

Assignment Report

Lau Rasmussen, Victor Lanng,

Peter Würtz & Oskar Minds

Department:

Mechanical and Production Engineering

Course:

|  |  |  |
| --- | --- | --- |
| *Titel:* |  | Computational Fluid Dynamics |
| *ID:* |  | F25.290232U010.A |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| *Group nr:* |  | 6 |
| *Assign. Nr.:* |  | 1 |

Table of Contents

[1.1 discretization 1](#_Toc190478597)

[1.2 Solution computed 4](#_Toc190478598)

[1.3 Order of truncation error 5](#_Toc190478599)

[Appendix A: Code 7](#_Toc190478600)

[A.1 Task 1.2 code 7](#_Toc190478601)

[A.2 Additional task 1.3 code 8](#_Toc190478602)

## discretization

The differential form of the 1D steady-state convection-diffusion transport equation,

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

can be discretized using the finite volume method by evaluating within cells along the -axis. For a cell the left neighbouring cell is denoted and the right is . The face between cell and is , and is between and . Evaluating (1) for any cell , equation (1) is integrated between the cell face locations, to . If is constant, the integration gives,

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

### Non-boundary cells

Starting with the convective term on the LHS of (2), the face values, and , must be described as a function of the neighbouring cell centre values. Using a 2nd order central difference scheme (CDS), and , can be expressed as,

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

If instead using the 1st-order upwind difference scheme (UDS) they can be expressed as,

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

For the diffusion term on the RHS of (2) the differentials can also be expressed using the central difference scheme, giving,

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Substituting (5) and (3) into (2) gives,

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

If using (4) instead of (3) the substitution gives,

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Realizing that the terms within the parentheses are constants (6) and (7) can be expressed for the ’th cell as,

|  |  |  |
| --- | --- | --- |
|  | where: | (8) |

for .

### Boundary cells

Using Dirichlet boundary conditions, (3) and (4) must be modified for the boundary cells to,

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

where and are known values. For the case of CDS the boundary values are exact, so no truncation error arises here, but for UDS it is still 1st order accurate at one of the boundaries but exact at the other.

Equation (5) must also be modified with a one-sided 2nd order scheme:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Substituting (9) and (10) into (2) gives (11) for cell

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

and (12) for cell ,

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

These doesn’t fit into the first equation in (8) but the definitions therein can be substituted into (11) and (12) to give (13) for cell ,

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

and (14) for cell ,

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

### Solving the system

An equation for every cell is now defined with equation (8), (13), and (14). These can be combined to give the linear system,

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

where is a tridiagonal matrix,

## Solution computed

To compute the solution of the linear system of equations given in (15), a boundary condition is given,

Three different cases will be tested,

* Case 1:
* Case 2:
* Case 3:

where is the Pechlan number, which is defined as,

For case 3, will be negative. The solution in code is shown in Appendix A.1. Figure 1 compares the three different cases against the exact analytical solution to the problem given by,

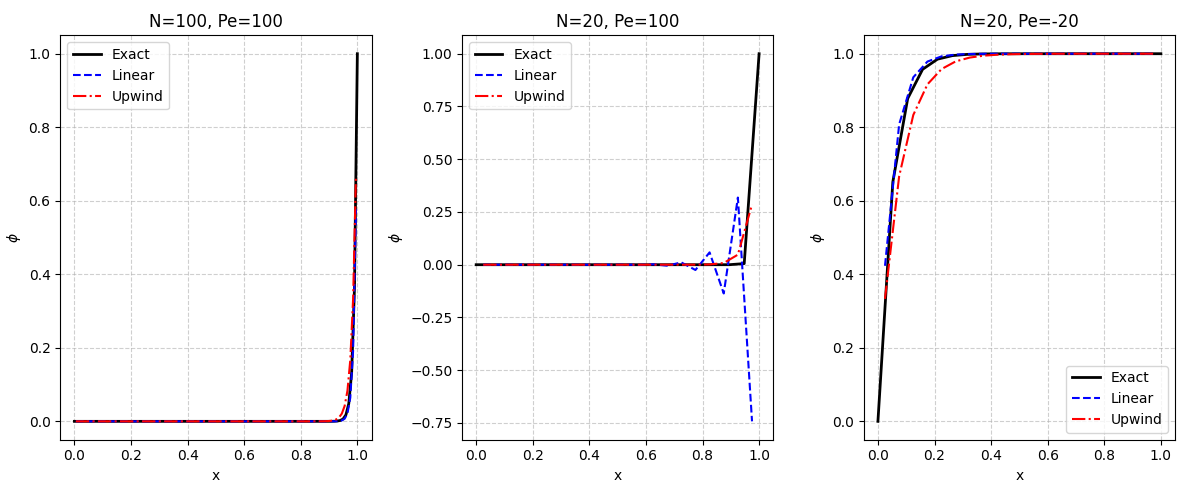


Figure 1: The solution for for three different cases of grid size and Pechlan number.

Notice that the linear and upwind graphs do not include the boundary values since they are not a part of the solution in the -vector. For case 1 both the CDS and UDS follows the exact solution well but for case 2, the CDS solution diverges because of instability. It can be shown that the tridiagonal matrix is ill-defined in this case because the grid size is too small compared to the Pechlan number, so the instability is expected. The third case shows that the solution also works for negative flows.

## Order of truncation error

To determine order of truncation error for both the UDS and CDS solution, simulations are run with and varying from 10 to 1000. The solution for each is compared against the exact solution using,

This is plotted in a double logarithmic plot using the code from Appendix A.2 and shown in Figure 2. As expected the CDS has a order of truncation error of 2.04 since the CDS is a 2nd order accuracy scheme. As mentioned, the UDS scheme is a 1st order scheme but the Figure states that it is 0.91 which is close to 1.

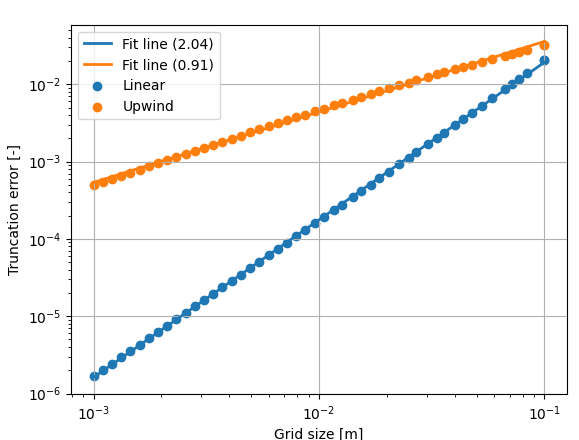


Figure 2: Global error as a function of in a double logarithmic plot for both the CDS and UDS case with .

##### Code

###### Task 1.2 code

import numpy as np

import matplotlib.pyplot as plt

def solve\_fvm(N, Pe, scheme='linear'):

    L = 1.0  # Domain length

    dx = L / N  # Grid spacing

    rho\_u = Pe  # product of density and velocity

    Gamma = 1  # Diffusion coefficient

    Q = np.zeros(N)

    A = np.zeros((N, N))

    A\_d = Gamma / dx  # Diffusion term

    # Choose advection scheme

    if scheme == 'linear':

        A\_c\_W, A\_c\_E = rho\_u / 2, rho\_u / 2  # Linear scheme

        # Apply boundary conditions (Dirichlet)

        # Right-hand side vector

        Q[0]  = ( 2\*A\_c\_W + 2\*A\_d)\*0    # Equation (13)

        Q[-1] = (-2\*A\_c\_W + 2\*A\_d)\*1    # Equation (14)

        # Left hand side matrix

        A[0, 0]   =  A\_c\_W + 3\*A\_d      # Equation (13)

        A[-1, -1] = -A\_c\_W + 3\*A\_d      # Equation (14)

    elif scheme == 'upwind':

        A\_c\_W, A\_c\_E = max(rho\_u, 0), min(rho\_u, 0)  # Upwind scheme

        # Apply boundary conditions (Dirichlet)

        # Right-hand side vector

        Q[0] =  ( A\_c\_W + 2\*A\_d)\*0      # Equation (13)

        Q[-1] = (-A\_c\_E + 2\*A\_d)\*1      # Equation (14)

        # Left hand side matrix

        A[0, 0]   = A\_c\_W-A\_c\_E + 3\*A\_d # Equation (13)

        A[-1, -1] = A\_c\_W-A\_c\_E + 3\*A\_d # Equation (14)

    else:

        raise ValueError("Unknown scheme. Use 'linear' or 'upwind'.")

    A[0, 1]   =  A\_c\_E - A\_d            # Equation (13)

    A[-1, -2] = -A\_c\_W - A\_d            # Equation (14)

    # Coefficients

    A\_W = -A\_c\_W - A\_d

    A\_E =  A\_c\_E - A\_d

    A\_P = -A\_W - A\_E

    # Construct coefficient matrix A

    for i in range(1, N-1):

        A[i, i-1] = A\_W

        A[i, i]   = A\_P

        A[i, i+1] = A\_E

    # Solve the system

    phi = np.linalg.solve(A, Q)

    x = np.linspace(dx/2, L - dx/2, N)  # Compute cell center positions

    return x, phi

def exact\_solution(x, Pe):

    return (np.exp(Pe \* x) - 1) / (np.exp(Pe) - 1)

def plot\_solutions():

    cases = [(100, 100), (20, 100), (20, -20)]  # (Grid size, Peclet number)

    plt.figure(figsize=(12, 5))

    for i, (N, Pe) in enumerate(cases, 1):

        x\_linear, phi\_linear = solve\_fvm(N, Pe, 'linear')

        x\_upwind, phi\_upwind = solve\_fvm(N, Pe, 'upwind')

        x\_exact = np.linspace(0, 1, N)

        phi\_exact = exact\_solution(x\_exact, Pe)

        plt.subplot(1, 3, i)

        plt.plot(x\_exact, phi\_exact, 'k-', label='Exact', linewidth=2)

        plt.plot(x\_linear, phi\_linear, 'b--', label='Linear', linewidth=1.5)

        plt.plot(x\_upwind, phi\_upwind, 'r-.', label='Upwind', linewidth=1.5)

        plt.xlabel('x')

        plt.ylabel('$\phi$')

        plt.title(f'N={N}, Pe={Pe}')

        plt.legend()

        plt.grid(True, linestyle='--', alpha=0.6)

    plt.tight\_layout()

    plt.show()

# Run the plot to visualize solutions

plot\_solutions()

###### Additional task 1.3 code

"""Truncation error plot"""

Pe = 20

L = 1

cases = np.logspace(1, 3, 50, dtype=int)

dx = L/cases

linear\_errors = np.empty(len(cases))

upwind\_errors = np.empty(len(cases))

fig, ax = plt.subplots()

for i, N in enumerate(cases):

    x\_linear, phi\_linear = solve\_fvm(N, Pe, 'linear')

    x\_upwind, phi\_upwind = solve\_fvm(N, Pe, 'upwind')

    phi\_exact = exact\_solution(x\_linear, Pe)

    linear\_errors[i] = np.sum(np.abs(phi\_exact - phi\_linear)) / N

    upwind\_errors[i] = np.sum(np.abs(phi\_exact - phi\_upwind)) / N

ax.set\_yscale('log')

ax.set\_xscale('log')

linear\_order\_of\_discretion\_error = np.polyfit(np.log10(dx), np.log10(linear\_errors), 1)

upwind\_order\_of\_discretion\_error = np.polyfit(np.log10(dx), np.log10(upwind\_errors), 1)

ax.plot((dx[0], dx[-1]), 10\*\*np.dot([[np.log10(dx[0]), 1], [np.log10(dx[-1]), 1]], linear\_order\_of\_discretion\_error),

        label=f'Fit line ({linear\_order\_of\_discretion\_error[0]:.2f})', linewidth=2.0)

ax.plot((dx[0], dx[-1]), 10\*\*np.dot([[np.log10(dx[0]), 1], [np.log10(dx[-1]), 1]], upwind\_order\_of\_discretion\_error),

        label=f'Fit line ({upwind\_order\_of\_discretion\_error[0]:.2f})', linewidth=2.0)

ax.scatter(dx, linear\_errors, label='Linear')

ax.scatter(dx, upwind\_errors, label='Upwind')

ax.grid()

ax.legend()

ax.set\_xlabel('Grid size [m]')

ax.set\_ylabel('Truncation error [-]')

plt.show()