REPORT DOCUMENTATION

Submitted By: ARNAW

Branch: CSE

Semester: 6^{th}

College Name: Dumka Engineering College Dumka, Jharkhand

Registration Number: 22040445002

Student Id: 239031001046

Youtube Video Link: https://youtu.be/zsAvoGNwhDE

Github Repository Link: https://github.com/Arnaw17/Fossee-Python-Occ-Pyplot.git

TASK-1 CODE EXPLAINATION

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


def plot_sfd_bmd(excel_path: str="C:/Users/arnaw/Downloads/SFS_Screening_SFDBMD.xlsx",
sheet_name: str = 'Sheet1'):
    """
    Plots the Shear Force Diagram (SFD) and Bending Moment Diagram (BMD)
    from an Excel sheet.

    Parameters:
        excel_path (str): Path to the Excel file.
        sheet_name (str): Sheet name containing the data (default is 'Sheet1').
    """
    # Read data
    df = pd.read_excel(excel_path, sheet_name='Sheet1')

    # Extract columns
    distance = df['Distance (m)']
```

```
    shear_force = df['SF (kN)']
    bending_moment = df['BM (kN-m)']

    # Create subplots for SFD and BMD
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8), sharex=True)

    # Plot Shear Force Diagram (SFD)
    ax1.plot(distance, shear_force, color='blue', marker='o')
    ax1.set_title('Shear Force Diagram (SFD)')
    ax1.set_ylabel('Shear Force (kN)')
    ax1.grid(True)

    # Plot Bending Moment Diagram (BMD)
    ax2.plot(distance, bending_moment, color='red', marker='o')
    ax2.set_title('Bending Moment Diagram (BMD)')
    ax2.set_xlabel('Distance (m)')
    ax2.set_ylabel('Bending Moment (kN·m)')
    ax2.grid(True)

    # Adjust layout
    plt.tight_layout()
    plt.show()

    #Calling the function
print("Plotting started...")
plot_sfd_bmd()
```

Full Explanation of the `plot_sfd_bmd()` Code

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

These are library imports:

- `pandas` is used to **read and handle Excel files** like tables.
- `matplotlib.pyplot` is used to **plot graphs** (SFD and BMD in this case).

Function Definition

```
def plot_sfd_bmd(excel_path: str =
r"C:\Users\arnaw\Downloads\SFS_Screening_SFDBMD.xlsx",
                 sheet_name: str = 'Sheet1'):
```

- This defines a function called `plot_sfd_bmd`.
- It takes:
    - `excel_path`: The path to your Excel file (default is your file).
    - `sheet_name`: The sheet inside Excel to use (defaults to `'Sheet1'`).
- `r""` makes it a **raw string**, so Windows paths work without escaping `\`.

Confirmation Message

```
print("Plotting started...")
```

- Just prints to the terminal to confirm the function actually started running.

Reading Excel Data

```
df = pd.read_excel(excel_path, sheet_name=sheet_name)
```

- Reads the Excel sheet and puts it into a **DataFrame** (like an in-memory Excel table).
- `df` will now hold your values like `Distance (m)`, `SF (kN)`, and `BM (kN-m)`.

Extracting Columns

```
distance = df['Distance (m)']
shear_force = df['SF (kN)']
bending_moment = df['BM (kN-m)']
```

- Pulls out the important columns:
  - `distance`: Position along the beam.
  - `shear_force`: Shear force values.
  - `bending_moment`: Bending moment values.

These are used to make the plots.

Creating Two Subplots

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8), sharex=True)
```

- Creates **two vertical plots**:
  - `ax1` for the Shear Force Diagram.
  - `ax2` for the Bending Moment Diagram.
- `figsize=(12, 8)`: Sets the size of the window.
- `sharex=True`: Makes both plots share the same X-axis (distance).

Plotting the Shear Force Diagram (SFD)

```
ax1.plot(distance, shear_force, color='blue', marker='o')
ax1.set_title('Shear Force Diagram (SFD)')
ax1.set_ylabel('Shear Force (kN)')
ax1.grid(True)
```

- Plots shear force values in blue with circle markers.
- Adds a title and labels.
- Turns on a grid for easier reading.

Plotting the Bending Moment Diagram (BMD)

```
ax2.plot(distance, bending_moment, color='red', marker='o')
ax2.set_title('Bending Moment Diagram (BMD)')
ax2.set_xlabel('Distance (m)')
ax2.set_ylabel('Bending Moment (kN·m)')
ax2.grid(True)
```

- Same as above but for bending moments, using red.
- X-label is only added here to avoid repeating it in both subplots.

Finalizing and Showing the Plot

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

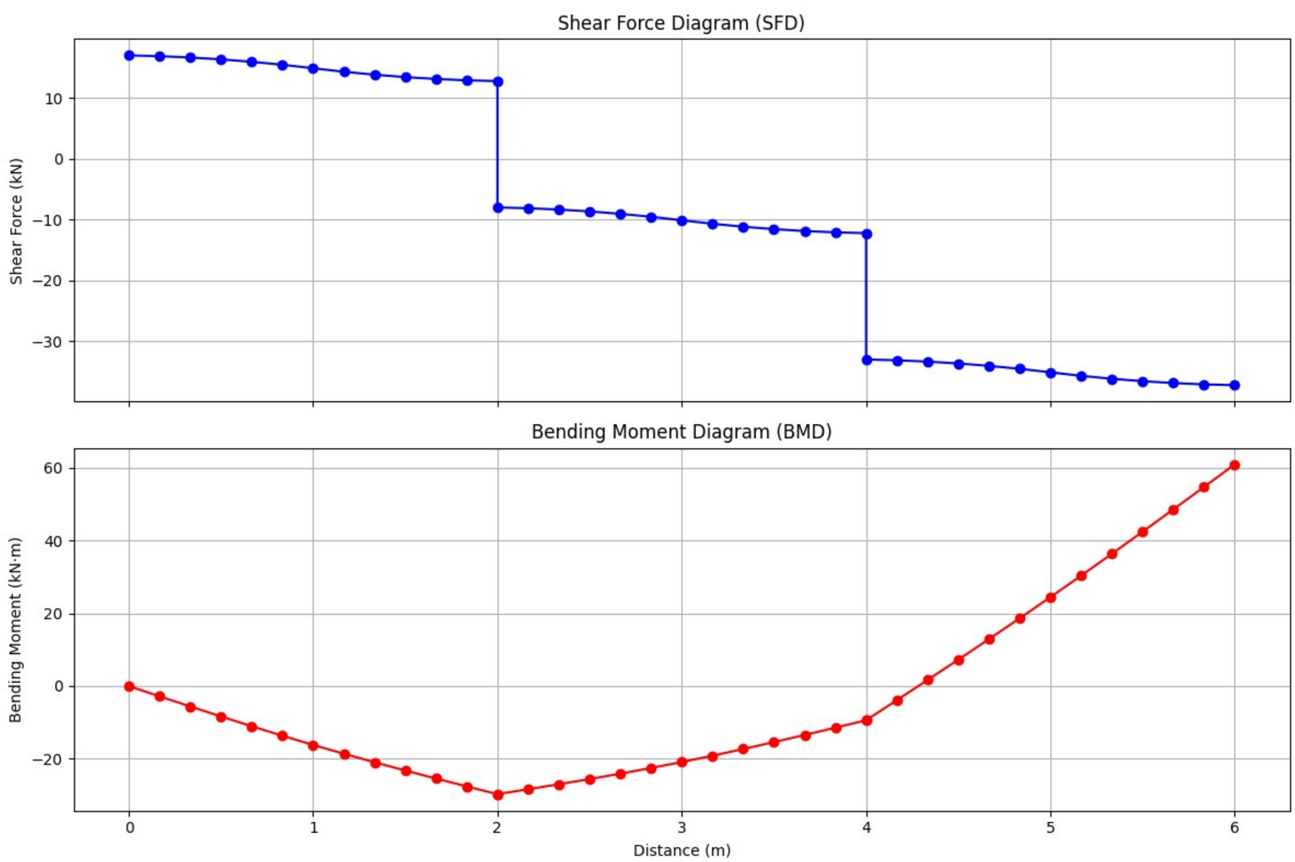
- `tight_layout()` fixes any overlapping labels and spacing.
- `show()` displays the plot window.

Outside the Function

```
plot_sfd_bmd()
```

- This **calls the function**, making everything above actually happen.

OUTPUT: - Plotting started...



TASK-2 CODE EXPLAINATION

```python
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.gp import gp_Pnt, gp_Dir, gp_Ax1
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Core.gp import gp_Trsf
from OCC.Display.SimpleGui import init_display
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.TopoDS import TopoDS_Shape
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeSolid
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox

# ================================
# Parameters (mm)
# ================================
column_height = 6100
column_spacing = 450
ismb_width = 100
ismb_depth = 200
plate_thickness = 10
plate_width = 430
plate_height = 300
lace_width = 100
lace_thickness = 8
lace_pitch = 450

# ================================
# Geometry Creation
# ================================

def create_ismb_column(origin_x):
    """Creates one ISMB section as a box (simplified shape)"""
    p1 = gp_Pnt(origin_x, 0, 0)
    return BRepPrimAPI_MakeBox(p1, ismb_width, ismb_depth, column_height).Shape()

def create_end_plate(z_pos):
    """Creates top or bottom plate"""
    p = gp_Pnt(-plate_width/2 + ismb_width/2, -plate_height/2 + ismb_depth/2, z_pos)
    return BRepPrimAPI_MakeBox(p, plate_width, plate_height, plate_thickness).Shape()

def create_lace(x_start, x_end, z_start, z_end):
    """Creates a diagonal lace bar between two ISMBs"""
    length = ((x_end - x_start)**2 + (z_end - z_start)**2)**0.5
    lace = BRepPrimAPI_MakeBox(gp_Pnt(0, 0, 0), lace_thickness, lace_width,
length).Shape()

    # Rotate and translate lace to fit between columns
    trsf = gp_Trsf()
    angle = gp_Dir(x_end - x_start, 0, z_end - z_start)
    axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 1, 0))
    trsf.SetRotation(axis, angle.Angle(gp_Dir(0, 0, 1)))

    t = BRepBuilderAPI_Transform(lace, trsf)
```

```python
    trsf_move = gp_Trsf()
    trsf_move.SetTranslation(gp_Pnt(0, 0, 0), gp_Pnt(x_start, 0, z_start))
    final = BRepBuilderAPI_Transform(t.Shape(), trsf_move)
    return final.Shape()

# ================================
# Main Assembly
# ================================
def build_column():
    shapes = []

    # Left and Right ISMBs
    shapes.append(create_ismb_column(0))
    shapes.append(create_ismb_column(column_spacing))

    # Top and Bottom Plates
    shapes.append(create_end_plate(0))
    shapes.append(create_end_plate(column_height - plate_thickness))

    # Lacing
    num_laces = int(column_height // lace_pitch)
    for i in range(num_laces):
        z1 = i * lace_pitch
        z2 = (i + 1) * lace_pitch
        # Diagonal lacing from left to right and vice versa
        shapes.append(create_lace(0, column_spacing, z1, z2))
        shapes.append(create_lace(column_spacing, 0, z1, z2))

    return shapes

# ================================
# Display
# ================================
if __name__ == "__main__":
    display, start_display, add_menu, add_function_to_menu = init_display()
    column_parts = build_column()
    for shape in column_parts:
        display.DisplayShape(shape, update=True)
    start_display()
```

---

1. Imports & Setup

```
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.gp import gp_Pnt, gp_Dir, gp_Ax1
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Core.gp import gp_Trsf
from OCC.Display.SimpleGui import init_display
```

- You're importing all the necessary functions from the OpenCASCADE toolkit.
- `gp_Pnt`, `gp_Dir`, and `gp_Ax1`: geometric primitives like points, directions, and axes.
- `BRepPrimAPI_MakeBox`: creates boxes (used to approximate I-beams and plates).

- `init_display`: launches a GUI window to show your 3D model.

---

2. Parameters

```
column_height = 6100  # Height of the column
column_spacing = 450  # Distance between the two I-beams
ismb_width = 100      # Width of one ISMB (simplified as a box)
ismb_depth = 200      # Depth of the ISMB
...
```

You define all geometric parameters (in mm) for the parts of the column:

- Widths, heights, thicknesses for beams, plates, and lacing.

---

3. Component Creation Functions

✅ ISMB (I-Beam, simplified as a box)

```
def create_ismb_column(origin_x):
    return BRepPrimAPI_MakeBox(gp_Pnt(origin_x, 0, 0), ismb_width, ismb_depth,
column_height).Shape()
```

This creates a vertical box at `origin_x`, representing one I-beam.

End Plates

```
def create_end_plate(z_pos):
    p = gp_Pnt(-plate_width/2 + ismb_width/2, -plate_height/2 + ismb_depth/2, z_pos)
    return BRepPrimAPI_MakeBox(p, plate_width, plate_height, plate_thickness).Shape()
```

This places a rectangular plate at the top or bottom (`z_pos`) of the column, centered between the two ISMBs.

Diagonal Lacing Bars

```
def create_lace(x_start, x_end, z_start, z_end):
```

This function:

- Creates a **diagonal box** (lace) between two points.
- Calculates the **true diagonal length** between two columns.
- Rotates and moves the lace using transformations so it fits between the points.

Transformation steps:

1. `gp_Trsf().SetRotation(...)`: Rotates the lace around Y-axis.
2. `gp_Trsf().SetTranslation(...)`: Moves the rotated shape to its position.

4. Main Assembly

```
def build_column():
    shapes = []
    shapes.append(create_ismb_column(0))  # Left column
    shapes.append(create_ismb_column(column_spacing))  # Right column
```

- Adds both ISMB columns.
- Adds top and bottom end plates.
- Adds a set of **diagonal laces**, in both directions, from bottom to top based on `lace_pitch`.

5. Display the Model

```
if __name__ == "__main__":
    display, start_display, ... = init_display()
    column_parts = build_column()
    for shape in column_parts:
        display.DisplayShape(shape, update=True)
    start_display()
```

- Launches a simple OpenCASCADE GUI.
- Calls the function to build the column and display each component.
- `start_display()` opens the interactive viewer window and **keeps it open** until you close it manually.

Output

Once run, you'll see a 3D model of:

- Two vertical boxes (columns),
- Plates on top and bottom,
- Criss-crossing diagonal laces.

To run the python program Step by Step Guide:

First: Install Anaconda if not install https://www.anaconda.com/download

Second: Install basic-miktex if not https://miktex.org/download

Third: Install Python 3.10.12 for running OCC if not: https://www.python.org/downloads/release/python-31012/

Fourth: first create an Environment by this Command: conda create --name=pyoccenv python=3.10

Fifth: type this command: activate pyoccenv

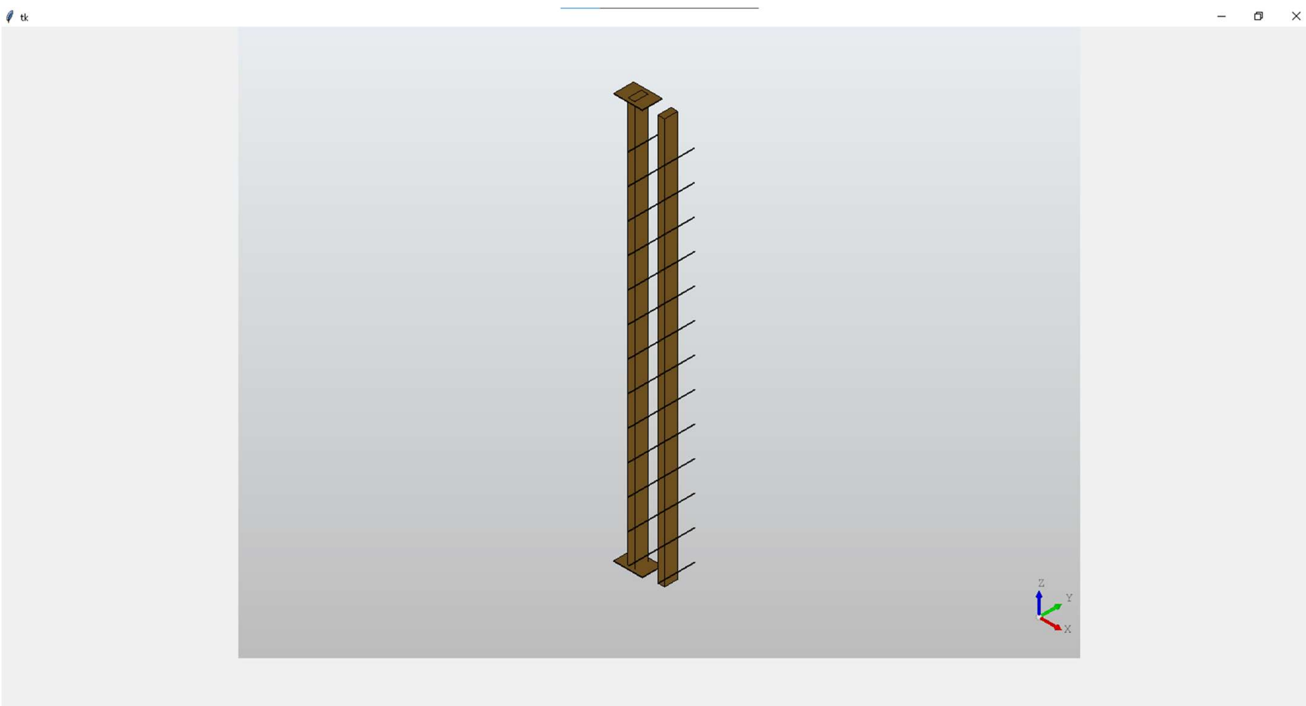
Sixth: type this command: conda install -c conda-forge pythonocc-core=7.8.1.1

Seventh: Now from cd command go to the directory where your program is save

Eight: Now type “Python File_name.py”

Now It’s Going to run

Note: This Method is for Python-occ Task which is task 2 in “PythonOCC and PyPlot” which is a task of [redacted] first program can run natively on any code editor with python install and the respective module using to plot the diagram.



<u>END</u>