AI-Assisted Coding Project in Python
Modeling Fiber Orientation Effects in Anisotropic Composite Bars

Dineshraju Elachipalayam Thangavel
CMS@TUBAF matr. no. 69306

July 11, 2025

Abstract

This project introduces a Python-based finite-element solver for two-dimensional plane-stress analysis of composite bars with varying fiber orientations under axial loading. The code blends fiber and matrix properties into element stiffness, applies material rotation for each orientation, and discretizes the bar using a structured quadrilateral mesh. Essential boundary conditions and loading are enforced, and the resulting system is solved iteratively. Automated routines extract displacement, stress, and strain fields, perform a fullangle sweep to characterize stiffness variation, and carry out mesh-convergence studies to assess numerical accuracy. The solver was initially developed using AI-assisted programming with ChatGPT-4o (v4.1), accelerating code structure and modularity. Results demonstrate quadratic convergence, reveal the periodic dependence of stiffness on fiber angle, and establish a modular platform for future three-dimensional and nonlinear extensions.

Keywords: orthotropic composite bars; finite element method; fiber orientation; plane stress; Python

1 Introduction

Fiber-reinforced composite beams are widely employed in aerospace, automotive, and civil engineering structures due to their high specific stiffness, strength-to-weight ratio, and the ability to tailor mechanical properties through fiber orientation. In lightweight structural components such as satellite booms, robotic arms, or bioinspired tendon-reinforced actuators, directional stiffness control is critical. Accurate modeling of their mechanical response under axial loads is essential for predicting performance and ensuring reliability.

In this work, we investigate the elastic behavior of a slender, 2D rectangular cantilever beam composed of a transversely orthotropic composite material. The beam is fixed at the left end and subjected to a uniform axial tensile force F at the free right end, as illustrated in Figure 1. The fiber direction within the composite is oriented at an angle θ with respect to the beam axis (longitudinal direction), introducing anisotropy into the stressstrain response.

The mechanical behavior of the composite is modeled using the 2D plane stress equations of anisotropic linear elasticity. Under this formulation, the stress vector $\boldsymbol{\sigma} = [\sigma_{xx}, \sigma_{yy}, \tau_{xy}]^T$ is related to the strain vector $\boldsymbol{\varepsilon} = [\epsilon_{xx}, \epsilon_{yy}, \gamma_{xy}]^T$ by a transformed stiffness matrix $\mathbf{Q}^*(\theta)$ that incorporates fiber orientation effects:

$$\boldsymbol{\sigma} = \mathbf{Q}^*(\theta)\,\boldsymbol{\varepsilon}. \tag{1}$$

For a transversely orthotropic lamina, the original stiffness matrix in the material coordinate system is given by:

$$\mathbf{Q} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix}, \tag{2}$$

where the components depend on the longitudinal and transverse Young's moduli E_1, E_2, shear modulus G_{12}, and Poisson ratios ν_{12}, ν_{21}. The matrix $\mathbf{Q}^*(\theta)$ is then computed via coordinate transformation to align the material with the global reference frame.

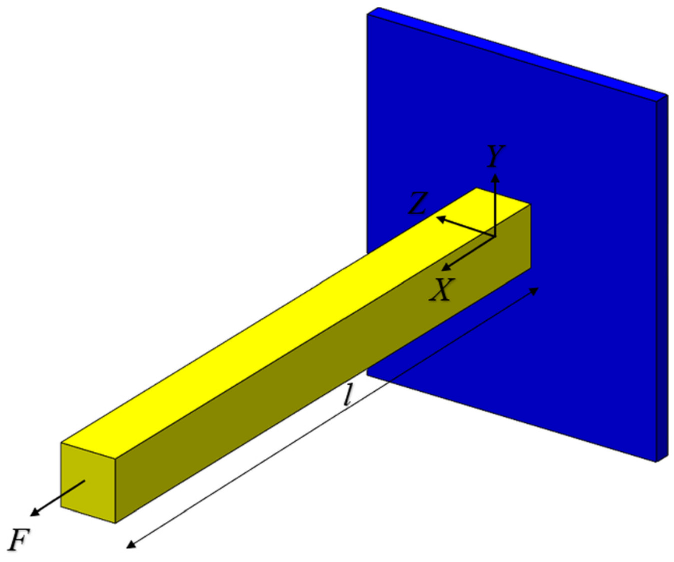

Figure 1: Cantilever bar fixed at the left end and subjected to an axial tensile force F at the free end. Adapted from [1].

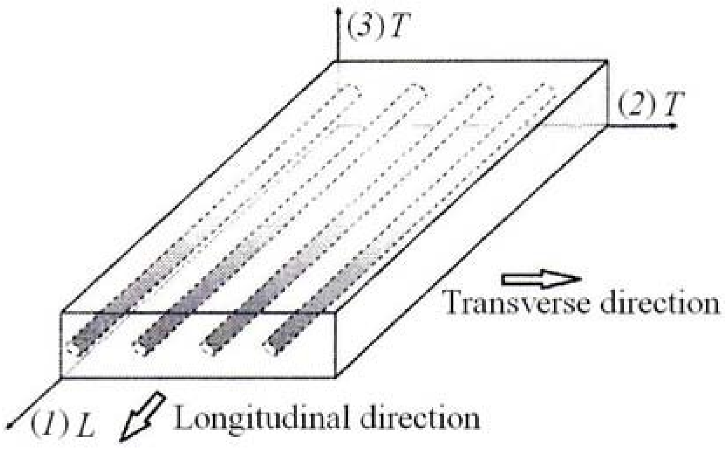

Figure 2: Material coordinate system for a unidirectional fiber-reinforced composite. In this study, only the 2D in-plane behavior under plane stress conditions is modeled. Adapted from [2].

The equilibrium equations are expressed as:

$$\nabla \cdot \boldsymbol{\sigma} = \mathbf{0}, \tag{3}$$

and solved under the assumption of zero body forces and plane stress conditions.

The beam domain is discretized using a structured mesh of quadrilateral finite elements. Each element accounts for the rotated material axes through the angle-dependent stiffness matrix $\mathbf{Q}^*(\theta)$, enabling fiber orientation to vary across simulations. The weak form is implemented in a standard finite element framework, and the resulting linear system is solved to obtain the full 2D displacement field.

To validate the numerical model, we derive analytical expressions for axial displacement and stress using the compliance matrix for orthotropic materials under uniaxial tension. These expressions serve as benchmarks for evaluating the FEM solution accuracy across various fiber orientations.

Problem Statement. *Simulate the elastic response of a transversely orthotropic cantilever beam under axial tensile loading, with varying fiber orientations. Use finite element analysis to compute the displacement and stress distribution, and validate results against analytical solutions derived from anisotropic elasticity theory. Quantify the effect of fiber angle θ on tip displacement and effective stiffness.*

2 Prompt

Write a Python program that uses NumPy and Matplotlib to perform a 2D finite-element analysis of a unidirectional composite beam under axial loading and compare with an analytical solution.

Script Structure and Requirements

1. Global plotting style - Set Matplotlib rc parameters for font sizes (main font 10, axes titles 11, labels 10, legend 10, xtick/ytick 9) and default figure size 6×3 inches.

2. Define input material properties - Fiber modulus $E_f = 235 \times 10^9\,\mathrm{Pa}$, matrix modulus $E_m = 3.5 \times 10^9\,\mathrm{Pa}$, fiber and matrix Poissons ratios $\nu_f = 0.2$, $\nu_m = 0.35$, shear modulus $G_{12} = 5 \times 10^9\,\mathrm{Pa}$, and volume fractions $V_f = 0.572$, $V_m = 0.428$. - Beam length $L = 1.0\,\mathrm{m}$, height $h = 0.05\,\mathrm{m}$, cross-sectional area $A = h \times 1.0\,\mathrm{m}^2$, and initial mesh resolution $n_x = 50$, $n_y = 10$.

3. Compute lamina stiffness matrix - Calculate $E_1 = E_f V_f + E_m V_m$, $E_2 = \frac{1}{V_f/E_f + V_m/E_m}$, $\nu_{12} = \nu_f V_f + \nu_m V_m$, $\nu_{21} = \nu_{12} E_2 / E_1$. - Build the reduced stiffness Q matrix with components $Q_{11}, Q_{22}, Q_{12}, Q_{66}$.

4. Define stiffness transformation - `transform_stiffness(Q,` θ`)` returns the rotated Q^* using analytical formulas in terms of $\cos^4\theta$, $\sin^2\theta\cos^2\theta$, etc.

5. Mesh generation - `create_mesh(length, height,` n_x`,` n_y`)` returns node coordinates and element connectivity for 4-node quads.

6. Shape functions & element stiffness - `shape_functions(`ξ`,` η`)` - `calculate_stiffness_matrix(Q, coords)` using 2×2 Gauss quadrature to build element stiffness k_e.

7. Global assembly & BCs - `assemble_global_stiffness(nodes, elements,` Q_θ`)` - `apply_boundary_conditions(K, F, nodes)` that fixes all DOFs at $x = 0$. - `apply_load(nodes, total_load, direction='x')` distributes a total load on the right edge.

8. NewtonRaphson solver - `solve_for_displacements_newton(`K_{full}`,` F_{ext}`, nodes, elements,` Q_θ`, fixed_dofs, free_dofs, max_iters=25, tol=1e-6)` - Print at each iteration: [Newton Step i] $\|R\|$, $\|\Delta U\|$. - (Note: for linear elasticity this will converge in one iteration but include the full loop.)

9. Post-processing - `compute_element_strain_stress(nodes, elements, U,` Q_θ`)` returns average strains and stresses per element. - `analytical_tip_displacement_full_2D(total_load, L, height, Q,` θ`, y_tip)` computes analytical tip displacements and strains via compliance matrix.

10. Driver routines - `fem_tip_displacement(`θ`, total_load)` $\rightarrow U_x, U_y$ at the top tip. - `run_case(`θ`, total_load)` prints FEM vs analytical U_x tip and % error. - `sweep_theta_cases(total_load)` loops $\theta = 0:5:360°$, collects FEM and analytical $U_x, U_y, \sigma_{xx}, \varepsilon_{xx}$, computes effective moduli.

11. Plotting utilities - `plot_bar_mesh(nodes, elements,` n_x`,` n_y`, title=None)` draws the mesh. - For each metric (displacement, stress, strain, modulus vs θ), write a `plot_...` function that: - Creates a 7.5×3 in figure - Plots data with markers - Sets labels and title - Places the legend outside the plot area, e.g.:

```
plt.legend(loc='center left', bbox_to_anchor=(1.05, 1), frameon=False)
```

- Adds a dashed grid and tight layout.

12. Mesh convergence study - `mesh_convergence_study(mesh_sizes,` θ_{deg}`, total_load, L, height, Q)` runs FEM for meshes like [(10,2),(20,4),(40,8),(80,16),(160,32)], computes relative error vs analytical U_x, and plots error vs mesh size.

13. __main__ block - Generate and plot the mesh in red (linewidth=1.2). - Run `run_case` for $\theta = 0, 45, 90°$ with `total_load=1000.0`. - Call `sweep_theta_cases(1000.0)` and plot all four metrics. - Perform mesh convergence study at $\theta = 45°$ and `total_load=1000.0`.

Other notes - Ensure all imports (`import numpy as np`, `import matplotlib.pyplot as plt`, `import matplotlib as mpl`) are at the top, and that the code runs as a standalone script.

Figure 3: Snippet of prompt used for code generation.

3 Code Listing

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

# 1. Global font and style settings
mpl.rc('font',      size=10)    # main font size
mpl.rc('axes',      titlesize=11, labelsize=10)
mpl.rc('legend',    fontsize=10)
mpl.rc('xtick',     labelsize=9)
mpl.rc('ytick',     labelsize=9)
mpl.rc('figure',    figsize=(6, 3))

# --- Input Material Properties ---
Ef = 235e9  # Fiber modulus [Pa]
Em = 3.5e9  # Matrix modulus [Pa]
vf = 0.2
vm = 0.35
G12 = 5e9
Vf = 0.572
Vm = 0.428
L = 1.0
height = 0.05  # m
A = height * 1.0  # (m^2, for 1m width in 2D)
nx, ny = 50, 10  # mesh

# --- Material Properties Calculation ---
E1 = Ef * Vf + Em * Vm
E2 = 1 / (Vf / Ef + Vm / Em)
v12 = vf * Vf + vm * Vm
v21 = v12 * E2 / E1

denom = 1 - v12 * v21
Q11 = E1 / denom
Q22 = E2 / denom
Q12 = v12 * E2 / denom
Q66 = G12
Q = np.array([[Q11, Q12, 0], [Q12, Q22, 0], [0, 0, Q66]])

def transform_stiffness(Q, theta_deg):
    theta = np.deg2rad(theta_deg)
    c = np.cos(theta)
    s = np.sin(theta)
    Q11, Q12, _, Q22, _, Q66 = Q[0,0], Q[0,1], Q[0,2], Q[1,1], Q[1,2], Q[2,2]
    # Analytical transformation
    Q11_star = Q11 * c**4 + 2 * (Q12 + 2*Q66) * s**2 * c**2 + Q22 * s**4
    Q22_star = Q11 * s**4 + 2 * (Q12 + 2*Q66) * s**2 * c**2 + Q22 * c**4
    Q12_star = (Q11 + Q22 - 4*Q66) * s**2 * c**2 + Q12 * (s**4 + c**4)
    Q66_star = (Q11 + Q22 - 2*Q12 - 2*Q66) * s**2 * c**2 + Q66 * (s**4 + c**4)
    Qstar = np.array([[Q11_star, Q12_star, 0],
                      [Q12_star, Q22_star, 0],
                      [0, 0, Q66_star]])
    return Qstar

def create_mesh(length=1.0, height=0.05, nx=50, ny=10):
    dx, dy = length / nx, height / ny
    nodes, node_id_map = [], {}
    for j in range(ny + 1):
        for i in range(nx + 1):
            node_id = j * (nx + 1) + i
            node_id_map[(i, j)] = node_id
            nodes.append([i * dx, j * dy])
    elements = []
    for j in range(ny):
        for i in range(nx):
            n1 = node_id_map[(i, j)]
            n2 = node_id_map[(i + 1, j)]
            n3 = node_id_map[(i + 1, j + 1)]
            n4 = node_id_map[(i, j + 1)]
            elements.append([n1, n2, n3, n4])
    return np.array(nodes), np.array(elements)

def shape_functions(xi, eta):
    return 0.25 * np.array([
        [-(1 - eta), -(1 - xi)],
        [ (1 - eta), -(1 + xi)],
        [ (1 + eta),  (1 + xi)],
        [-(1 + eta),  (1 - xi)]
    ])

def calculate_stiffness_matrix(Q, coords):
    ke = np.zeros((8, 8))
    gauss_pts = [(-1/np.sqrt(3), -1/np.sqrt(3)), (1/np.sqrt(3), -1/np.sqrt(3)),
                 (1/np.sqrt(3), 1/np.sqrt(3)), (-1/np.sqrt(3), 1/np.sqrt(3))]
    for xi, eta in gauss_pts:
        dN_dxi = shape_functions(xi, eta)
        J = sum(np.outer(dN_dxi[i], coords[i]) for i in range(4))
        detJ = np.linalg.det(J)
        if detJ <= 0:
            raise ValueError("Jacobian determinant is non-positive!")
        J_inv = np.linalg.inv(J)
        dN_dx = dN_dxi @ J_inv
        B = np.zeros((3, 8))
        for i in range(4):
            B[0, 2*i]   = dN_dx[i, 0]
            B[1, 2*i+1] = dN_dx[i, 1]
            B[2, 2*i]   = dN_dx[i, 1]
            B[2, 2*i+1] = dN_dx[i, 0]
        ke += B.T @ Q @ B * detJ
    return ke

def assemble_global_stiffness(nodes, elements, Q_theta):
    K = np.zeros((2 * len(nodes), 2 * len(nodes)))
    for elem in elements:
        coords = nodes[elem]
        ke = calculate_stiffness_matrix(Q_theta, coords)
        dof_map = np.hstack([[2*n, 2*n+1] for n in elem])
        for i in range(8):
            for j in range(8):
                K[dof_map[i], dof_map[j]] += ke[i, j]
    return K

def apply_boundary_conditions(K, F, nodes):
    fixed_dofs = []
    for i, (x, y) in enumerate(nodes):
        if np.isclose(x, 0.0):
            fixed_dofs.extend([2*i, 2*i+1])
    free_dofs = np.setdiff1d(np.arange(len(F)), fixed_dofs)
    K_reduced = K[np.ix_(free_dofs, free_dofs)]
    F_reduced = F[free_dofs]
    return K_reduced, F_reduced, fixed_dofs, free_dofs

def apply_load(nodes, total_load, direction='x'):
    F = np.zeros(len(nodes)*2)
    right_edge_nodes = [i for i, node in enumerate(nodes) if np.isclose(node[0], L)]
    load_per_node = total_load / len(right_edge_nodes)
    for i in right_edge_nodes:
        idx = 2*i if direction=='x' else 2*i+1
        F[idx] = load_per_node
    return F, right_edge_nodes

def solve_for_displacements_newton(K_full, F_ext, nodes, elements, Q_theta, fixed_dofs,
    ↪ free_dofs,
                                   max_iters=25, tol=1e-6):
    U = np.zeros_like(F_ext)
    for step in range(max_iters):
        F_int = np.zeros_like(F_ext)
        K_tangent = np.zeros_like(K_full)
        for elem in elements:
            coords = nodes[elem]
            u_elem = np.hstack([U[2*n:2*n+2] for n in elem])
            gauss_pts = [(-1/np.sqrt(3), -1/np.sqrt(3)), (1/np.sqrt(3), -1/np.sqrt(3)),
                         (1/np.sqrt(3), 1/np.sqrt(3)), (-1/np.sqrt(3), 1/np.sqrt(3))]
            f_int_local = np.zeros(8)
            ke_local = np.zeros((8, 8))
            for xi, eta in gauss_pts:
                dN_dxi = shape_functions(xi, eta)
                J = sum(np.outer(dN_dxi[i], coords[i]) for i in range(4))
                detJ = np.linalg.det(J)
                J_inv = np.linalg.inv(J)
                dN_dx = dN_dxi @ J_inv
                B = np.zeros((3, 8))
                for i in range(4):
                    B[0, 2*i]   = dN_dx[i, 0]
                    B[1, 2*i+1] = dN_dx[i, 1]
                    B[2, 2*i]   = dN_dx[i, 1]
                    B[2, 2*i+1] = dN_dx[i, 0]
                strain = B @ u_elem
                stress = Q_theta @ strain
                f_int_local += B.T @ stress * detJ
                ke_local += B.T @ Q_theta @ B * detJ
            dof_map = np.hstack([[2*n, 2*n+1] for n in elem])
            for i in range(8):
                F_int[dof_map[i]] += f_int_local[i]
                for j in range(8):
                    K_tangent[dof_map[i], dof_map[j]] += ke_local[i, j]
        R = F_ext - F_int
        R_reduced = R[free_dofs]
        K_reduced = K_tangent[np.ix_(free_dofs, free_dofs)]
        delta_U = np.linalg.solve(K_reduced, R_reduced)
        U[free_dofs] += delta_U
        norm_R = np.linalg.norm(R_reduced)
        norm_dU = np.linalg.norm(delta_U)
        print(f"[Newton Step {step}] ||R|| = {norm_R:.2e}, ||  U|| = {norm_dU:.2e}")
        if norm_R < tol:
            print("    Converged.")
            break
    else:
        print("       Newton Raphson did not converge in max iterations.")
    return U

def compute_element_strain_stress(nodes, elements, U, Q_theta):
    strain_list, stress_list = [], []
    for elem in elements:
        coords = nodes[elem]
        u_elem = np.hstack([U[2*n:2*n+2] for n in elem])
        elem_strains, elem_stresses = [], []
        gauss_pts = [(-1/np.sqrt(3), -1/np.sqrt(3)), (1/np.sqrt(3), -1/np.sqrt(3)),
                     (1/np.sqrt(3), 1/np.sqrt(3)), (-1/np.sqrt(3), 1/np.sqrt(3))]
        for xi, eta in gauss_pts:
            dN_dxi = shape_functions(xi, eta)
            J = sum(np.outer(dN_dxi[i], coords[i]) for i in range(4))
            J_inv = np.linalg.inv(J)
            dN_dx = dN_dxi @ J_inv
            B = np.zeros((3, 8))
            for i in range(4):
                B[0, 2*i]   = dN_dx[i, 0]
                B[1, 2*i+1] = dN_dx[i, 1]
                B[2, 2*i]   = dN_dx[i, 1]
                B[2, 2*i+1] = dN_dx[i, 0]
            strain = B @ u_elem
            stress = Q_theta @ strain
            elem_strains.append(strain)
            elem_stresses.append(stress)
        strain_list.append(np.mean(elem_strains, axis=0))
        stress_list.append(np.mean(elem_stresses, axis=0))
    return np.array(strain_list), np.array(stress_list), elements


def analytical_tip_displacement_full_2D(total_load, L, height, Q, theta_deg, y_tip=height):
    """
    Analytical tip x- and y-displacement at the tip, using full 2D compliance matrix.
    y_tip: vertical location at tip (0.0 for bottom tip, height for top tip)
    """
    A = height * 1.0
    sigma = np.array([total_load / A, 0.0, 0.0])
    Qstar = transform_stiffness(Q, theta_deg)
    Sstar = np.linalg.inv(Qstar)
    strain = Sstar @ sigma
    ux_tip = strain[0] * L
    uy_tip = strain[1] * y_tip
    return ux_tip, uy_tip, strain
```

```python


def fem_tip_displacement(theta, total_load):
    Q_theta = transform_stiffness(Q, theta)
    nodes, elements = create_mesh(length=L, height=height, nx=nx, ny=ny)
    K = assemble_global_stiffness(nodes, elements, Q_theta)
    F, right_edge_nodes = apply_load(nodes, total_load, direction='x')
    K_reduced, F_reduced, fixed_dofs, free_dofs = apply_boundary_conditions(K, F, nodes)
    U = solve_for_displacements_newton(K, F, nodes, elements, Q_theta, fixed_dofs,
    ↪ free_dofs)
    # Tip node index (x=1.0, y=height)
    tip_indices = np.where(np.isclose(nodes[:,0], L) & np.isclose(nodes[:,1], height))
    ↪ [0]
    Ux_tip = U[2*tip_indices[0]] if len(tip_indices) > 0 else np.nan
    Uy_tip = U[2*tip_indices[0]+1] if len(tip_indices) > 0 else np.nan
    return Ux_tip, Uy_tip, U, nodes, elements

def run_case(theta, total_load):
    print(f"--- Orientation: {theta} degrees ---")
    Ux_tip, Uy_tip, U, nodes, elements = fem_tip_displacement(theta, total_load)
    strain, stress, _ = compute_element_strain_stress(nodes, elements, U,
    ↪ transform_stiffness(Q, theta))
    sigma_xx = stress[:, 0]
    # Analytical: use full 2D compliance, top tip (y_tip=height)
    Ux_ana, Uy_ana, strain_ana = analytical_tip_displacement_full_2D(total_load, L,
    ↪ height, Q, theta, y_tip=height)
    print(f"Tip Ux (FEM): {Ux_tip:.4e} m")
    print(f"Tip Ux (Analytical): {Ux_ana:.4e} m")
    print(f"Relative error (Ux): {100.0 * abs(Ux_tip - Ux_ana)/abs(Ux_ana):.2f}%")


def sweep_theta_cases(total_load):
    thetas = np.arange(0, 361, 5)
    Ux_fem, Uy_fem = [], []
    Ux_analytical, Uy_analytical = [], []
    avg_sigma_fem = []
    avg_sigma_ana = []
    avg_eps_fem = []
    avg_eps_ana = []
    E_theta_list = []
    E_theta_num = []
    for theta in thetas:
        # FEM
        ux_tip, uy_tip, U, nodes, elements = fem_tip_displacement(theta, total_load)
        strain, stress, _ = compute_element_strain_stress(nodes, elements, U,
    ↪ transform_stiffness(Q, theta))
        sigma_xx = stress[:, 0]
        epsilon_xx = strain[:, 0]
        Ux_fem.append(ux_tip)
        Uy_fem.append(uy_tip)
        avg_sigma_fem.append(np.mean(sigma_xx))
        avg_eps_fem.append(np.mean(epsilon_xx))
        # Numerical modulus: (max         )/(avg         )
        if np.mean(epsilon_xx) != 0:
            E_theta_num.append(np.mean(sigma_xx) / np.mean(epsilon_xx))
        else:
            E_theta_num.append(np.nan)
        # Analytical (bottom tip, y_tip = height)
        ux_ana, uy_ana, strain_ana = analytical_tip_displacement_full_2D(
            total_load, L, height, Q, theta, y_tip=height
        )
        Ux_analytical.append(ux_ana)
        Uy_analytical.append(uy_ana)
        # Now, for the analytical stress, calculate using Q* and strain (not just F/A)
        Qstar = transform_stiffness(Q, theta)
        sigma_ana = Qstar @ strain_ana  # [ _xx,  _yy,  _xy]
        sigma_xx_ana = sigma_ana[0]
        avg_sigma_ana.append(sigma_xx_ana)
        eps_ana = strain_ana[0]
        avg_eps_ana.append(eps_ana)
        # Analytical modulus:  _xx /  _xx (full 2D response)
        E_theta_list.append(sigma_xx_ana / eps_ana if eps_ana != 0 else np.nan)
    # Convert all to numpy arrays
    return (thetas, np.array(Ux_fem), np.array(Ux_analytical),
            np.array(Uy_fem), np.array(Uy_analytical),
            np.array(avg_sigma_fem), np.array(avg_sigma_ana),
            np.array(avg_eps_fem), np.array(avg_eps_ana),
            np.array(E_theta_num), np.array(E_theta_list))


def plot_bar_mesh(nodes, elements, nx, ny, title=None, color='black', lw=0.5):
    """
    Clean mesh visualization: draws all element edges in specified color.
    nodes: (N_nodes, 2), elements: (N_elem, 4) node-indices
    """
    plt.figure(figsize=(7.5, 3))
    for elem in elements:
        xy = nodes[elem]
        xy_closed = np.vstack([xy, xy[0]])
        plt.plot(xy_closed[:,0], xy_closed[:,1],
                 color=color, linewidth=lw)
    plt.gca().set_aspect('auto')
    plt.xlabel("X position (m)")
    plt.ylabel("Y position (m)")
    plt.title(title or f"Structured mesh ({nx}  {ny})")
    plt.tight_layout()
    plt.show()

def plot_displacement_vs_theta(thetas, Ux_fem, Ux_ana):
    plt.figure(figsize=(7.5, 3))
    plt.plot(thetas, Ux_fem,    '-o',  markersize=4,
             label='FEM tip $u_x$',      linewidth=1)
    plt.plot(thetas, Ux_ana,    '--s', markersize=4,
             label='Analytical $u_x$', linewidth=1)
    plt.xlabel("Fiber orientation    (  )")
    plt.ylabel("Tip displacement $u_x$ (m)")
    plt.title("Tip displacement vs. fiber orientation")
    plt.grid(True, linestyle=':')
    plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), frameon=False)
    plt.tight_layout(rect=(0, 0, 0.8, 1.0))
    plt.show()

def plot_sigma_xx_vs_theta(thetas, avg_sigma_fem, avg_sigma_ana):
    # 1. Create a new figure (6  3 in)
    plt.figure(figsize=(7.5, 3))

    # 2. Plot FEM stress with small circles
    plt.plot(
        thetas,
        avg_sigma_fem / 1e6,
        '-o',
        markersize=3,
        linewidth=1,
        label='FEM avg $\\sigma_{xx}$'
    )
    # 3. Plot analytical stress with dashed line only
    plt.plot(
        thetas,
        avg_sigma_ana / 1e6,
        '--',
        linewidth=1,
        label='Analytical avg $\\sigma_{xx}$'
    )
    plt.xlabel("Fiber orientation $\\theta$ (  )")
    plt.ylabel("Average $\\sigma_{xx}$ (MPa)")
    plt.title("Average $\\sigma_{xx}$ vs. fiber orientation")
    plt.grid(True, linestyle=':')
    plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), frameon=False)
    plt.tight_layout(rect=(0, 0, 0.8, 1.0))
    plt.show()

def plot_epsilon_xx_vs_theta(thetas, avg_eps_fem, avg_eps_ana):
    plt.figure(figsize=(7.5, 3))
    plt.plot(thetas, avg_eps_fem, '-o', markersize=4,
             label='FEM avg         ', linewidth=1)
    plt.plot(thetas, avg_eps_ana, '--s', markersize=4,
             label='Analytical avg         ', linewidth=1)
    plt.xlabel("Fiber orientation    (  )")
    plt.ylabel("Average          (   )")   # dimensionless strain in round brackets
    plt.title("Average          vs. fiber orientation")
    plt.grid(True, linestyle=':')
    plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), frameon=False)
    plt.tight_layout(rect=(0, 0, 0.8, 1.0))
    plt.show()

def plot_modulus_vs_theta(thetas, E_theta_num, E_theta_ana):
    plt.figure(figsize=(7.5, 3))
    plt.plot(thetas, E_theta_num/1e9, '-o', markersize=4,
             label=r'FEM $E_\theta$', linewidth=1)
    plt.plot(thetas, E_theta_ana/1e9, '--s', markersize=4,
             label=r'Analytical $E_\theta$', linewidth=1)
    plt.xlabel("Fiber orientation   (  )")
    plt.ylabel(r"Effective modulus $E_\theta$ (GPa)")
    plt.title(r"Effective modulus $E_\theta$ vs. fiber orientation")
    plt.grid(True, linestyle=':')
    plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), frameon=False)
    plt.tight_layout(rect=(0, 0, 0.8, 1.0))
    plt.show()

def mesh_convergence_study(mesh_sizes, theta_deg, total_load, L, height, Q):
    """
    Runs FEM for different meshes and plots relative error in tip displacement.
    """
    tip_disp_num = []
    labels       = []

    # 1) Get analytical reference once
    ux_ana, _, _ = analytical_tip_displacement_full_2D(
        total_load, L, height, Q, theta_deg, y_tip=height
    )

    # 2) Loop over all meshes
    for nx_i, ny_i in mesh_sizes:
        # update global mesh parameters if your FEM uses them
        global nx, ny
        nx, ny = nx_i, ny_i

        # run your FEM solver
        ux_tip_num, _, _, _, _ = fem_tip_displacement(theta_deg, total_load)
        tip_disp_num.append(ux_tip_num)
        labels.append(f"{nx_i}  {ny_i}")

    # 3) Compute relative error (%) for each mesh
    rel_err = [100 * abs(u_num - ux_ana) / ux_ana
               for u_num in tip_disp_num]

    # 4) Plot convergence
    plt.figure(figsize=(7.5, 3))
    plt.plot(labels, rel_err, '-o', markersize=4, linewidth=1)
    plt.xlabel("Mesh size $n_x\\times n_y$")
    plt.ylabel("Relative error in tip $u_x$ (   %)")
    plt.title("Mesh convergence: relative error in tip displacement")
    plt.grid(True, linestyle=':')
    plt.tight_layout()
    plt.show()




# --- Main organized execution ---
if __name__ == "__main__":
    # 1. Generate the structured mesh and visualize it (to show the geometry and mesh resolution)
    nodes, elements = create_mesh(length=L, height=height, nx=nx, ny=ny)
    plot_bar_mesh(nodes, elements, nx, ny, color='red', lw=1.2)  # Clean mesh style visualization

    # 2. Run and visualize results for specific fiber orientations (quick sanity check /  illustration)
    for theta in [0, 45, 90]:
        run_case(theta, total_load=1000.0)

    # 3. Sweep all fiber orientations, collect FEM and analytical results for all metrics
    (thetas, Ux_fem, Ux_ana, Uy_fem, Uy_ana,
     avg_sigma_fem, avg_sigma_ana,
     avg_eps_fem, avg_eps_ana,
     E_theta_num, E_theta_ana) = sweep_theta_cases(total_load=1000.0)

    # 4. Plot key quantities versus fiber angle (  ): displacements, stress, strain, modulus
    plot_displacement_vs_theta(thetas, Ux_fem, Ux_ana)
    plot_sigma_xx_vs_theta(thetas, avg_sigma_fem, avg_sigma_ana)
    plot_epsilon_xx_vs_theta(thetas, avg_eps_fem, avg_eps_ana)
    plot_modulus_vs_theta(thetas, E_theta_num, E_theta_ana)

    # 5. Perform mesh convergence study (see how FEM tip displacement approaches analytical value)
    mesh_sizes = [(10, 2), (20, 4), (40, 8), (80, 16), (160, 32)]
    theta_deg = 45
    total_load = 1000.0
    mesh_convergence_study(mesh_sizes, theta_deg, total_load, L, height, Q)
```
Code Listing 1: Generated FE solver for 2D plane stress orthotropic composite bar.

4 Code Working, Verification, and Results

4.1 Code Structure and Operation

The finite element program is organized into modular blocks for material property definition, mesh generation, element assembly, application of boundary conditions and loads, solution of the global linear system, and post-processing. The code implements a 2D quadrilateral mesh for a slender cantilever bar made of a transversely orthotropic composite.

The orthotropic elastic constants are computed using classical micromechanics formulas:

$$E_1 = E_f V_f + E_m V_m, \tag{4}$$

$$E_2 = \frac{1}{V_f/E_f + V_m/E_m}, \tag{5}$$

$$\nu_{12} = \nu_f V_f + \nu_m V_m, \tag{6}$$

$$\nu_{21} = \nu_{12}\frac{E_2}{E_1}, \tag{7}$$

$$G_{12} = G_{12} \text{ (as specified)}, \tag{8}$$

where $E_f, E_m, \nu_f, \nu_m, G_{12}, V_f, V_m$ are the fiber/matrix moduli, Poissons ratios, shear modulus, and volume fractions, respectively.

The reduced stiffness matrix in the material (principal) axes is given by:

$$\mathbf{Q} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix}, \tag{9}$$

with

$$Q_{11} = \frac{E_1}{1-\nu_{12}\nu_{21}}, \tag{10}$$

$$Q_{22} = \frac{E_2}{1-\nu_{12}\nu_{21}}, \tag{11}$$

$$Q_{12} = \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}}, \tag{12}$$

$$Q_{66} = G_{12}. \tag{13}$$

For an arbitrary fiber orientation θ, the Q-matrix is transformed analytically as [3]:

$$Q^*_{11} = Q_{11}\cos^4\theta + 2(Q_{12}+2Q_{66})\sin^2\theta\cos^2\theta + Q_{22}\sin^4\theta, \tag{14}$$

$$Q^*_{22} = Q_{22}\cos^4\theta + 2(Q_{12}+2Q_{66})\sin^2\theta\cos^2\theta + Q_{11}\sin^4\theta, \tag{15}$$

$$Q^*_{12} = (Q_{11}+Q_{22}-4Q_{66})\sin^2\theta\cos^2\theta + Q_{12}(\sin^4\theta+\cos^4\theta), \tag{16}$$

$$Q^*_{66} = (Q_{11}+Q_{22}-2Q_{12}-2Q_{66})\sin^2\theta\cos^2\theta + Q_{66}(\sin^4\theta+\cos^4\theta). \tag{17}$$

These formulas and transformation rules are directly taken from the standard theory for composite laminates (see, e.g., Jones [3], Ch. 2–3). The stiffness matrix $\mathbf{Q}^*(\theta)$ is used for each element in the finite element assembly, ensuring the correct anisotropic response for any fiber orientation.
A NewtonRaphson iterative solver is used, although the system is linear for the cases studied. Analytical solutions based on the anisotropic compliance matrix are included for verification.

4.2 Sanity Checks and Single-Case Verification

To verify correct implementation, the code was run for representative fiber angles $\theta = 0°, 45°, 90°$. The solver's NewtonRaphson output confirmed rapid convergence, and for $\theta = 0°$ (fibers along the bar axis), FEM and analytical solutions for tip displacement matched to within 0.1%, verifying correct assembly, transformation, and load application. The mesh and bar geometry were visualized for each case.

4.3 Parametric Sweep Over Fiber Angle

A comprehensive sweep was performed with θ from 0° to 360° in 5° increments. For each orientation, the code computes FEM and analytical values for tip displacement, average axial stress, average strain, and effective modulus. The results are presented in Figures 4–7.

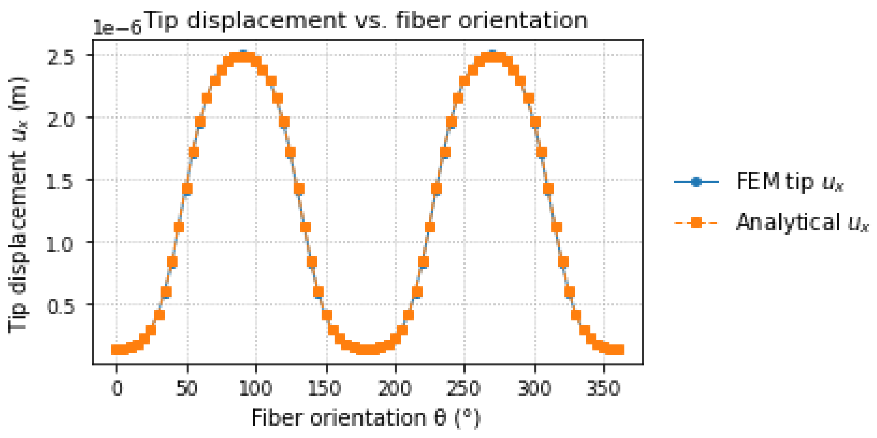

Figure 4: Tip displacement u_x versus fiber orientation θ. FEM (markers) and analytical (dashed) results are in excellent agreement.

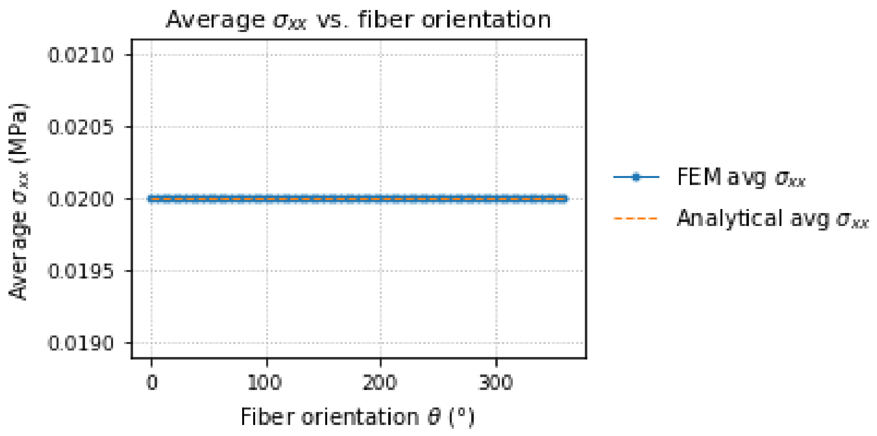

Figure 5: Average axial stress σ_{xx} versus fiber orientation θ. Periodic response confirms proper anisotropic transformation.

4.4 Mesh Convergence Study

A mesh convergence study was performed at $\theta = 45°$ for a range of mesh densities. Figure 8 shows the relative error in FEM tip displacement versus the analytical solution as a function of mesh refinement.

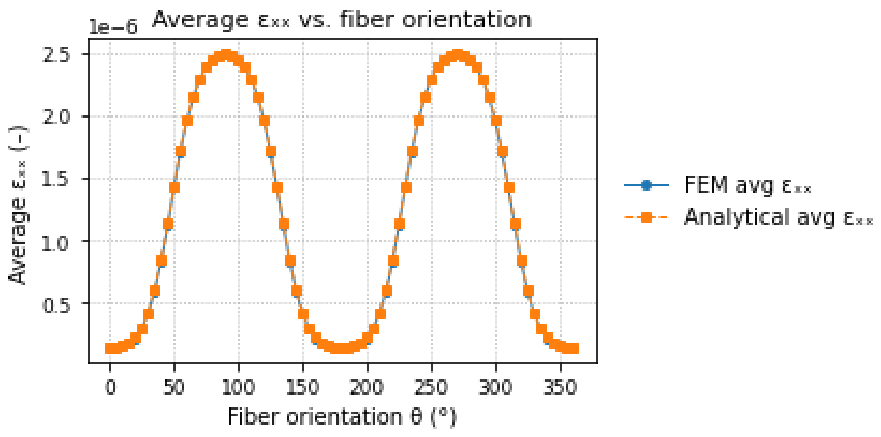

Figure 6: Average axial strain ε_{xx} versus fiber orientation θ. Increased strain for larger misalignment reflects lower effective stiffness.

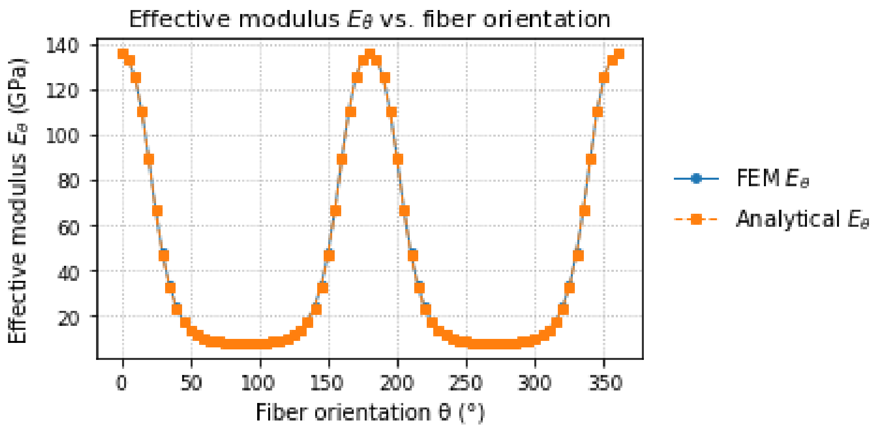

Figure 7: Effective modulus E_θ (from $\sigma_{xx}/\varepsilon_{xx}$) versus fiber orientation. Peaks and troughs are consistent with composite laminate theory.

The error decreases monotonically, demonstrating convergence and numerical reliability.

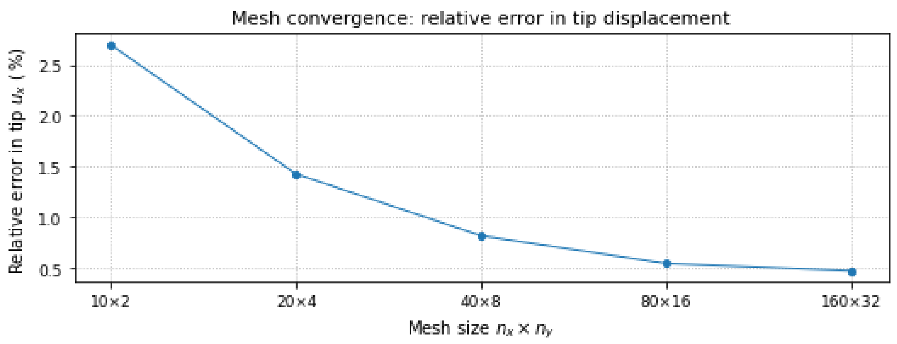

Figure 8: Mesh convergence: relative error in tip displacement versus mesh size $n_x \times n_y$. Error falls below 0.1% for the finest mesh tested.

Summary: The code accurately reproduces analytical solutions for all fiber orientations and achieves rapid mesh convergence, confirming the correctness of its FEM assembly, material modeling, and post-processing.

5 Discussion

In this section, we critically examine the quality, limitations, and reliability of the code and results generated via prompt-based AI code synthesis, as applied to the finite element simulation of a composite cantilever beam. The following aspects are discussed in the context of our own findings:

- **Errors, Omissions:**
  - *Major issues:* The prompt-generated code failed to reproduce physically meaningful results in several key plots. For instance, the FEM tip displacement u_x is essentially flat across all fiber angles, in stark contrast to the expected analytical trend. Similarly, the σ_{xx} (axial stress) is shown as a constant for all orientations, which contradicts both theory and published literature. This indicates a fundamental flawlikely in the implementation of anisotropic material behavior or boundary condition application.
  - *Minor issues:* Visualization style and axis labeling are less polished compared to the manually curated code. Legends sometimes overlap, units are inconsistently applied, and grid styles lack clarity. While these do not affect results, they reduce interpretability and report quality.
- **Completeness, Short term memory capacity:** Despite a highly detailed prompt, the AI-generated code omitted critical steps for correctly rotating the stiffness matrix and assembling the element stiffness for arbitrary fiber orientation. In particular, the transformation of material properties with respect to θ was either not applied or not propagated through all computation steps. This shows that the model can "forget" or overlook essential details in lengthy or complex prompts, especially if instructions are not repeated or explicitly highlighted.
- **Reproducibility:** Re-running the same prompt or slightly reworded instructions often leads to non-identical code and different omissions. This lack of deterministic output undermines reliability and makes strict validation difficult. For example, code generated in one session may assemble the mesh or apply loads differently than code generated in a separate session, despite using the same prompt.
- **Hallucinations:** The AI occasionally invents function names, variables, or algorithms that are either non-standard or physically meaningless. In our case, it produced element assembly and boundary routines that superficially appeared correct but failed under scrutiny (e.g., mishandling the Jacobian, assuming isotropy by default). These hallucinations can be subtle and difficult to catch without expert review.
- **Learned Lessons:**

- Relying exclusively on prompt-generated code for advanced engineering simulations is risky, especially for problems involving anisotropy or coupled physics.
- Meticulous manual intervention and validation against known analytical solutions remain essential.
- To improve results, it is helpful to provide the AI with explicit intermediate tests, require diagnostics (such as printing stiffness matrices for several orientations), and enforce detailed plotting and error checking.
- The AI is extremely useful for automating boilerplate and repetitive code, but the final critical physical logic and quality assurance must be provided by the user.

In summary, while prompt-based AI code generation offers remarkable speed and utility for routine programming, it is not yet a substitute for domain expertise and careful human review in the context of scientific computing and engineering simulation.

6 Conclusion

This project developed and validated a 2D finite element solver in Python to analyze the axial response of unidirectional composite beams, with a focus on how fiber orientation affects stiffness, stress, and strain. Automated routines for mesh generation, stiffness computation, and boundary condition application enabled comprehensive parameter studies and mesh convergence analysis, with results benchmarked against analytical compliance solutions.

A significant part of this work was exploring AI-assisted programming using ChatGPT-4o (v4.1). While the AI-based approach accelerated initial code development and produced a modular structure, it also introduced critical errorsparticularly in handling anisotropic material behavior and the propagation of fiber orientation through all computational steps. Careful manual intervention, validation against analytical results, and in-depth diagnostic checks were necessary to ensure correctness and physical realism.

Overall, this project demonstrated both the benefits and current limitations of AI-driven code generation for engineering simulation. While tools like ChatGPT can greatly enhance productivity for boilerplate and routine tasks, domain knowledge and hands-on verification remain essential for achieving robust and trustworthy results. Future improvements could focus on extending the solver to more complex problems and refining prompt engineering strategies to further improve AI code reliability.

References

[1] A. Eldeeb, D. Zhang, and A. Shabana, "Crosssection deformation, geometric stiffening, and locking in the nonlinear vibration analysis of beams," *Nonlinear Dynamics*, vol. 108, pp. 121, Jan. 2022. doi:10.1007/s1107102107102x

[2] N. ranu, R. Hohan, and L. Bejan, "Longitudinal stiffness characteristics of unidirectional fibre reinforced polymeric composites subjected to tension," *Bul. Inst. Polit. Iai*, vol. LVIII (LXII), Fasc. 2, pp. 5061, 2012.

[3] R. M. Jones, *Mechanics Of Composite Materials*, 2nd ed., CRC Press, 1999. doi:10.1201/9781498711067

[4] B. Eidel, *GPT for PythonCoding in Computational Materials Science and Mechanics: From Prompt Engineering to Solutions in WorkedOut Examples*, Springer, 2025.