OSDAG SCREENING TASK REPORT

TASK 1

Shear Force and Bending Moment Diagram Analysis

Introduction

The code is python script which generates Shear Force Diagrams (SFD) and Bending Moment Diagrams (BMD) using Python libraries. These diagrams are essential tools in structural engineering for analyzing the internal forces in beams and structural members under various loading conditions.

Theoretical Background

Shear Force
Shear force is the internal force acting perpendicular to the longitudinal axis of a structural member. It results from transverse loads and causes shearing stress in the member.

Bending Moment
Bending moment is the internal moment that causes bending in structural members. It is calculated by multiplying the force by the perpendicular distance from the point of interest.

Code Implementation

The Python implementation uses pandas for data handling and matplotlib for visualization. Below is a detailed analysis of the implementation.

Data Preparation

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the provided CSV file
file_path = r"C:\Users\\Downloads\SFS_Screening_SFDBMD.xlsx - Sheet1.csv"
data = pd.read_csv(file_path)

# Extracting data for plotting
x = data['Distance (m)']
shear_force = data['SF (kN)']
bending_moment = data['BM (kN-m)']

```

The code begins by importing necessary libraries which are pandas and matplotlib and loading the data from a CSV file. The distance, shear force, and bending moment values are extracted for visualization.

Plot Configuration

```
fig, axes = plt.subplots(2, 1, figsize=(10, 8))
```

A figure with two vertically stacked subplots is created - one for the SFD and one for the BMD.

Shear Force Diagram Implementation

```
# --- Shear Force Diagram (SFD) ---
axes[0].plot(x, shear_force, drawstyle='steps-post', color='b', linewidth=1.5, label='Shear Force')

# Vertical lines showing step changes between points
for i in range(len(x) - 1):
    axes[0].vlines(x[i+1], shear_force[i], shear_force[i+1], colors='blue', linewidth=1.5, linestyles='solid')

# Add dashed vertical lines from each point to the x-axis
for xi, sf in zip(x, shear_force):
    axes[0].vlines(xi, 0, sf, colors='blue', linestyles='solid', linewidth=1)

```

The SFD is plotted using a step function to accurately represent the discontinuous nature of shear force. Vertical lines are added to emphasize these step changes and to connect each point to the x-axis.

### Maximum Value Annotation for SFD

```python
# Printing max values in the plots
dist_value = 6
sfd_value = data.loc[data['Distance (m)'] == dist_value, 'SF (kN)'].values[0]
axes[0].text(dist_value, sfd_value, f"Max: {sfd_value:.2f} kN-m",
          fontsize=10, color='black', ha='left', va='bottom', fontweight='bold')
```

The maximum shear force value is identified at a distance of 6 meters and annotated on the plot.

SFD Formatting

```
axes[0].axhline(0, color='black', linewidth=0.8)
axes[0].set_xlabel("Distance (m)")
axes[0].set_ylabel("Shear Force (kN)")
axes[0].set_title("Shear Force Diagram (SFD)")
```

```
axes[0].legend()
axes[0].grid(True, linestyle='dashed', alpha=0.7)
```

The SFD plot is formatted with appropriate labels, title, legend, and grid.

Bending Moment Diagram Implementation

```
# --- Bending Moment Diagram (BMD) ---
axes[1].plot(x, bending_moment, color='r', linewidth=1.5, label='Bending Moment')

# Add dashed vertical lines from each point to the x-axis
for xi, bm in zip(x, bending_moment):
    axes[1].vlines(xi, 0, bm, colors='red', linestyles='solid', linewidth=1)
```

The BMD is plotted as a continuous line, reflecting the continuous nature of bending moment along the beam. Vertical lines connect each point to the x-axis.

### Maximum Value Annotations for BMD

```python
dist_value = 2
bm_value = data.loc[data['Distance (m)'] == dist_value, 'BM (kN-m)'].values[0]

dist_value1=6
bm_value1 = data.loc[data['Distance (m)'] == dist_value1, 'BM (kN-m)'].values[0]

axes[1].text(dist_value, bm_value, f"Max: {bm_value} kN-m",
             fontsize=10, color='black', ha='left', va='bottom', fontweight='bold')
axes[1].text(dist_value1, bm_value1, f"Max: {bm_value1} kN-m",
             fontsize=10, color='black', ha='left', va='bottom', fontweight='bold')
```

Two significant bending moment values are identified at distances of 2 and 6 meters and annotated on the plot.

BMD Formatting

```
axes[1].axhline(0, color='black', linewidth=0.8)
axes[1].set_xlabel("Distance (m)")
axes[1].set_ylabel("Bending Moment (kN-m)")
axes[1].set_title("Bending Moment Diagram (BMD)")
axes[1].legend()
axes[1].grid(True, linestyle='dashed', alpha=0.7)
```

The BMD plot is formatted with appropriate labels, title, legend, and grid.

Final Display

```
plt.tight_layout()
plt.show()
```

The plots are displayed with appropriate spacing using `tight_layout()`.

Interpretation of SFD

The Shear Force Diagram shows:
- Where the shear force is maximum
- Points where shear force changes sign (zero shear)
- Areas where the beam experiences high shear stress

Interpretation of BMD

The Bending Moment Diagram shows:
- Points of maximum positive and negative bending moments.
- Locations where the beam experiences maximum bending stress
- Points of contraflexure (where bending moment is zero)

Design Implications

1. Critical Sections: The maximum values in both diagrams identify critical sections where the beam needs to be designed for maximum stress.
2. Material Selection: Based on the maximum shear and bending values, appropriate materials can be selected.

Conclusion

The SFD and BMD provide critical information for structural analysis and design. The Python implementation presented here offers an efficient and accurate way to generate these diagrams from raw data using pyplots.

TASK 2

PythonOCC Code Analysis: Laced Compound Column

Introduction
The code is a Python script designed to generate a 3D model of a laced compound column using the Open Cascade Technology (OCCT) library through its Python wrapper (PythonOCC).

A laced compound column is a structural element consisting of two parallel I-sections connected by a system of end battens and diagonal lacing plates. This type of structure is commonly used in steel construction to provide increased load-bearing capacity while maintaining structural efficiency.

Libraries and Imports

The script uses the following libraries:

- **Open Cascade Technology (OCCT)**: A powerful CAD kernel for 3D modeling and computation.
    - gp_Vec, gp_Trsf, gp_Pnt, gp_Dir, gp_Ax1: Geometric primitives and transformations.
    - BRepPrimAPI_MakeBox: To create box shapes for components.
    - BRepAlgoAPI_Fuse: To perform Boolean operations (fusing shapes together).
    - BRepBuilderAPI_Transform: To apply geometric transformations to shapes.
    - BRepBuilderAPI_MakeEdge, BRepBuilderAPI_MakeWire, BRepBuilderAPI_MakeFace: To create more complex geometries.
    - BRepPrimAPI_MakePrism: To extrude 2D faces into 3D solids.
    - init_display: For visualizing the 3D model.

Functions

1. Create_i_section

This function creates an I-section (I-beam) shape.

- **Parameters**:
    - length: Length of the I-section.
    - width: Width of the I-section.
    - depth: Depth (height) of the I-section.
    - flange_thickness: Thickness of the flanges.
    - web_thickness: Thickness of the web.
- **Logic**:
    - The I-section is created by combining three boxes: bottom flange, top flange, and web.
    - The top flange is translated to the top of the section.
    - The web is positioned centrally between the two flanges.
    - All three components are fused into a single solid.

2. create_end_batten

This function creates the end batten plates that connect the two I-sections at the top and bottom.

- **Parameters**:
    - length: Length of the batten.
    - width: Width of the batten (matches I-section width).
    - depth: Thickness of the batten plate.
    - column_distance: Distance between the two I-sections.
- **Logic**:
    - Creates a rectangular plate spanning between the two columns.
- **Code**

```
def create_end_batten(length, width, depth, column_distance):
    batten = BRepPrimAPI_MakeBox(length, column_distance + width,
depth).Shape()
    return batten
```

3. create_straight_lace
This function creates horizontal lace plates.

- **Parameters**:
  - width: Width of the lace plate.
  - thickness: Thickness of the lace plate.
  - diagonal_length: Length of the lace plate.
- **Logic**:
  - Creates a box representing the horizontal connecting plate.
- **Code**

```
def create_straight_lace(width, thickness, length):
 # Create a horizontal lace plate
    lace = BRepPrimAPI_MakeBox(length, width, thickness).Shape()
    return lace
```

4. create_parallelogram_face
This function creates diagonal lacing plates with a parallelogram shape.

- **Parameters**:
  - p1, p2, p3: Three points defining the parallelogram.
  - thickness: Thickness of the lace plate.
- **Logic**:
  - Calculates the fourth point of the parallelogram based on the first three.
  - Creates edges connecting all four points.
  - Creates a wire from these edges.
  - Makes a face from the wire.
  - Extrudes the face to the specified thickness to create a solid.
- **Code**

```
def create_parallelogram_face(p1, p2, p3, thickness):
# Calculate the fourth point (Top-right)
    vec1 = gp_Vec(p1, p2)  # Base vector
    p4 = gp_Pnt(p3.XYZ() + vec1.XYZ())  # Top-right = Top-left + base 
vector
  # Create edges
  edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
  edge2 = BRepBuilderAPI_MakeEdge(p2, p4).Edge()
  edge3 = BRepBuilderAPI_MakeEdge(p4, p3).Edge()
  edge4 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

  # Make wire
  wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3, edge4).Wire()
    # Make face
  face = BRepBuilderAPI_MakeFace(wire).Face()

    # Extrude the face into a solid
  vec = gp_Vec(0, 0, thickness)
  prism = BRepPrimAPI_MakePrism(face, vec).Shape()
  return prism
```

Main Execution Block

Parameters

The script defines several parameters that control the dimensions of the laced compound column:

- column_length: Total length of the column (6100.0 mm).
- i_section_width: Width of the I-section (100.0 mm).
- i_section_height: Height of the I-section (200.0 mm).
- flange_thickness: Thickness of the I-section flanges (10.0 mm).
- web_thickness: Thickness of the I-section web (5.0 mm).
- column_distance: Distance between the two parallel I-sections (450.0 mm).
- batten_depth: Depth of the end battens (300.0 mm).
- batten_thickness: Thickness of the end battens (10.0 mm).
- lace_width: Width of the lacing plates (100.0 mm).
- lace_thickness: Thickness of the lacing plates (8.0 mm).

Construction Process

1. **Create I-Sections**:
   - Two identical I-sections are created.
   - The second I-section is translated to create the parallel configuration.
2. **Create End Battens**:
   - Four end battens are created (top-left, top-right, bottom-left, bottom-right).
   - Each is positioned using translation transformations.
3. **Combine Structure**:
   - The I-sections and end battens are fused into a single solid.
4. **Create Diagonal Laces**:
   - The code calculates the number and spacing of diagonal laces.
   - A loop creates multiple sets of diagonal laces along the column length.
   - For each position, it creates:
     - Forward-leaning diagonal lace (/)
     - Backward-leaning diagonal lace (\)
     - Two horizontal connecting laces
5. **Final Additions**:
   - Additional diagonal laces are added at the end of the structure.
   - All components are fused into the final composite structure.
6. **Display**:
   - The completed structure is displayed using the PythonOCC visualization system.

Calculations and Formulas

- **Lace Spacing**: The distance between diagonal laces is equal to the I-section height.
- **Number of Laces**: Calculated based on available space between end battens.
- **Lace Positioning**: Precise geometric points are calculated to ensure proper alignment of the diagonal and horizontal laces.
- **Rotations**: Horizontal laces are rotated 90 degrees to achieve proper orientation.

Units

- All dimensions are in millimeters (mm).
- Angles are in degrees and converted to radians when needed using math.radians().

Conclusion

The laced compound column generated by this code represents a classic structural engineering solution where two parallel members are connected by a system of diagonal bracing.