1 Methods

$$\int\limits_S \rho\overline{u}\phi \cdot \overline{n} dS = \int\limits_S \Gamma \frac{\partial \phi}{\partial x} \cdot \overline{n} dS + \int\limits_V q_\phi dV \tag{1}$$

We can approximate the two surface integrals as follows:

$$\int\limits_S \rho\overline{u}\phi \cdot \overline{n} dS = \Big(\rho\underline{(u)}S\phi\Big)_{i+1} - \Big(\rho\underline{(u)}S\phi\Big)_{i-1} \ ,$$

and

$$\int\limits_S \Gamma \frac{\partial \phi}{\partial x} \cdot \overline{n} dS = \left(\Gamma S \frac{\partial \phi}{\partial x}\right)_{i+1} - \left(\Gamma S \frac{\partial \phi}{\partial x}\right)_{i-1} \ .$$

We can approximate the volume integral as follows:

$$\int\limits_V q_\phi dV = \overline{q}V \ .$$

If we assume S is constant our transport equation becomes:

$$\Big(\rho\bar{(u)}\phi\Big)_{i+1} - \Big(\rho\bar{(u)}\phi\Big)_{i-1} = \left(\Gamma \frac{\partial \phi}{\partial x}\right)_{i+1} - \left(\Gamma \frac{\partial \phi}{\partial x}\right)_{i-1} + \overline{q}V \ .$$

We can apply the Central Differencing Scheme approximations:

$$\phi_{i+1} = \frac{\phi_{I+1} + \phi_I}{2} \qquad \phi_{i-1} = \frac{\phi_{I-1} + \phi_I}{2}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_{i+1} = \frac{\phi_{I+1} - \phi_I}{\delta x} \qquad \left(\frac{\partial \phi}{\partial x}\right)_{i-1} = \frac{\phi_I - \phi_{I-1}}{\delta x}$$

The transport equation now looks like:

$$\left(\frac{\rho\overline{u}}{2}\right)_{i+1} (\phi_{I+1} + \phi_I) - \left(\frac{\rho\overline{u}}{2}\right)_{i-1} (\phi_{I-1} + \phi_I) = \frac{\Gamma}{\Delta x} (\phi_{I+1} - \phi_I) - \frac{\Gamma}{\Delta x} (\phi_I - \phi_{I-1}) + \overline{q}_I \, \Delta x \ .$$

For this problem, the density and velocity are constant throughout the geometry. We can group terms and get the transport equation in a useful form to sovle.

$$\left[-\frac{\rho\overline{u}}{2} - \frac{\Gamma}{\Delta x}\right] \phi_{I-1} + \left[\frac{2\Gamma}{\Delta x}\right] \phi_I + \left[\frac{\rho\overline{u}}{2} - \frac{\Gamma}{\Delta x}\right] \phi_{I+1} = \overline{q}_I \, \Delta x \tag{2}$$

We can rewrite this equation as:

$$a_I \phi_{I-1} + b_I \phi_I + c_I \phi_{I+1} = Q_I \ . \tag{3}$$

This equation works for middle nodes, but the edge nodes do not have enough neighbor nodes. The first node does not have another node to the left of it $(I-1)$, so we have to only use the value at the left edge, which is ϕ_L. This gives us:

$$\left[\left(\frac{\rho\overline{u}}{2}\right)_{i+1} + \frac{3\Gamma}{\Delta x}\right] \phi_I + \left[\left(\frac{\rho\overline{u}}{2}\right)_{i+1} - \frac{\Gamma}{\Delta x}\right] \phi_{I+1} = \overline{q}_I \, \Delta x + (\rho\overline{u}\phi)_L + \frac{2\Gamma}{\Delta x}\phi_L \ , \tag{4}$$

which gives us an equation of the form:

$$b_1\phi_1 + c_1\phi_2 = Q_1 \,. \tag{5}$$

If we do the same on the right side we get:

$$\left[-\left(\frac{\rho\overline{u}}{2}\right)_{i-1} + \frac{3\Gamma}{\Delta x}\right]\phi_{I-1} + \left[-\left(\frac{\rho\overline{u}}{2}\right)_{i-1} - \frac{\Gamma}{\Delta x}\right]\phi_I = \overline{q}_I\,\Delta x - (\rho\overline{u}\phi)_R + \frac{2\Gamma}{\Delta x}\phi_R \,, \tag{6}$$

and:

$$a_N\phi_{N-1} + b_N\phi_N = Q_N \,. \tag{7}$$

2 Results

Figure 1 shows the two solutions for Case 1: 5 control volumes and $\overline{u} = 0.1\,\mathrm{m\,s^{-1}}$. Figure 2 shows the two solutions for Case 2: 5 control volumes and $\overline{u} = 2.5\,\mathrm{m\,s^{-1}}$. Figure 3 shows the two solutions for Case 3: 20 control volumes and $\overline{u} = 2.5\,\mathrm{m\,s^{-1}}$.

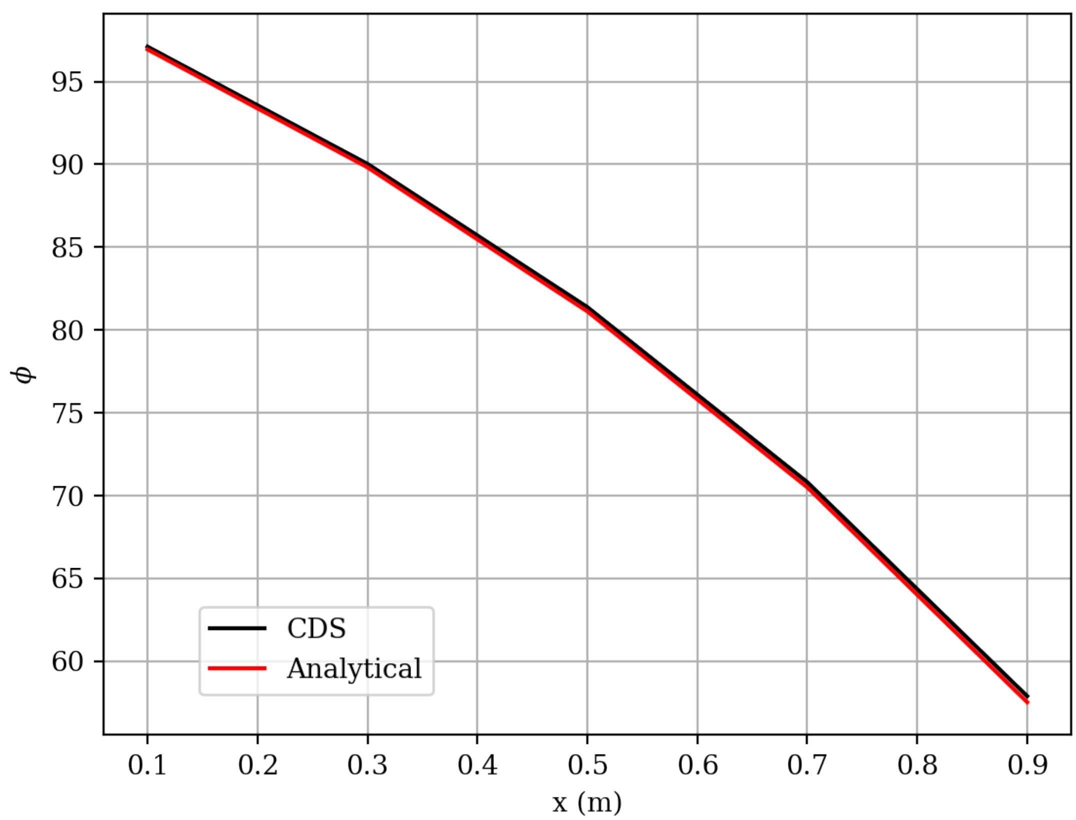

Figure 1: Central Difference Scheme and Analytical Solutions for Case 1.

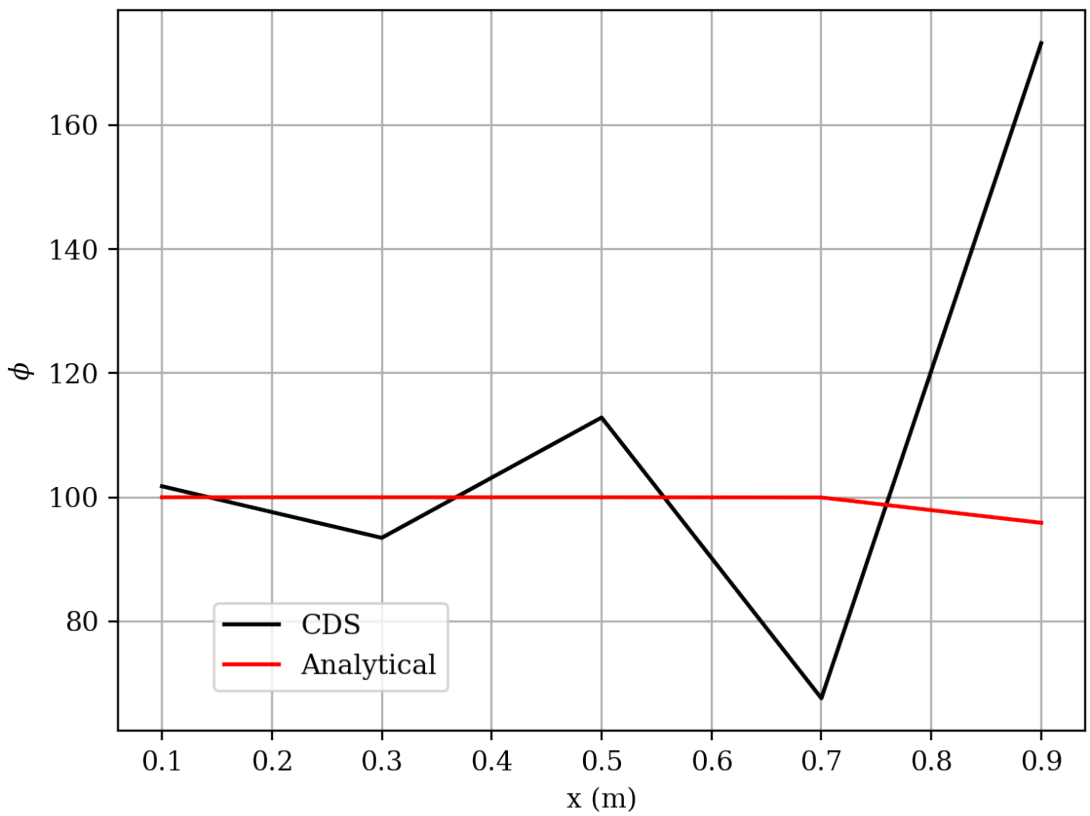

Figure 2: Central Difference Scheme and Analytical Solutions for Case 2.

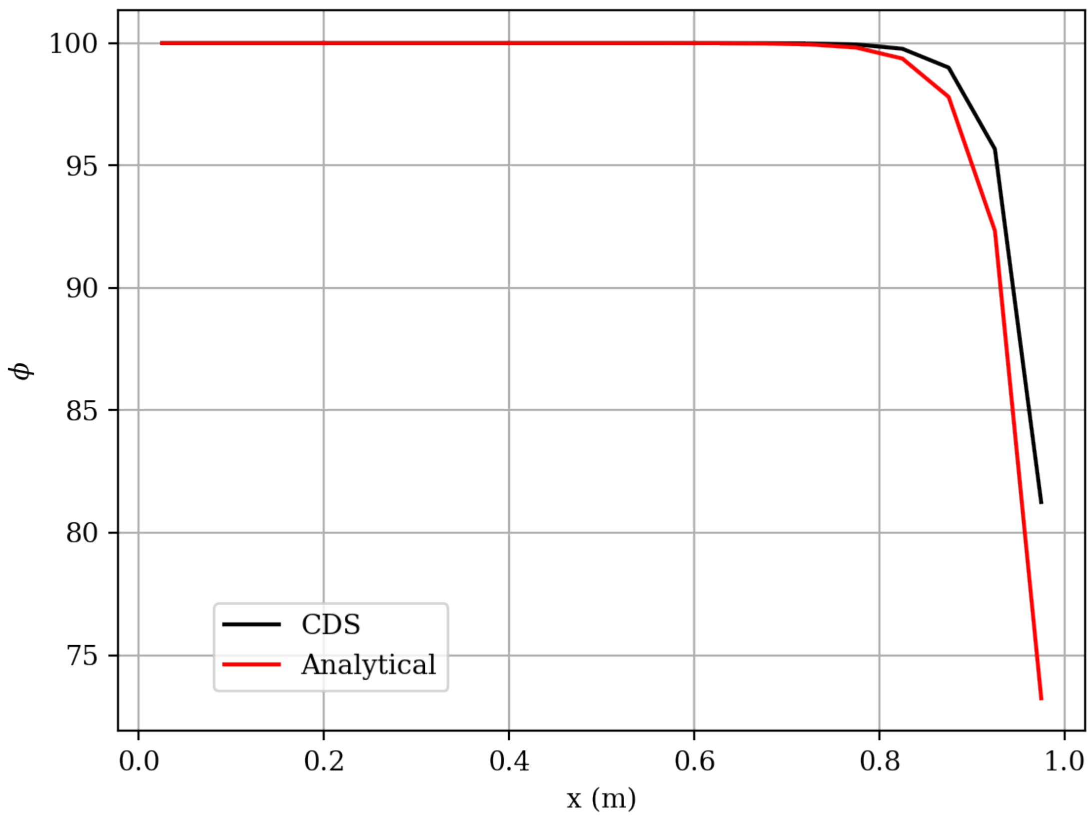

Figure 3: Central Difference Scheme and Analytical Solutions for Case 3.

Table 1 lists the error values calculated for each case.

Table 1: Error values for each case.

Case	Error
1	0.2629491288831474
2	26.174593696190538
3	0.6567406338821179

3 Discussion

We can use the Peclet number to see how stable each solution should be to determine if the solutions perform how we expect them to.

$$Pe = \frac{\rho \overline{u}}{\Gamma / \Delta x} \tag{8}$$

For this problem, we want $Pe < 2$. Table 2 shows the Peclet numbers for each case. As we can see, the second case does not meet the criteria for stability, so we expect the numerical solution to be unstable. The other two solutions are expected to be stable and they are. Also, the solutions go from a value of 100 on the left towards a value of 50 on the right, which we would expect from the boundary conditions.

Table 2: Peclet number for each case.

Case	Pe
1	0.20000000000000004
2	5.0
3	1.25

4 Code

```
# -----------------------------------------#
# -- main script for NSE 565 HW1 --#
# ----------- Austin Warren -----------#
# ------------ Winter 2022 ------------#
# -----------------------------------------#

import numpy as np
import matplotlib.pyplot as plt

```

```
# define central difference scheme function to use for each case
def cds_ss(num_volumes, tot_length, velocity, density, diffusion, source, 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.
    source : float
        The source term in [units].
    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



# define analytical solution function to use for each case
def analytical(num_volumes, tot_length, velocity, density, diffusion, left, right):
    """Function to analytically 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
```

```
    x = np.linspace(dx/2, tot_length-(dx/2), num=num_volumes, endpoint=True)
    phi = left + (right-left) * (np.exp(density*velocity*x/diffusion)-1)/(np.exp(density*velocity*tot_length/diffusion)-1)
    return phi



#
def error_calc(phi_exact, phi):
    """Function to calculate error.

    Parameters
    ----------
    phi_exact : numpy.ndarray
        The exact solution.
    phi : numpy.ndarray
        The approximate solution.

    Returns
    -------
    error : float
        The average error for the approximate solution.
    """
    error = np.sum(np.abs(phi_exact - phi)) / len(phi)
    return error


# define constants
tot_length = 1  # pipe length in meters
density = 1     # density in kg/m^3
diffusion = 0.1 # diffusion coefficient in kg-s/m
source = 0      # source
left = 100      # left hand boundary condition
right = 50      # right hand boundary condition

# define variables
velocity = np.array([0.1, 2.5, 2.5])
num_volumes = np.array([5,5,20])
Pe = np.zeros(len(velocity))


error = np.zeros(len(velocity))
# solve
```

```python
for k in range(len(velocity)):
    phi = cds_ss(num_volumes[k], tot_length, velocity[k], density, diffusion, source, left, right)
    phi_exact = analytical(num_volumes[k], tot_length, velocity[k], density, diffusion, left, right)
    error[k] = error_calc(phi_exact,phi)
    dx = tot_length / num_volumes[k]
    Pe[k] = density * velocity[k] * dx / diffusion


    x = np.linspace(dx/2, tot_length-(dx/2), num=num_volumes[k])

    plt.rcParams['font.family'] = 'serif'
    plt.rcParams['mathtext.fontset'] = 'dejavuserif'
    plt.figure(facecolor='w', edgecolor='k', dpi=200)
    plt.plot(x, phi, '-k', label='CDS')
    plt.plot(x, phi_exact, '-r', label='Analytical')
    plt.xlabel('x (m)')
    plt.ylabel(r'$\phi$')
    plt.figlegend(loc='right', bbox_to_anchor=(0.4,0.2))
    plt.grid(b=True, which='major', axis='both')
    plt.savefig('HW1/plots/graph_case'+str(k+1)+'.png', transparent=True)
    plt.savefig('HW1/plots/graph_case'+str(k+1)+'.svg', transparent=True)



# generate latex table for error
out_file = open('HW1/tabs/error_tab.tex','w')
out_file.write(
                '\\begin{table}[htbp]\n'+
                '\t \centering\n'+
                '\t \caption{Error values for each case.}\n'+
                '\t \\begin{tabular}{cc}\n'+
                '\t\t \\toprule\n'+
                '\t\t Case & Error \\\ \n'+
                '\t\t \midrule \n'+
                '\t\t 1 & '+str(error[0])+' \\\ \n'+
                '\t\t 2 & '+str(error[1])+' \\\ \n'+
                '\t\t 3 & '+str(error[2])+' \\\ \n'+
                '\t\t \\bottomrule \n'+
                '\t \end{tabular} \n'+
                '\t \label{tab:error} \n'+
                '\end{table}'
```

```
)


# generate latex table for Peclet numbers
out_file = open('HW1/tabs/Pe_tab.tex','w')
out_file.write(
                '\\begin{table}[htbp]\n'+
                '\t \centering\n'+
                '\t \caption{Peclet number for each case.}\n'+
                '\t \\begin{tabular}{cc}\n'+
                '\t\t \\toprule\n'+
                '\t\t Case & $Pe$ \\\ \n'+
                '\t\t \midrule \n'+
                '\t\t 1 & '+str(Pe[0])+' \\\ \n'+
                '\t\t 2 & '+str(Pe[1])+' \\\ \n'+
                '\t\t 3 & '+str(Pe[2])+' \\\ \n'+
                '\t\t \\bottomrule \n'+
                '\t \end{tabular} \n'+
                '\t \label{tab:Pe} \n'+
                '\end{table}'
)
```