\left(\frac{\partial \phi}{\partial x}\right)_{i+1} - \left(\frac{\partial \phi}{\partial x}\right)_{i-1}\right] \Delta t\,. \tag{4}$$

We can apply Equation (1) and Equation (2) for positive velocity. We can also divide by the surface to get:

$$\rho\, \Delta x\left(\phi_I^{n+1} - \phi_I^n\right) + \rho u_x \Delta t\left(\phi_I^X - \phi_{I-1}^X\right) = \Gamma\, \Delta t \left[\left(\frac{\phi_{I+1}^X - \phi_I^X}{\Delta x}\right) - \left(\phi_I^X - \phi_{I-1}^X\right)\right]. \tag{5}$$

We have three different time discretizations for this problem: Explicit Euler, Implicit Euler, and Trapezoidal. We can use Equation (5) for each scheme's inner nodes, but the boundary nodes will need to use Equation (4) since they have different gradients.

## 1.1 Explicit Euler

Explicit Euler uses the substitution: $\phi^X = \phi^n$. Inner Nodes:

$$\phi_I^{n+1} = \left[\frac{u_x\, \Delta t}{\Delta x} + \frac{\Gamma\, \Delta t}{\rho\, (\Delta x)^2)}\right] \phi_{I-1}^n + \left[1 - \frac{u_x\, \Delta t}{\Delta x} - \frac{2\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_I^n + \left[\frac{\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_{I+1}^n \tag{6}$$

Left Node (Node 1): $\phi_{i-1} = \phi_L$ and $\left(\frac{\partial \phi}{\partial x}\right)_{i-1} = \frac{\phi_I - \phi_L}{\Delta x/2}$

$$\phi_I^{n+1} = \left[\frac{u_x\, \Delta t}{\Delta x}\right] \phi_L + \left[1 - \frac{u_x\, \Delta t}{\Delta x} - \frac{3\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_I^n + \left[\frac{\Gamma\, \Delta t}{\Delta x}\right] \phi_{I+1}^n \tag{7}$$

Right Node (Node N): $\phi_{i+1} = \phi_R$ and $\left(\frac{\partial \phi}{\partial x}\right)_{i+1} = \frac{\phi_R - \phi_I}{\Delta x/2}$

$$\phi_I^{n+1} = \left[\frac{u_x\, \Delta t}{\Delta x} + \frac{\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_{I-1}^n + \left[1 - \frac{u_x\, \Delta t}{\Delta x} - \frac{3\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_I^n + \left[\frac{2\Gamma\, \Delta t}{\rho\, (\Delta x)^2}\right] \phi_R \tag{8}$$

## 1.2 Implicit Euler

Implicit Euler uses the substitution: $\phi^X = \phi^{n+1}$. Inner Nodes:

$$\left[\frac{-u_x\,\Delta t}{\Delta x} - \frac{\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_{I-1}^{n+1} + \left[1 + \frac{u_x\,\Delta t}{\Delta x} + \frac{2\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_I^{n+1} + \left[-\frac{\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_{I+1}^{n+1} = \phi_I^n \quad (9)$$

Left Node (Node 1): $\phi_{i-1} = \phi_L$ and $\left(\frac{\partial\phi}{\partial x}\right)_{i-1} = \frac{\phi_I - \phi_L}{\Delta x/2}$

$$\left[1 + \frac{u_x\,\Delta t}{\Delta x} + \frac{3\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_I^{n+1} + \left[-\frac{\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_{I+1}^{n+1} = \phi_I^n + \left[\frac{u_x\,\Delta t}{\Delta x} + \frac{2\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_L \quad (10)$$

Right Node (Node N): $\phi_{i+1} = \phi_R$ and $\left(\frac{\partial\phi}{\partial x}\right)_{i+1} = \frac{\phi_R - \phi_I}{\Delta x/2}$

$$\left[-\frac{u_x\,\Delta t}{\Delta x} - \frac{\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_{I-1}^{n+1} + \left[1 + \frac{u_x\,\Delta t}{\Delta x} + \frac{3\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_I^{n+1} = \phi_I^n + \left[\frac{2\Gamma\,\Delta t}{\rho\left(\Delta x\right)^2}\right]\phi_R \quad (11)$$

## 1.3 Trapezoidal

$$\begin{aligned}\left[-\frac{\rho u_x\,\Delta t}{2} - \frac{\Gamma\,\Delta t}{2\,\Delta x}\right]\phi_{I-1}^{n+1} + \left[\rho\,\Delta x + \frac{\rho u_x\,\Delta t}{2} + \frac{\Gamma\,\Delta t}{\Delta x}\right]\phi_I^{n+1} + \left[-\frac{\Gamma\,\Delta t}{2\,\Delta x}\right]\phi_{I+1}^{n+1} \\ = \left[\frac{\rho u_x\,\Delta t}{2} + \frac{\Gamma\,\Delta t}{2\,\Delta x}\right]\phi_{I-1}^n + \left[\rho\,\Delta x - \frac{\rho u_x\,\Delta t}{2} - \frac{\Gamma\,\Delta t}{\Delta x}\right]\phi_I^n + \left[\frac{\Gamma\,\Delta t}{2\,\Delta x}\right]\phi_{I+1}^n\end{aligned}$$

# 2 Results

## 2.1 Explict Euler

Figures 1, 2, and 3 show the results for the Explicit Euler solution using $K = 0.2, 2.0$, and 20.0, respectively. Table 1 shows the norm error for all Explicit Euler cases.

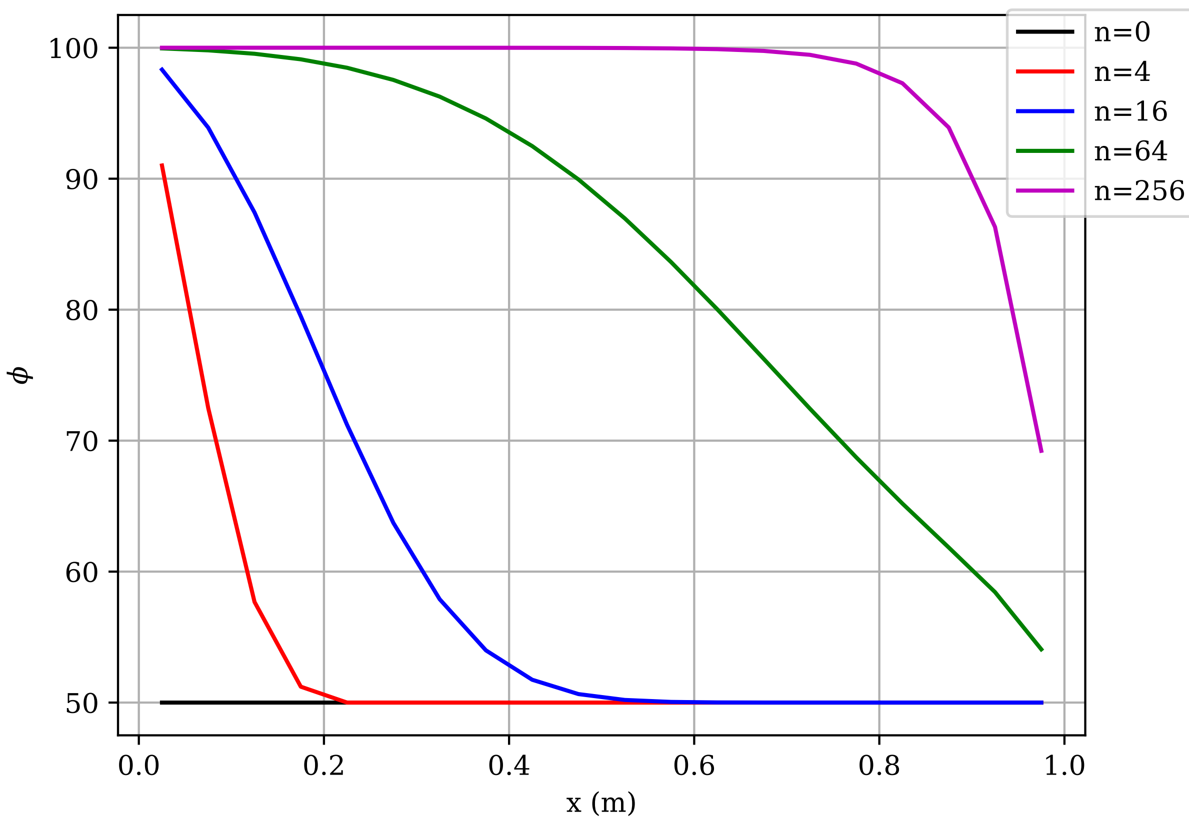


Figure 1: Explicit Euler solution for case 1: $K = 0.2$.

Norm:

Table 1: Norm values for Explicit Euler.

| $K$ | Norm |
|---|---|
| 0.2 | 1.55418029575927 |
| 2.0 | 8.3196861106867e+245 |
| 20.0 | inf |

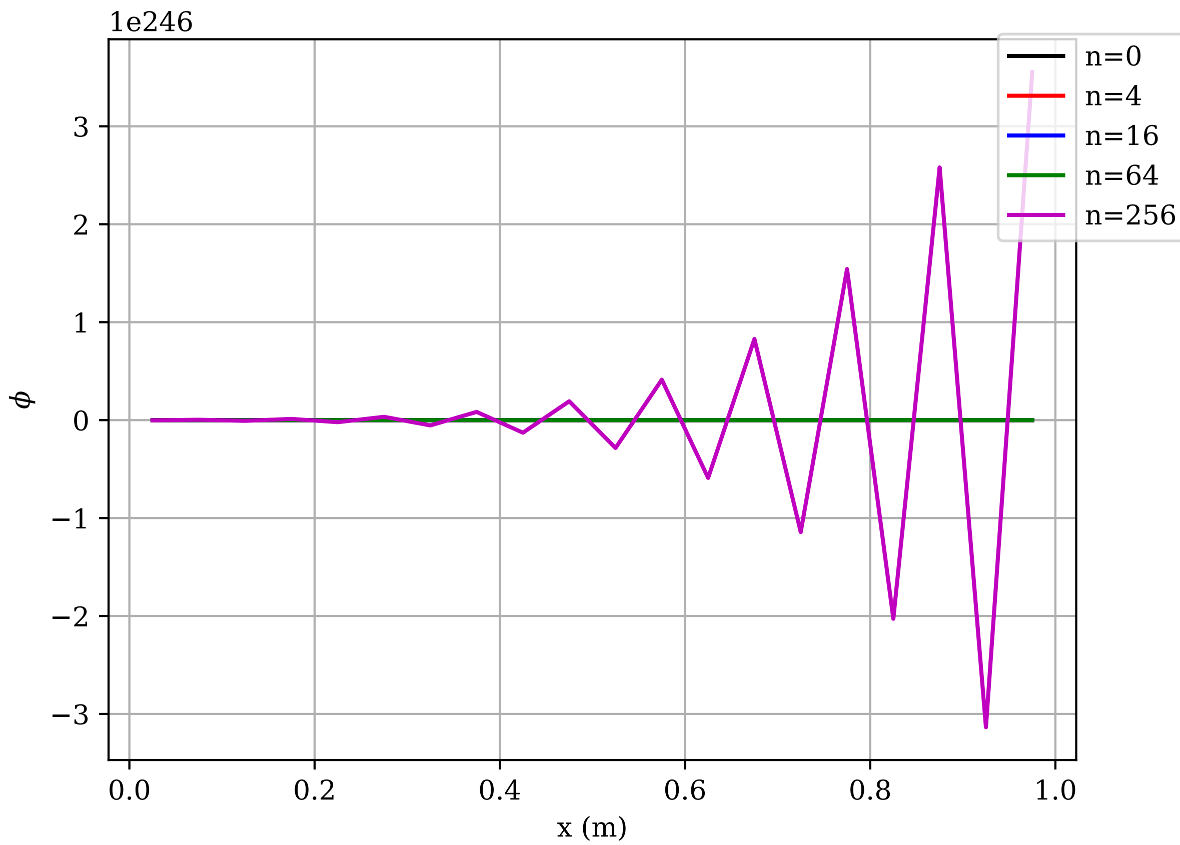


Figure 2: Explicit Euler solution for case 2: $K = 2.0$.

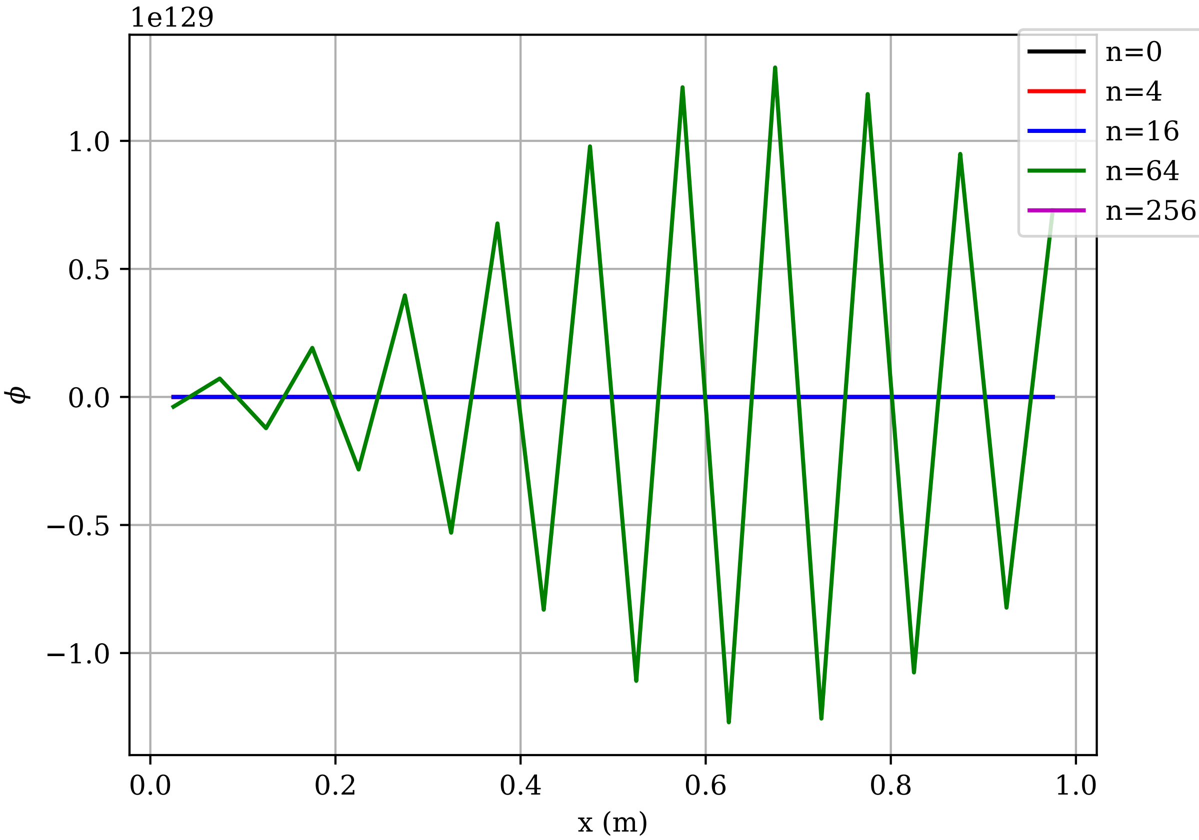


Figure 3: Explicit Euler solution for case 3: $K = 20.0$.

## 2.2 Implicit Euler

Figures 4, 5, and 6 show the results for the Explicit Euler solution using $K = 0.2, 2.0$, and 20.0, respectively. Table 2 shows the norm error for all Explicit Euler cases.

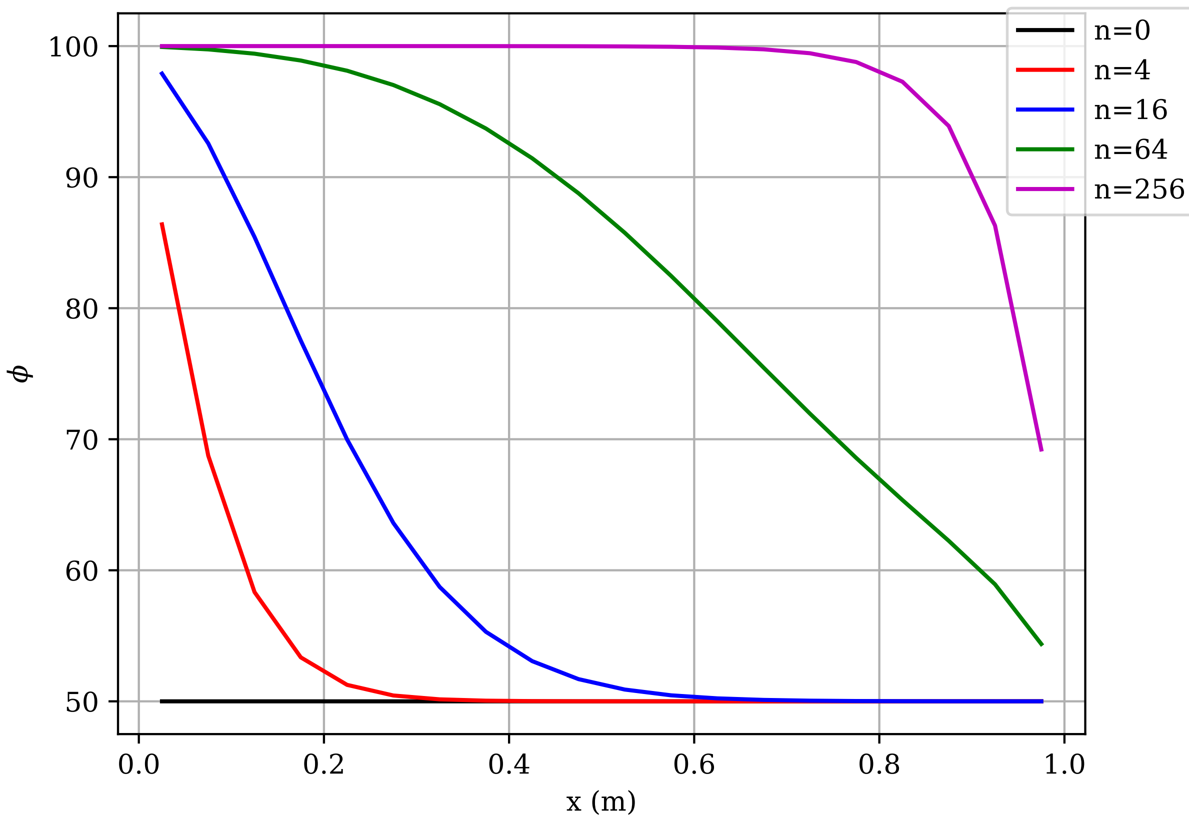


Figure 4: Implicit Euler solution for case 1: $K = 0.2$.

Norm:

Table 2: Norm values for Implicit Euler.

| $K$ | Norm |
|---|---|
| 0.2 | 1.5567368462357045 |
| 2.0 | 1.5504768792236276 |
| 20.0 | 1.5504768792236157 |

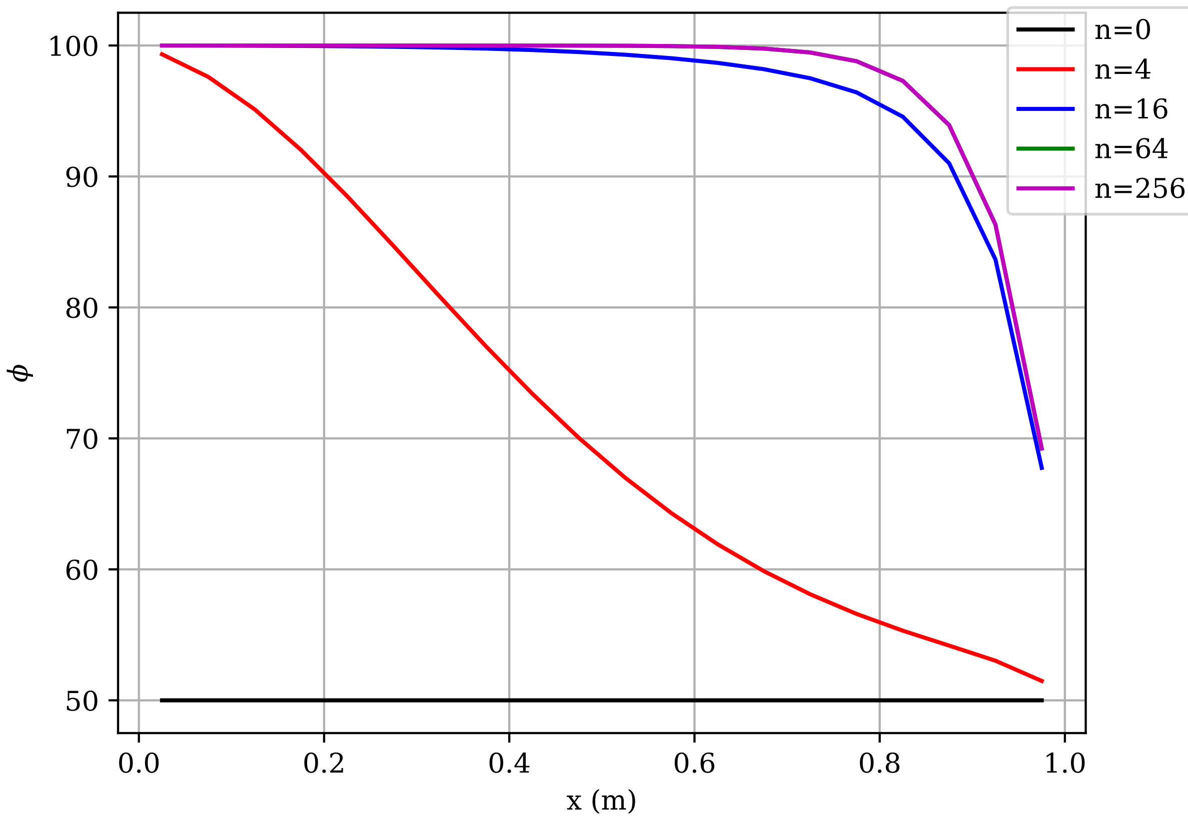


Figure 5: Implicit Euler solution for case 2: $K = 2.0$.

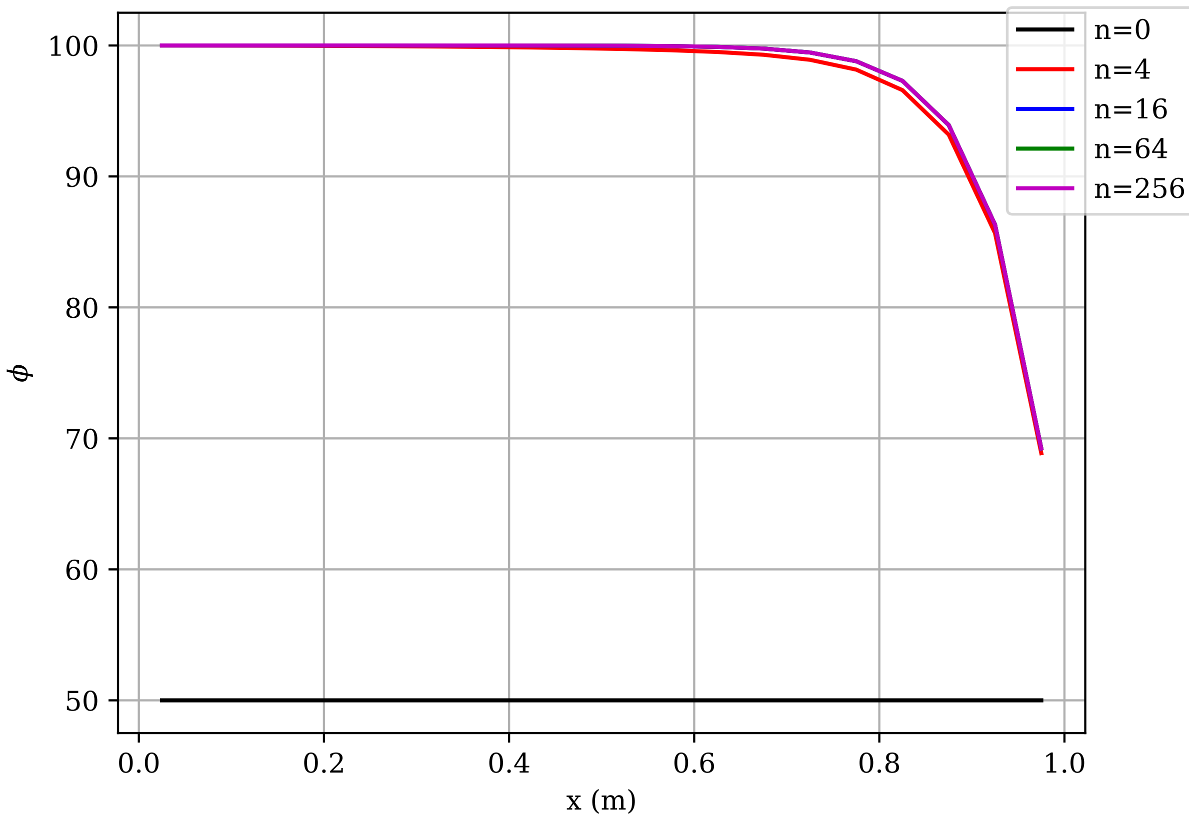


Figure 6: Implicit Euler solution for case 3: $K = 20.0$.

# 3 Discussion

# 4 Code

```
# ----------------------------------#
# -- main script for NSE 565 HW2 --#
# --------- Austin Warren ---------#
# ---------- Winter 2022 ----------#
# ----------------------------------#

import numpy as np
import matplotlib.pyplot as plt

# define functions for spatial and temporal discretizations
# - coefficient generation
# - time step

def UDS_positve(num_volumes, tot_length, density, velocity, diffusion, left, right):
    """ Upwind scheme solver for positive velocity
    """
    dx = tot_length/num_volumes
    phi = np.zeros(num_volumes)
    A = np.zeros((num_volumes,num_volumes))
    Q = np.zeros(num_volumes)



    for j in range(num_volumes):

        if j == 0:
            A[j,0] = density*velocity + 2*diffusion/dx
            A[j,1] = diffusion/dx
            Q[j] = density*velocity*left + diffusion*left/dx

        elif j == num_volumes-1:
            A[j,j-1] = -diffusion/dx
            A[j,j] = -density*velocity - 2*diffusion/dx
            Q[j] = -density*velocity*right + diffusion*right/dx

        else:
            A[j,j-1] = -density*velocity - diffusion/dx
            A[j,j] = density*velocity + 2*diffusion/dx
            A[j,j+1] = diffusion/dx
            Q[j] = 0

```

```
    phi = np.linalg.solve(A,Q)
    return phi


def EE_time_step(dt, dx, phi_in, density, velocity, diffusion, left, right):
    """ Explicit Euler solve for next time step
    """
    phi_out = np.zeros(len(phi_in))

    for j in range(len(phi_in)):
        if j==0:
            a = (velocity*dt/dx + 2*diffusion*dt/density/dx**2)
            b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
            c = (diffusion*dt/density/dx**2)
            phi_out[j] = a*left + b*phi_in[j] + c*phi_in[j+1]
        elif j==len(phi_in)-1:
            a = (velocity*dt/dx + diffusion*dt/density/dx**2)
            b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
            c = (2*diffusion*dt/density/dx**2)
            phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*right
        else:
            a = (velocity*dt/dx + diffusion*dt/density/dx**2)
            b = (1-velocity*dt/dx-2*diffusion*dt/density/dx**2)
            c = (diffusion*dt/density/dx**2)
            phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*phi_in[j+1]
    return phi_out


def IE_time_step(dt, dx, phi_in, density, velocity, diffusion, left, right):
    """ Implicit Euler solve for next time step
    """
    phi_out = np.zeros(len(phi_in))
    A = np.zeros((len(phi_in), len(phi_in)))
    Q = np.zeros(len(phi_in))

    for j in range(len(phi_in)):
        if j == 0:
            A[0,0] = (1 + velocity*dt/dx + 3*diffusion*dt/density/dx**2)
            A[0,1] = -diffusion*dt/density/dx**2
            Q[0] = phi_in[0] + (velocity*dt/dx + 2*diffusion*dt/density/dx**2)*left
```

```python
        elif j==len(phi_in)-1:
            A[j,j-1] = (-velocity*dt/dx - diffusion*dt/density/dx**2)
            A[j,j] = (1 + velocity*dt/dx + 3*diffusion*dt/density/dx**2)
            Q[j] = phi_in[j] + (2*diffusion*dt/density/dx**2)*right
        else:
            A[j,j-1] = (-velocity*dt/dx - diffusion*dt/density/dx**2)
            A[j,j] = (1 + velocity*dt/dx + 2*diffusion*dt/density/dx**2)
            A[j,j+1] = (-diffusion*dt/density/dx**2)
            Q[j] = phi_in[j]

    phi_out = np.linalg.solve(A,Q)
    return phi_out

# define central difference scheme function to use for each case
def cds_ss(num_volumes, tot_length, velocity, density, diffusion, left, right):
    """Function to perform central difference scheme to solve one-dimensional steady state transport with convection and diffusion.

    Parameters
    ----------
    num_volumes : float
        The number of discretized volumes.
    tot_length : float
        The total length of the pipe in meters.
    velocity : float
        The average velocity of the flow in meters per second.
    density : float
        The density of the flow in kilograms per cubic meter.
    diffusion : float
        The diffusion coefficient in kilogram-seconds per meter.
    left : float
        The left boundary condition.
    right : float
        The right boundary condition.

    Returns
    -------
    phi : numpy.ndarray
```

```
        Solved flux profile.
    """
    dx = tot_length/num_volumes
    phi = np.zeros(num_volumes)
    A = np.zeros((num_volumes,num_volumes))
    Q = np.zeros(num_volumes)



    for j in range(num_volumes):

        if j == 0:
            A[j,0] = density*velocity/2 + 3*diffusion/dx
            A[j,1] = density*velocity/2 - diffusion/dx
            Q[j] = density*velocity*left + 2*diffusion*left/dx

        elif j == num_volumes-1:
            A[j,j-1] = -density*velocity/2 - diffusion/dx
            A[j,j] = -density*velocity/2 + 3*diffusion/dx
            Q[j] = -density*velocity*right + 2*diffusion*right/dx

        else:
            A[j,j-1] = -density*velocity/2 - diffusion/dx
            A[j,j] = 2*diffusion/dx
            A[j,j+1] = density*velocity/2 - diffusion/dx
            Q[j] = 0

    phi = np.linalg.solve(A,Q)
    return phi



# variables
tot_length = 1.0
density = 1.0
diffusion = 0.1
left = 100
right = 50
velocity = 2.5
num_volumes = 20

phi_init = np.zeros(num_volumes)
phi_init[:] = 50


```

```
K = np.array([0.2, 2.0, 20])
dx = tot_length/num_volumes
x = np.linspace(dx/2, tot_length-dx/2,num_volumes)
#volume = dx
dt = K*dx/velocity
max_iter = 256

# steady state CDS
phi_ss = cds_ss(num_volumes, tot_length, velocity, density,
   diffusion, left, right)


# explicit euler
phi_in_ee = np.zeros(num_volumes)
error_ee = np.zeros(len(dt))

for j in range(len(dt)):
    phi_in_ee[:] = phi_init[:]
    m=0
    phi_plot_ee = np.zeros((5,num_volumes))

    for n in range(max_iter):
        phi_out_ee = EE_time_step(dt[j], dx, phi_in_ee, density,
           velocity, diffusion, left, right)
        if n==0 or n==4 or n==16 or n==64:
            phi_plot_ee[m,:] = phi_in_ee[:]
            m += 1
        elif n==255:
            phi_plot_ee[m,:] = phi_out_ee[:]
        phi_in_ee[:] = phi_out_ee[:]

    # plot
    plt.rcParams['font.family'] = 'serif'
    plt.rcParams['mathtext.fontset'] = 'dejavuserif'
    plt.figure(facecolor='w', edgecolor='k', dpi=300)
    plt.plot(x, phi_plot_ee[0,:], '-k', label='n=0')
    plt.plot(x, phi_plot_ee[1,:], '-r', label='n=4')
    plt.plot(x, phi_plot_ee[2,:], '-b', label='n=16')
    plt.plot(x, phi_plot_ee[3,:], '-g', label='n=64')
    plt.plot(x, phi_plot_ee[4,:], '-m', label='n=256')
    plt.xlabel('x (m)')
    plt.ylabel(r'$\phi$')
    plt.figlegend(bbox_to_anchor=(1.0,0.9))
    plt.grid(b=True, which='major', axis='both')
```

```
    plt.savefig('HW2/plots/graph_EE_case'+str(j+1)+'.pdf',
       transparent=True)

    # compare transient to steady
    error_ee[j] = np.sum(np.absolute(phi_plot_ee[4,:]-phi_ss)) /
       num_volumes


# implicit euler
phi_in_ie = np.zeros(num_volumes)
error_ie = np.zeros(len(dt))
for h in range(len(dt)):
    # reset inputs
    phi_in_ie[:] = phi_init[:]
    g=0
    phi_plot_ie = np.zeros((5,num_volumes))

    for k in range(max_iter):
        phi_out_ie = IE_time_step(dt[h], dx, phi_in_ie, density,
           velocity, diffusion, left, right)
        if k==0 or k==4 or k==16 or k==64:
            phi_plot_ie[g,:] = phi_in_ie[:]
            g += 1
        elif k==255:
            phi_plot_ie[g,:] = phi_out_ie[:]
        phi_in_ie[:] = phi_out_ie[:]

    # plot
    plt.rcParams['font.family'] = 'serif'
    plt.rcParams['mathtext.fontset'] = 'dejavuserif'
    plt.figure(facecolor='w', edgecolor='k', dpi=300)
    plt.plot(x, phi_plot_ie[0,:], '-k', label='n=0')
    plt.plot(x, phi_plot_ie[1,:], '-r', label='n=4')
    plt.plot(x, phi_plot_ie[2,:], '-b', label='n=16')
    plt.plot(x, phi_plot_ie[3,:], '-g', label='n=64')
    plt.plot(x, phi_plot_ie[4,:], '-m', label='n=256')
    plt.xlabel('x (m)')
    plt.ylabel(r'$\phi$')
    plt.figlegend(bbox_to_anchor=(1.0,0.9))
    plt.grid(b=True, which='major', axis='both')
    plt.savefig('HW2/plots/graph_IE_case'+str(h+1)+'.pdf',
       transparent=True)

    # compare transient to steady
```

```
    error_ie[h] = np.sum(np.absolute(phi_plot_ie[4,:]-phi_ss)) / num_volumes


# generate latex table for error
out_file = open('HW2/tabs/error_tab_ee.tex','w')
out_file.write(
                '\\begin{table}[htbp]\n'+
                '\t \centering\n'+
                '\t \caption{Norm values for Explicit Euler.}\n'+
                '\t \\begin{tabular}{cc}\n'+
                '\t\t \\toprule\n'+
                '\t\t $K$ & Norm \\\ \n'+
                '\t\t \midrule \n'+
                '\t\t 0.2 & '+str(error_ee[0])+' \\\ \n'+
                '\t\t 2.0 & '+str(error_ee[1])+' \\\ \n'+
                '\t\t 20.0 & '+str(error_ee[2])+' \\\ \n'+
                '\t\t \\bottomrule \n'+
                '\t \end{tabular} \n'+
                '\t \label{tab:error ee} \n'+
                '\end{table}'
)

out_file = open('HW2/tabs/error_tab_ie.tex','w')
out_file.write(
                '\\begin{table}[htbp]\n'+
                '\t \centering\n'+
                '\t \caption{Norm values for Implicit Euler.}\n'+
                '\t \\begin{tabular}{cc}\n'+
                '\t\t \\toprule\n'+
                '\t\t $K$ & Norm \\\ \n'+
                '\t\t \midrule \n'+
                '\t\t 0.2 & '+str(error_ie[0])+' \\\ \n'+
                '\t\t 2.0 & '+str(error_ie[1])+' \\\ \n'+
                '\t\t 20.0 & '+str(error_ie[2])+' \\\ \n'+
                '\t\t \\bottomrule \n'+
                '\t \end{tabular} \n'+
                '\t \label{tab:error ie} \n'+
                '\end{table}'
)
```