

# 1 Methods

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

$$\phi_{i+1} = \begin{cases} \phi_I & \bar{V} > 0 \\ \phi_{I+1} & \bar{V} < 0 \end{cases}, \quad (1)$$

and

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

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

$$\int_{\Delta t} \frac{\partial}{\partial t} \left( \int_{CV} \rho \phi dV \right) dt + \int_{\Delta t} \int_S \rho u_x \phi \cdot \bar{n} dS dt = \int_{\Delta t} \int_S \Gamma \frac{\partial \phi}{\partial x} \cdot \bar{n} dS dt \quad (3)$$

We can apply the surface integral approximations to get:

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

Performing the rest of the integrations, we get:

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

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

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

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:

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

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

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

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

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

## 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  $\left(\frac{\partial \phi}{\partial x}\right)_{i+1} = \frac{\phi_R - \phi_I}{\Delta x/2}$

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

## 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 Explicit 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.

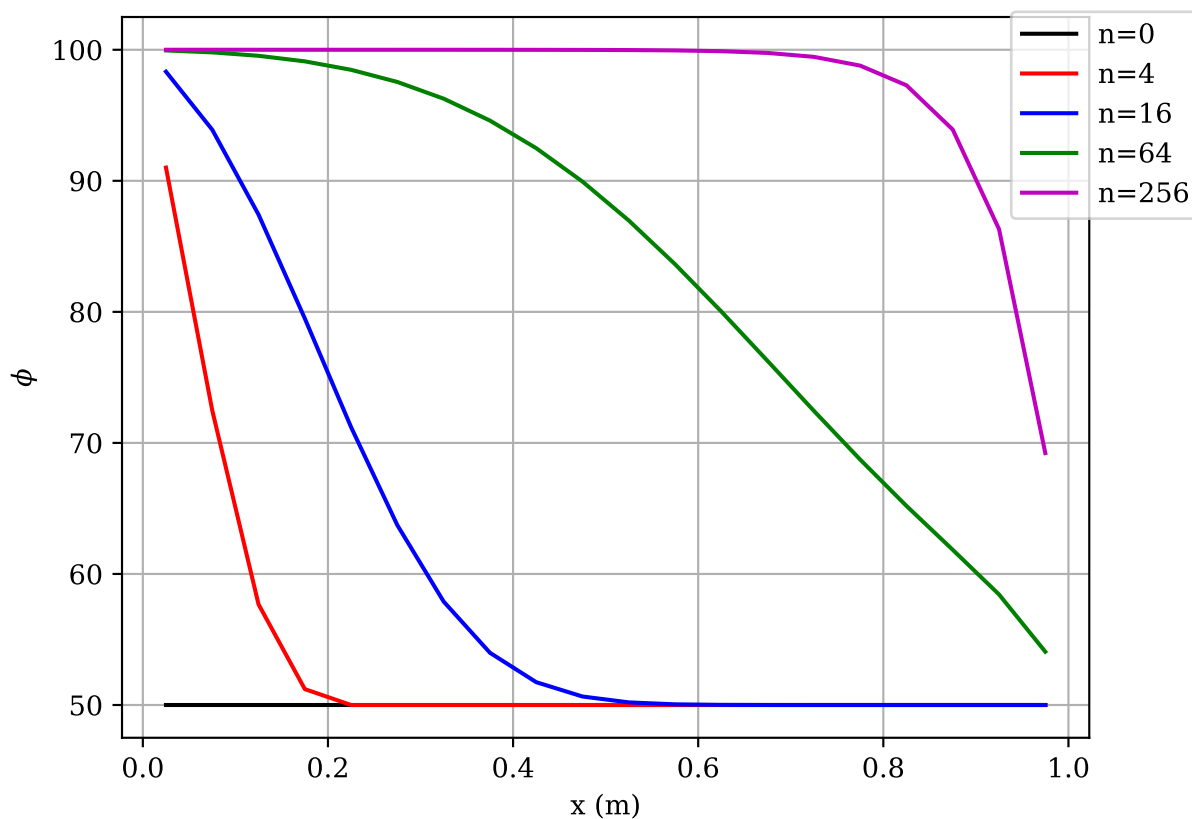


Figure 1: Explicit Euler solution for case 1:  $K = 0.2$ .

Norm:

Table 1: Norm values for Explicit Euler.

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

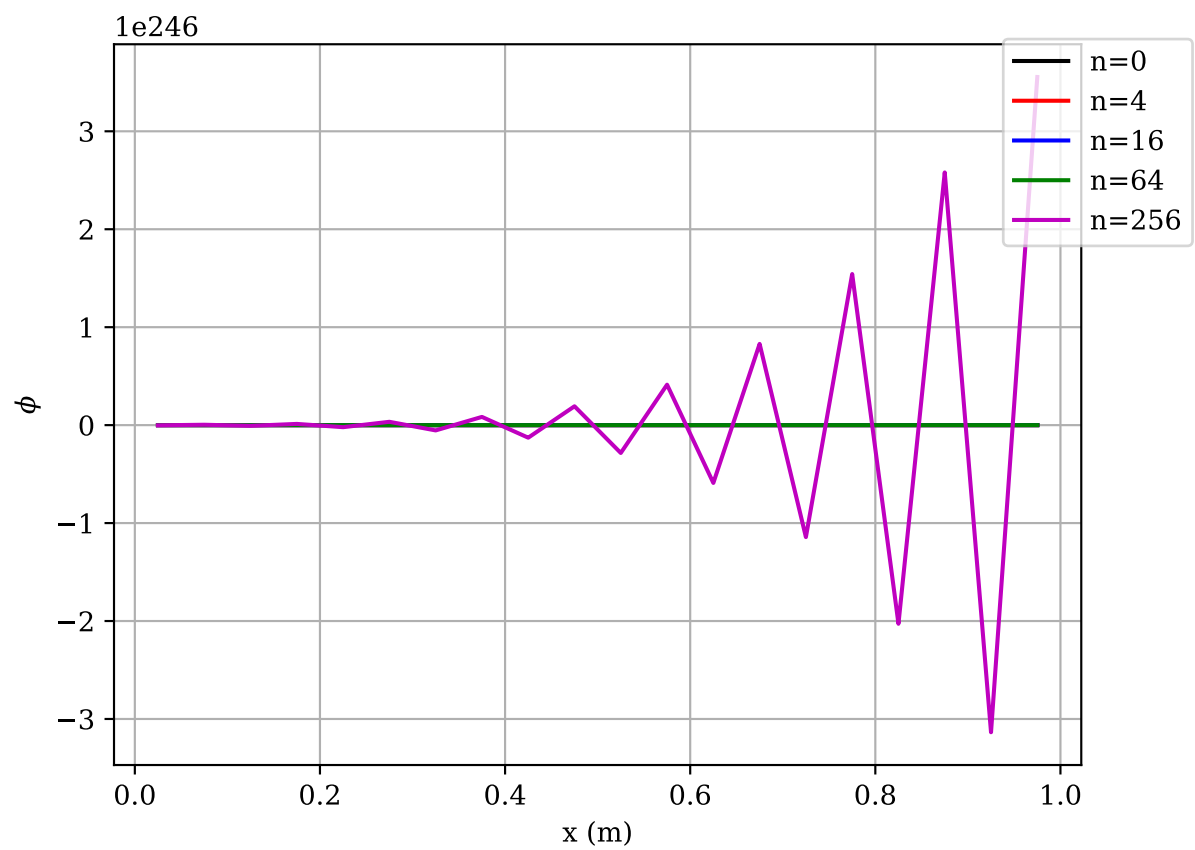


Figure 2: Explicit Euler solution for case 2:  $K = 2.0$ .

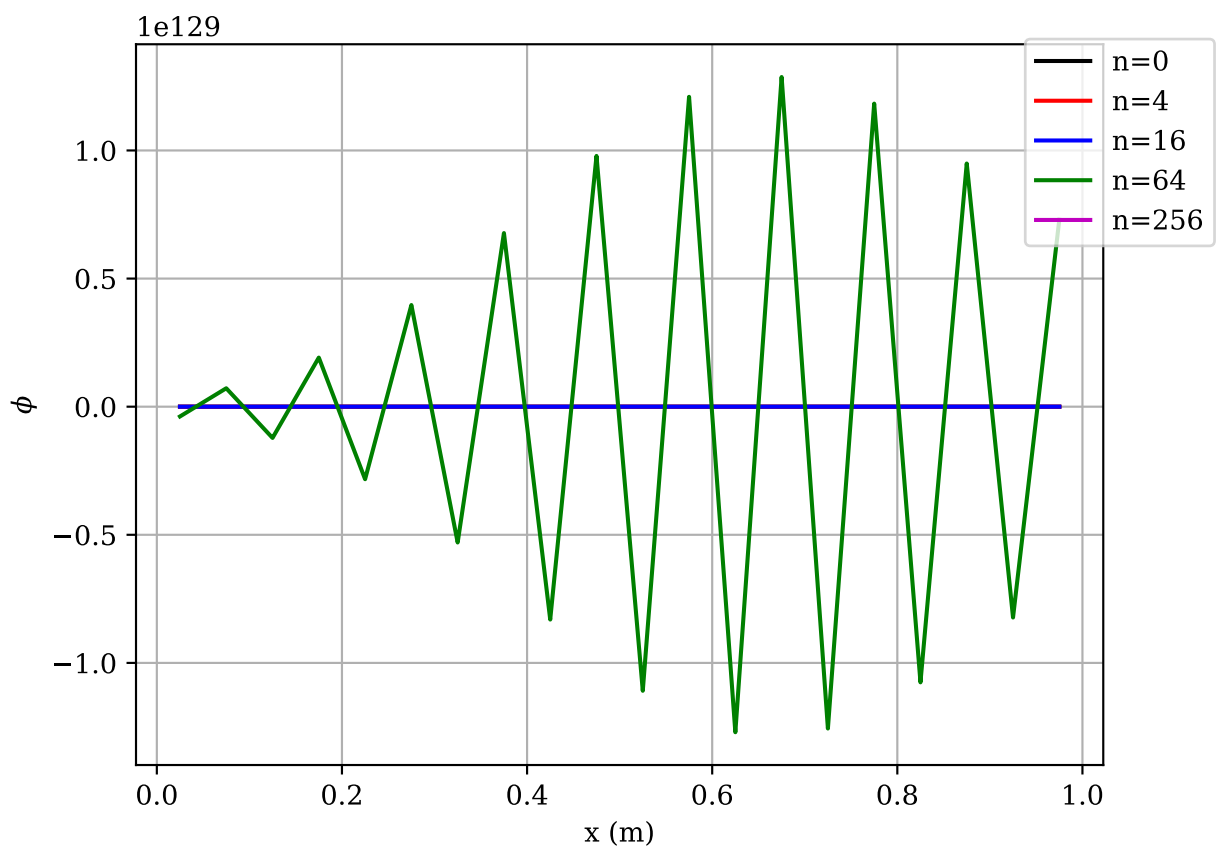


Figure 3: Explicit Euler solution for case 3:  $K = 20.0$ .

## 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.

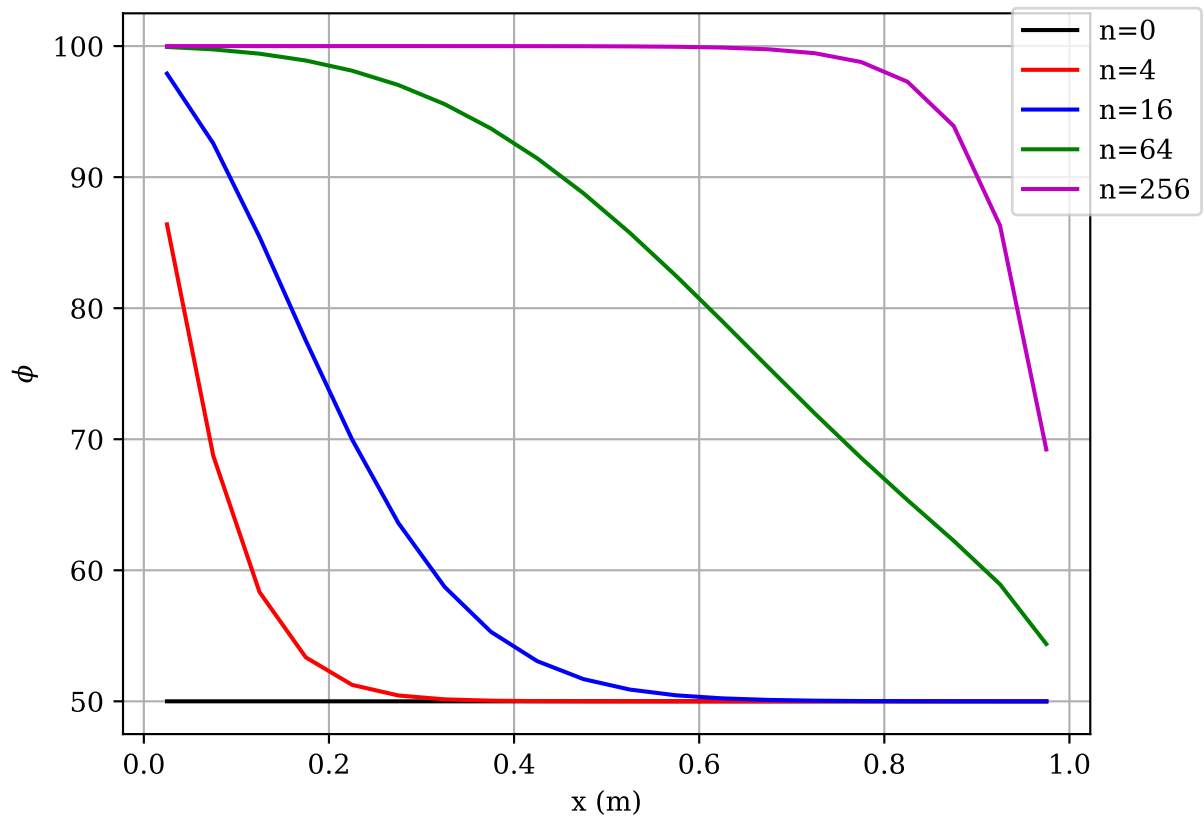


Figure 4: Implicit Euler solution for case 1:  $K = 0.2$ .

Norm:

Table 2: Norm values for Implicit Euler.

$K$	Norm
0.2	1.5567368462357045
2.0	1.5504768792236276
20.0	1.5504768792236157

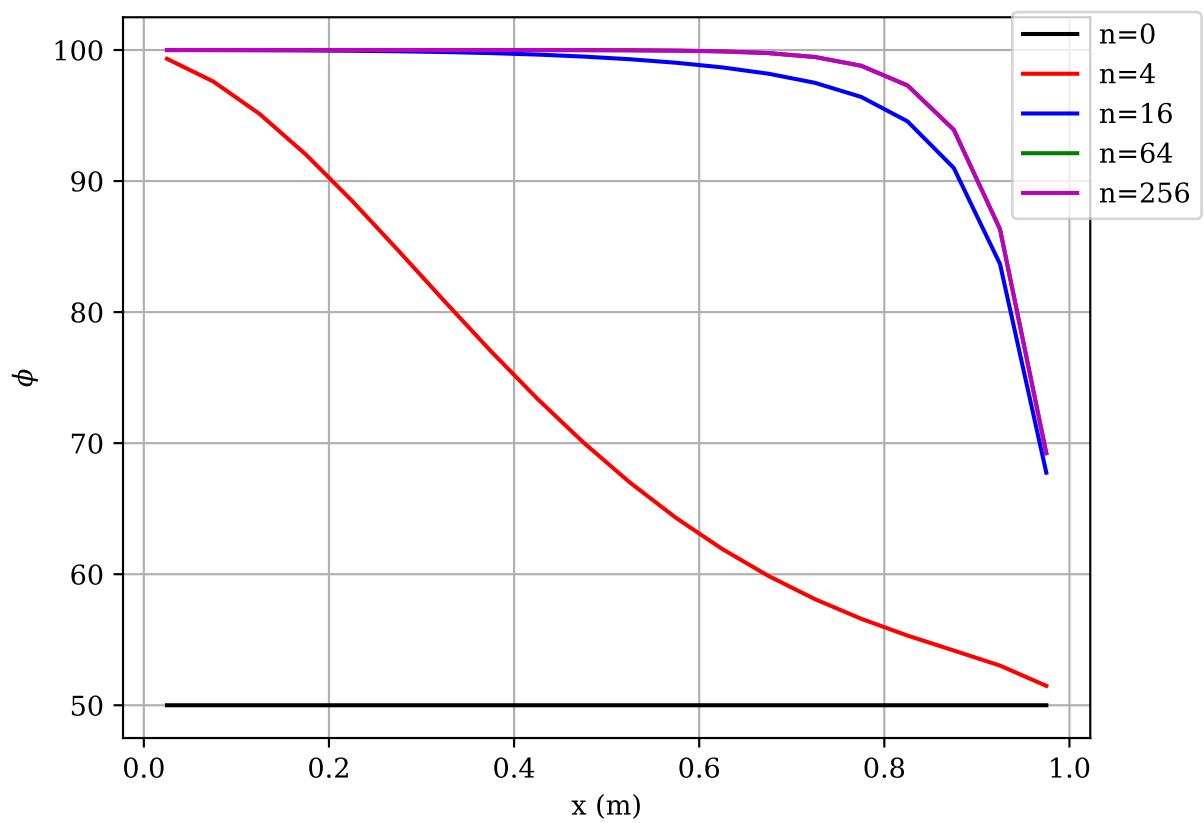


Figure 5: Implicit Euler solution for case 2:  $K = 2.0$ .

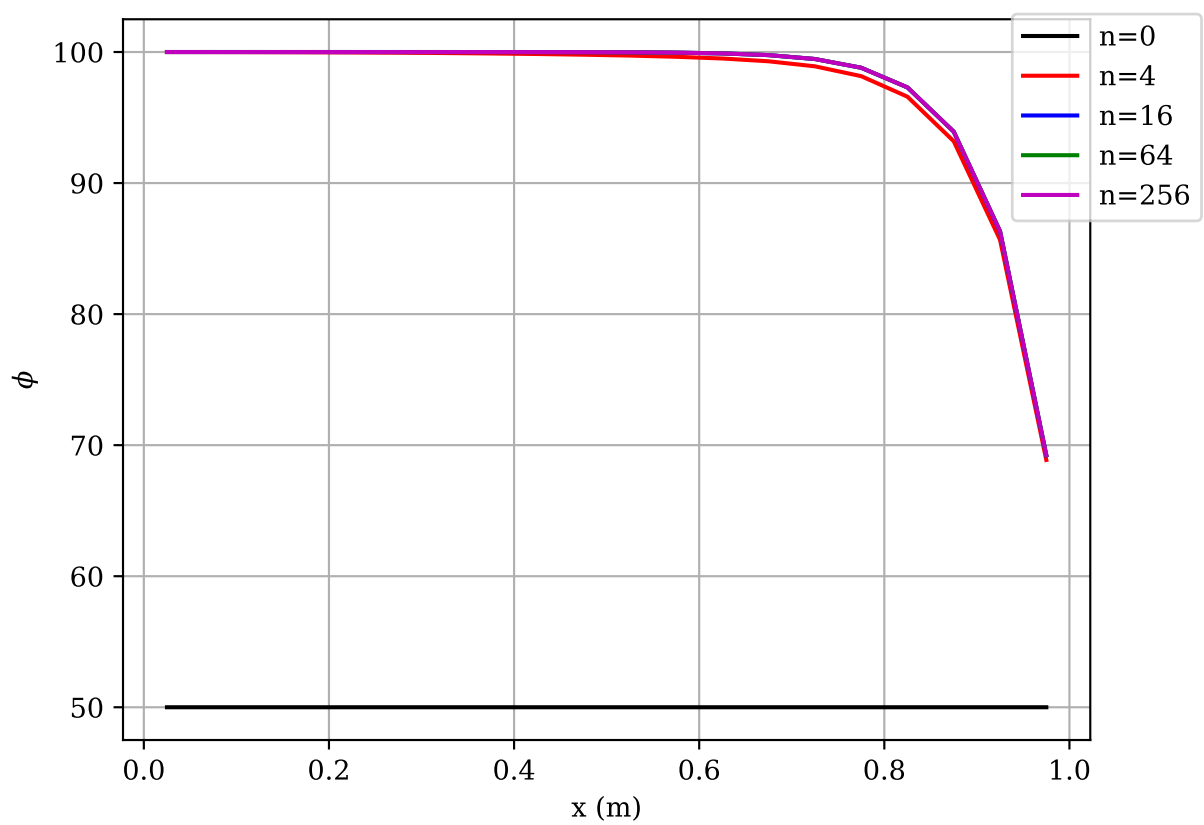


Figure 6: Implicit Euler solution for case 3:  $K = 20.0$ .



### 3 Discussion

## 4 Code

```
1 # -----#
2 # --- main script for NSE 565 HW2 ---#
3 # ----- Austin Warren -----#
4 # ----- Winter 2022 -----#
5 # -----#
6
7 import numpy as np
8 import matplotlib.pyplot as plt
9
10 # define functions for spatial and temporal discretizations
11 # - coefficient generation
12 # - time step
13
14 def UDS_positive(num_volumes, tot_length, density, velocity,
15                 diffusion, left, right):
16     """ Upwind scheme solver for positive velocity
17     """
18     dx = tot_length/num_volumes
19     phi = np.zeros(num_volumes)
20     A = np.zeros((num_volumes,num_volumes))
21     Q = np.zeros(num_volumes)
22
23
24     for j in range(num_volumes):
25
26         if j == 0:
27             A[j,0] = density*velocity + 2*diffusion/dx
28             A[j,1] = diffusion/dx
29             Q[j] = density*velocity*left + diffusion*left/dx
30
31         elif j == num_volumes-1:
32             A[j,j-1] = -diffusion/dx
33             A[j,j] = -density*velocity - 2*diffusion/dx
34             Q[j] = -density*velocity*right + diffusion*right/dx
35
36         else:
37             A[j,j-1] = -density*velocity - diffusion/dx
38             A[j,j] = density*velocity + 2*diffusion/dx
39             A[j,j+1] = diffusion/dx
40             Q[j] = 0
41
```

```
42     phi = np.linalg.solve(A,Q)
43     return phi
44
45
46 def EE_time_step(dt, dx, phi_in, density, velocity, diffusion,
47 left, right):
48     """ Explicit Euler solve for next time step
49     """
50     phi_out = np.zeros(len(phi_in))
51
52     for j in range(len(phi_in)):
53         if j==0:
54             a = (velocity*dt/dx + 2*diffusion*dt/density/dx**2)
55             b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
56             c = (diffusion*dt/density/dx**2)
57             phi_out[j] = a*left + b*phi_in[j] + c*phi_in[j+1]
58         elif j==len(phi_in)-1:
59             a = (velocity*dt/dx + diffusion*dt/density/dx**2)
60             b = (1-velocity*dt/dx-3*diffusion*dt/density/dx**2)
61             c = (2*diffusion*dt/density/dx**2)
62             phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*right
63         else:
64             a = (velocity*dt/dx + diffusion*dt/density/dx**2)
65             b = (1-velocity*dt/dx-2*diffusion*dt/density/dx**2)
66             c = (diffusion*dt/density/dx**2)
67             phi_out[j] = a*phi_in[j-1] + b*phi_in[j] + c*phi_in[j
68                 +1]
69     return phi_out
70
71 def IE_time_step(dt, dx, phi_in, density, velocity, diffusion,
72 left, right):
73     """ Implicit Euler solve for next time step
74     """
75     phi_out = np.zeros(len(phi_in))
76     A = np.zeros((len(phi_in), len(phi_in)))
77     Q = np.zeros(len(phi_in))
78
79     for j in range(len(phi_in)):
80         if j == 0:
81             A[0,0] = (1 + velocity*dt/dx + 3*diffusion*dt/density
82 /dx**2)
83             A[0,1] = -diffusion*dt/density/dx**2
84             Q[0] = phi_in[0] + (velocity*dt/dx + 2*diffusion*dt/
85 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,
97            left, right):
98     """Function to perform central difference scheme to solve one
99     -dimensional steady state transport with convection and
100     diffusion.
101
102     Parameters
103     -----
104     num_volumes : float
105         The number of discretized volumes.
106     tot_length : float
107         The total length of the pipe in meters.
108     velocity : float
109         The average velocity of the flow in meters per second.
110     density : float
111         The density of the flow in kilograms per cubic meter.
112     diffusion : float
113         The diffusion coefficient in kilogram-seconds per meter.
114     left : float
115         The left boundary condition.
116     right : float
117         The right boundary condition.
118
119     Returns
120     -----
121     phi : numpy.ndarray

```

```
119     Solved flux profile.
120     """
121     dx = tot_length/num_volumes
122     phi = np.zeros(num_volumes)
123     A = np.zeros((num_volumes,num_volumes))
124     Q = np.zeros(num_volumes)
125
126
127
128     for j in range(num_volumes):
129
130         if j == 0:
131             A[j,0] = density*velocity/2 + 3*diffusion/dx
132             A[j,1] = density*velocity/2 - diffusion/dx
133             Q[j] = density*velocity*left + 2*diffusion*left/dx
134
135         elif j == num_volumes-1:
136             A[j,j-1] = -density*velocity/2 - diffusion/dx
137             A[j,j] = -density*velocity/2 + 3*diffusion/dx
138             Q[j] = -density*velocity*right + 2*diffusion*right/dx
139
140         else:
141             A[j,j-1] = -density*velocity/2 - diffusion/dx
142             A[j,j] = 2*diffusion/dx
143             A[j,j+1] = density*velocity/2 - diffusion/dx
144             Q[j] = 0
145
146     phi = np.linalg.solve(A,Q)
147     return phi
148
149
150
151 # variables
152 tot_length = 1.0
153 density = 1.0
154 diffusion = 0.1
155 left = 100
156 right = 50
157 velocity = 2.5
158 num_volumes = 20
159
160 phi_init = np.zeros(num_volumes)
161 phi_init[:] = 50
162
163
```

```
164 K = np.array([0.2, 2.0, 20])
165 dx = tot_length/num_volumes
166 x = np.linspace(dx/2, tot_length-dx/2,num_volumes)
167 #volume = dx
168 dt = K*dx/velocity
169 max_iter = 256
170
171 # steady state CDS
172 phi_ss = cds_ss(num_volumes, tot_length, velocity, density,
    diffusion, left, right)
173
174
175 # explicit euler
176 phi_in_ee = np.zeros(num_volumes)
177 error_ee = np.zeros(len(dt))
178
179 for j in range(len(dt)):
180     phi_in_ee[:] = phi_init[:]
181     m=0
182     phi_plot_ee = np.zeros((5,num_volumes))
183
184     for n in range(max_iter):
185         phi_out_ee = EE_time_step(dt[j], dx, phi_in_ee, density,
            velocity, diffusion, left, right)
186         if n==0 or n==4 or n==16 or n==64:
187             phi_plot_ee[m,:] = phi_in_ee[:]
188             m += 1
189         elif n==255:
190             phi_plot_ee[m,:] = phi_out_ee[:]
191             phi_in_ee[:] = phi_out_ee[:]
192
193     # plot
194     plt.rcParams['font.family'] = 'serif'
195     plt.rcParams['mathtext.fontset'] = 'dejavuserif'
196     plt.figure(facecolor='w', edgecolor='k', dpi=300)
197     plt.plot(x, phi_plot_ee[0,:], '-k', label='n=0')
198     plt.plot(x, phi_plot_ee[1,:], '-r', label='n=4')
199     plt.plot(x, phi_plot_ee[2,:], '-b', label='n=16')
200     plt.plot(x, phi_plot_ee[3,:], '-g', label='n=64')
201     plt.plot(x, phi_plot_ee[4,:], '-m', label='n=256')
202     plt.xlabel('x (m)')
203     plt.ylabel(r'$\phi$')
204     plt.figlegend(bbox_to_anchor=(1.0,0.9))
205     plt.grid(b=True, which='major', axis='both')
```

```
206 plt.savefig('HW2/plots/graph_EE_case'+str(j+1)+'.pdf',
207             transparent=True)
208
209 # compare transient to steady
210 error_ee[j] = np.sum(np.absolute(phi_plot_ee[4,:]-phi_ss)) /
211 num_volumes
212
213 # implicit euler
214 phi_in_ie = np.zeros(num_volumes)
215 error_ie = np.zeros(len(dt))
216 for h in range(len(dt)):
217     # reset inputs
218     phi_in_ie[:] = phi_init[:]
219     g=0
220     phi_plot_ie = np.zeros((5,num_volumes))
221
222     for k in range(max_iter):
223         phi_out_ie = IE_time_step(dt[h], dx, phi_in_ie, density,
224                                   velocity, diffusion, left, right)
225         if k==0 or k==4 or k==16 or k==64:
226             phi_plot_ie[g,:] = phi_in_ie[:]
227             g += 1
228         elif k==255:
229             phi_plot_ie[g,:] = phi_out_ie[:]
230             phi_in_ie[:] = phi_out_ie[:]
231
232 # plot
233 plt.rcParams['font.family'] = 'serif'
234 plt.rcParams['mathtext.fontset'] = 'dejavuserif'
235 plt.figure(facecolor='w', edgecolor='k', dpi=300)
236 plt.plot(x, phi_plot_ie[0,:], '-k', label='n=0')
237 plt.plot(x, phi_plot_ie[1,:], '-r', label='n=4')
238 plt.plot(x, phi_plot_ie[2,:], '-b', label='n=16')
239 plt.plot(x, phi_plot_ie[3,:], '-g', label='n=64')
240 plt.plot(x, phi_plot_ie[4,:], '-m', label='n=256')
241 plt.xlabel('x (m)')
242 plt.ylabel(r'$\phi$')
243 plt.figlegend(bbox_to_anchor=(1.0,0.9))
244 plt.grid(b=True, which='major', axis='both')
245 plt.savefig('HW2/plots/graph_IE_case'+str(h+1)+'.pdf',
246             transparent=True)
247
248 # compare transient to steady
```

```

246     error_ie[h] = np.sum(np.absolute(phi_plot_ie[4,:]-phi_ss)) /
        num_volumes
247
248
249 # generate latex table for error
250 out_file = open('HW2/tabs/error_tab_ee.tex','w')
251 out_file.write(
252     '\\begin{table}[htbp]\n'+
253     '\\t \\centering\n'+
254     '\\t \\caption{Norm values for Explicit Euler.}\n'+
255     '\\t \\begin{tabular}{cc}\n'+
256     '\\t\\t \\toprule\n'+
257     '\\t\\t $K$ & Norm \\\\ \\n'+
258     '\\t\\t \\midrule\n'+
259     '\\t\\t 0.2 & '+str(error_ee[0])+ ' \\\\ \\n'+
260     '\\t\\t 2.0 & '+str(error_ee[1])+ ' \\\\ \\n'+
261     '\\t\\t 20.0 & '+str(error_ee[2])+ ' \\\\ \\n'+
262     '\\t\\t \\bottomrule\n'+
263     '\\t \\end{tabular} \n'+
264     '\\t \\label{tab:error ee} \n'+
265     '\\end{table}'
266 )
267
268 out_file = open('HW2/tabs/error_tab_ie.tex','w')
269 out_file.write(
270     '\\begin{table}[htbp]\n'+
271     '\\t \\centering\n'+
272     '\\t \\caption{Norm values for Implicit Euler.}\n'+
273     '\\t \\begin{tabular}{cc}\n'+
274     '\\t\\t \\toprule\n'+
275     '\\t\\t $K$ & Norm \\\\ \\n'+
276     '\\t\\t \\midrule\n'+
277     '\\t\\t 0.2 & '+str(error_ie[0])+ ' \\\\ \\n'+
278     '\\t\\t 2.0 & '+str(error_ie[1])+ ' \\\\ \\n'+
279     '\\t\\t 20.0 & '+str(error_ie[2])+ ' \\\\ \\n'+
280     '\\t\\t \\bottomrule\n'+
281     '\\t \\end{tabular} \n'+
282     '\\t \\label{tab:error ie} \n'+
283     '\\end{table}'
284 )

```