hw1

January 21, 2026

1 AEROSP 423 Computational Methods: Homework 1

Jason Chen

1.1 Problem 1

$$\text{state:}\quad \mathbf{u} = \begin{bmatrix} E \\ f_x/c^2 \\ f_y/c^2 \end{bmatrix}, \quad \text{flux:}\quad \mathbf{F} = \begin{bmatrix} f_x n_x + f_y n_y \\ \frac{1}{3}E n_x \\ \frac{1}{3}E n_y \end{bmatrix}$$

### Part (a) Calculate the flux Jacobian matrix.

Generally speaking, the Jacobian matrix is:

$$\mathbf{A} = \frac{\partial \mathbf{F}}{\partial \mathbf{u}} = \begin{bmatrix} \frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2} & \frac{\partial F_1}{\partial u_3} \\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2} & \frac{\partial F_2}{\partial u_3} \\ \frac{\partial F_3}{\partial u_1} & \frac{\partial F_3}{\partial u_2} & \frac{\partial F_3}{\partial u_3} \end{bmatrix}$$

However, we need to write each component of the flux in terms of the state variables, e.g. u_1. So the flux matrix becomes:

$$\mathbf{F} = \begin{bmatrix} c^2(u_2 n_x + u_3 n_y) \\ \frac{1}{3}u_1 n_x \\ \frac{1}{3}u_1 n_y \end{bmatrix}$$

So the Jacobian:

$$\mathbf{A} = \begin{bmatrix} 0 & c^2 n_x & c^2 n_y \\ \frac{1}{3}n_x & 0 & 0 \\ \frac{1}{3}n_y & 0 & 0 \end{bmatrix}$$

1.1.1 Part (b)

Compute the eigenvalues and right eigenvectors of $\mathbf{A}$.

We will use the eigenvalue characteristic equation:

$$\mathbf{det}(A - \lambda I) = 0$$

$$\begin{vmatrix} -\lambda & c^2 n_x & c^2 n_y \\ \frac{1}{3}n_x & -\lambda & 0 \\ \frac{1}{3}n_y & 0 & -\lambda \end{vmatrix} = 0$$

$$= -\lambda(\lambda^2) - c^2 n_x \left(-\lambda \frac{1}{3} n_x\right) + c^2 n_y \left(\lambda \frac{1}{3} n_y\right)$$

$$= -\lambda^3 + \lambda\frac{c^2}{3}(n_x^2 + n_y^2)$$

Since $\vec{n}$ is a normal vector, the term $(n_x^2 + n_y^2)$ equals 1. So the characteristic equation becomes:

$$\lambda\left(-\lambda^2 + \frac{c^2}{3}\right) = 0$$

Eigenvalues:

$$\lambda_1 = 0, \quad \lambda_2 = \frac{c}{\sqrt{3}}, \quad \lambda_3 = -\frac{c}{\sqrt{3}}$$

To solve for the eigenvectors, we go through each eigenvalue and solve again for the equation $A\vec{v} = \lambda\vec{v}$. For $\lambda_1 = 0$:

$$\begin{bmatrix} 0 & c^2 n_x & c^2 n_y \\ \frac{1}{3}n_x & 0 & 0 \\ \frac{1}{3}n_y & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

From rows 2 and 3, we know immediately that $v_1 = 0$ since each component n_x and n_y are given to be nonzero. The first row gives:

$$c^2 n_x v_2 + c^2 n_y v_3 = 0$$

So, $v_3 = -\frac{n_x}{n_y} v_2$. If $v_2 = n_y$:

$$\mathbf{v_1} = \begin{bmatrix} 0 \\ n_y \\ -n_x \end{bmatrix}$$

Repeating this process for $\lambda_2 = \frac{c}{\sqrt{3}}$, we once again need to solve the characteristic equation:

$$\begin{bmatrix} 0 - \frac{c}{\sqrt{3}} & c^2 n_x & c^2 n_y \\ \frac{1}{3}n_x & 0 - \frac{c}{\sqrt{3}} & 0 \\ \frac{1}{3}n_y & 0 & 0 - \frac{c}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Equation 1 (row 1):

$$-\frac{c}{\sqrt{3}}v_1 + c^2 n_x v_2 + c^2 n_y v_3 = 0$$

Equation 2 (row 2):

$$\frac{1}{3}n_x v_1 - \frac{c}{\sqrt{3}}v_2 = 0$$

Equation 3 (row 3):

$$\frac{1}{3}n_y v_1 - \frac{c}{\sqrt{3}}v_3 = 0$$

We can choose to solve row 2 for v_2 and row 3 for v_3:

$$v_2 = \frac{n_x v_1}{3}\left(\frac{\sqrt{3}}{c}\right) = \frac{\sqrt{3}n_x v_1}{3c} = \frac{n_x v_1}{\sqrt{3}c}$$

$$v_3 = \frac{n_y v_1}{3}\left(\frac{\sqrt{3}}{c}\right) = \frac{\sqrt{3}n_y v_1}{3c} = \frac{n_y v_1}{\sqrt{3}c}$$

And choosing $v_1 = \sqrt{3}c$ to eliminate the fractions and simplify:

$$v_2 = \frac{n_x(\sqrt{3}c)}{\sqrt{3}c} = n_x$$

$$v_3 = \frac{n_y(\sqrt{3}c)}{\sqrt{3}c} = n_y$$

We finally get:

$$\mathbf{v_2} = \begin{bmatrix} \sqrt{3}c \\ n_x \\ n_y \end{bmatrix}$$

Repeating the above process for $\lambda_3 = -\frac{c}{\sqrt{3}}$ (and using MATLAB `eig()` to check):

$$\mathbf{v_3} = \begin{bmatrix} \sqrt{3}c \\ -n_x \\ -n_y \end{bmatrix}$$

1.1.2 Part (c)

Determine the null space of **A**.

The null space (kernel) of **A** is all vectors which get transformed to the zero vector after applying **A**, i.e. we need to solve:

$$\begin{bmatrix} 0 & c^2 n_x & c^2 n_y \\ \frac{1}{3}n_x & 0 & 0 \\ \frac{1}{3}n_y & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We can notice that this is the same equation we solved for in the $\lambda_1 = 0$ case, which makes sense. Thus the null space consists of all vectors that satisfy:

$$\mathbf{x} = t\begin{bmatrix} 0 \\ n_y \\ -n_x \end{bmatrix}$$

Above, t is any scalar value (any vector that is linearly dependent to $\mathbf{x}$ is part of the null space).

1.2 Problem 2

Consider a mass, $m = 0.1$ kg, attached to a spring with stiffness $k = 200$ N/m and damping c, in the absence of a gravitational field. The horizontal force on the mass is $F = -kx - c\dot{x}$, where x is the position of the mass measured to the right from the equilibrium length of the spring.

1.2.1 Part (a)

Write the dynamics of the system in state space form.

Given that the state is:

$$\mathbf{u} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

The derivative is:

$$\dot{\mathbf{u}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix}$$

Since we know $F = m\ddot{x}$, we can rearrange the force equation to:

$$\ddot{x} = -\frac{1}{m}(kx + c\dot{x})$$

Thus substituting into the derivative:

$$\dot{\mathbf{u}} = \begin{bmatrix} u_2 \\ -\frac{1}{m}(ku_1 + cu_2) \end{bmatrix}$$

To put this in the form of $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u}$, we can just extract the state vector from above to separate out $\mathbf{A}$:

$$\dot{\mathbf{u}} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

1.2.2 Part (b)

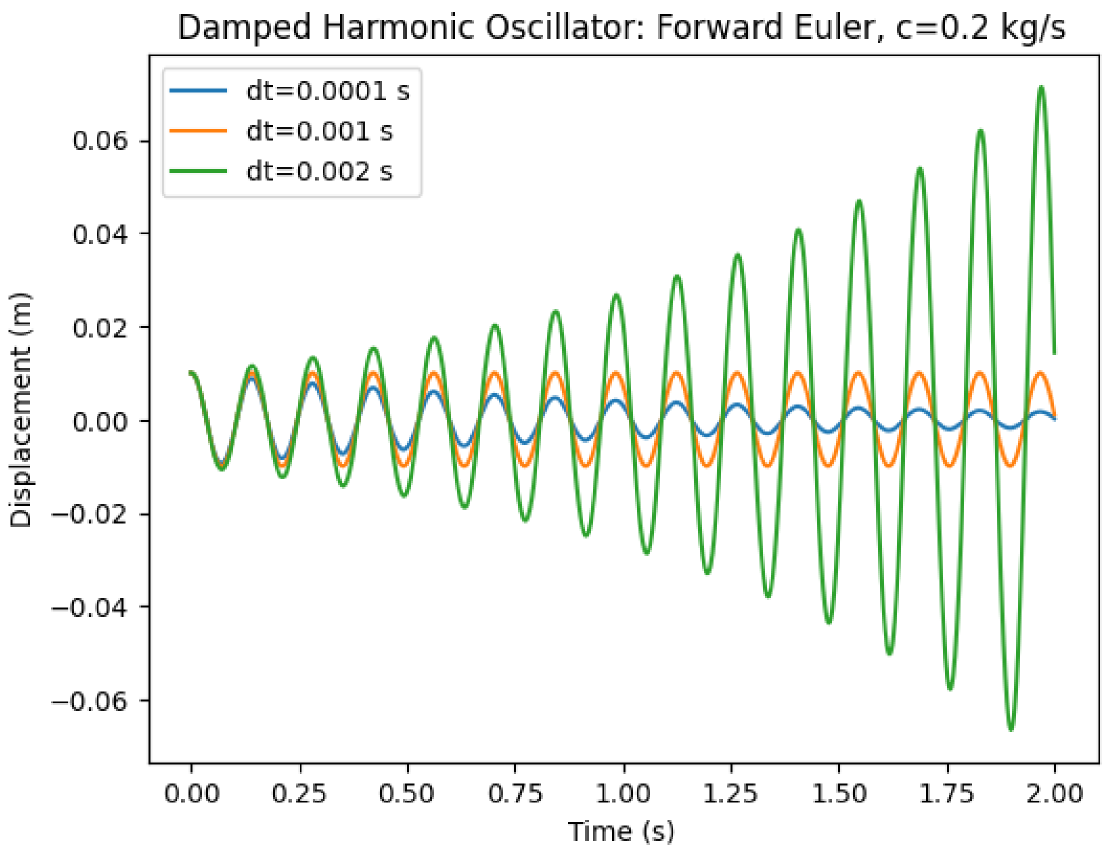
We're asked to perform Forward Euler to simulate this system using: - $\mathbf{u_0} = [0.1, 0]^T$ - Time horizon $T = 2$ seconds - Try $c = 2$ kg/s and $c = 0.2$ kg/s and determine the largest T we can use before obtaining unstable results

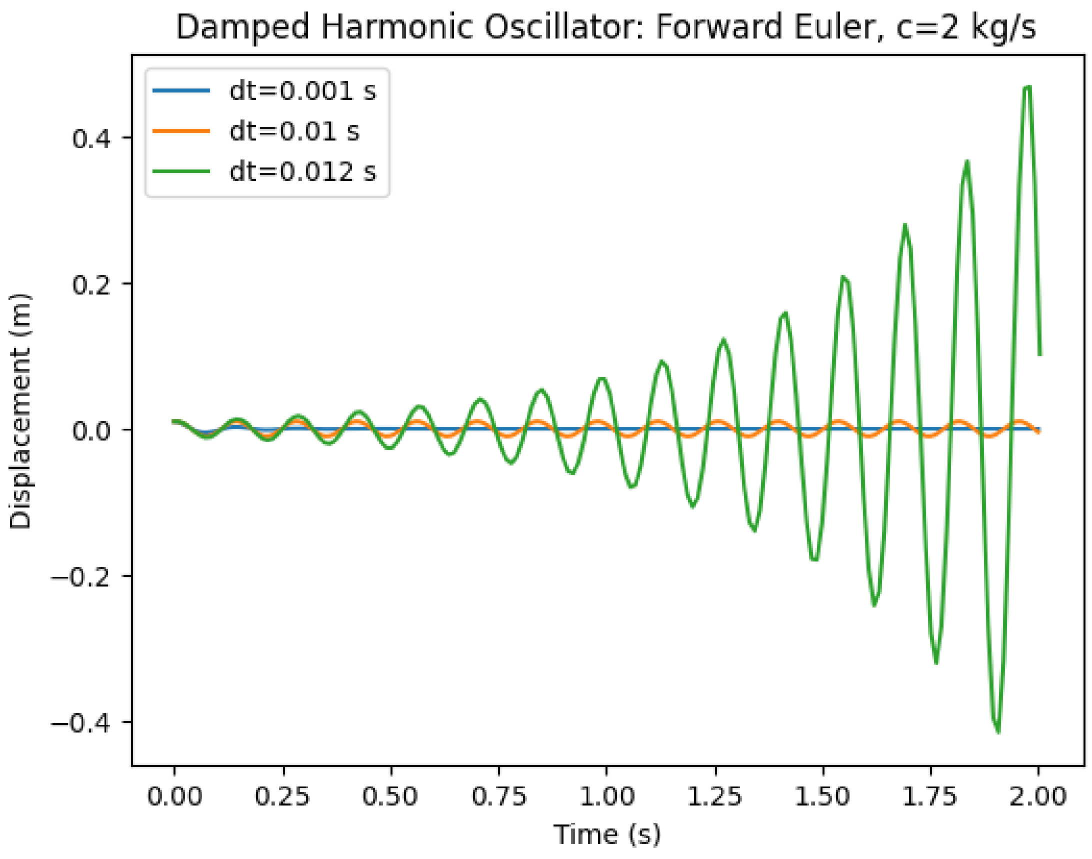
We know the update equation for the state in FE is simply:

$$\boxed{\mathbf{u^{n+1}} = \mathbf{u^n} + \Delta t \mathbf{f}(\mathbf{u^n})}$$

We will do this simulation in Python, shown in the following snippet:

[21]:
```
import numpy as np
import matplotlib.pyplot as plt

def problem_2b(c, dt_values):
    """Forward Euler solution for a damped harmonic oscillator, with variable␣
 ↪damping c."""
    def f(u, c):
        """Computes the derivative of the state vector u."""
        A = np.array([[0, 1], [-k/m, -c/m]])
        return A @ u
    m = 0.1
    k = 200
    u_0 = np.array([[0.01], [0]])
    T = 2
    for dt in dt_values:
        time = np.arange(0, T + dt, dt)
        u_states = np.zeros((2, len(time)))
        u_states[:, 0] = u_0.flatten()
        for i in range(1, len(time)):
            u_states[:, i] = u_states[:, i-1] + dt * f(u_states[:, i-1], c)
        plt.plot(time, u_states[0, :], label=f'dt={dt} s')

    # Plotting settings
    plt.title(f'Damped Harmonic Oscillator: Forward Euler, c={c} kg/s')
    plt.xlabel('Time (s)')
    plt.ylabel('Displacement (m)')
    plt.legend()
    plt.show()

problem_2b(0.2, [1e-4, 1e-3, 2e-3])
problem_2b(2, [0.001, 0.01, 0.012])
```





From empirical iteration, it was found that $\boxed{\Delta t > 1 \text{ ms}}$ caused the amplitude to diverge for $c = 0.2$ kg/s, and $\boxed{\Delta t > 10 \text{ ms}}$ for $c = 2$ kg/s.

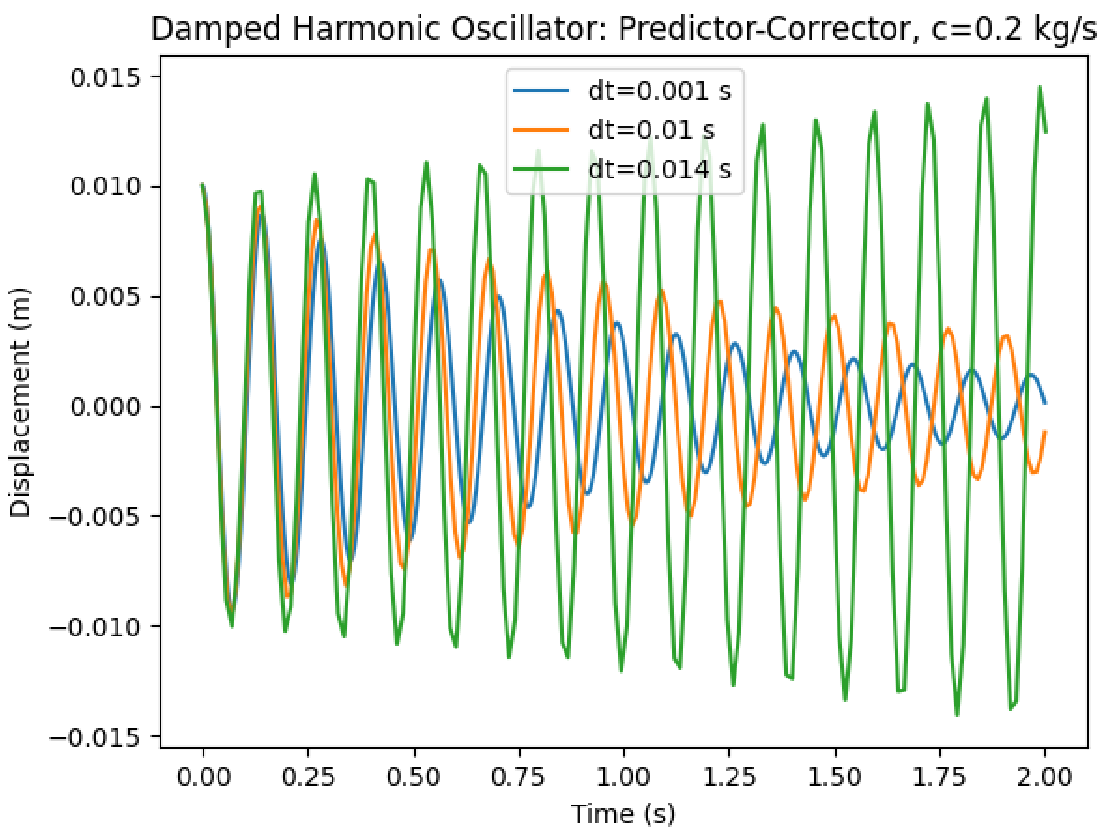
1.2.3 Part (c)

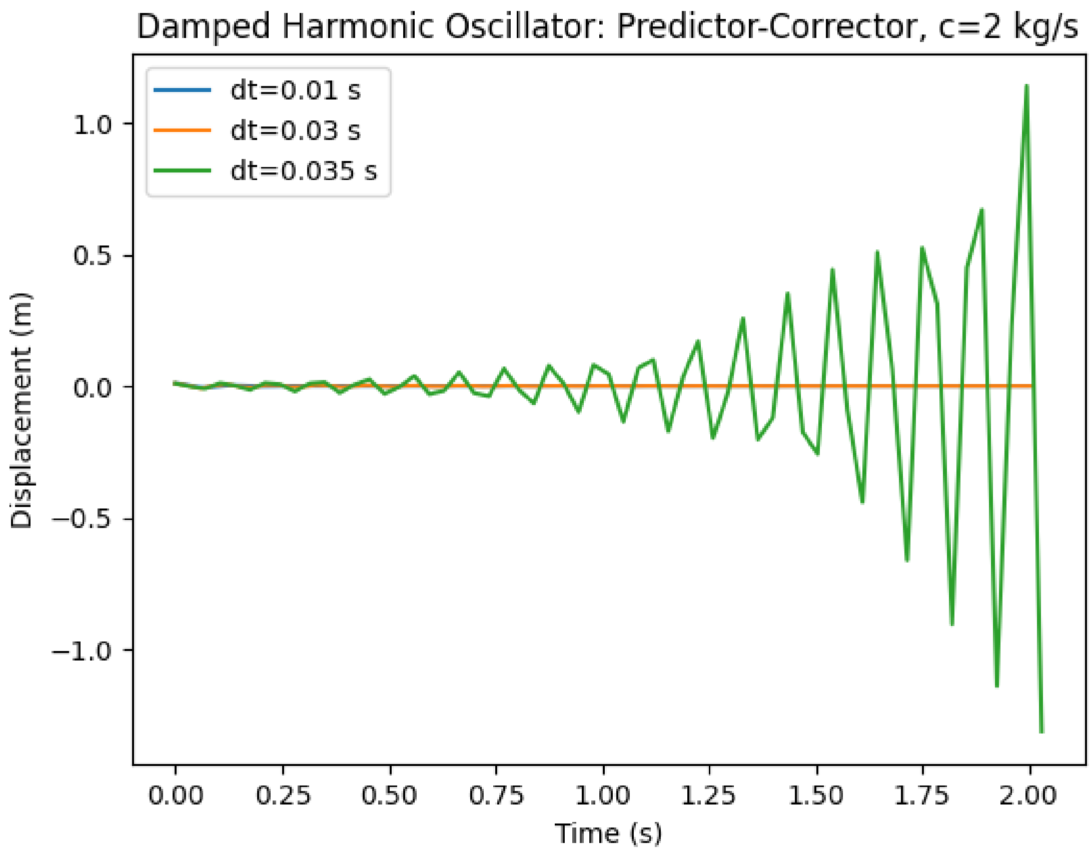
Repeat the above with the Pedictor-Corrector method.

The Pedictor-Corrector method composes of two steps: it first just does Forward Euler (to obtain $\tilde{\mathbf{u}}^{n+1}$), and then uses this new state to get a better estimate for the derivative at $\mathbf{u^n}$ by averaging:

$$\boxed{\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \frac{\mathbf{f}(\mathbf{u}^n) + \mathbf{f}(\tilde{\mathbf{u}}^{n+1})}{2}}$$

[39]:
```python
def problem_2c(c, dt_values):
    """Predictor-Corrector solution for a damped harmonic oscillator, with␣
 ↪variable damping c."""
    def f(u, c):
        """Computes the derivative of the state vector u."""
        A = np.array([[0, 1], [-k/m, -c/m]])
        return A @ u
    m = 0.1
    k = 200
    u_0 = np.array([[0.01], [0]])
    T = 2
    for dt in dt_values:
        time = np.arange(0, T + dt, dt)
        u_states = np.zeros((2, len(time)))
        u_states[:, 0] = u_0.flatten()
        for i in range(1, len(time)):
            u_tilde = u_states[:, i-1] + dt * f(u_states[:, i-1], c)
            u_states[:, i] = u_states[:, i-1] + dt * (f(u_states[:, i-1], c) +␣
 ↪f(u_tilde, c)) / 2
        plt.plot(time, u_states[0, :], label=f'dt={dt} s')

    # Plotting settings
    plt.title(f'Damped Harmonic Oscillator: Predictor-Corrector, c={c} kg/s')
    plt.xlabel('Time (s)')
    plt.ylabel('Displacement (m)')
    plt.legend()
    plt.show()

problem_2c(0.2, [1e-3, 1e-2, 0.014])
problem_2c(2, [0.01, 0.03, 0.035])
```





From empirical iteration, it was found that $\boxed{\Delta t > 0.013 \text{ s}}$ caused the amplitude to diverge for $c = 0.2$ kg/s, and $\boxed{\Delta t > 0.034 \text{ s}}$ for $c = 2$ kg/s.

Fundamentally, the difference between FE and PC is a slight trade in computational complexity (PC needs to do an additional averaging operation) to buy a potentially smaller timestep size. The reason the timestep size can be decreased is because a more accurate slope that is used during integration reduces the compounding of error in the integration process. We can see above from our experiment that this is true, as the PC method is able to get away with a smaller timestep size before divergence in the simulation when compared to FE.

1.3 Problem 3

We consider N aircraft flying in side-by-side formation, each modeled as a horseshoe vortex with span b. The key physics here is that each aircraft's trailing vortices induce a vertical velocity (downwash) on neighboring aircraft, which affects the induced drag. The goal is to find the weight distribution W_i such that all aircraft have the same lift-to-induced-drag ratio.

From the problem statement, we have: - Lift: $L = \rho b U_\infty \Gamma_i$ - Induced drag: $D = -\rho b w_i \Gamma_i$

The lift-to-induced-drag ratio is:

$$\frac{L}{D} = \frac{\rho b U_\infty \Gamma_i}{-\rho b w_i \Gamma_i} = -\frac{U_\infty}{w_i}$$

For all aircraft to have the same L/D, they must all experience the same downwash $w_i = w$ (constant). Since w_i depends on the circulation distribution (and thus the weight distribution), we need to find Γ_i (equivalently W_i) that makes w_i constant for all aircraft. The induced velocity at aircraft i due to all aircraft's trailing vortices is:

$$w_i = \sum_{j=1}^{N} \Delta w_{ij} = \sum_{j=1}^{N} \left[\frac{\Gamma_j}{4\pi(y_i - y_j - b/2)} - \frac{\Gamma_j}{4\pi(y_i - y_j + b/2)} \right]$$

Now we can set up a system of equations. Setting $w_i = w$ for all aircraft gives us N equations:

$$\sum_{j=1}^{N} \left[\frac{\Gamma_j}{4\pi(y_i - y_j - b/2)} - \frac{\Gamma_j}{4\pi(y_i - y_j + b/2)} \right] = w, \quad i = 1, \ldots, N$$

This is N equations with $N + 1$ unknowns ($\Gamma_1, \ldots, \Gamma_N$ and w). We can solve the system with a normalization constraint that the average horseshoe vertex strength equals some reference value $\bar{\Gamma}$:

$$\frac{1}{N} \sum_{j=1}^{N} \Gamma_j = \bar{\Gamma}$$

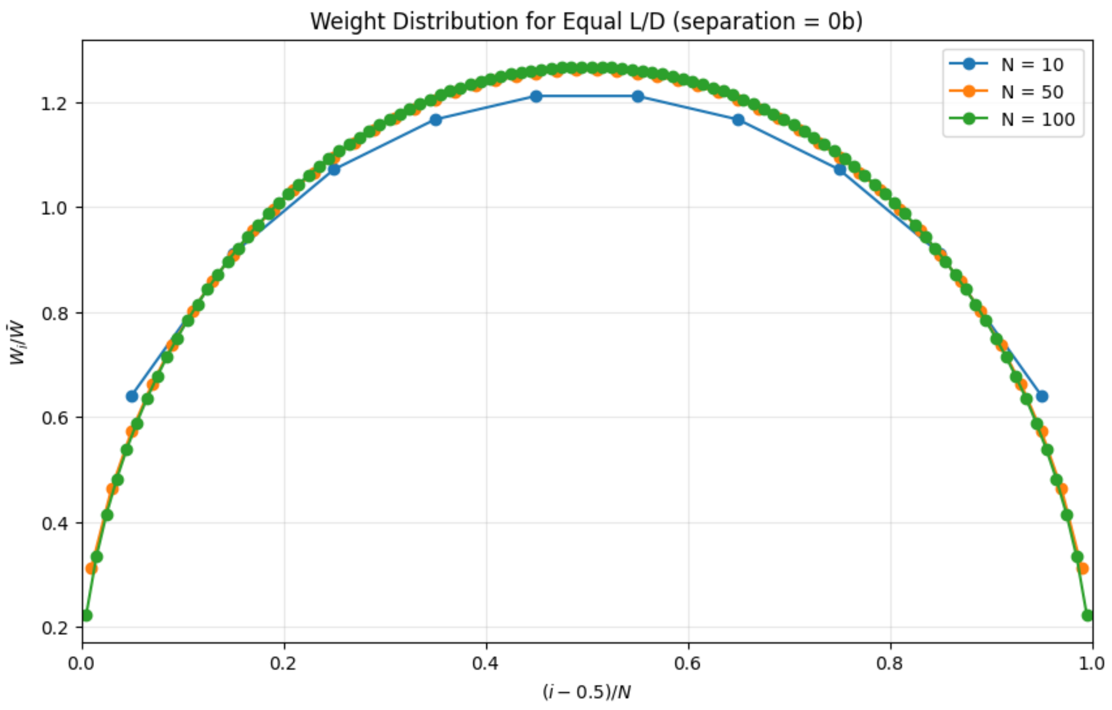
Since lift is proportional to Γ and weight equals lift in steady flight, we have $W_i/\bar{W} = \Gamma_i/\bar{\Gamma}$.

Finally we can define $a_{ij} = \frac{1}{4\pi(y_i - y_j - b/2)} - \frac{1}{4\pi(y_i - y_j + b/2)}$ for compactness, and write:

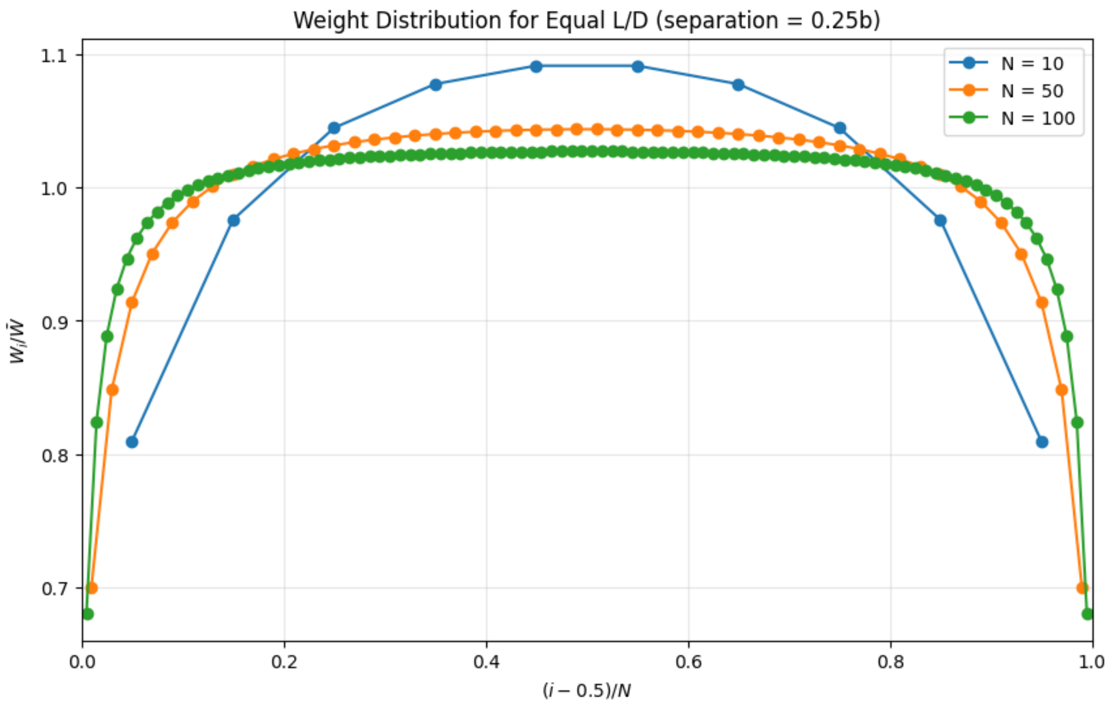
$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} & -1 \\ a_{21} & a_{22} & \cdots & a_{2N} & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} & -1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \vdots \\ \Gamma_N \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ N\bar{\Gamma} \end{bmatrix}$$

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

def formation_flight(N_values, separation):
    b = 1.0  # Normalize span to 1
    plt.figure(figsize=(10, 6))

    for N in N_values:
        # Aircraft positions: evenly spaced, centered at y=0, each aircraft is␣
↪separated by (b + separation)
        spacing = b + separation
        y = np.array([(i - (N-1)/2) * spacing for i in range(N)])
        A = np.zeros((N+1, N+1))
        for i in range(N):
            for j in range(N):
                dy = y[i] - y[j]
                denom_left = dy - b/2
                denom_right = dy + b/2
                # a_ij from the induced velocity formula
                if abs(denom_left) > 1e-10:
                    A[i, j] += 1 / (4 * np.pi * denom_left)
                if abs(denom_right) > 1e-10:
                    A[i, j] -= 1 / (4 * np.pi * denom_right)
        A[:N, N] = -1
        # Normalization constraint: sum of all gamma = N * gamma_bar
        A[N, :N] = 1
        A[N, N] = 0
        gamma_bar = 1.0
        rhs = np.zeros(N+1)
        rhs[N] = N * gamma_bar
        # Solve the linear system
        solution = np.linalg.solve(A, rhs)
        gamma = solution[:N]
        W_ratio = gamma / gamma_bar

        # Plot against normalized position (i - 0.5) / N
        x_plot = (np.arange(1, N+1) - 0.5) / N
        plt.plot(x_plot, W_ratio, 'o-', label=f'N = {N}', markersize=6)
    plt.xlabel(r'$(i - 0.5)/N$')
    plt.ylabel(r'$W_i / \bar{W}$')
    plt.title(f'Weight Distribution for Equal L/D (separation = {separation}b)')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.xlim([0, 1])
    plt.show()

# Part 1: Separation distance of 0
print("Case 1: Separation distance = 0")
formation_flight([10, 50, 100], separation=0)

# Part 2: Separation distance of b/4
print("\nCase 2: Separation distance = b/4")
formation_flight([10, 50, 100], separation=0.25)
```

```
Case 1: Separation distance = 0
```



```
Case 2: Separation distance = b/4
```



We can see above that when the aircraft fly with zero gap between wingtips, the weight distribution shows a characteristic shape where aircraft near the edges of the formation need to be lighter than those in the center. This is because the edge aircraft only receive beneficial upwash from neighbors on one side, while center aircraft benefit from neighbors on both sides. The distribution converges to a smooth curve as N increases.

With a gap of $b/4$ between aircraft, the overall trend is similar, but the magnitude of the weight variation is reduced. The beneficial interference effects are weaker because the trailing vortices are farther from neighboring aircraft. This means the formation is more “democratic” — the weight penalty for edge aircraft is less severe, but the overall efficiency gain from formation flight is also reduced.