AERO 430 HW 5

Judson Upchurch

2024-12-01

Table of Contents

1 Problem Formulation

We consider a vibrating bar of uniform cross-sectional area A, Young's modulus E, and density ρ. The bar undergoes longitudinal forced vibrations with displacement $u(x,t)$ of the form:

$$u(x,t) = U(x)\sin(\Omega t), \tag{1}$$

where $U(x)$ is the amplitude displacement function to be determined.

1.1 Dynamic Equilibrium

The equation of motion for a small segment of the bar of length $\mathrm{d}x$ can be derived from the force balance:

$$\sum F = m\ddot{u}. \tag{2}$$

The axial forces at $x+\frac{\mathrm{d}x}{2}$ and $x-\frac{\mathrm{d}x}{2}$ are given by:

$$N\left(x+\frac{\mathrm{d}x}{2},t\right) - N\left(x-\frac{\mathrm{d}x}{2},t\right) = \rho A\ddot{u}(x,t)\mathrm{d}x. \tag{3}$$

Taking the limit as $\mathrm{d}x \to 0$, we obtain:

$$\frac{\partial N(x,t)}{\partial x} = \rho A\ddot{u}(x,t). \tag{4}$$

1.2 Constitutive Relation

Assuming linear elasticity, the axial force $N(x,t)$ is related to the displacement gradient by:

$$N = AE\frac{\partial u(x,t)}{\partial x}. \tag{5}$$

Substituting this into the equilibrium equation gives:

$$\frac{\partial}{\partial x}\left(AE\frac{\partial u(x,t)}{\partial x}\right) = \rho A\frac{\partial^2 u(x,t)}{\partial t^2}. \tag{6}$$

1.3 Separation of Variables

Assuming a solution of the form $u(x,t) = U(x)\sin(\Omega t)$, we substitute into the governing equation:

$$AE\frac{\partial^2 U(x)}{\partial x^2}\sin(\Omega t) = -\rho A\Omega^2 U(x)\sin(\Omega t). \tag{7}$$

Canceling $\sin(\Omega t)$ (non-trivial for harmonic motion) gives:

$$AE\frac{\mathrm{d}^2 U(x)}{\mathrm{d}x^2} = -\rho A\Omega^2 U(x). \tag{8}$$

Rewriting:

$$\frac{\mathrm{d}^2 U(x)}{\mathrm{d}x^2} + \alpha^2 U(x) = 0, \tag{9}$$

where $\alpha^2 = \frac{\rho\Omega^2}{E}$.

1.4 General Solution

The general solution for $U(x)$ is:

$$U(x) = C_1\sin(\alpha x) + C_2\cos(\alpha x). \tag{10}$$

1.5 Boundary Conditions and Specific Solution

Given the boundary conditions:

$$U(0) = 0 \quad \text{and} \quad U(1) = 100, \tag{11}$$

we find:

$$C_2 = 0, \quad C_1 = \frac{100}{\sin(\alpha)}.$$

Thus, the specific solution is:

$$U(x) = \frac{100\sin(\alpha x)}{\sin(\alpha)}. \tag{12}$$

1.6 Strain Amplitude

The strain amplitude, defined as the gradient of the displacement $U(x)$, is given by:

$$\text{Strain amplitude: } \frac{\mathrm{d}U(x)}{\mathrm{d}x}. \tag{13}$$

Differentiating $U(x)$:

$$\frac{\mathrm{d}U(x)}{\mathrm{d}x} = \frac{100\alpha\cos(\alpha x)}{\sin(\alpha)}. \tag{14}$$

1.7 Resonance Condition

Resonance occurs when $\alpha = n\pi$, where n is an integer. This corresponds to:

$$\Omega = n\pi\sqrt{\frac{E}{\rho}}. \tag{15}$$

1.8 Summary

The amplitude displacement function $U(x)$ is determined as:

$$U(x) = \frac{100\sin(\alpha x)}{\sin(\alpha)} \quad \text{with} \quad \alpha = \sqrt{\frac{\rho\Omega^2}{E}}. \tag{16}$$

The strain amplitude is:

$$\frac{\mathrm{d}U(x)}{\mathrm{d}x} = \frac{100\alpha\cos(\alpha x)}{\sin(\alpha)}. \tag{17}$$

The resonance frequencies are given by:

$$\Omega = n\pi\sqrt{\frac{E}{\rho}}, \quad n \in \mathbb{Z}^+. \tag{18}$$

2 Analytical Solution

For the displacement $U(x)$ and strain amplitude $\frac{\mathrm{d}U(x)}{\mathrm{d}x}$, we analyze their behavior for a range of α values:

$$\alpha = \{0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 20.0\}.$$

2.1 Displacement

The displacement function $U(x)$ is given by:

$$U(x) = \frac{100\sin(\alpha x)}{\sin(\alpha)}. \tag{19}$$

The figure below illustrates the displacement $U(x)$ as a function of x for various α values. Each curve represents a different value of α, showing how the displacement profile changes with increasing or decreasing α. This provides insight into the spatial response of the bar under different vibration conditions.

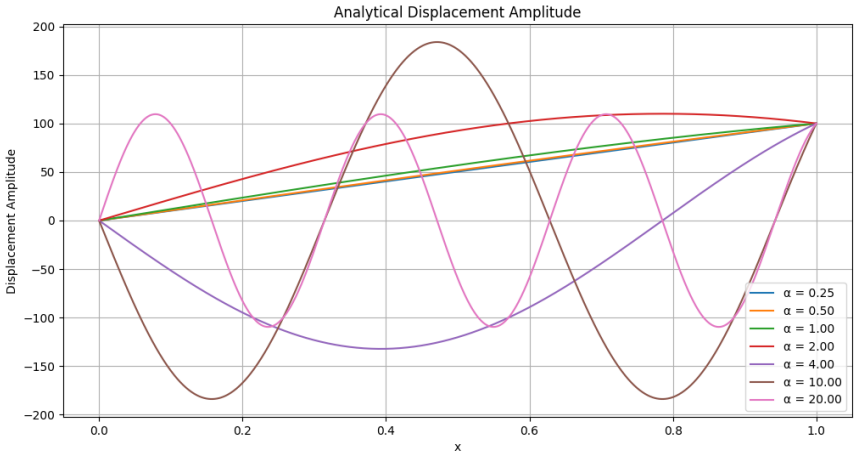

Figure 1: Displacement $U(x)$ for various values of α.

The table below shows the numerical values of $U(x)$ at specific points along the bar.

x	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 4.0$	$\alpha = 10.0$	$\alpha = 20.0$
0.000	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000
0.125	12.6291	13.0279	14.8163	27.2083	-63.3488	-174.4389	65.5540
$\frac{1}{2\pi}$	16.0783	16.5810	18.8341	34.4180	-78.5516	-183.7768	-4.5451
0.250	25.2459	26.0050	29.4014	52.7248	-111.1877	-110.0090	-105.0363
0.375	37.8380	38.8806	43.5276	74.9632	-131.8039	105.0623	102.7444
0.500	50.3932	51.6043	56.9747	92.5408	-120.1499	176.2660	-59.5897
0.625	62.8992	64.1264	69.5327	104.3646	-79.0790	6.0989	-7.2646
0.750	75.3437	76.3982	81.0056	109.6995	-18.6469	-172.4198	71.2297
0.875	87.7147	88.3716	91.2145	108.2139	46.3507	-114.8345	-106.8658
1.000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000

Table 1: Numerical values of $U(x)$ for various values of α at selected points x.

2.2 Strain

The strain amplitude is given by:

$$\frac{dU(x)}{dx} = \frac{100\alpha\cos(\alpha x)}{\sin(\alpha)}. \tag{20}$$

The figure below depicts the strain amplitude $\frac{dU(x)}{dx}$ as a function of x for the same α values. Each curve highlights how the strain distribution varies with different vibration parameters, showcasing the behavior of the material under oscillatory loads.

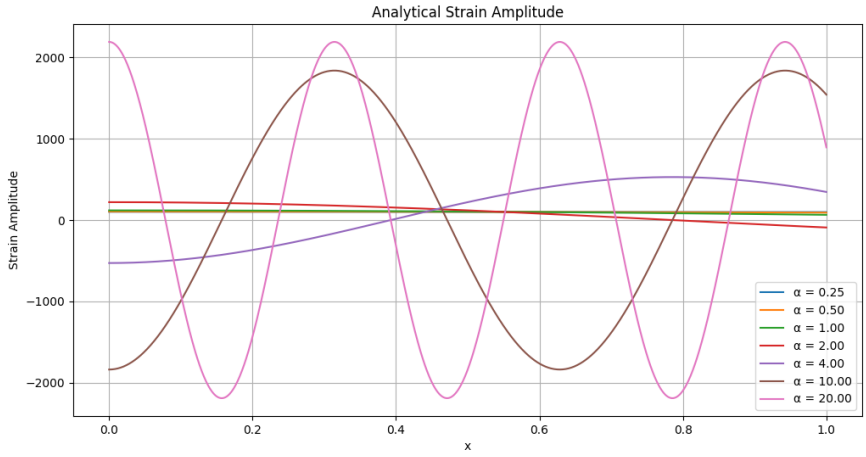

Figure 2: Strain amplitude $\frac{dU(x)}{dx}$ for various values of α.

The table below lists the numerical values of $\frac{dU(x)}{dx}$ at the same points.

x	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 4.0$	$\alpha = 10.0$	$\alpha = 20.0$
0.000	101.0493	104.2915	118.8395	219.9500	-528.5395	-1838.1640	2190.7119
0.125	101.0000	104.0879	117.9123	213.1123	-463.8370	-579.6142	-1755.0748
$\frac{1}{2\pi}$	100.9693	103.9614	117.3376	208.9010	-425.0038	38.1449	-2188.8251
0.250	100.8520	103.4778	115.1451	193.0243	-285.5711	1472.6333	621.4221
0.375	100.6056	102.4636	110.5811	160.9350	-37.3874	1508.3226	759.3781
0.500	100.2609	101.0493	104.2915	118.8395	219.9500	-521.4176	-1838.1640
0.625	99.8183	99.2404	96.3745	69.3552	423.4360	-1837.1519	2185.8885
0.750	99.2783	97.0440	86.9535	15.5586	523.2501	-637.1725	-1664.2573
0.875	98.6413	94.4687	76.1758	-39.2052	494.9543	1435.3224	480.7297
1.000	97.9079	91.5244	64.2093	-91.5315	345.4765	1542.3510	893.9902

Table 2: Numerical values of $\frac{dU(x)}{dx}$ for various values of α at selected points x.

2.3 Frequency Sweep Analysis

The frequency sweep examines the displacement $U(x)$ at $x = \frac{1}{2\pi}$ as a function of the forcing frequency Ω. This analysis helps identify resonances and overall behavior as the frequency varies.
The figure below shows the displacement at $x = \frac{1}{2\pi}$ for a range of forcing frequencies Ω.

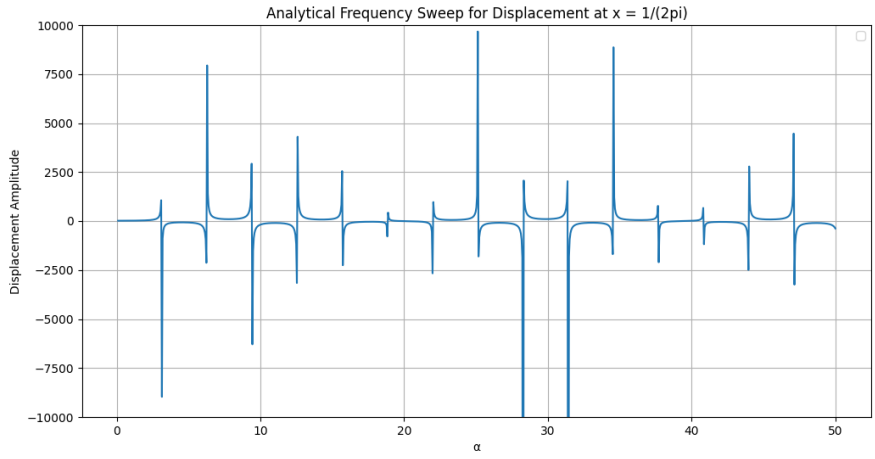

Figure 3: Displacement at $x = \frac{1}{2\pi}$ as a function of forcing frequency Ω.

3 Finite Difference Method (FDM) Derivation

The second-order differential equation governing the forced vibration of a bar is:

$$\frac{\mathrm{d}^2U(x)}{\mathrm{d}x^2} = -\alpha^2 U(x),$$

where $\alpha^2 = \frac{\rho\Omega^2}{E}$.
The finite difference method will seek to approximate the second derivative of $U(x)$ with 2nd and 4th order accuracy, then apply the governing differential equation over each sub element to form a system of equations.

3.1 Taylor Series Expansions

To approximate the derivatives, the Taylor series expansions of $U(x)$ around a point x_i are used:

$$U_{i-1} = U_i - \Delta x U_i' + \frac{\Delta x^2}{2}U_i'' - \frac{\Delta x^3}{6}U_i''' + \frac{\Delta x^4}{24}U_i^{(4)} + \mathscr{O}(\Delta x^5), \tag{21}$$

$$U_{i+1} = U_i + \Delta x U_i' + \frac{\Delta x^2}{2}U_i'' + \frac{\Delta x^3}{6}U_i''' + \frac{\Delta x^4}{24}U_i^{(4)} + \mathscr{O}(\Delta x^5). \tag{22}$$

Adding and subtracting these equations:

$$U_{i-1} + U_{i+1} = 2U_i + \Delta x^2 U_i'' + \frac{\Delta x^4}{12}U_i^{(4)}, \tag{23}$$

$$U_{i+1} - U_{i-1} = 2\Delta x U_i' + \frac{\Delta x^3}{3}U_i'''. \tag{24}$$

3.2 Second-Order Approximation

The second-order approximation for U_i'' is obtained by ignoring the fourth-order term in the summation equation:

$$\Delta x^2 U_i'' = U_{i-1} - 2U_i + U_{i+1}.$$

Rearranging:

$$U_i'' = \frac{U_{i-1} - 2U_i + U_{i+1}}{\Delta x^2}.$$

Substituting U_i'' into the governing equation:

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{\Delta x^2} + \alpha^2 U_i = 0.$$

Rewriting:

$$U_{i-1} - \left(2 - \alpha^2 \Delta x^2\right) U_i + U_{i+1} = 0.$$

Define:

$$\kappa_{\text{FDM2}} = 2 - \alpha^2 \Delta x^2,$$

so the equation becomes:

$$U_{i-1} - \kappa_{\text{FDM2}} U_i + U_{i+1} = 0.$$

3.3 Fourth-Order Approximation

For the fourth-order approximation, the fourth-order term in the summation equation is retained:

$$\Delta x^2 U_i'' = U_{i-1} - 2U_i + U_{i+1} - \frac{\Delta x^4}{12} U_i^{(4)}.$$

From the governing equation, $U_i^{(4)} = \alpha^4 U_i$. Substituting this:

$$\Delta x^2 U_i'' = U_{i-1} - 2U_i + U_{i+1} - \frac{\Delta x^4 \alpha^4}{12} U_i.$$

Rearranging:

$$U_i'' = \frac{U_{i-1} - 2U_i + U_{i+1}}{\Delta x^2} - \frac{\Delta x^2 \alpha^4}{12} U_i.$$

Substituting U_i'' into the governing equation:

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{\Delta x^2} - \frac{\Delta x^2 \alpha^4}{12} U_i + \alpha^2 U_i = 0.$$

Rewriting:

$$U_{i-1} - \left(2 + \alpha^2 \Delta x^2 + \frac{\Delta x^4 \alpha^4}{12}\right) U_i + U_{i+1} = 0.$$

Define:

$$\kappa_{\text{FDM4}} = 2 + \alpha^2 \Delta x^2 + \frac{\Delta x^4 \alpha^4}{12},$$

so the equation becomes:

$$U_{i-1} - \kappa_{\text{FDM4}} U_i + U_{i+1} = 0.$$

3.4 Final System of Equations

The finite difference equations result in a tridiagonal system of equations. For both second- and fourth-order FDM schemes, the matrix representation is given as:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & \kappa & -1 & \cdots & 0 & 0 \\ 0 & -1 & \kappa & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & -1 & 0 \\ 0 & 0 & 0 & \cdots & \kappa & -1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{N-2} \\ U_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 100 \end{bmatrix}.$$

Here: - $\kappa = \kappa_{\text{FDM2}}$ for the second-order scheme. - $\kappa = \kappa_{\text{FDM4}}$ for the fourth-order scheme. - Boundary conditions $U_1 = 0$ (fixed at the left end) and $U_{N-1} = 100$ (fixed at the right end) are incorporated into the system by setting the first and last rows of the matrix.

3.5 Summary

The second-order FDM equation is:

$$U_{i-1} - \kappa_{\text{FDM2}} U_i + U_{i+1} = 0.$$

The fourth-order FDM equation is:

$$U_{i-1} - \kappa_{\text{FDM4}} U_i + U_{i+1} = 0.$$

The boundary conditions are applied directly in the system of equations to enforce $U_1 = 0$ and $U_{N-1} = 100$.

4 Finite Element Method (FEM) Derivation

For the derivation of the finite element method, we will first consider the p=2 system that involves 2 linear shape functions and a third quadratic shape function. After assembling a governing system of equations with a local stiffness matrix for each sub element, a constraint will be placed to recover the p=1 solution through an ignorance of the quadratic shape function.

4.1 Weak Form

The weak form of the governing equation is:

$$\int \left(U'(x)\phi'_m(x) + \alpha^2 U(x)\phi_m(x) \right) \mathrm{d}x = 0 \quad \text{for } m = 1,2,3.$$

Where ϕ_m represents the mth shape function with domain (i, i+1) over a sub element of the discretized bar.

4.2 Discretization of $U(x)$ and $U'(x)$

The displacement $U(x)$ is discretized as:

$$U(x) \approx U_{i-1}\phi_1(x) + U_i\phi_2(x) + \varepsilon_i\phi_3(x),$$

where U_{i-1} and U_i are the nodal displacements, ε_i is the intermediate (quadratic) displacement, and ϕ_1, ϕ_2, ϕ_3 are the shape functions.
The derivative $U'(x)$ is:

$$U'(x) \approx U_{i-1}\phi'_1(x) + U_i\phi'_2(x) + \varepsilon_i\phi'_3(x).$$

The shape functions for p=2 are:

$$\phi_1(x) = 1 - \frac{x}{\Delta x}, \qquad \phi'_1(x) = -\frac{1}{\Delta x}, \tag{25}$$

$$\phi_2(x) = \frac{x}{\Delta x}, \qquad \phi'_2(x) = \frac{1}{\Delta x}, \tag{26}$$

$$\phi_3(x) = \phi_1(x)\phi_2(x) = \left(1 - \frac{x}{\Delta x}\right)\frac{x}{\Delta x}, \qquad \phi'_3(x) = \frac{1}{\Delta x}\left(-1 + \frac{2x}{\Delta x}\right). \tag{27}$$

Substituting the discretized forms of $U(x)$ and $U'(x)$, the weak form becomes:

$$\int \left(\left(U_{i-1}\phi'_1(x) + U_i\phi'_2(x) + \varepsilon_i\phi'_3(x) \right)\phi'_m(x) + \alpha^2 \left(U_{i-1}\phi_1(x) + U_i\phi_2(x) + \varepsilon_i\phi_3(x) \right)\phi_m(x) \right) \mathrm{d}x = -U'^{-}\phi_m(i) + U'^{+}\phi_m(i-1).$$

The weak form is valid for any domain of the bar. Discretized elements of length Δx are chosen such that the quadratic shape function is 0 at both ends and one of the linear shape functions is unity with the other nil. This reduces the weak form equation to linear combinations of the quantity U and integral coefficients with respect to the shape functions. The right hand side reduces to the strain at either end of the sub element.

4.3 Equations for $m = 1, 2, 3$

Expanding the weak form for each test function m gives the following element equations:
For $m = 1$:

$$\int \left((-\phi_1'\phi_1' + \alpha^2\phi_1\phi_1)U_{i-1} + (-\phi_1'\phi_2' + \alpha^2\phi_1\phi_2)U_i + (-\phi_1'\phi_3' + \alpha^2\phi_1\phi_3)\varepsilon_i\right) \mathrm{d}x = U_{i-1}'^{+}.$$

For $m = 2$:

$$\int \left((-\phi_2'\phi_1' + \alpha^2\phi_2\phi_1)U_{i-1} + (-\phi_2'\phi_2' + \alpha^2\phi_2\phi_2)U_i + (-\phi_2'\phi_3' + \alpha^2\phi_2\phi_3)\varepsilon_i\right) \mathrm{d}x = U_i'^{-}.$$

For $m = 3$:

$$\int \left((-\phi_3'\phi_1' + \alpha^2\phi_3\phi_1)U_{i-1} + (-\phi_3'\phi_2' + \alpha^2\phi_3\phi_2)U_i + (-\phi_3'\phi_3' + \alpha^2\phi_3\phi_3)\varepsilon_i\right) \mathrm{d}x = 0.$$

4.4 Local Stiffness Matrix Representation

The local stiffness matrix for an element i is written as:

$$\begin{bmatrix} k_{11}^i & k_{12}^i & k_{13}^i \\ k_{21}^i & k_{22}^i & k_{23}^i \\ k_{31}^i & k_{32}^i & k_{33}^i \end{bmatrix} \begin{bmatrix} U_{i-1} \\ U_i \\ \varepsilon_i \end{bmatrix} = \begin{bmatrix} U_{i-1}'^{+} \\ -U_i'^{-} \\ 0 \end{bmatrix}.$$

The local stiffness matrix for an element $i+1$ is written as:

$$\begin{bmatrix} k_{11}^{i+1} & k_{12}^{i+1} & k_{13}^{i+1} \\ k_{21}^{i+1} & k_{22}^{i+1} & k_{23}^{i+1} \\ k_{31}^{i+1} & k_{32}^{i+1} & k_{33}^{i+1} \end{bmatrix} \begin{bmatrix} U_i \\ U_{i+1} \\ \varepsilon_{i+1} \end{bmatrix} = \begin{bmatrix} U_i'^{+} \\ -U_{i+1}'^{-} \\ 0 \end{bmatrix}.$$

4.5 Conservation of Force Across Elements

For each element, conservation of force ensures that:

$$AEN_i^- = AEN_i^+.$$

For a prismatic bar of uniform material, this reduces to:

$$U_i'^{+} - U_i'^{-} = 0.$$

4.6 p=2 FEM Derivation

For p=2, we retain the quadratic term $\phi_3(x)$ and the intermediate displacement ε_i. This leads to three equations for each element: two from force conservation and one from the weak form.
Conservation of force across the element boundaries ensures continuity:

$$U_i'^{+} - U_i'^{-} = 0.$$

The weak form for $m = 3$ provides an additional equation:

$$\int \left((-\phi_3'\phi_1' + \alpha^2\phi_3\phi_1)U_{i-1} + (-\phi_3'\phi_2' + \alpha^2\phi_3\phi_2)U_i + (-\phi_3'\phi_3' + \alpha^2\phi_3\phi_3)\varepsilon_i\right) \mathrm{d}x = 0.$$

4.6.1 Combined System of Equations for p=2

The system of equations for element i becomes:

$$\begin{bmatrix} k_{21}^i U_{i-1} + (k_{22}^i + k_{12}^{i+1}) U_i + k_{23}^{i+1} U_{i+1} + k_{13}^i \varepsilon_i + k_{23}^{i+1} \varepsilon_{i+1} \\ k_{31}^i U_{i-1} + k_{32}^i U_i + k_{33}^i \varepsilon_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

4.7 p=1 FEM Derivation

For p=1, we disregard the quadratic term $\phi_3(x)$ and the intermediate displacement ε_i. This reduces the weak form to two equations: one from force conservation and one from the reduced stiffness matrix.

4.7.1 Reduced Weak Form for $m = 1$ and $m = 2$

With the quadratic term ignored, the weak form reduces to:

$$\int \left((-\phi_m' \phi_n' + \alpha^2 \phi_m \phi_n) U_n \right) \mathrm{d}x = 0 \quad \text{for } m, n = 1, 2.$$

4.7.2 Combined System of Equations for p=1

The system of equations for element i becomes:

$$\left[k_{21}^i U_{i-1} + (k_{22} + k_{11})^{i+1} U_i + k_{23}^{i+1} U_{i+1} \right] = \left[0 \right].$$

4.8 Global Assembly for p=2 and p=1

The global stiffness matrix is assembled by summing the local stiffness contributions across all elements. The entries of the local stiffness matrix $\mathbf{K}^i$ for each element i are given by:

$$k_{11}^i = -\frac{1}{\Delta x} + \frac{\alpha^2 \Delta x}{3}, \quad k_{12}^i = k_{21}^i = \frac{1}{\Delta x} + \frac{\alpha^2 \Delta x}{6}, \quad k_{13}^i = k_{31}^i = \frac{\alpha^2 \Delta x}{12},$$
$$k_{22}^i = -\frac{1}{\Delta x} + \frac{\alpha^2 \Delta x}{3}, \quad k_{23}^i = k_{32}^i = \frac{\alpha^2 \Delta x}{12}, \quad k_{33}^i = -\frac{1}{3\Delta x} + \frac{\alpha^2 \Delta x}{30}.$$

The boundary conditions are applied directly:

$$U_0 = 0, \quad U_L = 100.$$

The resulting global system for p=2 or p=1 is:

$$\mathbf{KU} = \mathbf{F},$$

where $\mathbf{K}$ is the global stiffness matrix, $\mathbf{U}$ is the global displacement vector, and $\mathbf{F}$ incorporates the boundary conditions as well as any internal forces of the system (assumed to be zero in this analysis).
In regards to p=2 FEM, the final system of equations per element can either be reduced by hand to achieve a similar form to the p=1 single equation per element. However, the solution can be computed through use of unknown quadratic coefficient terms ε without need for reduction. That can lead to more direct access for weak form interpolation methods due to having all coefficients per sub element in the resultant solution vector.

5 Discrete Interpolation of $U(x)$ and $U'(x)$

To interpolate the displacement $U(x)$ and its derivative $U'(x)$ at any arbitrary point within a discrete domain, we employ a 4th Taylor series expansion. This ensures continuity and accuracy of interpolation using known nodal values and their derivatives. Both Taylor series interpolations will require the use of nodal strain U_i' which is calculated after the end of the displacement vector calculation.

5.1 Interpolation of $U(x)$

The displacement $U(x)$ can be expressed as a Taylor series about a known nodal value U_i:

$$U(x+h) = U_i + hU_i' + \frac{h^2}{2}U_i'' + \frac{h^3}{6}U_i''' + \frac{h^4}{24}U_i^{(4)}.$$

Substituting the governing equation $U''(x) = -\alpha^2 U(x)$ and its higher-order derivatives:

$$U'''(x) = -\alpha^2 U'(x),$$
$$U^{(4)}(x) = \alpha^4 U(x),$$

the expression becomes:

$$U(x+h) = U_i + hU_i' - \frac{h^2\alpha^2}{2}U_i - \frac{h^3\alpha^2}{6}U_i' + \frac{h^4\alpha^4}{24}U_i.$$

Rearranging terms:

$$U(x+h) = \left(1 - \frac{h^2\alpha^2}{2} + \frac{h^4\alpha^4}{24}\right)U_i + \left(h - \frac{h^3\alpha^2}{6}\right)U_i'.$$

5.2 Interpolation of $U'(x)$

The derivative $U'(x)$ can also be expressed as a Taylor series about U_i':

$$U'(x+h) = U_i' + hU_i'' + \frac{h^2}{2}U_i''' + \frac{h^3}{6}U_i^{(4)}.$$

Using the governing equations for $U''(x), U'''(x)$, and $U^{(4)}(x)$, this becomes:

$$U'(x+h) = U_i' - h\alpha^2 U_i - \frac{h^2\alpha^2}{2}U_i' + \frac{h^3\alpha^4}{6}U_i.$$

Rearranging terms:

$$U'(x+h) = \left(\frac{h^3\alpha^4}{6} - h\alpha^2\right)U_i + \left(1 - \frac{h^2\alpha^2}{2}\right)U_i'.$$

5.3 Nodal Strain Using Taylor Series Expansion

The nodal strain U_i' can be approximated using the Taylor series expansion from either the left or right neighboring points. These expansions ensure the continuity of strain calculations and provide the necessary relationship between neighboring nodal values.

5.3.1 Left-Side Taylor Expansion

The Taylor expansion from the left neighboring point $i-1$ is:

$$U_{i-1} = U_i - \Delta x U_i' + \frac{\Delta x^2}{2} U_i'' - \frac{\Delta x^3}{6} U_i''' + \frac{\Delta x^4}{24} U_i^{(4)}.$$

Rearranging to solve for U_i', and substituting $U_i'' = -\alpha^2 U_i$, $U_i''' = -\alpha^2 U_i'$, and $U_i^{(4)} = \alpha^4 U_i$, we get:

$$U_i' = \frac{\left(1 - \frac{\Delta x^2 \alpha^2}{2} + \frac{\Delta x^4 \alpha^4}{24}\right) U_i - U_{i-1}}{\Delta x - \frac{\Delta x^3 \alpha^2}{6}}.$$

5.3.2 Right-Side Taylor Expansion

Similarly, the Taylor expansion from the right neighboring point $i+1$ is:

$$U_{i+1} = U_i + \Delta x U_i' + \frac{\Delta x^2}{2} U_i'' + \frac{\Delta x^3}{6} U_i''' + \frac{\Delta x^4}{24} U_i^{(4)}.$$

Rearranging to solve for U_i', and substituting the same governing equations, we get:

$$U_i' = \frac{\left(1 - \frac{\Delta x^2 \alpha^2}{2} + \frac{\Delta x^4 \alpha^4}{24}\right) U_i - U_{i+1}}{-\Delta x + \frac{\Delta x^3 \alpha^2}{6}}.$$

5.3.3 Selection of Nodal Strain

To compute U_i', either the left or right expansion can be used depending on the node chosen. Both expansions provide consistent results when applied to a uniform and continuous domain. The chosen U_i' is then plugged back into the interpolation equations for $U(x)$ and $U'(x)$, ensuring the continuity and consistency of the numerical solution.

5.4 Summary of Interpolation Equations

The final expressions for interpolation are:

$$U(x+h) = \left(1 - \frac{h^2\alpha^2}{2} + \frac{h^4\alpha^4}{24}\right) U_i + \left(h - \frac{h^3\alpha^2}{6}\right) U_i', \tag{28}$$

$$U'(x+h) = \left(\frac{h^3\alpha^4}{6} - h\alpha^2\right) U_i + \left(1 - \frac{h^2\alpha^2}{2}\right) U_i'. \tag{29}$$

These equations enable the interpolation of displacement $U(x)$ and its derivative $U'(x)$ at any point within the element, given the nodal values U_i calculated in the solution of the global assembly system and U_i' calculated with a 4th order Taylor series scheme after the nodal displacements are found.

6 FDM and FEM Convergence of Displacement

The convergence analysis evaluates the numerical methods' accuracy and the rate at which they approach the exact solution as the mesh is refined. The analysis includes error computation, convergence rate (β), and Richardson extrapolation. These metrics are critical for assessing the performance of numerical methods.

6.1 Error Calculation

The relative error is computed to quantify the deviation of the numerical solution from the exact (analytical) solution. The formula for relative error is:

$$\text{Error} = \left| \frac{U_{\text{exact}} - U_{\text{num}}}{U_{\text{exact}}} \right|,$$

where U_{exact} is the exact displacement value and U_{num} is the numerical solution. This provides insight into the accuracy of the solution for a given mesh refinement level.

6.2 Convergence Rate (β)

The convergence rate (β) measures how quickly the numerical solution approaches the exact solution as the mesh is refined. It is calculated using the formula:

$$\beta = \frac{\log \left| \frac{U_{\text{exact}} - U_{\text{num},1}}{U_{\text{exact}} - U_{\text{num},2}} \right|}{\log \left(\frac{\Delta x_1}{\Delta x_2} \right)},$$

where:

- $U_{\text{num},1}$ and $U_{\text{num},2}$ are the numerical solutions at two different mesh sizes, Δx_1 and Δx_2, respectively.
- Δx_1 and Δx_2 are the mesh spacings.

The convergence rate indicates the order of the numerical method.

6.3 Richardson Extrapolation

Richardson extrapolation improves the numerical solution's accuracy by combining results from multiple mesh sizes. The extrapolated solution is computed as:

$$U_{\text{extrap}} = \frac{U_1 U_3 - U_2^2}{U_1 + U_3 - 2U_2},$$

where:

- U_1, U_2, and U_3 are the numerical solutions at three successive mesh sizes.
- If the denominator is close to zero, the extrapolated value is set to NaN to avoid instability.

Richardson extrapolation provides an estimate of the solution at a higher order of accuracy. If the denominator becomes too small, the result is automatically truncated to NaN.

6.4 FDM Order 2 and FEM p=1

The figure below shows the percentage error versus Δx for FDM Order 2, FDM Order 2 Extrapolated, FEM $p = 1$, and FEM $p = 1$ Extrapolated.

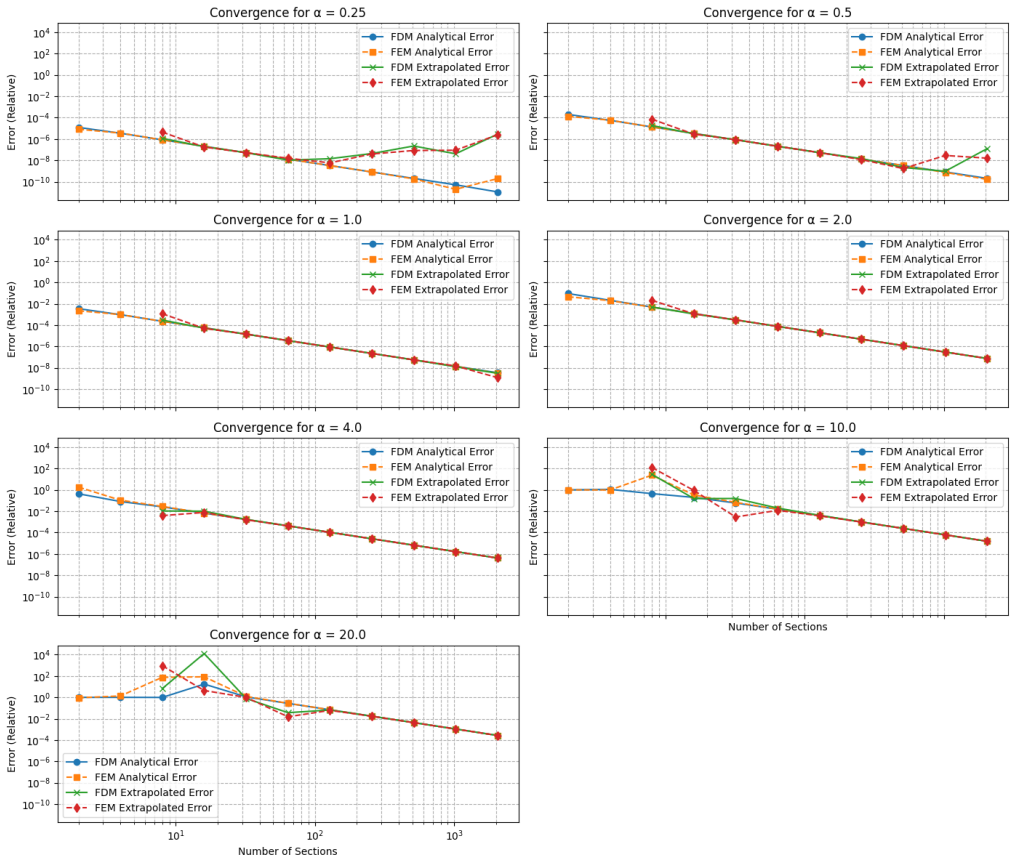

Figure 4: Convergence of $U(1)$ for FDM Order 2 and FEM $p = 1$.

Δx	Analytical Value	Method Value	Extrap Value	% Error	β	% Extrap Error	β Extrap
0.5000	34.4179	37.5522	37.5522	9.1062	nan	nan	nan
0.2500	34.4179	35.1507	35.1507	2.1289	2.0968	nan	nan
0.1250	34.4179	34.5889	34.5889	0.4970	2.1002	0.4985	2.0957
0.0625	34.4179	34.4604	34.4604	0.1231	2.0119	0.1107	2.1281
0.0313	34.4179	34.4285	34.4285	0.0306	2.0063	0.0304	2.0138
0.0156	34.4179	34.4206	34.4206	0.0077	2.0008	0.0076	2.0081
0.0078	34.4179	34.4186	34.4186	0.0019	2.0002	0.0019	2.0010
0.0039	34.4179	34.4181	34.4181	0.0005	2.0001	0.0005	2.0002
0.0020	34.4179	34.4180	34.4180	0.0001	2.0000	0.0001	2.0001
0.0010	34.4179	34.4179	34.4179	0.0000	2.0000	0.0000	1.9998
0.0005	34.4179	34.4178	34.4178	0.0000	2.0001	0.0000	2.0018

Table 3: Convergence for FDM Order 2 with $\alpha = 2.0$

Δx	Analytical Value	Method Value	Extrap Value	% Error	β	% Extrap Error	β Extrap
0.5000	34.4179	37.5522	37.5522	9.1062	nan	nan	nan
0.2500	34.4179	35.1507	35.1507	2.1289	1.1929	nan	nan
0.1250	34.4179	34.5889	34.5889	0.4832	2.0363	2.1043	0.7662
0.0625	34.4179	34.4604	34.4604	0.1224	1.9808	0.1144	2.0547
0.0313	34.4179	34.4285	34.4285	0.0306	2.0005	0.0313	1.9742
0.0156	34.4179	34.4206	34.4206	0.0076	1.9992	0.0076	2.0010
0.0078	34.4179	34.4186	34.4186	0.0019	1.9998	0.0019	1.9990
0.0039	34.4179	34.4181	34.4181	0.0005	1.9999	0.0005	1.9997
0.0020	34.4179	34.4180	34.4180	0.0001	2.0000	0.0001	1.9999
0.0010	34.4179	34.4179	34.4179	0.0000	2.0002	0.0000	1.9998
0.0005	34.4179	34.4178	34.4178	0.0000	1.9999	0.0000	2.0044

Table 4: Convergence for FEM $p = 1$ with $\alpha = 2.0$

Convergence for $\alpha = 2.0$ is provided. All convergence data is located in appendix.

6.5 FDM Order 4 and FEM p=2

The figure below shows the percentage error versus Δx for FDM Order 4, FDM Order 4 Extrapolated, FEM $p = 2$, and FEM $p = 2$ Extrapolated.

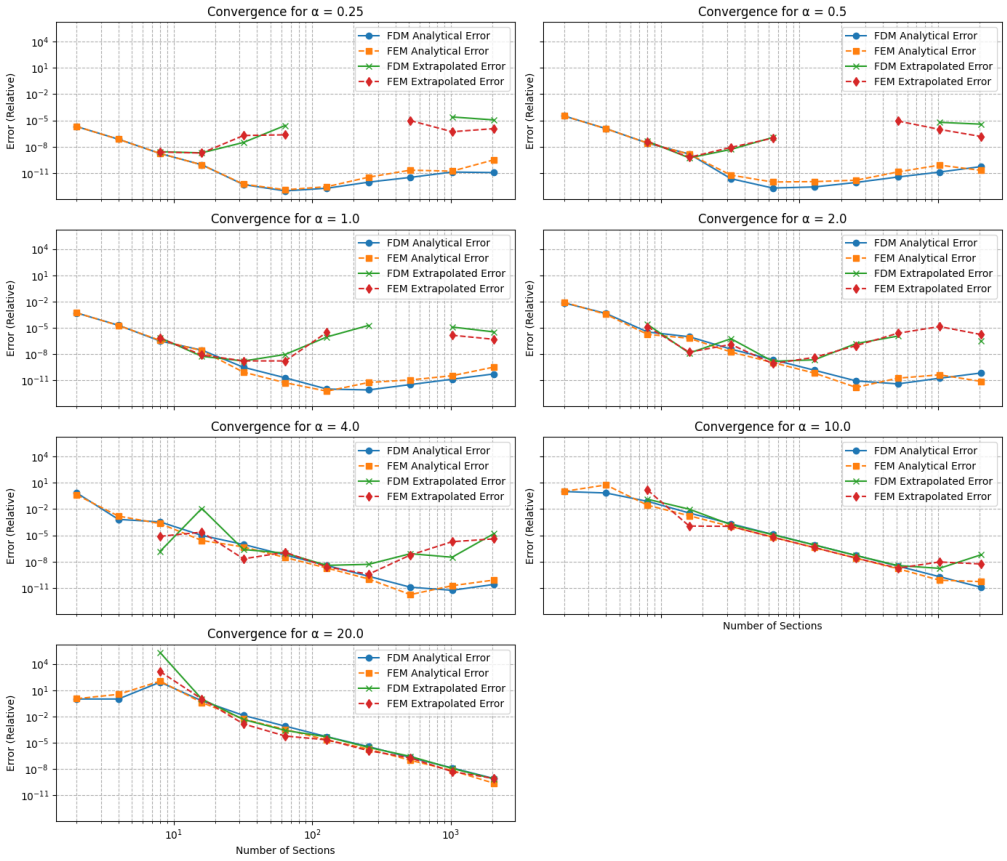

Figure 5: Convergence of $U(1)$ for FDM Order 4 and FEM $p = 2$.

Δx	Analytical Value	Method Value	Extrap Value	% Error	β	% Error Extrap	β Extrap
0.5000	34.4179	34.6635	34.6635	0.7134	nan	nan	nan
0.2500	34.4179	34.4026	34.4026	0.0446	3.9983	nan	nan
0.1250	34.4179	34.4179	34.4179	0.0003	7.0379	0.0024	4.0969
0.0625	34.4179	34.4179	34.4179	0.0001	1.8086	0.0000	7.5118
0.0313	34.4179	34.4178	34.4178	0.0000	4.6872	0.0001	1.3810
0.0156	34.4179	34.4178	34.4178	0.0000	4.0006	0.0000	4.5925
0.0078	34.4179	34.4178	34.4178	0.0000	4.0009	0.0000	-5.5122
0.0039	34.4179	34.4178	34.4178	0.0000	4.0569	0.0000	-0.0012
0.0020	34.4179	34.4178	34.4178	0.0000	1.1194	0.0001	0.0000
0.0010	34.4179	34.4178	34.4178	0.0000	-2.1286	nan	nan
0.0005	34.4179	34.4178	34.4178	0.0000	-2.0054	0.0000	0.0002

Table 5: Convergence for FDM Order 4 with $\alpha = 2.0$

Δx	Analytical Value	Method Value	Extrap Value	% Error	β	% Error Extrap	β Extrap
0.5000	34.4179	34.6635	34.6635	0.7134	nan	nan	nan
0.2500	34.4179	34.4026	34.4026	0.0446	3.9983	nan	nan
0.1250	34.4179	34.4179	34.4179	0.0003	7.0379	0.0024	4.0969
0.0625	34.4179	34.4179	34.4179	0.0001	1.8086	0.0000	7.5118
0.0313	34.4179	34.4178	34.4178	0.0000	4.6872	0.0001	1.3810
0.0156	34.4179	34.4178	34.4178	0.0000	4.0006	0.0000	4.5925
0.0078	34.4179	34.4178	34.4178	0.0000	4.0009	0.0000	-5.5122
0.0039	34.4179	34.4178	34.4178	0.0000	4.0569	0.0000	-0.0012
0.0020	34.4179	34.4178	34.4178	0.0000	1.1194	0.0001	0.0000
0.0010	34.4179	34.4178	34.4178	0.0000	-2.1286	nan	nan
0.0005	34.4179	34.4178	34.4178	0.0000	-2.0054	0.0000	0.0002

Table 6: Convergence for FEM $p = 2$ with $\alpha = 2.0$

6.6 Discussion

From Figure 4, the 2nd order schemes only achieve there expected 2nd order rate of convergence when sufficient granularity is achieved. For higher α values, a finer mesh size is required before convergence is achieved. As expected, for the rate of convergence in reference to extrapolated values, the rate of convergence appears to "bounce" when the double precision of the numerical computation can no longer precisely represent the floating point values.

While similar, the Figure 5 shows a slightly different story. The 4th order schemes experience much faster convergence to errors within tolerance, however, particularly for lower α values, the convergence is not as clear as the 2nd order schemes. The methods "hit the floor" of double precision much quicker, with extrapolated values almost immediately representing the limit of machine precision. To better characterize this system for convergence, more values of Δx must be analyzed with more granularity than $\left(\frac{1}{2}\right)^N$

7 Frequency Sweep Analysis

7.1 Purpose of Frequency Sweep

A frequency sweep is performed to analyze the behavior of the vibrating bar over a range of forcing frequencies. By studying the response, we can identify resonant frequencies and compare the accuracy of different numerical methods, such as FDM and FEM, against the analytical solution. Resonance occurs when the natural frequency of the bar aligns with the forcing frequency, leading to amplified displacement and strain amplitudes. This analysis provides insights into the bar's dynamic behavior and validates the numerical methods for various discretizations and orders of accuracy.

7.2 Frequency Sweep Results

For both cases, the displacement amplitude $U(x = \frac{1}{2\pi})$ is evaluated as a function of the forcing frequency.

7.2.1 Frequency Sweep: Analytical, FDM Order 2, FEM p=1

The figure below illustrates the displacement amplitude $U(x = \frac{1}{2\pi})$ as a function of the forcing frequency. The results from the analytical solution are compared with FDM (2nd order) and FEM (p=1) to demonstrate the accuracy of these numerical methods at lower-order approximations.

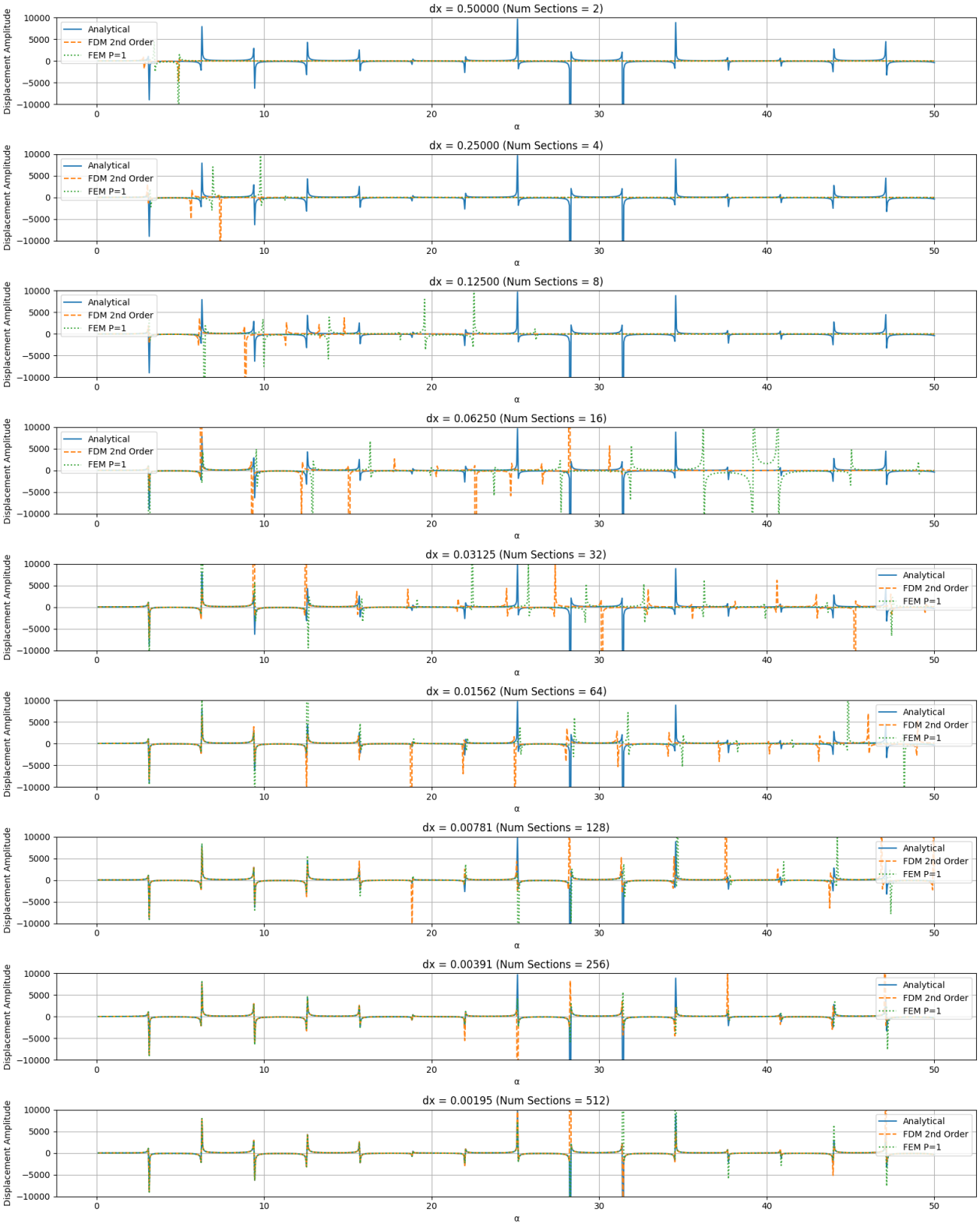

Figure 6: Frequency sweep results for analytical solution, FDM (2nd order), and FEM (p=1) at $x = \frac{1}{2\pi}$.

7.2.2 Frequency Sweep: Analytical, FDM Order 4, FEM p=2

The figure below illustrates the displacement amplitude $U(x = \frac{1}{2\pi})$ as a function of the forcing frequency for higher-order methods. The analytical solution is compared with FDM (4th order) and FEM (p=2) to evaluate the benefits of higher-order discretization in capturing the dynamic response.

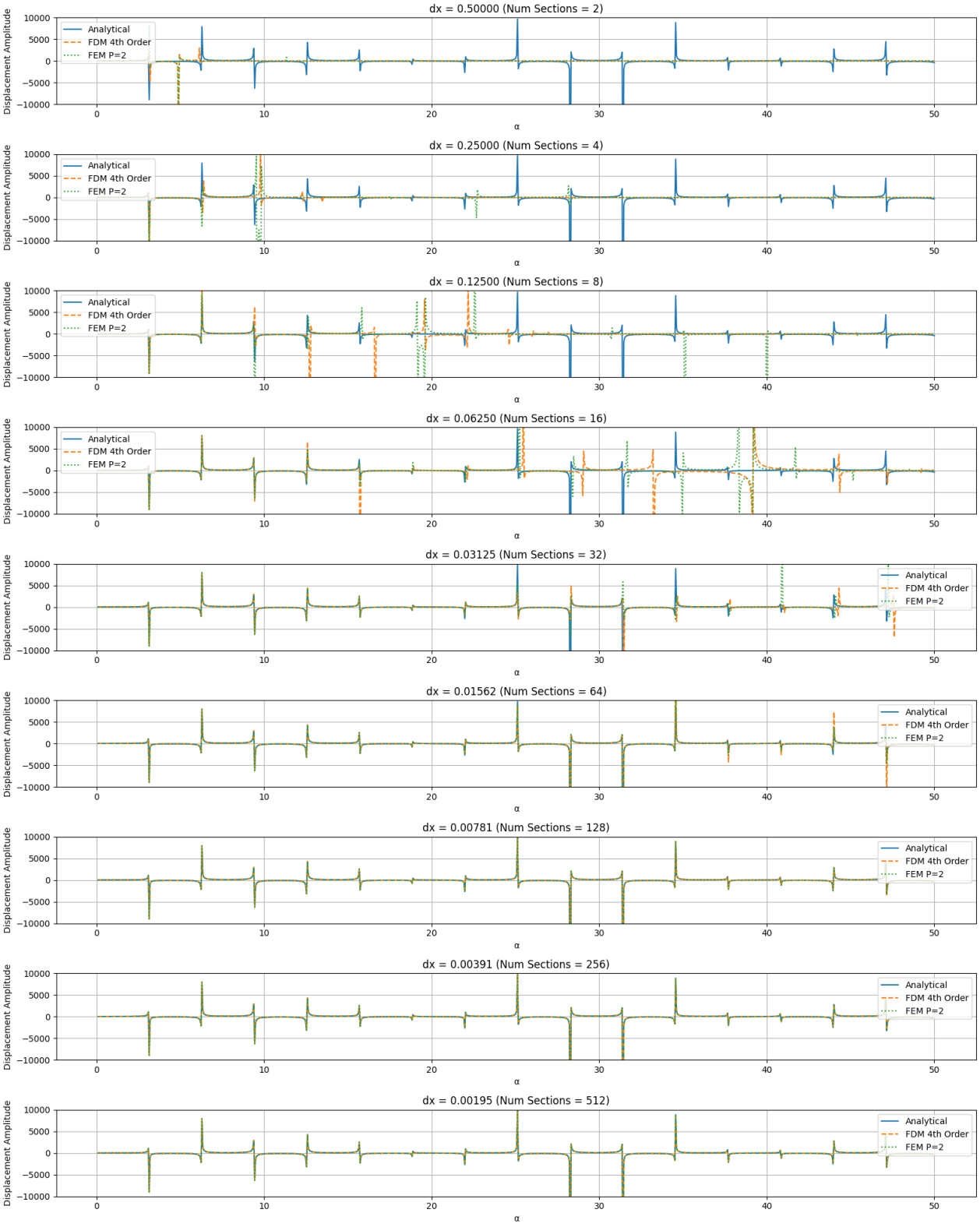

Figure 7: Frequency sweep results for analytical solution, FDM (4th order), and FEM (p=2) at $x = \frac{1}{2\pi}$.

7.3 Discussion

In reference to Figure 6 and Figure 7, it is clear that higher order schemes are more accurately able to predict the natural frequencies of the bar in reference to the analytical values. This, however, is not the primary finding of the frequency sweep.

In the sweep with only 2 sections, all methods, no matter the order, were able to identify the first natural frequency of the bar. However, there is a false natural frequency reported by the methods. This corresponds to not the natural frequency of the bar, but the scheme itself. As the number of sections increases, the number of correctly identified natural frequencies increases depending on the order of the method. Once the method has enough sections to identify all natural frequencies in the sweep range, increasing the number of sections serves to increase the accuracy of the reported natural frequency.

8 Raw Code Output

```
Analytical Results
Analytical Displacement
x          = 0.25        = 0.50        = 1.00        = 2.00        = 4.00        = 10.00       = 20.00
-------------------------------------------------------------------------------------------------------
0.000     0.0000        0.0000        0.0000        0.0000        -0.0000       -0.0000       0.0000
0.125     12.6291       13.0279       14.8163       27.2083       -63.3488      -174.4389     65.5540
0.159     16.0783       16.5810       18.8341       34.4180       -78.5516      -183.7768     -4.5451
0.250     25.2459       26.0050       29.4014       52.7248       -111.1877     -110.0090     -105.0363
0.375     37.8380       38.8806       43.5276       74.9632       -131.8039     105.0623      102.7444
0.500     50.3932       51.6043       56.9747       92.5408       -120.1499     176.2660      -59.5897
0.625     62.8992       64.1264       69.5327       104.3646      -79.0790      6.0989        -7.2646
0.750     75.3437       76.3982       81.0056       109.6995      -18.6469      -172.4198     71.2297
0.875     87.7147       88.3716       91.2145       108.2139      46.3507       -114.8345     -106.8658
1.000     100.0000      100.0000      100.0000      100.0000      100.0000      100.0000      100.0000

Analytical Strain
x          = 0.25        = 0.50        = 1.00        = 2.00        = 4.00        = 10.00       = 20.00
-------------------------------------------------------------------------------------------------------
0.000     101.0493      104.2915      118.8395      219.9500      -528.5395     -1838.1640    2190.7119
0.125     101.0000      104.0879      117.9123      213.1123      -463.8370     -579.6142     -1755.0748
0.159     100.9693      103.9614      117.3376      208.9010      -425.0038     38.1449       -2188.8251
0.250     100.8520      103.4778      115.1451      193.0243      -285.5711     1472.6333     621.4221
0.375     100.6056      102.4636      110.5811      160.9350      -37.3874      1508.3226     759.3781
0.500     100.2609      101.0493      104.2915      118.8395      219.9500      -521.4176     -1838.1640
0.625     99.8183       99.2404       96.3745       69.3552       423.4360      -1837.1519    2185.8885
0.750     99.2783       97.0440       86.9535       15.5586       523.2501      -637.1725     -1664.2573
0.875     98.6413       94.4687       76.1758       -39.2052      494.9543      1435.3224     480.7297
1.000     97.9079       91.5244       64.2093       -91.5315      345.4765      1542.3510     893.9902
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
Convergence Results
2nd Order FDM and P=1 FEM Convergence of U(x=1/(2pi))
Convergence Table for  = 0.25
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             16.078254       16.078452       16.078452       0.000012        nan                  nan              nan
0.25            16.078254       16.078310       16.078310       0.000003        1.814778             nan              nan
0.125           16.078254       16.078268       16.078268       0.000001        2.068323             0.000001         1.725316
0.0625          16.078254       16.078258       16.078258       0.000000        1.998164             0.000000         2.091062
0.03125         16.078254       16.078255       16.078255       0.000000        2.004412             0.000000         1.993886
0.015625        16.078254       16.078255       16.078255       0.000000        2.000002             0.000000         2.233528
0.0078125       16.078254       16.078254       16.078254       0.000000        1.999997             0.000000         0.752269
0.00390625      16.078254       16.078254       16.078254       0.000000        1.999987             0.000000         -0.082906
0.001953125     16.078254       16.078254       16.078254       0.000000        2.000812             0.000000         -0.003978
0.0009765625    16.078254       16.078254       16.078254       0.000000        1.992182             0.000000         0.005319
0.00048828125   16.078254       16.078254       16.078254       0.000000        2.201742             0.000003         -0.000020
Convergence Table for  = 0.5
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             16.580992       16.584311       16.584311       0.000200        nan                  nan              nan
0.25            16.580992       16.581930       16.581930       0.000057        1.823229             nan              nan
0.125           16.580992       16.581215       16.581215       0.000013        2.069082             0.000018         1.736830
0.0625          16.580992       16.581048       16.581048       0.000003        1.998625             0.000003         2.091852
0.03125         16.580992       16.581006       16.581006       0.000001        2.004462             0.000001         1.996719
0.015625        16.580992       16.580995       16.580995       0.000000        2.000028             0.000000         2.006221
0.0078125       16.580992       16.580993       16.580993       0.000000        2.000007             0.000000         1.994125
0.00390625      16.580992       16.580992       16.580992       0.000000        2.000035             0.000000         1.891296
0.001953125     16.580992       16.580992       16.580992       0.000000        2.000010             0.000000         1.843723
0.0009765625    16.580992       16.580992       16.580992       0.000000        1.999286             0.000000         0.668336
0.00048828125   16.580992       16.580992       16.580992       0.000000        2.003076             0.000000         0.007129
Convergence Table for  = 1.0
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             18.834148       18.899840       18.899840       0.003488        nan                  nan              nan
0.25            18.834148       18.852241       18.852241       0.000961        1.860259             nan              nan
0.125           18.834148       18.838449       18.838449       0.000228        2.072629             0.000299         1.787081
0.0625          18.834148       18.835222       18.835222       0.000057        2.000605             0.000052         2.095846
0.03125         18.834148       18.834415       18.834415       0.000014        2.004686             0.000014         1.999248
0.015625        18.834148       18.834214       18.834214       0.000004        2.000145             0.000004         2.006198
0.0078125       18.834148       18.834164       18.834164       0.000001        2.000036             0.000001         2.000161
0.00390625      18.834148       18.834152       18.834152       0.000000        2.000039             0.000000         1.999982
0.001953125     18.834148       18.834149       18.834149       0.000000        2.000002             0.000000         1.998332
0.0009765625    18.834148       18.834148       18.834148       0.000000        2.000046             0.000000         2.109196
0.00048828125   18.834148       18.834148       18.834148       0.000000        2.000014             0.000000         2.133319
Convergence Table for  = 2.0
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             34.417979       37.552156       37.552156       0.091062        nan                  nan              nan
0.25            34.417979       35.150695       35.150695       0.021289        2.096760             nan              nan
0.125           34.417979       34.588873       34.588873       0.004965        2.100152             0.004985         2.095727
0.0625          34.417979       34.460351       34.460351       0.001231        2.011902             0.001107         2.128103
0.03125         34.417979       34.428526       34.428526       0.000306        2.006294             0.000304         2.013755
0.015625        34.417979       34.420614       34.420614       0.000077        2.000814             0.000076         2.008115
0.0078125       34.417979       34.418638       34.418638       0.000019        2.000204             0.000019         2.001018
0.00390625      34.417979       34.418144       34.418144       0.000005        2.000073             0.000005         2.000246
0.001953125     34.417979       34.418020       34.418020       0.000001        2.000013             0.000001         2.000097
0.0009765625    34.417979       34.417989       34.417989       0.000000        2.000004             0.000000         1.999770
0.00048828125   34.417979       34.417981       34.417981       0.000000        2.000059             0.000000         2.001842
Convergence Table for  = 4.0
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             -78.551640      -44.521329      -44.521329      0.433222        nan                  nan              nan
0.25            -78.551640      -72.147403      -72.147403      0.081529        2.409722             nan              nan
0.125           -78.551640      -76.519925      -76.519925      0.025865        1.656329             0.010631         2.659493
0.0625          -78.551640      -78.029415      -78.029415      0.006648        1.959954             0.010096         1.534405
0.03125         -78.551640      -78.419329      -78.419329      0.001684        1.980741             0.001729         1.952832
0.015625        -78.551640      -78.518501      -78.518501      0.000422        1.997361             0.000431         1.975144
0.0078125       -78.551640      -78.543351      -78.543351      0.000106        1.999341             0.000106         1.996700
0.00390625      -78.551640      -78.549567      -78.549567      0.000026        1.999770             0.000026         1.999198
0.001953125     -78.551640      -78.551121      -78.551121      0.000007        1.999957             0.000007         1.999708
0.0009765625    -78.551640      -78.551510      -78.551510      0.000002        1.999982             0.000002         1.999958
0.00048828125   -78.551640      -78.551607      -78.551607      0.000000        1.999999             0.000000         1.999993
4th Order FDM and P=2 FEM Convergence of U(x=1/(2pi))
Convergence Table for  = 0.25
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             16.078254       16.078287       16.078287       0.000002        nan                  nan              nan
0.25            16.078254       16.078253       16.078253       0.000000        4.811274             nan              nan
0.125           16.078254       16.078254       16.078254       0.000000        5.424900             0.000000         4.730211
0.0625          16.078254       16.078254       16.078254       0.000000        4.272605             0.000000         0.909708
0.03125         16.078254       16.078254       16.078254       0.000000        7.515896             0.000000         -0.004096
0.015625        16.078254       16.078254       16.078254       0.000000        2.391135             0.000003         0.000000
0.0078125       16.078254       16.078254       16.078254       0.000000        -1.007072            nan              nan
0.00390625      16.078254       16.078254       16.078254       0.000000        -2.314502            nan              nan
0.001953125     16.078254       16.078254       16.078254       0.000000        -1.799075            nan              nan
0.0009765625    16.078254       16.078254       16.078254       0.000000        -2.029648            0.000025         0.000001
0.00048828125   16.078254       16.078254       16.078254       0.000000        0.207580             0.000011         -0.000000
Convergence Table for  = 0.5
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             16.580992       16.581525       16.581525       0.000032        nan                  nan              nan
0.25            16.580992       16.580973       16.580973       0.000001        4.786253             nan              nan
0.125           16.580992       16.580992       16.580992       0.000000        5.521549             0.000000         4.806285
0.0625          16.580992       16.580992       16.580992       0.000000        4.118326             0.000000         5.636740
0.03125         16.580992       16.580992       16.580992       0.000000        9.365088             0.000000         -0.509052
0.015625        16.580992       16.580992       16.580992       0.000000        3.536667             0.000000         -0.000026
0.0078125       16.580992       16.580992       16.580992       0.000000        -0.410172            nan              nan
0.00390625      16.580992       16.580992       16.580992       0.000000        -1.704184            nan              nan
0.001953125     16.580992       16.580992       16.580992       0.000000        -2.067930            nan              nan
0.0009765625    16.580992       16.580992       16.580992       0.000000        -1.875636            0.000006         0.000002
0.00048828125   16.580992       16.580992       16.580992       0.000000        -2.081360            0.000004         -0.000016
Convergence Table for  = 1.0
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             18.834148       18.843758       18.843758       0.000510        nan                  nan              nan
0.25            18.834148       18.833772       18.833772       0.000020        4.677470             nan              nan
0.125           18.834148       18.834153       18.834153       0.000000        5.985224             0.000001         4.710179
0.0625          18.834148       18.834147       18.834147       0.000000        3.448116             0.000000         5.882540
0.03125         18.834148       18.834148       18.834148       0.000000        6.498357             0.000000         4.031577
0.015625        18.834148       18.834148       18.834148       0.000000        4.002814             0.000000         -0.047321
0.0078125       18.834148       18.834148       18.834148       0.000000        4.262898             0.000001         0.000029
0.00390625      18.834148       18.834148       18.834148       0.000000        0.326859             0.000019         -0.000000
0.001953125     18.834148       18.834148       18.834148       0.000000        -2.024489            nan              nan
0.0009765625    18.834148       18.834148       18.834148       0.000000        -2.030078            0.000013         0.000001
0.00048828125   18.834148       18.834148       18.834148       0.000000        -1.997818            0.000003         0.000017
Convergence Table for  = 2.0
dx              Analytical Value     Method Value         Extrap Value         % Error                    % Extrap Error   Extrap
-------------------------------------------------------------------------------------------------------------------------------
0.5             34.417979       34.663528       34.663528       0.007134        nan                  nan              nan
0.25            34.417979       34.402614       34.402614       0.000446        3.998268             nan              nan
0.125           34.417979       34.417862       34.417862       0.000003        7.037851             0.000024         4.096854
0.0625          34.417979       34.417945       34.417945       0.000001        1.808584             0.000000         7.511752
0.03125         34.417979       34.417978       34.417978       0.000000        4.687210             0.000001         1.381023
0.015625        34.417979       34.417979       34.417979       0.000000        4.000567             0.000000         4.592474
0.0078125       34.417979       34.417979       34.417979       0.000000        4.000935             0.000000         -5.512228
0.00390625      34.417979       34.417979       34.417979       0.000000        4.056899             0.000000         -0.001227
0.001953125     34.417979       34.417979       34.417979       0.000000        1.119402             0.000001         0.000016
0.0009765625    34.417979       34.417979       34.417979       0.000000        -2.128619            nan              nan
0.00048828125   34.417979       34.417979       34.417979       0.000000        -2.005417            0.000000         0.000241
Convergence Table for  = 4.0
```

dx	Analytical Value	Method Value	Extrap Value	% Error	% Extrap Error	Extrap
0.5	-78.551640	-133.563988	-133.563988	0.700334	nan	nan	nan
0.25	-78.551640	-78.603880	-78.603880	0.000665	10.040387	nan	nan
0.125	-78.551640	-78.578508	-78.578508	0.000342	0.959249	0.000000	11.080952
0.0625	-78.551640	-78.552404	-78.552404	0.000010	5.135275	0.011709	-0.041042
0.03125	-78.551640	-78.551711	-78.551711	0.000001	3.412791	0.000000	5.235885
0.015625	-78.551640	-78.551644	-78.551644	0.000000	4.002608	0.000000	3.363585
0.0078125	-78.551640	-78.551640	-78.551640	0.000000	4.001591	0.000000	3.942435
0.00390625	-78.551640	-78.551640	-78.551640	0.000000	4.071443	0.000000	-1.652138
0.001953125	-78.551640	-78.551640	-78.551640	0.000000	4.137881	0.000000	-0.003593
0.0009765625	-78.551640	-78.551640	-78.551640	0.000000	1.177470	0.000000	0.000789
0.00048828125	-78.551640	-78.551640	-78.551640	0.000000	-2.229793	0.000015	-0.000002

9 Python Code

9.1 main.py

```
# main.py

import numpy as np
import matplotlib.pyplot as plt

from Bar import Bar
from Analytical import Analytical
from DiscreteMethod import DiscreteMethod
from FDM import FDM
from FEM import FEM
from Convergence import Convergence
from common import freq_from_alpha

bar = Bar()
alpha_values = np.array([0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 20.0])
alpha_sweep_values = np.linspace(0.1, 50, 1000)

def analytical():
    analytical = Analytical(bar)

    x_vals = np.linspace(0, 1, 1000)
    x_vals_table = np.array([0.0, 0.125, 1/(2*np.pi), 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0])
    displacement_data = []
    strain_data = []

    # Plotting displacement amplitude
    plt.figure(figsize=(12, 6))
    for alpha in alpha_values:
        freq = freq_from_alpha(bar, alpha)
        disp_amplitude = analytical.get_disp_amp(freq, x_vals)
        displacement_data.append(analytical.get_disp_amp(freq, x_vals_table))
        plt.plot(x_vals, disp_amplitude, label=f' = {alpha:.2f}')

    plt.title("Analytical Displacement Amplitude")
    plt.xlabel("x")
    plt.ylabel("Displacement Amplitude")
    plt.legend()
    plt.grid(True)
    plt.savefig("Images/1_Displacement_vs_Position.png")
    plt.close()

    # Plotting strain amplitude
    plt.figure(figsize=(12, 6))
    for alpha in alpha_values:
        freq = freq_from_alpha(bar, alpha)
        strain_amplitude = analytical.get_strain_amp(freq, x_vals)
```

```
    strain_data.append(analytical.get_strain_amp(freq, x_vals_table))
    plt.plot(x_vals, strain_amplitude, label=f' = {alpha:.2f}')

plt.title("Analytical Strain Amplitude")
plt.xlabel("x")
plt.ylabel("Strain Amplitude")
plt.legend()
plt.grid(True)
plt.savefig("Images/1_Strain_vs_Position.png")
plt.close()

# Display Displacement Table
print("\nAnalytical Results")
print("Analytical Displacement")
header = ["x"] + [f" = {alpha:.2f}" for alpha in alpha_values]
print("{:<10}".format(header[0]) + "".join(f"{val:<15}" for val in header[1:]))
print("-" * (10 + 15 * len(alpha_values)))

for i, x in enumerate(x_vals_table):
    row = [f"{x:.3f}"] + [f"{disp[i]:.4f}" for disp in displacement_data]
    print("{:<10}".format(row[0]) + "".join(f"{val:<15}" for val in row[1:]))

# Display Strain Table
print("\nAnalytical Strain")
print("{:<10}".format(header[0]) + "".join(f"{val:<15}" for val in header[1:]))
print("-" * (10 + 15 * len(alpha_values)))

for i, x in enumerate(x_vals_table):
    row = [f"{x:.3f}"] + [f"{strain[i]:.4f}" for strain in strain_data]
    print("{:<10}".format(row[0]) + "".join(f"{val:<15}" for val in row[1:]))

# Frequency sweep of strain at x = 1
plt.figure(figsize=(12, 6))
disp_amplitudes = np.array([])
for alpha in alpha_sweep_values:
    freq = freq_from_alpha(bar, alpha)
    disp_amplitude = analytical.get_disp_amp(freq, x_vals=np.array([1/(2*np.pi)]))
    disp_amplitudes = np.append(disp_amplitudes, disp_amplitude)

plt.plot(alpha_sweep_values, disp_amplitudes)
plt.ylim((-10000, 10000))
plt.title("Analytical Frequency Sweep for Displacement at x = 1/(2pi)")
plt.xlabel(" ")
plt.ylabel("Displacement Amplitude")
plt.legend()
plt.grid(True)
#plt.show()
plt.savefig("Images/1_Strain_Frequency_Sweep.png")
plt.close()
```

```
def convergence():
    def convergence_metric(method: 'DiscreteMethod', forcing_freq: float = None):
        """Extract U(1/(2pi)) from the method."""
        x = 1 / (2 * np.pi)
        if forcing_freq is None: # Finite method
            return method.get_disp_amp(x)
        else:
            return method.get_disp_amp(forcing_freq, x) # Analytical

    print("\nConvergence Results")
    print("\n2nd Order FDM and P=1 FEM Convergence of U(x=1/(2pi))")

    # Initialize Bar, FDM, FEM
    bar = Bar()
    analy = Analytical(bar)
    fdm = FDM(bar, desired_order="2")
    fem = FEM(bar, desired_order="2")

    # Set boundary conditions
    fdm.set_boundary_conditions(U_0=0, U_L=100)
    fem.set_boundary_conditions(U_0=0, U_L=100)

    # Initialize Convergence
    convergence = Convergence(analy, fdm, fem)

    num_sections_range = [2**n for n in range(1, 12)]
    alpha_values = [0.25, 0.5, 1.0, 2.0, 4.0]

    # Create a figure with 2 columns of subplots
    fig, axs = plt.subplots((len(alpha_values) + 1) // 2, 2, figsize=(14, 12), sharex=True,
        sharey=True)
    axs = axs.flatten()

    for i, alpha in enumerate(alpha_values):
        forcing_freq = freq_from_alpha(bar, alpha)

        # Run convergence
        results = convergence.run_convergence(forcing_freq, num_sections_range, convergence_metric)

        # Plot results in the subplot
        ax = axs[i]
        ax.loglog(results['fdm']['num_sections'], results['fdm']['errors'], '-o', label='FDM
            Analytical Error')
        ax.loglog(results['fem']['num_sections'], results['fem']['errors'], '--s', label='FEM
            Analytical Error')
        ax.loglog(results['fdm']['num_sections'], results['fdm']['extrapolated_errors'], '-x',
            label='FDM Extrapolated Error')
        ax.loglog(results['fem']['num_sections'], results['fem']['extrapolated_errors'], '--d',
            label='FEM Extrapolated Error')
```

```
    ax.set_title(f"Convergence for  = {alpha}")
    ax.set_ylabel("Error (Relative)")
    ax.grid(True, which="both", linestyle="--")
    if i >= len(alpha_values) - 2:
        ax.set_xlabel("Number of Sections")
    ax.legend()

    # Output table for this alpha
    print(f"\nConvergence Table for  = {alpha}")
    print("{:<15} {:<20} {:<20} {:<20} {:<15} {:<15} {:<15}".format(
        "dx", "Analytical Value", "Method Value", "Extrap Value",
        "% Error", " ", "% Extrap Error", " Extrap"
    ))
    print("-" * 135)
    for j in range(len(results['fdm']['num_sections'])):
        num_sections = results['fdm']['num_sections'][j]
        dx = 1 / num_sections
        analytical_value = results['fdm']['analytical_values'][j]
        fdm_value = results['fdm']['extrapolated_values'][j]
        fdm_error = results['fdm']['errors'][j]
        fdm_beta = results['fdm']['betas'][j] if j >= 1 else float('nan') # Beta starts at index 1
        fdm_extrap_err = results['fdm']['extrapolated_errors'][j]
        fdm_extrap_beta = results['fdm']['extrapolated_betas'][j] if j >= 2 else float('nan') #
            Extrap Beta starts at index 2

        print("{:<15} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f}
            {:<15.6f}".format(
            dx, analytical_value, fdm_value, fdm_value, fdm_error, fdm_beta, fdm_extrap_err,
                fdm_extrap_beta
        ))

# Hide any unused subplots
for j in range(len(alpha_values), len(axs)):
    fig.delaxes(axs[j])

# Save and show the plot
plt.tight_layout()
plt.savefig("Images/2_Second_Order_Convergence.png")
plt.show()



print("\n4th Order FDM and P=2 FEM Convergence of U(x=1/(2pi))")
# Initialize Bar, FDM, FEM for 4th order convergence
bar = Bar()
analy = Analytical(bar)
fdm = FDM(bar, desired_order="4")
fem = FEM(bar, desired_order="4")

# Set boundary conditions
```

```
fdm.set_boundary_conditions(U_0=0, U_L=100)
fem.set_boundary_conditions(U_0=0, U_L=100)

# Initialize Convergence
convergence = Convergence(analy, fdm, fem)

num_sections_range = [2**n for n in range(1, 12)]
alpha_values = [0.25, 0.5, 1.0, 2.0, 4.0]

# Create a figure with 2 columns of subplots
fig, axs = plt.subplots((len(alpha_values) + 1) // 2, 2, figsize=(14, 12), sharex=True,
    sharey=True)
axs = axs.flatten()

for i, alpha in enumerate(alpha_values):
    forcing_freq = freq_from_alpha(bar, alpha)

    # Run convergence
    results = convergence.run_convergence(forcing_freq, num_sections_range, convergence_metric)

    # Plot results in the subplot
    ax = axs[i]
    ax.loglog(results['fdm']['num_sections'], results['fdm']['errors'], '-o', label='FDM
        Analytical Error')
    ax.loglog(results['fem']['num_sections'], results['fem']['errors'], '--s', label='FEM
        Analytical Error')
    ax.loglog(results['fdm']['num_sections'], results['fdm']['extrapolated_errors'], '-x',
        label='FDM Extrapolated Error')
    ax.loglog(results['fem']['num_sections'], results['fem']['extrapolated_errors'], '--d',
        label='FEM Extrapolated Error')
    ax.set_title(f"Convergence for  = {alpha}")
    ax.set_ylabel("Error (Relative)")
    ax.grid(True, which="both", linestyle="--")
    if i >= len(alpha_values) - 2:
        ax.set_xlabel("Number of Sections")
    ax.legend()

    # Output table for this alpha
    print(f"\nConvergence Table for  = {alpha}")
    print("{:<15} {:<20} {:<20} {:<20} {:<15} {:<15} {:<15}".format(
        "dx", "Analytical Value", "Method Value", "Extrap Value",
        "% Error", " ", "% Extrap Error", " Extrap"
    ))
    print("-" * 135)
    for j in range(len(results['fdm']['num_sections'])):
        num_sections = results['fdm']['num_sections'][j]
        dx = 1 / num_sections
        analytical_value = results['fdm']['analytical_values'][j]
        fdm_value = results['fdm']['extrapolated_values'][j]
        fdm_error = results['fdm']['errors'][j]
```

```
            fdm_beta = results['fdm']['betas'][j] if j >= 1 else float('nan') # Beta starts at index 1
            fdm_extrap_err = results['fdm']['extrapolated_errors'][j]
            fdm_extrap_beta = results['fdm']['extrapolated_betas'][j] if j >= 2 else float('nan') #
                Extrap Beta starts at index 2

            print("{:<15} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f} {:<15.6f}
                {:<15.6f}".format(
                dx, analytical_value, fdm_value, fdm_value, fdm_error, fdm_beta, fdm_extrap_err,
                    fdm_extrap_beta
            ))

    # Hide any unused subplots
    for j in range(len(alpha_values), len(axs)):
        fig.delaxes(axs[j])

    # Save and show the plot
    plt.tight_layout()
    plt.savefig("Images/2_Fourth_Order_Convergence.png")
    plt.show()


import numpy as np
import matplotlib.pyplot as plt

def frequency_sweep():
    # Alpha sweep values
    alpha_sweep_values = np.linspace(0.1, 50, 1000)

    # Section sweep values
    num_sections_range = [2**n for n in range(1, 10)]

    # Initialize Bar, Analytical, FDM, FEM
    bar = Bar()
    analytical = Analytical(bar)

    # Plot 1: Analytical, FDM 2nd Order, FEM P=1
    fdm_2nd = FDM(bar, desired_order="2")
    fem_p1 = FEM(bar, desired_order="2")

    fdm_2nd.set_boundary_conditions(U_0=0, U_L=100)
    fem_p1.set_boundary_conditions(U_0=0, U_L=100)

    fig, axs = plt.subplots(len(num_sections_range), 1, figsize=(16, 20)) # 10 rows, 1 column
    axs = axs.flatten()

    for idx, num_sections in enumerate(num_sections_range):
        dx = 1 / num_sections
        disp_amplitudes_analytical = []
        disp_amplitudes_fdm_2nd = []
        disp_amplitudes_fem_p1 = []
```

```
    for alpha in alpha_sweep_values:
        freq = freq_from_alpha(bar, alpha)

        # Analytical
        disp_amplitudes_analytical.append(analytical.get_disp_amp(freq, x_vals=np.array([1 / (2 *
            np.pi)])))

        # FDM 2nd Order
        fdm_2nd.run(freq, num_sections)
        disp_amplitudes_fdm_2nd.append(fdm_2nd.get_disp_amp(x=np.array([1 / (2 * np.pi)])))

        # FEM P=1
        fem_p1.run(freq, num_sections)
        disp_amplitudes_fem_p1.append(fem_p1.get_disp_amp(x=np.array([1 / (2 * np.pi)])))

    # Subplot for this number of sections
    ax = axs[idx]
    ax.plot(alpha_sweep_values, disp_amplitudes_analytical, label="Analytical", linestyle="-")
    ax.plot(alpha_sweep_values, disp_amplitudes_fdm_2nd, label="FDM 2nd Order", linestyle="--")
    ax.plot(alpha_sweep_values, disp_amplitudes_fem_p1, label="FEM P=1", linestyle=":")
    ax.set_ylim((-10000, 10000)) # Set uniform y-limits for all subplots
    ax.set_title(f"dx = {dx:.5f} (Num Sections = {num_sections})")
    ax.set_xlabel(" ")
    ax.set_ylabel("Displacement Amplitude")
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.suptitle("Frequency Sweep (Displacement Amplitude, FDM 2nd Order, FEM P=1)", y=1.02)
plt.savefig("Images/3_Frequency_Sweep_FDM2_FEM1.png")
plt.close()

# Plot 2: Analytical, FDM 4th Order, FEM P=2
fdm_4th = FDM(bar, desired_order="4")
fem_p2 = FEM(bar, desired_order="4")

fdm_4th.set_boundary_conditions(U_0=0, U_L=100)
fem_p2.set_boundary_conditions(U_0=0, U_L=100)

fig, axs = plt.subplots(len(num_sections_range), 1, figsize=(16, 20)) # 10 rows, 1 column
axs = axs.flatten()

for idx, num_sections in enumerate(num_sections_range):
    dx = 1 / num_sections
    disp_amplitudes_analytical = []
    disp_amplitudes_fdm_4th = []
    disp_amplitudes_fem_p2 = []

    for alpha in alpha_sweep_values:
```

```
            freq = freq_from_alpha(bar, alpha)

            # Analytical
            disp_amplitudes_analytical.append(analytical.get_disp_amp(freq, x_vals=np.array([1 / (2 *
                np.pi)])))

            # FDM 4th Order
            fdm_4th.run(freq, num_sections)
            disp_amplitudes_fdm_4th.append(fdm_4th.get_disp_amp(x=np.array([1 / (2 * np.pi)])))

            # FEM P=2
            fem_p2.run(freq, num_sections)
            disp_amplitudes_fem_p2.append(fem_p2.get_disp_amp(x=np.array([1 / (2 * np.pi)])))

        # Subplot for this number of sections
        ax = axs[idx]
        ax.plot(alpha_sweep_values, disp_amplitudes_analytical, label="Analytical", linestyle="-")
        ax.plot(alpha_sweep_values, disp_amplitudes_fdm_4th, label="FDM 4th Order", linestyle="--")
        ax.plot(alpha_sweep_values, disp_amplitudes_fem_p2, label="FEM P=2", linestyle=":")
        ax.set_ylim((-10000, 10000)) # Set uniform y-limits for all subplots
        ax.set_title(f"dx = {dx:.5f} (Num Sections = {num_sections})")
        ax.set_xlabel(" ")
        ax.set_ylabel("Displacement Amplitude")
        ax.legend()
        ax.grid(True)

    plt.tight_layout()
    plt.suptitle("Frequency Sweep (Displacement Amplitude, FDM 4th Order, FEM P=2)", y=1.02)
    plt.savefig("Images/3_Frequency_Sweep_FDM4_FEM2.png")
    plt.close()




if __name__ == "__main__":
    import os
    os.system('cls' if os.name == 'nt' else 'clear')
    analytical()

    convergence()

    frequency_sweep()

# if __name__ == "__main__":
#     import os
#     import sys
#     os.system('cls' if os.name == 'nt' else 'clear')
#     # Redirect stdout and stderr to a file
#     with open("output.txt", "w", encoding="utf-8") as output_file:
```

```
#         # Redirect stdout
#         sys.stdout = output_file
#         # Redirect stderr
#         sys.stderr = output_file

#         # Run the main functions
#         try:
#             analytical()
#             convergence()
#             # frequency_sweep()
#         except Exception as e:
#             print(f"An error occurred: {e}")

#     # Restore stdout and stderr
#     sys.stdout = sys.__stdout__
#     sys.stderr = sys.__stderr__

#     print("Execution completed. Output saved to output.txt.")
```

9.2 common.py

```
# common.py
from Bar import Bar

def freq_from_alpha(bar:'Bar', desired_alpha):
    '''Returns the forcing frequency that corresponds to the desired alpha'''
    return desired_alpha * (bar.E / bar.density)**0.5
```

9.3 Bar.py

```
# Bar.py

from numpy import pi

class Bar():
    def __init__(self, radius:float=0.1, k:float=0.5, h:float=0.0025, E:float=7e9,
        density:float=2710):
        self.radius:float = radius
        self.k:float = k
        self.h:float = h
        self.E:float = E
        self.density:float = density

        self.update_properties()

    def update_properties(self):
        self.area:float = pi*self.radius**2
```

```
        self.p: float = 2*pi*self.radius
        self.alpha = ((self.h*self.p)/(self.k*self.area))**0.5
```

9.4 Analytical.py

```
# analytical.py

import numpy as np
from Bar import Bar


class Analytical():
    def __init__(self, bar:'Bar'):
        self.bar:'Bar' = bar

    def get_disp_amp(self, freq, x_vals:np.ndarray=None):
        '''Gets the analytical displacement amplitude values for a given forcing frequency.
        Provides output for a given x_vals. If x_vals is not provided, returns 1000 points'''
        if x_vals is None:
            x_vals = np.linspace(0, 1, 1000)

        alpha = freq*(self.bar.density / self.bar.E)**0.5
        return 100 * np.sin(alpha*x_vals) / np.sin(alpha)

    def get_strain_amp(self, freq, x_vals:np.ndarray=None):
        '''Gets the analytical strain amplitude values for a given forcing frequency.
        Provides output for a given x_vals. If x_vals is not provided, returns 1000 points'''
        if x_vals is None:
            x_vals = np.linspace(0, 1, 1000)

        alpha = freq*(self.bar.density / self.bar.E)**0.5
        return 100 * alpha * np.cos(alpha*x_vals) / np.sin(alpha)

    def get_force_amp(self, freq, x_vals:np.ndarray=None):
        '''Gets the analytical force amplitude values for a given forcing frequency.
        Provides output for a given x_vals. If x_vals is not provided, returns 1000 points.
        Units are in MN'''
        return self.bar.area * self.bar.E * self.get_strain_amp(freq, x_vals) / (1e9)


if __name__ == "__main__":
    import matplotlib.pyplot as plt
    from common import freq_from_alpha
    bar = Bar()
    analytical = Analytical(bar)

    x_vals = np.linspace(0, 1, 1000)
```

```
    # Specific alpha values
    alpha_values = np.linspace(0, 20, 6)

    freq = freq_from_alpha(bar, 4)
    strain_amplitude = analytical.get_strain_amp(freq, x_vals=np.array([0.31]))
    print(f"Analytical strain at x = {0.31}: {strain_amplitude}")

    # Plotting displacement amplitude
    plt.figure(figsize=(12, 6))
    for alpha in [20]:
        freq = freq_from_alpha(bar, alpha)
        disp_amplitude = analytical.get_disp_amp(freq, x_vals)
        plt.plot(x_vals, disp_amplitude, label=f' = {alpha:.2f}')

    plt.title("Displacement Amplitude vs x for Various  Values")
    plt.xlabel("x (Normalized Position)")
    plt.ylabel("Displacement Amplitude")
    plt.legend()
    plt.grid(True)
    plt.show()

    # # Plotting strain amplitude
    # plt.figure(figsize=(12, 6))
    # for alpha in alpha_values:
    #     freq = freq_from_alpha(bar, alpha)
    #     strain_amplitude = analytical.strain_amp(freq, x_vals)
    #     plt.plot(x_vals, strain_amplitude, label=f' = {alpha:.2f}')

    # plt.title("Strain Amplitude vs x for Various  Values")
    # plt.xlabel("x (Normalized Position)")
    # plt.ylabel("Strain Amplitude")
    # plt.legend()
    # plt.grid(True)
    # plt.show()
```

9.5 DiscreteMethod.py

```
# DiscreteMethod.py

import numpy as np

from Bar import Bar

class DiscreteMethod():
    '''Provides the class for solving via FDM or FEM.
    The bar is provided at initiation, as well as the desired order.
    The user can change the forcing frequency and num_sections as desired'''
    def __init__(self, bar: 'Bar', desired_order:str="2"):
```

```
        self.bar: 'Bar' = bar
        self.desired_order:str = desired_order

        # Finite method solving
        self.num_sections: int = None
        self.dx: float = None
        self.alpha: float = None
        self.stiffness_matrix: np.ndarray = None
        self.constant_vector: np.ndarray = None

        # Result vectors
        self.x_vals: np.ndarray = None
        self.forcing_freq: float = None
        self.disp_amp: np.ndarray = None
        self.strain_amp: np.ndarray = None

    def set_boundary_conditions(self, U_0:float=0, U_L:float=100):
        '''Set the left and right amplitude boundary conditions'''
        self.U_0 = U_0
        self.U_L = U_L

    def nodal_strain(self, node_idx: int):
        '''Calculates the strain (U') at a given node using the 4th-order scheme.'''
        dx = self.dx
        alpha = self.alpha
        if node_idx == 0: # Left boundary
            # Use forward difference at the boundary
            return (
                (1 - dx**2 * alpha**2 / 2 + dx**4 * alpha**4 / 24) / (-dx + (dx**3*alpha**2/6)) *
                    self.disp_amp[node_idx]
                - (1 / (-dx + dx**3 * alpha**2 / 6)) * self.disp_amp[node_idx + 1]
            )
        elif node_idx == len(self.disp_amp) - 1: # Right boundary, use backward difference
            return (
                (1 - dx**2 * alpha**2 / 2 + dx**4 * alpha**4 / 24) / (dx - (dx**3*alpha**2/6)) *
                    self.disp_amp[node_idx]
                - (1 / (dx - (dx**3*alpha**2/6))) * self.disp_amp[node_idx - 1]
            )
        else:  # Return the average of the backward and forward derivative
            return (
                ((1 - dx**2 * alpha**2 / 2 + dx**4 * alpha**4 / 24) / (dx - (dx**3*alpha**2/6)) *
                    self.disp_amp[node_idx]
                - (1 / (dx - (dx**3*alpha**2/6))) * self.disp_amp[node_idx - 1]
                +
                (1 - dx**2 * alpha**2 / 2 + dx**4 * alpha**4 / 24) / (-dx + (dx**3*alpha**2/6)) *
                    self.disp_amp[node_idx]
                - (1 / (-dx + dx**3 * alpha**2 / 6)) * self.disp_amp[node_idx + 1])
                / 2.0
            )
```

```
def calc_all_nodal_strain(self):
    '''Calculates all of the nodal strains after running the method'''
    nodal_strains = np.array([])
    for i in range(self.num_sections+1):
        nodal_strains = np.append(nodal_strains, self.nodal_strain(i))
    self.strain_amp = nodal_strains

def get_interpolated_disp_amp(self, x_val: float):
    """Returns the interpolated displacement amplitude for a given x_val."""
    # Find the closest node to x_val
    idx = np.argmin(np.abs(self.x_vals - x_val)) # Closest node index
    alpha = self.alpha

    # Calculate h (distance from the nearest node)
    h = x_val - self.x_vals[idx]

    U_i = self.disp_amp[idx]

    if h == 0:
        return U_i

    # Calculate U' at the closest node
    U_prime = self.strain_amp[idx]

    # Apply the fourth-order Taylor series

    interpolated_value = (
        (1 - h**2 * alpha**2 / 2 + h**4 * alpha**4 / 24) * U_i
        + (h - h**3 * alpha**2 / 6) * U_prime
    )

    return interpolated_value

def get_interpolated_strain_amp(self, x_val: float):
    '''Returns the interpolated strain (U') amplitude for a given x_val.'''
    # Find the closest node to x_val
    idx = np.argmin(np.abs(self.x_vals - x_val)) # Closest node index
    dx = 1 / (len(self.x_vals) - 1) # Spacing between nodes
    alpha = self.forcing_freq * (self.bar.density / self.bar.E)**0.5

    # Calculate h (distance from the nearest node)
    h = x_val - self.x_vals[idx]

    # Get U at the closest node
    U_at_node = self.disp_amp[idx]

    # Calculate U' at the closest node
    U_prime_at_node = self.strain_amp[idx]

    if h == 0:
```

```
            return U_prime_at_node

        # Apply the fourth-order Taylor series
        # Taylor series expansion for strain
        U_prime_interpolated = (
            (-h*alpha**2 + h**3 * alpha**4 / 6) * U_at_node
            + (1 - h**2 * alpha**2 / 2 + h**4 * alpha**4 / 24) * U_prime_at_node
        )

        return U_prime_interpolated

    def build_stiffness_matrix(self):
        pass

    def build_constant_vector(self):
        pass

    def run(self, forcing_freq:float, num_sections:int=4):
        '''Runs the method for the given forcing frequency and num_sections.
        Stores displacment ampltitude data in self.disp_amp
        Stores strain amplitude data in self.strain_amp'''
        # Save the appropriate values
        self.num_sections = num_sections
        self.dx = 1 / num_sections
        self.alpha = forcing_freq*(self.bar.density / self.bar.E)**0.5
        self.forcing_freq = forcing_freq

        self.build_stiffness_matrix()
        self.build_constant_vector()

        # Solve the system
        self.disp_amp = np.linalg.solve(self.stiffness_matrix, self.constant_vector)

        # Calculate all of the nodal strains
        self.calc_all_nodal_strain()

        # Save the x values for the nodal values
        self.x_vals = np.linspace(0, 1, num_sections+1) # sections + endpoints

    def get_disp_amp(self, x):
        '''Returns the displacement amplitude at x'''
        return self.get_interpolated_disp_amp(x)

    def get_strain_amp(self, x):
        '''Returns the strain amplitude at x'''
        return self.get_interpolated_strain_amp(x)

    def get_force_amp(self, x):
        '''Returns the force amplitude at x.
        Units are in MN'''
```

```
        return self.bar.area * self.bar.E* self.get_interpolated_strain_amp(x) / (1e9)
```

9.6 FDM.py

```
# FDM.py

import numpy as np

from Bar import Bar
from DiscreteMethod import DiscreteMethod

class FDM(DiscreteMethod):
    def __init__(self, bar:'Bar', desired_order="2"):
        super().__init__(bar, desired_order)

    def get_kappa(self):
        '''Returns the 2nd or 4th order kappa value'''
        dx = self.dx
        alpha = self.alpha

        kappa = 2 - alpha**2 * dx**2

        if self.desired_order == "4": # Add the 4th order term
            kappa += alpha**4 * dx**4 / 12

        return kappa

    def build_stiffness_matrix(self):
        '''Given a number of sections, build the stiffness_matrix'''
        # The stiffness matrix follows the form [1, -kappa, 1] in a diagonal
        kappa = self.get_kappa()

        size = self.num_sections + 1 # Include boundary points
        self.stiffness_matrix = np.zeros((size, size))

        for i in range(1, size - 1): # Internal points
            self.stiffness_matrix[i, i - 1] = 1 # Left neighbor
            self.stiffness_matrix[i, i] = -kappa # Center
            self.stiffness_matrix[i, i + 1] = 1 # Right neighbor

        # Enforce boundary conditions in the stiffness matrix
        self.stiffness_matrix[0, 0] = 1 # Left boundary
        self.stiffness_matrix[-1, -1] = 1 # Right boundary

    def build_constant_vector(self):
        '''Given a number of sections, build the constant_vector where the
        first and last elements come from boundary conditions'''
        self.constant_vector = np.zeros(self.num_sections + 1) # Include endpoints
```

```
        self.constant_vector[0] = self.U_0 # Left boundary
        self.constant_vector[-1] = self.U_L # Right boundary

    def run(self, forcing_freq:float, num_sections:int=4):
        '''Runs FDM for the given forcing frequency and num_sections.
        Stores displacment ampltitude data in self.disp_amp
        Stores strain amplitude data in self.strain_amp'''
        # Save the appropriate values
        self.num_sections = num_sections
        self.dx = 1 / num_sections
        self.alpha = forcing_freq*(self.bar.density / self.bar.E)**0.5
        self.forcing_freq = forcing_freq

        self.build_stiffness_matrix()
        self.build_constant_vector()

        # Solve the system
        self.disp_amp = np.linalg.solve(self.stiffness_matrix, self.constant_vector)

        # Calculate all of the nodal strains
        self.calc_all_nodal_strain()

        # Save the x values for the nodal values
        self.x_vals = np.linspace(0, 1, num_sections+1) # sections + endpoints


if __name__ == "__main__":
    import matplotlib.pyplot as plt
    from common import freq_from_alpha
    bar = Bar()

    # Initialize FDM
    fdm_2 = FDM(bar, desired_order="2")

    # Define alpha values for testing
    alpha_values = np.linspace(0, 20, 6)

    # Set boundary conditions
    fdm_2.set_boundary_conditions(U_0=0, U_L=100)

    # Plot displacement amplitudes for varying alpha values
    plt.figure(figsize=(12, 6))
    for alpha in [100]:
        forcing_freq = freq_from_alpha(bar, alpha)
        fdm_2.run(forcing_freq, num_sections=4)
        plt.plot(fdm_2.x_vals, fdm_2.disp_amp, label=f' = {alpha:.2f}')

    plt.title("FDM Displacement Amplitude vs x for Various  Values")
    plt.xlabel("x (Normalized Position)")
    plt.ylabel("Displacement Amplitude")
```

```
    plt.legend()
    plt.grid(True)
    plt.show()

    # Set boundary conditions and run the solver
    fdm_4 = FDM(bar, desired_order="4")
    fdm_4.set_boundary_conditions(U_0=0, U_L=100)
    freq = freq_from_alpha(bar, 4)
    num_sections = 10
    fdm_4.run(freq, num_sections)
    fdm_2.run(freq, num_sections)

    # Interpolate at a specific x_val
    x_val = 0.31
    interpolated_strain = fdm_4.get_interpolated_strain_amp(x_val)
    print(f"Interpolated Strain Amplitude at x = {x_val}: {interpolated_strain}")

    interpolated_strain = fdm_2.get_interpolated_strain_amp(x_val)
    print(f"Interpolated Strain Amplitude at x = {x_val}: {interpolated_strain}")
```

9.7 FEM.py

```
# FEM.py

import numpy as np

from Bar import Bar
from DiscreteMethod import DiscreteMethod

class FEM(DiscreteMethod):
    def __init__(self, bar:'Bar', desired_order="2"):
        super().__init__(bar, desired_order)

    def get_kappa(self, row:int, col:int):
        dx = self.dx
        alpha = self.alpha

        if (row == 1 and col == 1) or (row == 2 and col == 2):
            return -1/dx + alpha**2*dx/ 3.0
        elif row == 3 and col == 3:
            return -1/(3*dx) + alpha**2*dx/30.0
        elif (row == 1 and col == 2) or (row == 2 and col == 1):
            return 1/dx + alpha**2*dx/6.0
        elif (row == 1 and col == 3) or (row == 3 and col == 1) or (row == 2 and col == 3) or (row ==
            3 and col == 2):
            return alpha**2*dx/12.0

    def get_all_kappa(self):
```

```
    '''Returns the kappa matrix for p=1 or p=2 FEM.
    Note that both matrices are equivalent regardless of p=1 or p=2, but p=1 does not use row 3 or
        col 3'''

    K = np.zeros((3, 3))
    for row in range(3):
        for col in range(3):
            K[row, col] = self.get_kappa(row+1, col+1)
    return K


def build_stiffness_matrix(self):
    '''Given a number of sections, build the stiffness_matrix for P=1 FEM'''
    num_sections = self.num_sections
    if self.desired_order == "2": # p=1
        size = num_sections + 1 # Number of nodes for P=1
        self.stiffness_matrix = np.zeros((size, size))

        # Get local stiffness matrix (K) for a single section
        K_local = self.get_all_kappa()

        # Extract local stiffness terms for P=1
        k11 = K_local[0, 0]
        k12 = K_local[0, 1]
        k21 = K_local[1, 0]
        k22 = K_local[1, 1]

        # Loop through each section and assemble the global stiffness matrix
        for section in range(num_sections):
            self.stiffness_matrix[section, section - 1] = k21
            self.stiffness_matrix[section, section] = k22 + k11
            self.stiffness_matrix[section, section + 1] = k12

        # Enforce boundary conditions by fixing the first and last nodes
        self.stiffness_matrix[0, :] = 0 # Set first row to zero
        self.stiffness_matrix[0, 0] = 1 # Fix the left boundary

        self.stiffness_matrix[-1, :] = 0 # Set last row to zero
        self.stiffness_matrix[-1, -1] = 1 # Fix the right boundary

    elif self.desired_order == "4": # p=2
        size = 2 * num_sections + 1 # Number of linear + quadratic nodes
        self.stiffness_matrix = np.zeros((size, size))

        # Get local stiffness matrix (K) for a single section
        K_local = self.get_all_kappa()

        # Extract local stiffness terms for P=2
        k11 = K_local[0, 0]
        k12 = K_local[0, 1]
```

```
        k13 = K_local[0, 2]
        k21 = K_local[1, 0]
        k22 = K_local[1, 1]
        k23 = K_local[1, 2]
        k31 = K_local[2, 0]
        k32 = K_local[2, 1]
        k33 = K_local[2, 2]

        # Loop through each section and assemble the global stiffness matrix for similar to p=1
        for section in range(num_sections):
            # Assemble contributions from K_local
            self.stiffness_matrix[section, section - 1] = k21
            self.stiffness_matrix[section, section] = k22 + k11
            self.stiffness_matrix[section, section + 1] = k12

            # Add the k23 and k13 terms in that correspond to Ei and Ei+1
            self.stiffness_matrix[section, num_sections+section] = k23
            self.stiffness_matrix[section, num_sections+section+1] = k13


        # Add the quadratic element relations
        for section in range(num_sections):
            # Assemble contributions from K_local
            self.stiffness_matrix[num_sections+section+1, section] = k31
            self.stiffness_matrix[num_sections+section+1, section+1] = k32
            self.stiffness_matrix[num_sections+section+1, num_sections+section+1] = k33

        # Enforce boundary conditions by fixing the first and last nodes
        self.stiffness_matrix[0, :] = 0 # Set first row to zero
        self.stiffness_matrix[0, 0] = 1 # Fix the left boundary

        self.stiffness_matrix[num_sections, :] = 0 # Set last temperature row to 0
        self.stiffness_matrix[num_sections, num_sections] = 1 # Fix the right boundary

def build_constant_vector(self):
    '''Given a number of sections, build the constant_vector where the
    first and last elements come from boundary conditions'''
    num_sections = self.num_sections
    if self.desired_order == "2": # p=1
        self.constant_vector = np.zeros(num_sections + 1) # Include endpoints
        self.constant_vector[0] = self.U_0 # Left boundary
        self.constant_vector[-1] = self.U_L # Right boundary
    if self.desired_order == "4": # p=2
        self.constant_vector = np.zeros(2*num_sections + 1) # Include endpoints
        self.constant_vector[0] = self.U_0 # Left boundary
        self.constant_vector[num_sections] = self.U_L # Right boundary

def run(self, forcing_freq:float, num_sections:int=4):
    '''Runs FDM for the given forcing frequency and num_sections.
    Stores displacment ampltitude data in self.disp_amp
```

```
        Stores strain amplitude data in self.strain_amp'''
        # Save the appropriate values
        self.num_sections = num_sections
        self.dx = 1 / num_sections
        self.alpha = forcing_freq*(self.bar.density / self.bar.E)**0.5
        self.forcing_freq = forcing_freq

        self.build_stiffness_matrix()
        self.build_constant_vector()

        # Solve the system
        if self.desired_order == "2": # p=1
            self.disp_amp = np.linalg.solve(self.stiffness_matrix, self.constant_vector)
        elif self.desired_order == "4": # p=3
            self.disp_amp = np.linalg.solve(self.stiffness_matrix, self.constant_vector)
            self.disp_amp = self.disp_amp[0:self.num_sections+1]

        # Calculate all of the nodal strains
        self.calc_all_nodal_strain()

        # Save the x values for the nodal values
        self.x_vals = np.linspace(0, 1, self.num_sections+1) # sections + endpoints



if __name__ == "__main__":
    import matplotlib.pyplot as plt
    from common import freq_from_alpha
    bar = Bar()

    # Initialize FDM
    fem = FEM(bar, desired_order="4")

    # Define alpha values for testing
    alpha_values = np.linspace(0, 20, 6)

    # Set boundary conditions
    fem.set_boundary_conditions(U_0=0, U_L=100)

    # Plot displacement amplitudes for varying alpha values
    plt.figure(figsize=(12, 6))
    for alpha in [20]:
        forcing_freq = freq_from_alpha(bar, alpha)
        fem.run(forcing_freq, num_sections=100)
        plt.plot(fem.x_vals, fem.disp_amp, label=f' = {alpha:.2f}')

    plt.title("FEM Displacement Amplitude vs x for Various  Values")
    plt.xlabel("x (Normalized Position)")
    plt.ylabel("Displacement Amplitude")
    plt.legend()
```

```
    plt.grid(True)
    plt.show()

    # Set boundary conditions and run the solver
    fem.set_boundary_conditions(U_0=0, U_L=100)
    freq = freq_from_alpha(bar, 4)
    num_sections = 100
    fem.run(freq, num_sections)

    # Interpolate at a specific x_val
    x_val = 0.31
    interpolated_strain = fem.get_interpolated_strain_amp(x_val)
    print(f"Interpolated Strain Amplitude at x = {x_val}: {interpolated_strain}")
```

9.8 Convergence.py

```
# convergence.py

import numpy as np
import matplotlib.pyplot as plt

from DiscreteMethod import DiscreteMethod

class Convergence:
    def __init__(self, analytical, fdm, fem):
        """
        Initialize the Convergence class.

        Parameters:
        - analytical:
        - fdm: Instance of the FDM class.
        - fem: Instance of the FEM class.
        """
        self.analytical = analytical
        self.fdm: 'DiscreteMethod' = fdm
        self.fem: 'DiscreteMethod' = fem

    def calc_error(self, exact, q_1):
        """Calculate relative error."""
        return np.abs((exact - q_1) / exact)

    def calc_beta(self, exact, q_1, q_2, dx_1, dx_2):
        """Calculate convergence rate beta."""
        return np.log(np.abs((exact - q_1) / (exact - q_2))) / np.log(dx_1 / dx_2)

    def calc_extrapolated(self, q1, q2, q3, tolerance=1e-10):
        """Calculate Richardson extrapolation, returns NaN if denominator is too small."""
        numerator = q1 * q3 - q2**2
```

```
    denominator = q1 + q3 - 2 * q2
    if abs(denominator) < tolerance:
        return float('NaN') # Return NaN if denominator is close to zero
    return numerator / denominator

def run_convergence(self, forcing_freq: float, num_sections_range, metric_func):
    """
    Run convergence analysis for FDM and FEM.

    Parameters:
    - forcing_freq: The forcing frequency to test.
    - num_sections_range: Array of num_sections to test.
    - metric_func: Callable defining the metric to analyze (e.g., U'(L) or U(1 / (2pi))).

    Returns:
    - results: Dictionary containing errors, betas, extrapolated values, and analytical values for
        both methods.
    """
    results = {
        "fdm": {
            "num_sections": [],
            "errors": [],
            "betas": [],
            "extrapolated_errors": [],
            "extrapolated_values": [],
            "analytical_values": [],
        },
        "fem": {
            "num_sections": [],
            "errors": [],
            "betas": [],
            "extrapolated_errors": [],
            "extrapolated_values": [],
            "analytical_values": [],
        },
    }

    # Analytical value for the metric
    analytical_value = metric_func(self.analytical, forcing_freq)

    for num_sections in num_sections_range:
        dx = 1 / num_sections

        # Run FDM
        self.fdm.run(forcing_freq, num_sections)
        fdm_metric = metric_func(self.fdm)
        fdm_error = self.calc_error(analytical_value, fdm_metric)

        # Run FEM
        self.fem.run(forcing_freq, num_sections)
```

```
        fem_metric = metric_func(self.fem)
        fem_error = self.calc_error(analytical_value, fem_metric)

        # Store results
        results["fdm"]["num_sections"].append(num_sections)
        results["fdm"]["errors"].append(fdm_error)
        results["fdm"]["extrapolated_values"].append(fdm_metric)
        results["fdm"]["analytical_values"].append(analytical_value)

        results["fem"]["num_sections"].append(num_sections)
        results["fem"]["errors"].append(fem_error)
        results["fem"]["extrapolated_values"].append(fem_metric)
        results["fem"]["analytical_values"].append(analytical_value)

    # Compute extrapolated errors and betas
    for method in ["fdm", "fem"]:
        extrapolated_errors = [np.nan] # Padding for the first run
        betas = [np.nan] # Padding for the first run
        extrapolated_betas = [np.nan] # Padding for the first run

        values = results[method]["extrapolated_values"]
        num_sections = results[method]["num_sections"]

        # Extrapolation and beta calculations
        for i in range(len(num_sections) - 1):
            q1 = values[i]
            q2 = values[i + 1]
            dx1 = 1 / num_sections[i]
            dx2 = 1 / num_sections[i + 1]

            beta = self.calc_beta(analytical_value, q1, q2, dx1, dx2)

            if i >= 1:
                extrapolated_value = self.calc_extrapolated(
                    values[i - 1], q1, q2
                )  # Requires 3 consecutive values for extrapolation
                extrapolated_error = self.calc_error(extrapolated_value, q2)
                extrapolated_beta = self.calc_beta(extrapolated_value, q1, q2, dx1, dx2)
            else:
                extrapolated_value = np.nan
                extrapolated_error = np.nan
                extrapolated_beta = np.nan

            if i >= 1:
                extrapolated_errors.append(extrapolated_error)
                extrapolated_betas.append(extrapolated_beta)
            else:
                extrapolated_errors.append(np.nan)
                extrapolated_betas.append(np.nan)
```

```
            betas.append(beta)

        results[method]["extrapolated_errors"] = extrapolated_errors
        results[method]["betas"] = betas
        results[method]["extrapolated_betas"] = extrapolated_betas

    return results



def plot_convergence(self, results):
    """
    Plot convergence results for FDM and FEM, including errors relative to analytical
    and extrapolated values.

    Parameters:
    - results: Dictionary from run_convergence.
    """
    plt.figure(figsize=(12, 6))

    for method in ["fdm", "fem"]:
        num_sections = results[method]["num_sections"]
        errors = results[method]["errors"]
        extrapolated_errors = results[method]["extrapolated_errors"]

        # Pad num_sections to match extrapolated_errors length
        padded_num_sections = [np.nan] * 2 + num_sections[:-2]

        # Plot true error relative to the analytical solution
        plt.loglog(
            num_sections, errors, '-o', label=f"{method.upper()} Analytical Error"
        )

        # Plot error relative to the extrapolated solution
        plt.loglog(
            num_sections, extrapolated_errors, '--s', label=f"{method.upper()} Extrapolated Error"
        )

    plt.xlabel("Number of Sections")
    plt.ylabel("Error (Relative)")
    plt.title("Convergence Analysis: True and Extrapolated Errors")
    plt.legend()
    plt.grid(True, which="both", linestyle="--")
    plt.show()


if __name__ == "__main__":
    from Analytical import Analytical
    from Bar import Bar
    from common import freq_from_alpha
```

```python
from DiscreteMethod import DiscreteMethod
from FDM import FDM
from FEM import FEM

def metric_u_prime_l(method:'DiscreteMethod', forcing_freq:float = None):
    """Extract U'(L) from the method."""
    if forcing_freq is None: # Finite method
        return method.get_strain_amp(1.0)
    else:
        return method.get_strain_amp(forcing_freq, 1.0) # Analytical

def metric_u_half_pi(method:'DiscreteMethod', forcing_freq:float = None):
    """Extract U(1 / (2pi)) from the method."""
    x_half_pi = 1 / (2 * np.pi)
    if forcing_freq is None: # Finite method
        return method.get_disp_amp(x_half_pi)
    else:
        return method.get_disp_amp(forcing_freq, x_half_pi) # Analytical

# Initialize Bar, FDM, FEM
bar = Bar()
analy = Analytical(bar)
fdm = FDM(bar, desired_order="2")
fem = FEM(bar, desired_order="2")

# Set boundary conditions
fdm.set_boundary_conditions(U_0=0, U_L=100)
fem.set_boundary_conditions(U_0=0, U_L=100)

# Initialize Convergence
convergence = Convergence(analy, fdm, fem)

alpha = 0.25
forcing_freq = freq_from_alpha(bar, alpha)

# Run convergence for U'(L)
num_sections_range = [2**n for n in range(1, 12)]

results = convergence.run_convergence(forcing_freq, num_sections_range, metric_u_half_pi)

# Plot results
convergence.plot_convergence(results)
```