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.

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

We can apply the surface integral approximations to get:

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

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:

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

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

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

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

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

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.

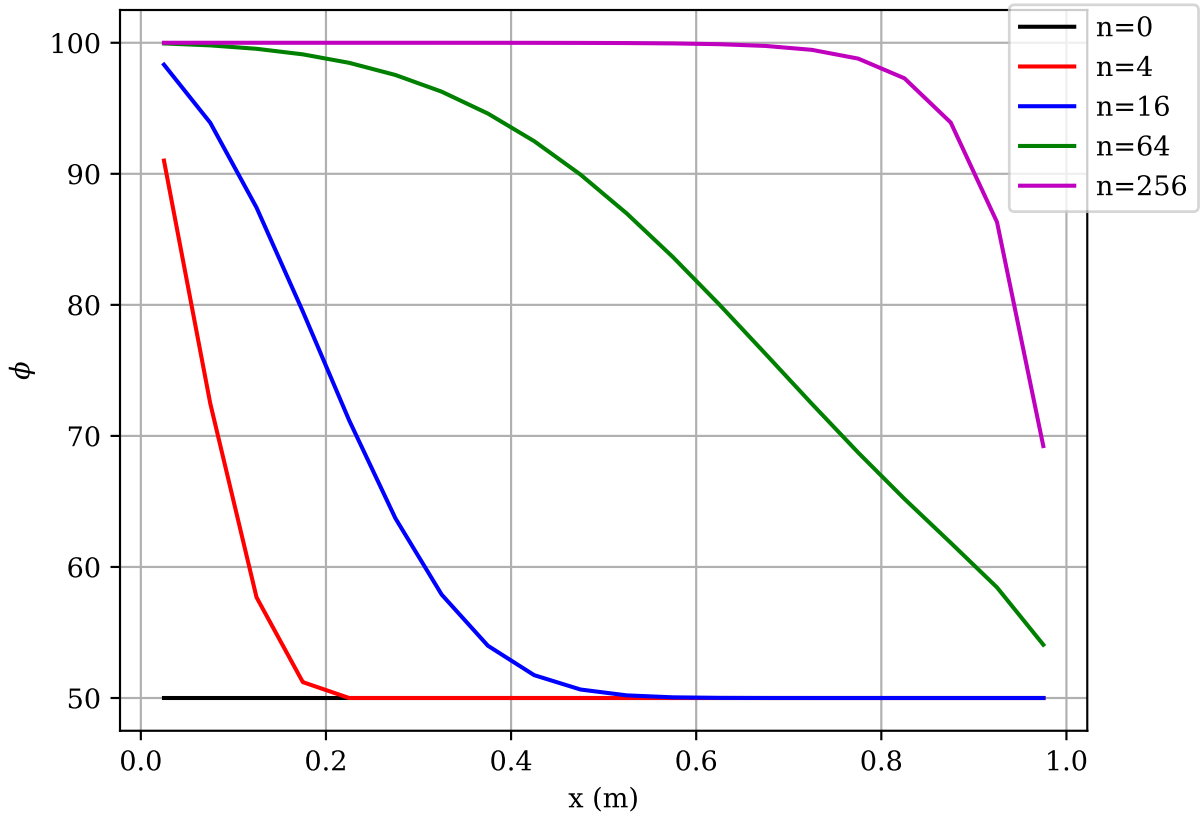

Figure 1: Explicit Euler solution for case 1: $K = 0.2$.

Norm:

Table 1: Norm values for Explicit Euler.

K	Norm
0.2	1.55418029575927
2.0	8.3196861106867e+245
20.0	inf

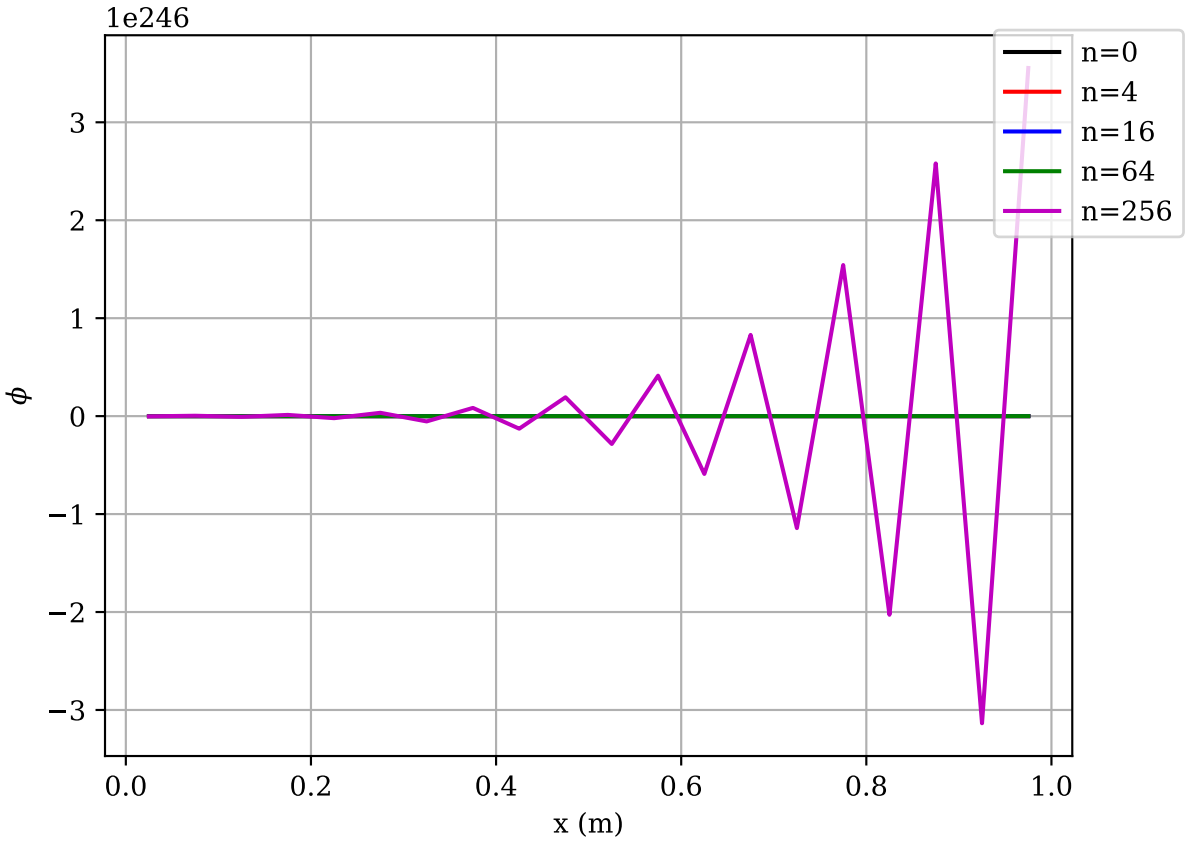

Figure 2: Explicit Euler solution for case 2: $K = 2.0$.

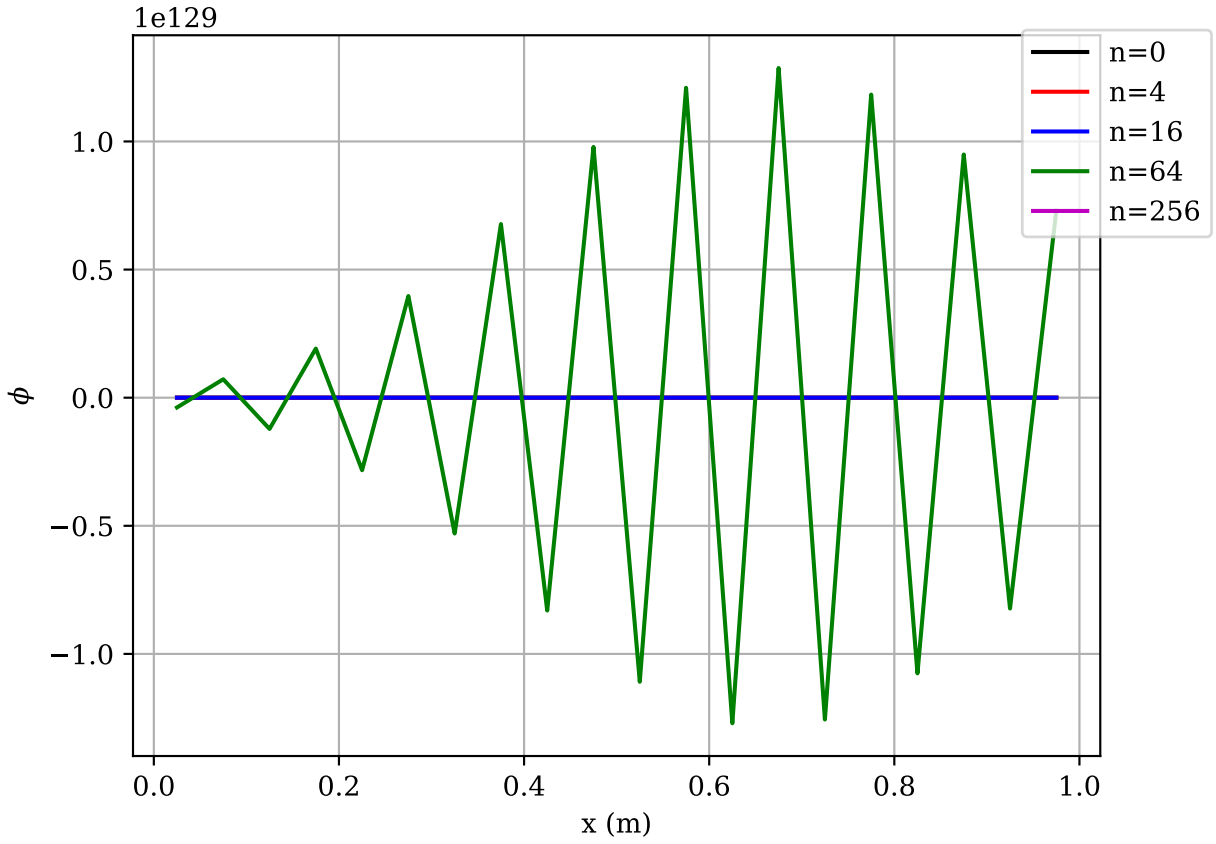

Figure 3: Explicit Euler solution for case 3: $K = 20.0$.

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.

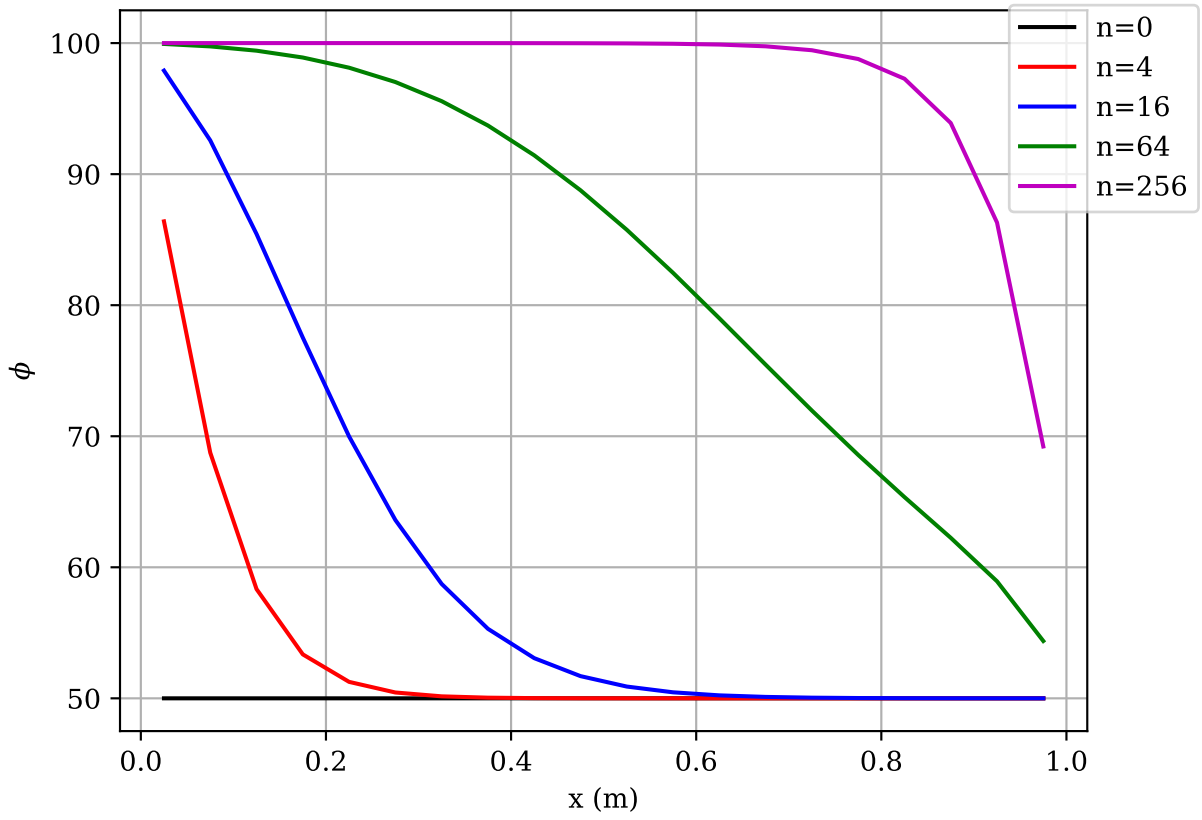

Figure 4: Implicit Euler solution for case 1: $K = 0.2$.

Norm:

Table 2: Norm values for Implicit Euler.

K	Norm
0.2	1.5567368462357045
2.0	1.5504768792236276
20.0	1.5504768792236157

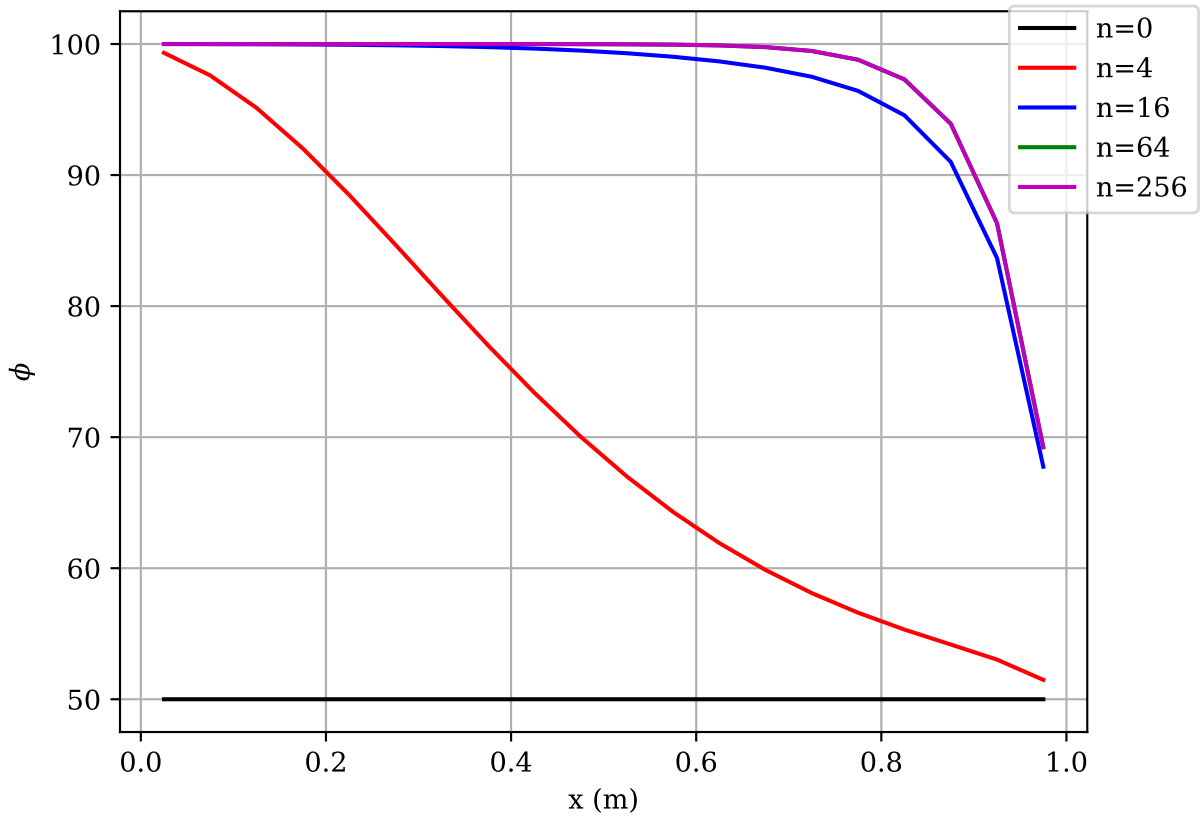

Figure 5: Implicit Euler solution for case 2: $K = 2.0$.

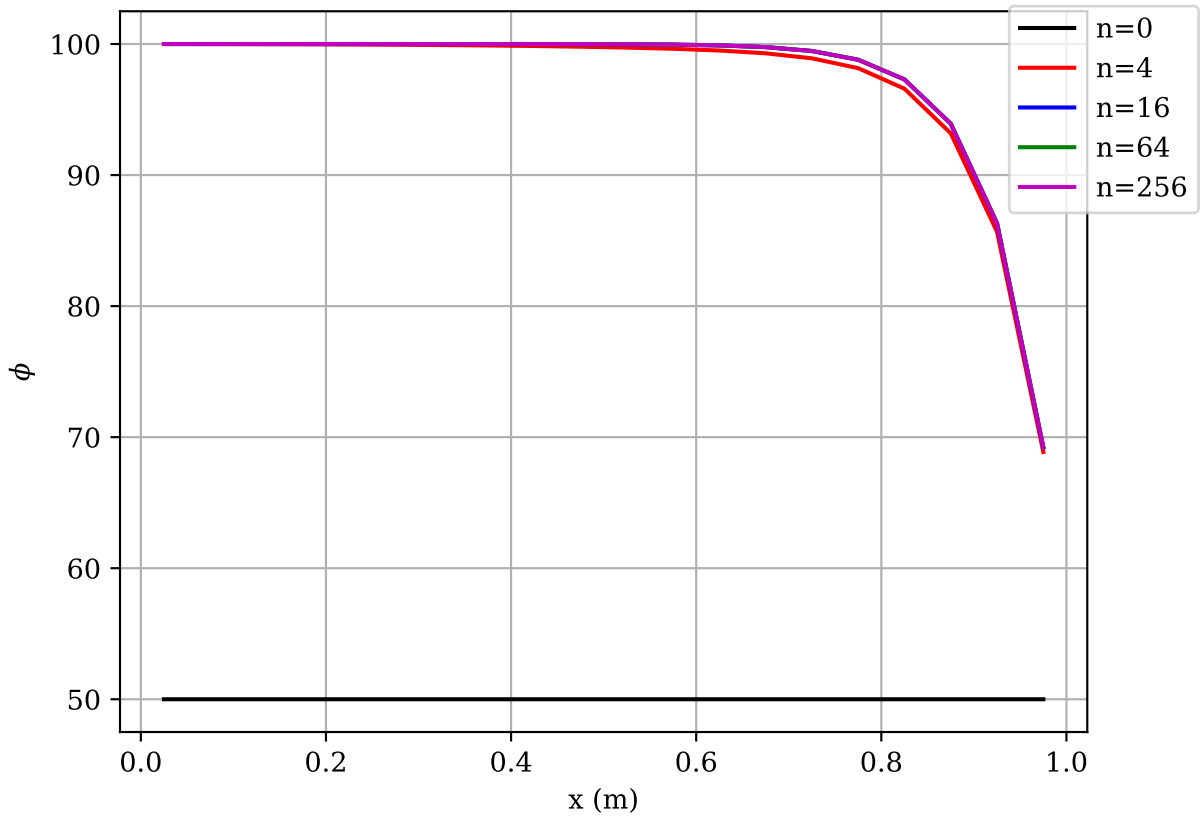

Figure 6: Implicit Euler solution for case 3: $K = 20.0$.

3 Discussion

For the Explicit Euler results, we see that for other than the smallest time step, the solution is unbounded. We can perform a stability analysis to see what the time step needs to be for Explicit Euler to be stable. We will begin by writing the inner node equation in terms of the time step to characteristic diffusion time ratio, $d = \frac{\Gamma \Delta t}{\rho(\Delta x)^2}$; and the Courant number, $c = \frac{u_x \Delta t}{\Delta x}$.

$$\phi_I^{n+1} = [c + d]\,\phi_{I-1}^n + [1 - c - 2d]\,\phi_I^n + [d]\,\phi_{I+1}^n$$

Then we can substitute the eigenvalue into the equation.

$$\sigma^{n+1} e^{i\alpha I} = [c + d]\,\sigma^n e^{i\alpha(I-1)} + [1 - c - 2d]\,\sigma^n e^{i\alpha I} + [d]\,\sigma^n e^{i\alpha(I+1)}$$

We can divide by σ^n and $e^{i\alpha I}$.

$$\sigma = [c + d]\,e^{-i\alpha} + [1 - c - 2d] + [d]\,e^{i\alpha}$$

We can then substitute the identity for $e^{i\alpha}$ and rearrange to get the solution for the eigenvalue.

$$\sigma = 1 + c\,(\cos(\alpha) - 1) + 2d\,(\cos(\alpha) - 1) - ic\sin(\alpha) \tag{12}$$

To be stable, $\sigma < 1$ and $\sigma^2 < 1$ for all values of α. We can find the bounding values for c and d by looking at the extreme cases. Firstly, when $d = 0$:

$$\sigma = 1 + c\,(\cos(\alpha) - 1) - ic\sin(\alpha)$$

$$\sigma^2 = [1 + c\,(\cos(\alpha) - 1)]^2 + c^2 \sin^2(\alpha)$$

We can see that for any value of c, $sigma^2 < 1$, so this is always unstable. We do not get any restrictions on c from this case. Next is when $c = 0$.

$$\sigma = 1 + 2d\,(\cos(\alpha) - 1)$$

The worst case is when $\cos(\alpha) = -1$.

$$1 > 1 + 2d\,(-1 - 1)$$

$$d < \frac{1}{2}$$

Now for σ^2 with the same conditions:

$$\sigma^2 = [1 + 2d\,(\cos(\alpha) - 1)]^2$$

$$1 > [1 + 2d\,(-1 - 1)]^2$$

$$1 > [1 - 4d]^2$$

$$1 > 1 - 8d + 16d^2$$

$$d < \frac{1}{2}$$

Table 3: Time step to characteristic diffusion time ratio.

K	d
0.2	0.16000000000000003
2.0	1.5999999999999996
20.0	16.0

So both conditions come to the same bounds that d must be less than $1/2$. Table 3 shows d for each time step size and we can see that only the smallest time step is less than $1/2$ and the others will not be stable. This matches up with the results as the plots for the second two solutions for Explicit Euler show the instability.

The results for Implicit Euler are all stable since Implicit Euler is an unconditionally stable solution method. We can also see that for the smallest time step, we can see the convergence over time towards the steady solution more clearly. The largest time step gets to the steady soltuion almost immediately. This is expected since we are plotting the solution at certain time steps, not certain times. So the solutions with larger time steps are closer to the steady solution in time at the same time step number than with smaller time steps.

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

```

```
42      phi = np.linalg.solve(A,Q)
43      return phi
44
45
46  def EE_time_step(dt, dx, phi_in, density, velocity, diffusion,
        left, right):
47      """ Explicit Euler solve for next time step
48      """
49      phi_out = np.zeros(len(phi_in))
50
51      for j in range(len(phi_in)):
52          if j==0:
53              a = (velocity*dt/dx + 2*diffusion*dt/density/dx**2)
54              b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
55              c = (diffusion*dt/density/dx**2)
56              phi_out[j] = a*left + b*phi_in[j] + c*phi_in[j+1]
57          elif j==len(phi_in)-1:
58              a = (velocity*dt/dx + diffusion*dt/density/dx**2)
59              b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
60              c = (2*diffusion*dt/density/dx**2)
61              phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*right
62          else:
63              a = (velocity*dt/dx + diffusion*dt/density/dx**2)
64              b = (1-velocity*dt/dx-2*diffusion*dt/density/dx**2)
65              c = (diffusion*dt/density/dx**2)
66              phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*phi_in[j
                    +1]
67      return phi_out
68
69
70  def IE_time_step(dt, dx, phi_in, density, velocity, diffusion,
        left, right):
71      """ Implicit Euler solve for next time step
72      """
73      phi_out = np.zeros(len(phi_in))
74      A = np.zeros((len(phi_in), len(phi_in)))
75      Q = np.zeros(len(phi_in))
76
77      for j in range(len(phi_in)):
78          if j == 0:
79              A[0,0] = (1 + velocity*dt/dx + 3*diffusion*dt/density
                    /dx**2)
80              A[0,1] = -diffusion*dt/density/dx**2
81              Q[0] = phi_in[0] + (velocity*dt/dx + 2*diffusion*dt/
                    density/dx**2)*left
```

```
82          elif j==len(phi_in)-1:
83              A[j,j-1] = (-velocity*dt/dx - diffusion*dt/density/dx
                    **2)
84              A[j,j] = (1 + velocity*dt/dx + 3*diffusion*dt/density
                    /dx**2)
85              Q[j] = phi_in[j] + (2*diffusion*dt/density/dx**2)*
                    right
86          else:
87              A[j,j-1] = (-velocity*dt/dx - diffusion*dt/density/dx
                    **2)
88              A[j,j] = (1 + velocity*dt/dx + 2*diffusion*dt/density
                    /dx**2)
89              A[j,j+1] = (-diffusion*dt/density/dx**2)
90              Q[j] = phi_in[j]
91
92      phi_out = np.linalg.solve(A,Q)
93      return phi_out
94
95  # define central difference scheme function to use for each case
96  def cds_ss(num_volumes, tot_length, velocity, density, diffusion,
        left, right):
97      """Function to perform central difference scheme to solve one
            -dimensional steady state transport with convection and
            diffusion.
98
99      Parameters
100     ----------
101     num_volumes : float
102         The number of discretized volumes.
103     tot_length : float
104         The total length of the pipe in meters.
105     velocity : float
106         The average velocity of the flow in meters per second.
107     density : float
108         The density of the flow in kilograms per cubic meter.
109     diffusion : float
110         The diffusion coefficient in kilogram-seconds per meter.
111     left : float
112         The left boundary condition.
113     right : float
114         The right boundary condition.
115
116     Returns
117     -------
118     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

# d -- ratio of time step to characteristic diffusion time
d = diffusion * dt / density / dx**2

# 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}'
)


out_file = open('HW2/tabs/diff_ratio.tex','w')
out_file.write(
```

```
                    '\\begin{table}[htbp]\n'+
                    '\t \centering\n'+
                    '\t \caption{Time step to characteristic
                       diffusion time ratio.}\n'+
                    '\t \\begin{tabular}{cc}\n'+
                    '\t\t \\toprule\n'+
                    '\t\t $K$ & $d$ \\\ \n'+
                    '\t\t \midrule \n'+
                    '\t\t 0.2 & '+str(d[0])+' \\\ \n'+
                    '\t\t 2.0 & '+str(d[1])+' \\\ \n'+
                    '\t\t 20.0 & '+str(d[2])+' \\\ \n'+
                    '\t\t \\bottomrule \n'+
                    '\t \end{tabular} \n'+
                    '\t \label{tab:diff ratio} \n'+
                    '\end{table}'
)
```