FOSSEE Winter Internship Report

On

Development of Purlin Module for Osdag

Submitted by

Debayan Ghosh

3rd Year B.Tech Student, Department of Computer Science and Engineering

(Artificial Intelligence & Machine Learning)

Institute of Engineering & Management

Kolkata

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

January 15, 2025

Acknowledgments

- I would like to sincerely thank the **Osdag** and **FOSSEE** teams at **IIT Bombay** for giving me this wonderful opportunity. Through this internship, I was able to learn new things, and it was a truly great experience for me.

- A special thanks to our mentors, **Parth Karia**, **Ajinkya Dahale**, and **Ajmal Babu M. S.**, project staff at the Osdag team, for their constant support during the internship and for teaching us new skills that have greatly enriched our learning experience.

- I am deeply grateful to **Prof. Siddhartha Ghosh**, Principal Investigator (PI) of the Osdag Project and Professor in the Department of Civil Engineering at IIT Bombay, for his invaluable guidance and support throughout the project.

- I extend my sincere thanks to **Prof. Kannan M. Moudgalya**, Principal Investigator (PI) of the FOSSEE Project, Department of Chemical Engineering, IIT Bombay, for providing me with this opportunity and for his continuous encouragement.

- My special thanks to FOSSEE Managers, **Ms. Usha Viswanathan** and **Ms. Vineeta Parmar**, and their entire team for their support and coordination, which greatly facilitated the project's progress.

- I extend my sincere thanks to the **National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India** for their support and their role in facilitating this project

- I extend my heartfelt thanks to my colleagues **Shubham** and **Koustav** for their help and collaboration throughout this internship.

- I also want to express my gratitude to my professors from the **Computer Science and Engineering(Artificial Intelligence & Machine Learning)** Department at **IEM, Kolkata** for their guidance and support during my academic journey.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	**Resource**	**For Students/Researchers**	**For Institutions**
		Audio-Video e-content	
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
		Digital Content Access	
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
		Hands-on Learning	
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
		E-Governance	
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
		Tracking and Research Tools	
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

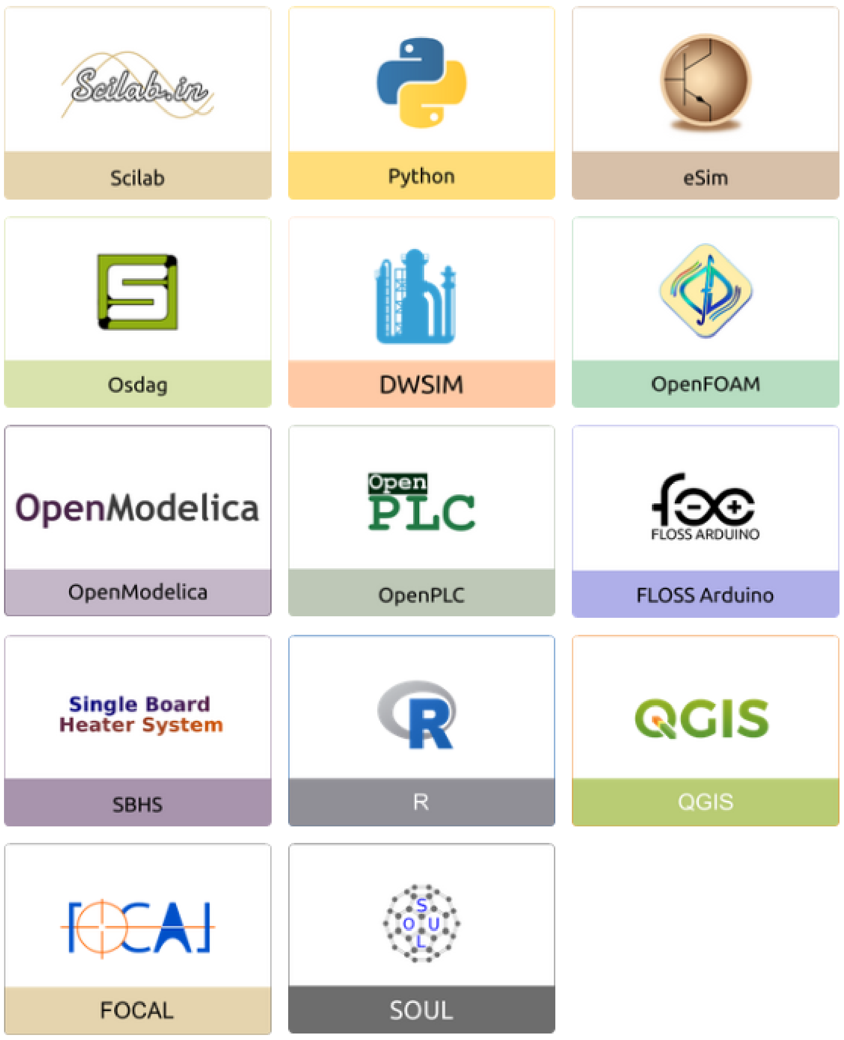


Figure 1.1: FOSSEE Projects and Activities

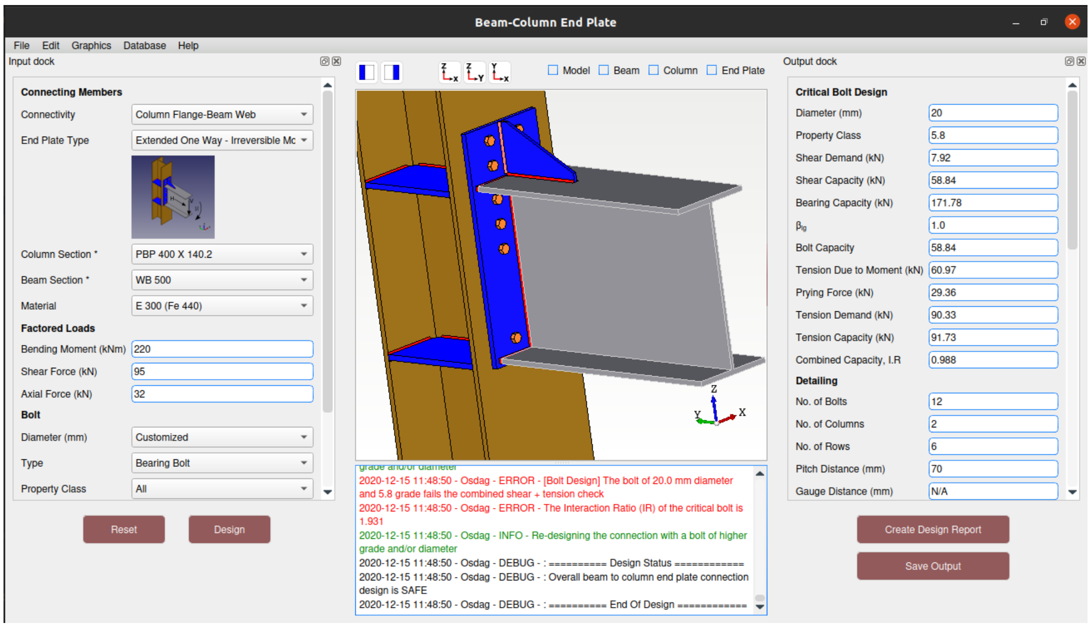
1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.



Connecting Members

Connectivity	Column Flange-Beam Web
End Plate Type	Extended One Way - Irreversible Mc
Column Section *	PBP 400 X 140.2
Beam Section *	WB 500
Material	E 300 (Fe 440)

Factored Loads

Bending Moment (kNm)	220
Shear Force (kN)	95
Axial Force (kN)	32

Bolt

Diameter (mm)	Customized
Type	Bearing Bolt
Property Class	All

Critical Bolt Design

Diameter (mm)	20
Property Class	5.8
Shear Demand (kN)	7.92
Shear Capacity (kN)	58.84
Bearing Capacity (kN)	171.78
β_{lg}	1.0
Bolt Capacity	58.84
Tension Due to Moment (kN)	60.97
Prying Force (kN)	29.36
Tension Demand (kN)	90.33
Tension Capacity (kN)	91.73
Combined Capacity, I.R	0.988

Detailing

No. of Bolts	12
No. of Columns	2
No. of Rows	6
Pitch Distance (mm)	70
Gauge Distance (mm)	N/A

```
grade and/or diameter
2020-12-15 11:48:50 - Osdag - ERROR - [Bolt Design] The bolt of 20.0 mm diameter
and 5.8 grade fails the combined shear + tension check
2020-12-15 11:48:50 - Osdag - ERROR - The Interaction Ratio (IR) of the critical bolt is
1.931
2020-12-15 11:48:50 - Osdag - INFO - Re-designing the connection with a bolt of higher
grade and/or diameter
2020-12-15 11:48:50 - Osdag - DEBUG - : ========== Design Status ============
2020-12-15 11:48:50 - Osdag - DEBUG - : Overall beam to column end plate connection
design is SAFE
2020-12-15 11:48:50 - Osdag - DEBUG - : ========== End Of Design ============
```

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The Screening Task consisted of performing certain tasks on a PyQt5 based prism viewer application. The following tasks had to be performed on the application:

- Prepare unit tests to check if the application is working correctly (for surface area and volume)
- Package the application for Conda by writing a Conda recipe
- Submit a report on the work done during the screening task

The specific deliverables for this task were:

- Unit test code, setup.py or pyproject.toml for PIP
- Conda Recipe
- Report consisting of:
  - Methodology involved
  - Elaborate on the tests chosen
  - Challenges while building on Conda
  - References

2.2 Tasks Done

2.2.1 Development of Unit Tests

The prism_calculator.py contains helper functions to calculate the surface area and the volume of the rectangular prism. I wrote unit tests test these functions and ensure they conform to following:

- Functions can be called and behave normally.
- Functions return correct values for integer value parameters.
- Functions return correct values for floating value parameters.

The tests were implemented using unitest package in python to use its powerful features and ensure that the tests are written in a standard format. In total, 6 tests were written and line coverage for the file prism_calculator.py was 100%.

2.2.2 Python Code for Unit tests

```python
from prism_calculator import PrismCalculator
import unittest


class TestPrismCalculator(unittest.TestCase):
    def test_surface_area_exists_and_executes(self):
        try:
            PrismCalculator.surface_area(1, 1, 1)
        except NameError:
            self.fail("PrismCalculator.surface_area() does not exist.")
        except TypeError as e:
            self.fail(f"PrismCalculator.surface_area() exists but raised a TypeError: {e}")
        except Exception as e:
            self.fail(f"PrismCalculator.surface_area() raised an unexpected error: {e}")
        else:
            print("PrismCalculator.surface_area() executed
               successfully.")

    def test_volume_exists_and_executes(self):
        try:
            PrismCalculator.volume(1, 1, 1)
        except NameError:
            self.fail("PrismCalculator.volume() does not exist.")
        except TypeError as e:
            self.fail(f"PrismCalculator.volume() exists but
               raised a TypeError: {e}")
        except Exception as e:
            self.fail(f"PrismCalculator.volume() raised an
               unexpected error: {e}")
        else:
            print("PrismCalculator.volume() executed successfully
               .")

    def test_surface_area_integer(self):
        """
        Tests surface_area method of the PrismCalculator class
           with integer values.
        """
        self.assertEqual(PrismCalculator.surface_area(2, 2, 3),
           32)
        self.assertNotEqual(PrismCalculator.surface_area(2, 2, 3)
           , 27)

    def test_surface_area_float(self):
        """
        Tests surface_area method of the PrismCalculator class
           with float values.
        """
        self.assertEqual(PrismCalculator.surface_area
           (2.0,2.0,3.0), 32.0)
        self.assertNotEqual(PrismCalculator.surface_area
           (2.0,2.0,3.0), 27.0)

    def test_volume_integer(self):
        """
        Tests volume method of the PrismCalculator class with
           integer values.
        """
        self.assertEqual(PrismCalculator.volume(2, 2, 3), 12)
        self.assertNotEqual(PrismCalculator.volume(2, 2, 3), 13)

    def test_volume_float(self):
        """
        Tests volume method of the PrismCalculator class with
           float values.
        """
        self.assertEqual(PrismCalculator.volume(2.0, 2.0, 3.0),
           12.0)
        self.assertNotEqual(PrismCalculator.volume(2.0, 2.0, 3.0)
           , 13.0)

if __name__ == '__main__':
    unittest.main()
```

2.2.3 Conda Packaging using Conda Recipe

Upon going through the files provided in the Screening task, I realized that the main dependencies of the project are numpy, PyQt5 and PythonOCC. To streamline building the project, I wrote a a pyproject.toml file, specifying the project's dependencies and build configuration.

To ensure that the project can be packaged for Conda, I also wrote a Conda recipe using a meta.yaml file to define the application's dependencies and metadata. It includes all the dependencies and build instructions along with other instructions required for building the conda package.

2.2.4 pyproject.toml - For PIP

```
[build-system]
requires = ["setuptools>=61.0", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "reactangular-prism-viewer"
version = "0.1.0"
description = "This project involves creating a desktop
   application using PyQt5 that allows users to view and analyze
   3D models of rectangular prisms. The application retrieves
   prism dimensions from a SQLite database, calculates surface
   area and volume, and displays a 3D CAD model using PythonOCC.
   "
readme = "README.md"
requires-python = ">=3.6"
dependencies = [
    "numpy>=2.1.2",
    "PyQt5>=5.15.11",
    "PyQt5-Qt5>=5.15.2",
    "PyQt5_sip>=12.15.0",
]

[project.urls]
Source = "https://github.com/OsdagScreeningTasks/
   RectangularPrismViewer"
```

2.2.5 meta.yaml - For Conda Packaging

```
package:
  name: reactangular-prism-viewer
  version: "0.1.0"

source:
  path: ..

build:
    script: {{ PYTHON }} -m pip install --no-deps --ignore-
       installed .

requirements:
  build:
    - python ==3.10
    - pip
  host:
    - python ==3.10
    - pip
  run:
    - python ==3.10
    - numpy >=1.21.2
    - pyqt >=5.15.10
    - conda-forge::pythonocc-core

about:
  home: "https://github.com/OsdagScreeningTasks/
     RectangularPrismViewer"
  summary: "A PyQt5 desktop application that allows users to view
      and analyze 3D models of rectangular prisms."
  description: |
    This project involves creating a desktop application using
       PyQt5 that allows users to view and analyze 3D models of
       rectangular prisms. The application retrieves prism
       dimensions from a SQLite database, calculates surface area
        and volume, and displays a 3D CAD model using PythonOCC.
```

Chapter 3

Internship Task 1: Development of the Purlin Module

3.1 Task 1: Problem Statement

Development of the Purlin Module in Osdag and make the necessary changes to integrate it into the software.

3.2 Task 1: Tasks Done

Worked on developing an algorithm for the Purlin module and then implementing this after going through the code written in the flexure module and other module's and consultation with mentors and colleagues. Further work was done to ensure that the ensure that the developed module didnt cause any errors and integrated well into the current software.

3.2.1 Input Dock

The input dock was designed to take inputs for section detail of the members, section data and factored loads. The input items are:

- Section Profile: The only and default value is "Channels".
- Section Size: Store the section size's.

- Material: Store the value of Material.
- Cladding: Store the type of cladding.
- Torsional Restraint: Store the type of torsional restraint.
- Warping Restraint: Store the type of warping restraint.
- Effective Span: Store the length of effective span.
- Bending Moment (y-y): Store the bending moment in Y-Y plane.
- Bending Moment (z-z): Store the bending moment in Z-Z plane.
- Shear Force (y-y): Store the shear force in Y-Y plane.
- Shear Force (z-z): Store the shear force in Z-Z plane.

3.2.2 Purlin Module Code

The flow of logic in the Purlin module is as follows:

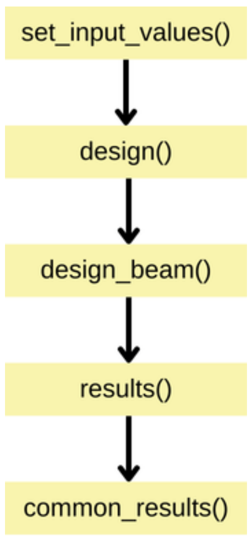


Figure 3.1: Logic Flow of the Purlin Module

- The set_input_values() function sets various values that will later be used in calculations.
- The main logic of purlin calculations and various checks are performed in the design_beam() function and based on these checks and calculations, values are sent to results() for each valid section.

- Lastly the section with the best utilisation ratio is selected and all the output values are assigned.

3.2.3 Output Dock

Values displayed in the output dock are for the section with the best utilisation ratio and if all sections fail the checks, then the values are displayed as NA. The items displayed in the output dock are as follows:

- Designation
- Utilization Ratio
- Section Classification
- Betab
- Eff. Sectional Area (m2)
- Eff. Length (m)
- Shear Strength (y-y) (kN)
- Shear Strength (z-z) (kN)
- Moment Strength (y-y) (kNm)
- Moment Strength (z-z) (kNm)
- High Shear Check (y-y)
- High Shear Check (z-z)
- Bending Compressive Stress (y-y)
- Bending Compressive Stress (z-z)
- Critical Moment (y-y) (Mcr)
- Critical Moment (z-z) (Mcr)
- Non-dimensional Effective SR (y-y)

- Non-dimensional Effective SR (z-z)
- Buckling Class
- Imperfection
- Bending Stress Reduction Factor (y-y)
- Bending Stress Reduction Factor (z-z)
- Moment (y-y)
- Moment (z-z)

3.3 Task 1: Python Code

3.3.1 Description of the Script

An Overview of the code is:

- input_values(): Takes input of different values via the input dock.
- set_input_values(): Sets the values for different variables that are later used in calculations.
- func_for_validation(): Used to validate the input and output values.
- output_values(): Sets the values for output dock in GUI after all calculations are completed.
- design_beam(): Executes the main calculations and checks for Purlin sections.
- results(): Based on the highest utilisation ration, calls common_results() to set values for output.
- common_results(): Sets all the values to be displayed in the output dock.
- section_classification(): Performs section classification.
- common_checks_1(): Takes input of an option and based on it executes a helper function. It is used to compile all the calculated values and results of different checks and their associated name into a dictionary for ease of use later.

3.3.2 Python Code

3.3.3 Full code

```
"""

@Author:    Rutvik Joshi - Osdag Team, IIT Bombay [(P)
    rutvikjoshi63@gmail.com / 30005086@iitb.ac.in]

@Module - Beam Design - Cantilever
           - Laterally Supported Beam [Moment + Shear]
           - Laterally Unsupported Beam [Moment + Shear]


@Reference(s): 1) IS 800: 2007, General construction in steel
     - Code of practice (Third revision)
               2) IS 808: 1989, Dimensions for hot rolled
    steel beam, column, channel, and angle sections and
                              it's subsequent revision(s)
               3) Design of Steel Structures by N.
    Subramanian (Fifth impression, 2019, Chapter 15)
               4) Limit State Design of Steel Structures by S
     K Duggal (second edition, Chapter 11)

other          8)
references     9)

"""
import logging
import math
import numpy as np
from Common import *
# from design_type.connection.moment_connection import
    MomentConnection
```

```
from utils.common.material import *
from utils.common.load import Load
from utils.common.component import ISection, Material
from utils.common.component import *
from design_type.member import Member
from Report_functions import *
from design_report.reportGenerator_latex import CreateLatex
from utils.common.common_calculation import *
from design_type.tension_member import *
from utils.common.Section_Properties_Calculator import
   BBAngle_Properties
from utils.common import is800_2007
from utils.common.component import *


# TODO DEBUG
class Flexure_Purlin(Member):

    def __init__(self):
        # print(f"Here10")
        super(Flexure_Purlin, self).__init__()

    ###############################################
    # Design Preference Functions Start
    ###############################################
    def tab_list(self):
        """

        :return: This function returns the list of tuples.
   Each tuple will create a tab in design preferences, in the
        order they are appended. Format of the Tuple is:
        [Tab Title, Type of Tab, function for tab content)
        Tab Title : Text which is displayed as Title of Tab,
        Type of Tab: There are Three types of tab layouts.
```

```
            Type_TAB_1: This have "Add", "Clear", "Download
xlsx file" "Import xlsx file"
            TYPE_TAB_2: This contains a Text box for side
note.
            TYPE_TAB_3: This is plain layout
        function for tab content: All the values like labels,
 input widgets can be passed as list of tuples,
        which will be displayed in chosen tab layout

        """
        tabs = []

        t1 = (KEY_DISP_COLSEC, TYPE_TAB_1, self.tab_section)
        tabs.append(t1)

        t2 = ("Optimization", TYPE_TAB_2, self.
optimization_tab_flexure_design)
        tabs.append(t2)

        t5 = ("Design", TYPE_TAB_2, self.design_values)
        tabs.append(t5)

        return tabs

    def tab_value_changed(self):
        change_tab = []

        t1 = (KEY_DISP_COLSEC, [KEY_SEC_MATERIAL], [
KEY_SEC_FU, KEY_SEC_FY], TYPE_TEXTBOX, self.
get_fu_fy_I_section)
        change_tab.append(t1)

        t4 = (KEY_DISP_COLSEC, ['Label_1', 'Label_2', '
Label_3', 'Label_4', 'Label_5'],
            ['Label_11', 'Label_12', 'Label_13', 'Label_14'
, 'Label_15', 'Label_16', 'Label_17', 'Label_18',
             'Label_19', 'Label_20', 'Label_21', 'Label_22'
, KEY_IMAGE], TYPE_TEXTBOX, self.get_I_sec_properties)
        change_tab.append(t4)

        t5 = (KEY_DISP_COLSEC, ['Label_HS_1', 'Label_HS_2', '
Label_HS_3'],
            ['Label_HS_11', 'Label_HS_12', 'Label_HS_13', '
Label_HS_14', 'Label_HS_15', 'Label_HS_16', 'Label_HS_17',
             'Label_HS_18',
             'Label_HS_19', 'Label_HS_20', 'Label_HS_21', '
Label_HS_22', KEY_IMAGE], TYPE_TEXTBOX,
            self.get_SHS_RHS_properties)
        change_tab.append(t5)

        t6 = (KEY_DISP_COLSEC, ['Label_CHS_1', 'Label_CHS_2',
 'Label_CHS_3'],
            ['Label_CHS_11', 'Label_CHS_12', 'Label_CHS_13'
, 'Label_HS_14', 'Label_HS_15', 'Label_HS_16', 'Label_21',
             'Label_22',
             KEY_IMAGE], TYPE_TEXTBOX, self.
get_CHS_properties)
        change_tab.append(t6)

        t6 = (KEY_DISP_COLSEC, [KEY_SECSIZE], [KEY_SOURCE],
TYPE_TEXTBOX, self.change_source)
        change_tab.append(t6)

        return change_tab

    def edit_tabs(self):
        """ This function is required if the tab name changes
 based on connectivity or profile or any other key.
```

```
                Not required for this module but empty list
should be passed"""
        return []

    def input_dictionary_design_pref(self):
        """

        :return: This function is used to choose values of
design preferences to be saved to design dictionary.

         It returns list of tuple which contains, tab name,
input widget type of keys, keys whose values to be saved,

         [(Tab Name, input widget type of keys, [List of keys
 to be saved])]

         """
        design_input = []

        t1 = (KEY_DISP_COLSEC, TYPE_COMBOBOX, [
KEY_SEC_MATERIAL])  # Need to check
        design_input.append(t1)

        t1 = (KEY_DISP_COLSEC, TYPE_TEXTBOX, [KEY_SEC_FU,
KEY_SEC_FY])
        design_input.append(t1)

        t2 = ("Optimization", TYPE_TEXTBOX,
              [KEY_EFFECTIVE_AREA_PARA, KEY_LENGTH_OVERWRITE,
 KEY_BEARING_LENGTH])  # , KEY_STEEL_COST
        design_input.append(t2)

        t2 = ("Optimization", TYPE_COMBOBOX, [KEY_ALLOW_CLASS
, KEY_LOAD])  # , KEY_STEEL_COST, KEY_ShearBucklingOption
```

```
        design_input.append(t2)

        t6 = ("Design", TYPE_COMBOBOX, [KEY_DP_DESIGN_METHOD
])
        design_input.append(t6)

        return design_input

    def input_dictionary_without_design_pref(self):

        design_input = []

        t1 = (KEY_MATERIAL, [KEY_SEC_MATERIAL], 'Input Dock')
        design_input.append(t1)

        t2 = (None, [KEY_ALLOW_CLASS, KEY_EFFECTIVE_AREA_PARA
, KEY_LENGTH_OVERWRITE, KEY_BEARING_LENGTH, KEY_LOAD,
                    KEY_DP_DESIGN_METHOD], '')   #
KEY_ShearBucklingOption
        design_input.append(t2)

        return design_input

    def refresh_input_dock(self):

        add_buttons = []

        t2 = (KEY_DISP_COLSEC, KEY_SECSIZE, TYPE_COMBOBOX,
KEY_SECSIZE, None, None, "Columns")
        add_buttons.append(t2)

        return add_buttons
    def get_values_for_design_pref(self, key,
design_dictionary):
        if design_dictionary[KEY_MATERIAL] != 'Select
Material':
            material = Material(design_dictionary[
KEY_MATERIAL], 41)
            fu = material.fu
            fy = material.fy
        else:
            fu = ''
            fy = ''

        val = {
            KEY_ALLOW_CLASS: 'Yes',
            KEY_EFFECTIVE_AREA_PARA: '1.0',
            KEY_LENGTH_OVERWRITE: 'NA',
            KEY_BEARING_LENGTH: 'NA',
            KEY_LOAD: 'Normal',
            KEY_DP_DESIGN_METHOD: "Limit State Design",
            # KEY_ShearBucklingOption: KEY_DISP_SB_Option[0],
        }[key]

        return val

    ####################################
    # Design Preference Functions End
    ####################################

    # Setting up logger and Input and Output Docks
    ####################################
    def module_name(self):
        return KEY_DISP_FLEXURE3

    def set_osdaglogger(key):
        """
        Set logger for Column Design Module.
        """
        global logger
        logger = logging.getLogger('Osdag')

        logger.setLevel(logging.DEBUG)
        handler = logging.StreamHandler()
        formatter = logging.Formatter(fmt='%(asctime)s - %(
    name)s - %(levelname)s - %(message)s',
                                      datefmt='%Y-%m-%d %H:%M
    :%S')

        handler.setFormatter(formatter)
        logger.addHandler(handler)
        handler = logging.FileHandler('logging_text.log')

        formatter = logging.Formatter(fmt='%(asctime)s - %(
    name)s - %(levelname)s - %(message)s',
                                      datefmt='%Y-%m-%d %H:%M
    :%S')
        handler.setFormatter(formatter)
        logger.addHandler(handler)

        if key is not None:
            handler = OurLog(key)
            formatter = logging.Formatter(fmt='%(asctime)s -
    %(name)s - %(levelname)s - %(message)s',
                                          datefmt='%Y-%m-%d %
    H:%M:%S')
            handler.setFormatter(formatter)
            logger.addHandler(handler)

    def customized_input(self):
        c_lst = []

        t1 = (KEY_SECSIZE, self.fn_profile_section)
        c_lst.append(t1)

        return c_lst

    def input_values(self):

        '''
            Fuction to return a list of tuples to be
displayed as the UI.(Input Dock)
        '''

        self.module = KEY_DISP_FLEXURE3
        options_list = []

        t1 = (None, DISP_TITLE_CM, TYPE_TITLE, None, True, '
No Validator')
        options_list.append(t1)

        t1 = (KEY_MODULE, KEY_DISP_FLEXURE3, TYPE_MODULE,
None, True, "No Validator")
        options_list.append(t1)

        t2 = (KEY_SEC_PROFILE, KEY_DISP_SEC_PROFILE,
TYPE_COMBOBOX, VALUES_SEC_PROFILE4, True,
                'No Validator')  # 'Beam and Column'
        options_list.append(t2)

        t4 = (KEY_SECSIZE, KEY_DISP_SECSIZE,
TYPE_COMBOBOX_CUSTOMIZED, ['All', 'Customized'], True, 'No
 Validator')
```

```
        options_list.append(t4)

        t4 = (KEY_MATERIAL, KEY_DISP_MATERIAL, TYPE_COMBOBOX,
    VALUES_MATERIAL, True, 'No Validator')
        options_list.append(t4)

        t1 = (None, KEY_SECTION_DATA, TYPE_TITLE, None, True,
    'No Validator')
        options_list.append(t1)

        t4 = (KEY_CLADDING, KEY_DISP_CLADDING, TYPE_COMBOBOX,
    VALUES_CLADDING, True, 'No Validator')
        options_list.append(t4)

        # t2 = (
        #     KEY_DESIGN_TYPE_FLEXURE,
        #     KEY_BEAM_SUPP_TYPE,
        #     TYPE_COMBOBOX,
        #     VALUES_SUPP_TYPE_temp,
        #     True,
        #     "No Validator",
        # )
        # options_list.append(t2)

        #
        # t3 = (KEY_BENDING, KEY_DISP_BENDING, TYPE_COMBOBOX,
    VALUES_BENDING_TYPE, False, 'No Validator')
        # options_list.append(t3)
        #
        #
        # t4 = (KEY_SUPPORT, KEY_DISP_SUPPORT, TYPE_NOTE,
   KEY_DISP_SUPPORT3, True, 'No Validator')
        # options_list.append(t4)
        # t12 = (KEY_IMAGE, None, TYPE_IMAGE, Purlin_img,
True, 'No Validator')
        # options_list.append(t12)
        #
        t10 = (KEY_TORSIONAL_RES, DISP_TORSIONAL_RES,
TYPE_COMBOBOX, Torsion_Restraint_list, True, 'No Validator
')
        options_list.append(t10)
        #
        t11 = (KEY_WARPING_RES, DISP_WARPING_RES,
TYPE_COMBOBOX, Warping_Restraint_list, True, 'No Validator
')
        options_list.append(t11)

        # t11 = (KEY_SUPPORT_TYPE, DISP_SUPPORT_RES,
TYPE_COMBOBOX, Supprt_Restraint_list, True, 'No Validator
')
        # options_list.append(t11)
        #
        # t11 = (KEY_SUPPORT_TYPE2, DISP_TOP_RES,
TYPE_COMBOBOX, Top_Restraint_list, False, 'No Validator')
        # options_list.append(t11)

        t5 = (KEY_LENGTH, KEY_DISP_LENGTH_BEAM, TYPE_TEXTBOX,
 None, True, 'Int Validator')
        options_list.append(t5)

        t7 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, '
No Validator')
        options_list.append(t7)

        t8 = (KEY_MOMENT_YY, KEY_DISP_MOMENT_YY + '*',
TYPE_TEXTBOX, None, True, 'No Validator')
        options_list.append(t8)
        t8 = (KEY_MOMENT_ZZ, KEY_DISP_MOMENT_ZZ + '*',
TYPE_TEXTBOX, None, True, 'No Validator')
        options_list.append(t8)

        t8 = (KEY_SHEAR_YY, KEY_DISP_SHEAR_YY + '*',
TYPE_TEXTBOX, None, True, 'No Validator')
        options_list.append(t8)

        t8 = (KEY_SHEAR_ZZ, KEY_DISP_SHEAR_ZZ + '*',
TYPE_TEXTBOX, None, True, 'No Validator')
        options_list.append(t8)

        return options_list

    def fn_profile_section(self):

        profile = self[0]
        if profile == 'Beams':  # Beam and Column
            return connectdb("Beams", call_type="popup")
            profile2 = connectdb("Columns", call_type="popup"
)
        if profile == 'Columns':  # Beam and Column
            return connectdb("Columns", call_type="popup")
            # profile2 = connectdb("Columns", call_type="
popup")
        if profile == 'Beams and Columns':  # Beam and Column
            res1 = connectdb("Beams", call_type="popup")
            res2 = connectdb("Columns", call_type="popup")
            return list(set(res1 + res2))
        if profile == 'Channels':
            return connectdb("Channels", call_type="popup")

    def fn_torsion_warping(self):
        print('Inside fn_torsion_warping', self)
        if self[0] == Torsion_Restraint1:
            return Warping_Restraint_list
        elif self[0] == Torsion_Restraint2:
            return [Warping_Restraint5]
        else:
            return [Warping_Restraint5]

    def fn_supp_image(self):
        print('Inside fn_supp_image', self)
        if self[0] == KEY_DISP_SUPPORT1:
            return Simply_Supported_img
        else:
            return Cantilever_img

    def axis_bending_change(self):
        design = self[0]
        print('Inside fn_supp_image', self)
        if self[0] == KEY_DISP_DESIGN_TYPE_FLEXURE:
            return ['NA']
        else:
            return VALUES_BENDING_TYPE

    # def show_error_message(self):
    #     QMessageBox.about(self, 'information', "Your
message!")
    def input_value_changed(self):

        lst = []

        t1 = ([KEY_SEC_PROFILE], KEY_SECSIZE,
TYPE_COMBOBOX_CUSTOMIZED, self.fn_profile_section)
        lst.append(t1)

        # t3 = ([KEY_SUPPORT], KEY_IMAGE, TYPE_IMAGE, self.
fn_supp_image)
        # lst.append(t3)

        # t3 = ([KEY_DESIGN_TYPE_FLEXURE], KEY_BENDING,
TYPE_COMBOBOX, self.axis_bending_change)
        # lst.append(t3)

        t3 = ([KEY_MATERIAL], KEY_MATERIAL,
TYPE_CUSTOM_MATERIAL, self.new_material)
        lst.append(t3)

        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_T_constatnt, TYPE_OUT_LABEL, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_T_constatnt, TYPE_OUT_DOCK, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_W_constatnt, TYPE_OUT_LABEL, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_W_constatnt, TYPE_OUT_DOCK, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_IMPERFECTION_FACTOR_LTB, TYPE_OUT_LABEL,
 self.output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_IMPERFECTION_FACTOR_LTB, TYPE_OUT_DOCK,
self.output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_SR_FACTOR_LTB, TYPE_OUT_LABEL, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_SR_FACTOR_LTB, TYPE_OUT_DOCK, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_NON_DIM_ESR_LTB, TYPE_OUT_LABEL, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_NON_DIM_ESR_LTB, TYPE_OUT_DOCK, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_DESIGN_STRENGTH_COMPRESSION,
TYPE_OUT_LABEL, self.output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_DESIGN_STRENGTH_COMPRESSION,
TYPE_OUT_DOCK, self.output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_Elastic_CM, TYPE_OUT_LABEL, self.
output_modifier)
        # lst.append(t18)
        #
        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         KEY_Elastic_CM, TYPE_OUT_DOCK, self.
output_modifier)
        # lst.append(t18)

        # t18 = ([KEY_DESIGN_TYPE_FLEXURE],
        #         'After checking Non-dimensional slenderness
ratio for given section, some sections maybe be ignored by
 Osdag.[Ref IS 8.2.2] ', TYPE_WARNING, self.
major_bending_warning)
        # lst.append(t18)

        return lst

    def output_modifier(self):
        print(self)
        if self[0] == VALUES_SUPP_TYPE_temp[2]:
            return False
        # elif self[0] == VALUES_SUPP_TYPE_temp[0] or self[0]
 == VALUES_SUPP_TYPE_temp[1] :
        #     return True
        else:
            return True
    def major_bending_warning(self):

        if self[0] == VALUES_SUPP_TYPE_temp[2]:
            return True
        else:
            return False

    def output_values(self, flag):

        out_list = []

        t1 = (None, DISP_TITLE_STRUT_SECTION, TYPE_TITLE,
None, True)

        out_list.append(t1)

        t1 = (KEY_TITLE_OPTIMUM_DESIGNATION,
KEY_DISP_TITLE_OPTIMUM_DESIGNATION, TYPE_TEXTBOX,
           self.result_designation if flag else '', True)
        out_list.append(t1)

        t1 = (
            KEY_OPTIMUM_UR_COMPRESSION,
KEY_DISP_OPTIMUM_UR_COMPRESSION, TYPE_TEXTBOX,
            round(self.result_UR, 3) if flag else '', True)
        out_list.append(t1)

        t1 = (KEY_OPTIMUM_SC, KEY_DISP_OPTIMUM_SC,
TYPE_TEXTBOX, self.result_section_class if flag else '',
True)
        out_list.append(t1)

        t2 = (KEY_betab_constatnt, KEY_DISP_betab_constatnt,
TYPE_TEXTBOX,
              round(self.result_betab, 2) if flag else '',
True)
        out_list.append(t2)

        t2 = (
            KEY_EFF_SEC_AREA, KEY_DISP_EFF_SEC_AREA,
TYPE_TEXTBOX, self.result_effective_area if flag else '',
            True)
        out_list.append(t2)

        t2 = (KEY_EFF_LEN, KEY_DISP_EFF_LEN, TYPE_TEXTBOX,
self.result_eff_len if flag else '',
              True)
        out_list.append(t2)

        t1 = (None, KEY_DESIGN_COMPRESSION, TYPE_TITLE, None,
 True)
        out_list.append(t1)

        t1 = (KEY_SHEAR_STRENGTH_YY,
KEY_DISP_DESIGN_STRENGTH_SHEAR_YY, TYPE_TEXTBOX,
              self.result_shear_yy if flag else
              '', True)
        out_list.append(t1)

        t1 = (KEY_SHEAR_STRENGTH_ZZ,
KEY_DISP_DESIGN_STRENGTH_SHEAR_ZZ, TYPE_TEXTBOX,
              self.result_shear_zz if flag else
              '', True)
        out_list.append(t1)
        #
        t1 = (KEY_MOMENT_STRENGTH_YY,
KEY_DISP_DESIGN_STRENGTH_MOMENT_YY, TYPE_TEXTBOX,
              self.result_bending_yy if flag else
            '', True)
        out_list.append(t1)

        t1 = (KEY_MOMENT_STRENGTH_ZZ,
KEY_DISP_DESIGN_STRENGTH_MOMENT_ZZ, TYPE_TEXTBOX,
              self.result_bending_zz if flag else
              '', True)
        out_list.append(t1)

        # t1 = (KEY_BUCKLING_STRENGTH,
KEY_DISP_BUCKLING_STRENGTH, TYPE_TEXTBOX,
        #        self.result_capacity if flag else
        #        '', True)
        # out_list.append(t1)
        # t1 = (KEY_WEB_CRIPPLING,
KEY_DISP_CRIPPLING_STRENGTH, TYPE_TEXTBOX,
        #        self.result_crippling if flag else
        #        '', True)
        # out_list.append(t1)

        t1 = (KEY_HIGH_SHEAR_YY, KEY_DISP_HIGH_SHEAR_YY,
TYPE_TEXTBOX,
              self.result_high_shear_yy if flag else
              '', True)
        out_list.append(t1)

        t1 = (KEY_HIGH_SHEAR_ZZ, KEY_DISP_HIGH_SHEAR_ZZ,
TYPE_TEXTBOX,
              self.result_high_shear_yy if flag else
              '', True)
        out_list.append(t1)

        # t1 = (None, KEY_DISP_LTB, TYPE_TITLE, None, False)
        # out_list.append(t1)
        #
        # t2 = (KEY_T_constatnt, KEY_DISP_T_constatnt,
TYPE_TEXTBOX,
        #       self.result_tc if flag else '', False)
        # out_list.append(t2)
        #
        # t2 = (KEY_W_constatnt, KEY_DISP_W_constatnt,
TYPE_TEXTBOX, self.result_wc if flag else '', False)
        # out_list.append(t2)
        #
        # t2 = (
        #     KEY_IMPERFECTION_FACTOR_LTB,
KEY_DISP_IMPERFECTION_FACTOR, TYPE_TEXTBOX, self.
result_IF_lt if flag else '',
        #     False)
        # out_list.append(t2)
        #
        # t2 = (KEY_SR_FACTOR_LTB, KEY_DISP_SR_FACTOR,
TYPE_TEXTBOX, self.result_srf_lt if flag else '', False)
        # out_list.append(t2)
        #
        # t2 = (KEY_NON_DIM_ESR_LTB, KEY_DISP_NON_DIM_ESR,
TYPE_TEXTBOX, self.result_nd_esr_lt if flag else '', False
)
        # out_list.append(t2)
        #
        # t1 = (KEY_DESIGN_STRENGTH_COMPRESSION,
KEY_DISP_COMP_STRESS, TYPE_TEXTBOX,
        #       self.result_nd_esr_lt if flag else
        #       '', False)
        # out_list.append(t1)
        #
        # t2 = (KEY_Elastic_CM, KEY_DISP_Elastic_CM,
TYPE_TEXTBOX, self.result_mcr if flag else '', False)
```

```
      # out_list.append(t2)

      # TODO
      # t1 = (None, KEY_DISP_LTB, TYPE_TITLE, None, False)
      # out_list.append(t1)

      # t2 = (KEY_T_constatnt, KEY_DISP_T_constatnt,
TYPE_TEXTBOX,
      #        self.result_tc if flag else '', False)
      # out_list.append(t2)

      # t2 = (KEY_W_constatnt, KEY_DISP_W_constatnt,
TYPE_TEXTBOX, self.result_wc if flag else '', False)
      # out_list.append(t2)

      # t2 = (
      #     KEY_IMPERFECTION_FACTOR_LTB,
KEY_DISP_IMPERFECTION_FACTOR, TYPE_TEXTBOX, self.
result_IF_lt if flag else '',
      #     False)
      # out_list.append(t2)

      # t2 = (KEY_SR_FACTOR_LTB, KEY_DISP_SR_FACTOR,
TYPE_TEXTBOX, self.result_srf_lt if flag else '', False)
      # out_list.append(t2)

      # t2 = (KEY_NON_DIM_ESR_LTB, KEY_DISP_NON_DIM_ESR,
TYPE_TEXTBOX, self.result_nd_esr_lt if flag else '', False
)
      # out_list.append(t2)

      # t1 = (KEY_DESIGN_STRENGTH_COMPRESSION,
KEY_DISP_COMP_STRESS, TYPE_TEXTBOX,
      #        self.result_fcd__lt if flag else
```

```
        #         '', False)
        # out_list.append(t1)

        # t2 = (KEY_Elastic_CM, KEY_DISP_Elastic_CM,
    TYPE_TEXTBOX, self.result_mcr if flag else '', False)
        # out_list.append(t2)

        t1 = (None, KEY_WEB_RESISTANCE, TYPE_TITLE, None,
    True)
        out_list.append(t1)

        t2 = (KEY_BENDING_COMPRESSIVE_STRESS_YY,
    KEY_DISP_BENDING_COMPRESSIVE_STRESS_YY, TYPE_TEXTBOX,
              self.result_Fcrb_yy if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_BENDING_COMPRESSIVE_STRESS_ZZ,
    KEY_DISP_BENDING_COMPRESSIVE_STRESS_ZZ, TYPE_TEXTBOX,
              self.result_Fcrb_zz if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_Elastic_CM_YY, KEY_DISP_Elastic_CM_YY,
    TYPE_TEXTBOX,
              self.result_mcr_yy if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_Elastic_CM_ZZ, KEY_DISP_Elastic_CM_ZZ,
    TYPE_TEXTBOX,
              self.result_mcr_zz if flag else
              '', True)
        out_list.append(t2)
        t2 = (KEY_NON_DIM_ESR_YY, KEY_DISP_NON_DIM_ESR_YY,
TYPE_TEXTBOX,
              self.result_lambda_lt_yy if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_NON_DIM_ESR_ZZ, KEY_DISP_NON_DIM_ESR_ZZ,
TYPE_TEXTBOX,
              self.result_lambda_lt_zz if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_BUCKLING_CLASS, KEY_DISP_BUCKLING_CLASS,
TYPE_TEXTBOX,
              self.result_buckling_class if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_IMPERFECTION_FACTOR,
KEY_DISP_IMPERFECTION_FACTOR, TYPE_TEXTBOX,
              self.result_IF_lt if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_BENDING_STRESS_RF_YY,
KEY_DISP_BENDING_STRESS_RF_YY, TYPE_TEXTBOX,
              self.result_Fbd_yy if flag else
              '', True)
        out_list.append(t2)

        t2 = (KEY_BENDING_STRESS_RF_ZZ,
KEY_DISP_BENDING_STRESS_RF_ZZ, TYPE_TEXTBOX,
              self.result_Fbd_zz if flag else
```

```
                '', True)
        out_list.append(t2)

        t2 = (KEY_RESISTANCE_MOMENT_YY,
KEY_DISP_RESISTANCE_MOMENT_YY,TYPE_TEXTBOX,
                self.result_resistance_bending_yy if flag else
                '', True)
        out_list.append(t2)

        t2 = (KEY_RESISTANCE_MOMENT_ZZ,
KEY_DISP_RESISTANCE_MOMENT_ZZ, TYPE_TEXTBOX,
                self.result_resistance_bending_zz if flag else
                '', True)
        out_list.append(t2)

        # t2 = (KEY_ESR, KEY_DISP_ESR, TYPE_TEXTBOX, self.
result_eff_sr if flag else '', True)
        # out_list.append(t2)
        #
        # t2 = (KEY_EULER_BUCKLING_STRESS,
KEY_DISP_EULER_BUCKLING_STRESS, TYPE_TEXTBOX,
        #       self.result_ebs if flag else '', True)
        # out_list.append(t2)
        #
        # t2 = (KEY_BUCKLING_CURVE, KEY_DISP_BUCKLING_CURVE,
TYPE_TEXTBOX, self.result_bc if flag else '', True)
        # out_list.append(t2)
        #
        # t2 = (
        #     KEY_IMPERFECTION_FACTOR,
KEY_DISP_IMPERFECTION_FACTOR, TYPE_TEXTBOX, self.result_IF
 if flag else '',
        #     True)
        # out_list.append(t2)
        #
        # t2 = (KEY_SR_FACTOR, KEY_DISP_SR_FACTOR,
TYPE_TEXTBOX, self.result_srf if flag else '', True)
        # out_list.append(t2)
        #
        # t2 = (KEY_NON_DIM_ESR, KEY_DISP_NON_DIM_ESR,
TYPE_TEXTBOX, self.result_nd_esr if flag else '', True)
        # out_list.append(t2)

        t2 = ()

        return out_list

    def func_for_validation(self, design_dictionary):
        print(f"func_for_validation here")
        all_errors = []
        self.design_status = False
        flag = False
        self.output_values(self, flag)

        flag1 = False
        flag2 = False
        flag3 = False
        flag4 = False
        flag5 = False
        option_list = self.input_values(self)
        missing_fields_list = []
        print(f'func_for_validation option_list {option_list}
'
              f"\n  design_dictionary {design_dictionary}"
              )
        for option in option_list:
            print(option,len(option))
            if option[2] == TYPE_TEXTBOX or option[0] ==
   KEY_LENGTH or option[0] == KEY_SHEAR_YY or option[0] ==
   KEY_SHEAR_ZZ or option[
                0] == KEY_MOMENT_YY or option[0] ==
   KEY_MOMENT_ZZ:
                try:

                    if design_dictionary[option[0]] == '':
                        missing_fields_list.append(option[1])
                        continue
                    if option[0] == KEY_LENGTH:
                        if float(design_dictionary[option
   [0]]) <= 0.0:
                               print("Input value(s) cannot be
   equal or less than zero.")
                               error = "Input value(s) cannot be
    equal or less than zero."
                               all_errors.append(error)
                        else:
                               flag1 = True
                    elif option[0] == KEY_SHEAR_YY:
                        if float(design_dictionary[option
   [0]]) <= 0.0:
                               print("Input value(s) cannot be
   equal or less than zero.")
                               error = "Input value(s) cannot be
    equal or less than zero."
                               all_errors.append(error)
                        else:
                               flag2 = True
                    elif option[0] == KEY_SHEAR_ZZ:
                        if float(design_dictionary[option
   [0]]) <= 0.0:
```

```
                            print("Input value(s) cannot be
equal or less than zero.")
                            error = "Input value(s) cannot be
 equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag3 = True
                    elif option[0] == KEY_MOMENT_YY:
                        if float(design_dictionary[option
[0]]) <= 0.0:
                            print("Input value(s) cannot be
equal or less than zero.")
                            error = "Input value(s) cannot be
 equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag4 = True
                    elif option[0] == KEY_MOMENT_ZZ:
                        if float(design_dictionary[option
[0]]) <= 0.0:
                            print("Input value(s) cannot be
equal or less than zero.")
                            error = "Input value(s) cannot be
 equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag5 = True
                except:
                    error = "Input value(s) are not valid"
                    all_errors.append(error)
                    # elif type(design_dictionary[option[0]])
 != 'float':
        #               print("Input value(s) are not valid
")
```

```
        #            error = "Input value(s) are not
valid"
        #            all_errors.append(error)

        # elif option[2] == TYPE_COMBOBOX and option[0]
not in [KEY_SEC_PROFILE, KEY_END1, KEY_END2,
KEY_DESIGN_TYPE_FLEXURE, KEY_BENDING, KEY_SUPPORT]:
        #     val = option[3]
        #     if design_dictionary[option[0]] == val[0]:
        #         missing_fields_list.append(option[1])

    if len(missing_fields_list) > 0:
        error = self.generate_missing_fields_error_string
(self, missing_fields_list)
        all_errors.append(error)
    else:
        flag = True

    if flag and flag1 and flag2 and flag3 and flag4 and
flag5:
        print(f"\n design_dictionary{design_dictionary}")
        self.set_input_values(self, design_dictionary)
        if self.design_status == False and len(self.
failed_design_dict) > 0:
            logger.error(
                "Design Failed, Check Design Report"
            )
            return  # ['Design Failed, Check Design
Report'] @TODO
        elif self.design_status:
            pass
        else:
            logger.error(
                "Design Failed. Selender Sections
Selected"
            )
            return  # ['Design Failed. Selender Sections
Selected']
    else:
        return all_errors

def get_3d_components(self):

    components = []

    # t3 = ('Column', self.call_3DColumn)
    # components.append(t3)

    return components

# warn if a beam of older version of IS 808 is selected
def warn_text(self):
    """ give logger warning when a beam from the older
version of IS 808 is selected """
    global logger
    red_list = red_list_function()

    if (self.sec_profile == VALUES_SEC_PROFILE[0]) or (
            self.sec_profile == VALUES_SEC_PROFILE[1]):
# Beams or Columns
        for section in self.sec_list:
            if section in red_list:
                logger.warning(
                    " : You are using a section ({}) (in
red color) that is not available in latest version of IS
808".format(
                            section))
```

```
 # Setting inputs from the input dock GUI
 def set_input_values(self, design_dictionary):
     '''
     TODO
             self.bending_type == KEY_DISP_BENDING1:
             self.lambda_lt = self.
lambda_lt_check_member_type
             if self.lambda_lt < 0.4:
                 self.design_type ==
KEY_DISP_DESIGN_TYPE_FLEXURE
     '''
     super(Flexure_Purlin, self).set_input_values(self,
design_dictionary)

     # section properties
     self.module = design_dictionary[KEY_MODULE]
     self.mainmodule = KEY_Flexure_Member_MAIN_MODULE
     self.sec_profile = design_dictionary[KEY_SEC_PROFILE]
     self.sec_list = design_dictionary[KEY_SECSIZE]
     print(f"\n Inside set_input_values{self.sec_profile}"
)
     print(f"\n sec_profile{self.sec_list}")
     self.main_material = design_dictionary[KEY_MATERIAL]
     self.material = design_dictionary[KEY_SEC_MATERIAL]

     # design type
     '''
         Temporarily has been set to Major Laterally
Supported, further on will be changed
     '''
     self.design_type_temp = KEY_DISP_BENDING1 + " " +
KEY_DISP_DESIGN_TYPE_FLEXURE  # or
KEY_DISP_DESIGN_TYPE2_FLEXURE
```

```
        self.latex_design_type = KEY_DISP_BENDING1 + " " +
    KEY_DISP_DESIGN_TYPE_FLEXURE  # or
    KEY_DISP_DESIGN_TYPE2_FLEXURE
        if self.design_type_temp == VALUES_SUPP_TYPE_temp[0]:
            self.design_type = VALUES_SUPP_TYPE[0]  # or
    KEY_DISP_DESIGN_TYPE2_FLEXURE
            self.bending_type = KEY_DISP_BENDING1
            # TODO self.support_cndition_shear_buckling
            self.support_cndition_shear_buckling = 'NA'  #
    design_dictionary[KEY_ShearBucklingOption]
        elif self.design_type_temp == VALUES_SUPP_TYPE_temp
    [1]:
            self.design_type = VALUES_SUPP_TYPE[0]
            self.bending_type = KEY_DISP_BENDING2  # if
    design_dictionary[KEY_BENDING] != 'Disabled' else 'NA'
            self.support_cndition_shear_buckling = 'NA'

        elif self.design_type_temp == VALUES_SUPP_TYPE_temp
    [2]:
            self.design_type = VALUES_SUPP_TYPE[1]
            self.bending_type = KEY_DISP_BENDING1
            self.support_cndition_shear_buckling = 'NA'

        # section user data
        self.length = float(design_dictionary[KEY_LENGTH])

        # end condition
        self.support = 'Supported'

        # factored loads
        self.load = Load(
            shear_force_yy=design_dictionary[KEY_SHEAR_YY],
            shear_force_zz=design_dictionary[KEY_SHEAR_ZZ],
            axial_force="",
            moment_yy=design_dictionary[KEY_MOMENT_YY],
            moment_zz=design_dictionary[KEY_MOMENT_ZZ],
            unit_kNm=True,
        )

        self.cladding = design_dictionary[KEY_CLADDING]

        # design preferences
        # self.allowable_utilization_ratio = float(
design_dictionary[KEY_ALLOW_UR])
        self.latex_efp = design_dictionary[
KEY_LENGTH_OVERWRITE]
        self.effective_area_factor = float(design_dictionary[
KEY_EFFECTIVE_AREA_PARA])
        self.allowable_utilization_ratio = 1.0
        self.optimization_parameter = "Utilization Ratio"
        self.allow_class = design_dictionary[KEY_ALLOW_CLASS]
   # if 'Semi-Compact' is available
        self.steel_cost_per_kg = 50
        # Step 2 - computing the design compressive stress
for web_buckling & web_crippling
        self.bearing_length = design_dictionary[
KEY_BEARING_LENGTH]
        # TAKE from Design Dictionary
        self.allowed_sections = []
        if self.allow_class == "Yes":
            self.allowed_sections == KEY_SemiCompact

        print(f"self.allowed_sections {self.allowed_sections}
")
        print("==================")
        # print(f"self.load_type {self.load_type}")

        print(f"self.module{self.module}")
        print(f"self.sec_list {self.sec_list}")
        print(f"self.material {self.material}")
        print(f"self.length {self.length}")
        print(f"self.load {self.load}")
        print("==================")

        # safety factors
        self.gamma_m0 = IS800_2007.cl_5_4_1_Table_5["gamma_m0
"]["yielding"]
        self.gamma_m1 = IS800_2007.cl_5_4_1_Table_5["gamma_m1
"]["ultimate_stress"]
        self.material_property = Material(material_grade=self
.material, thickness=0)
        self.fyf = self.material_property.fy
        self.fyw = self.material_property.fy

        print(f"self.material_property {self.
material_property}]")
        # print( "self.material_property",self.
material_property.fy)
        # initialize the design status
        self.design_status_list = []
        self.design_status = False
        self.sec_prop_initial_dict = {}
        self.failed_design_dict = {}
        self.design(self, design_dictionary)
        if self.flag:
            self.results(self, design_dictionary)

    # Simulation starts here
    def design(self, design_dictionary, flag=0):
        '''
        TODO optimimation_tab_check changes to include self.
material_property = Material(material_grade=self.material,
 thickness=0)
            for each section
        '''
        # flag = self.section_classification(self)
        print(f"\n Inside design")
        # self.show_error_message(self)
        """Perform design of struct"""
        # checking DP inputs

        self.optimization_tab_check(self)
        # print( "self.material_property",self.
material_property.fy)
        self.input_modifier(self)
        # print( "self.material_property",self.
material_property.fy)

        self.design_beam(self, design_dictionary)

 def optimization_tab_check(self):
        '''
        TODO add button to give user option to take Tension
holes or not
        '''
        print(f"\n Inside optimization_tab_check")
        self.latex_tension_zone = False
        if (self.effective_area_factor <= 0.10) or (self.
effective_area_factor > 1.0):
            logger.error(
                "The defined value of Effective Area Factor
in the design preferences tab is out of the suggested
range."
            )
            logger.info("Provide an appropriate input and re-
design.")
```

```
            logger.warning("Assuming a default value of 1.0."
)
            self.effective_area_factor = 1.0
            # self.design_status = False
            # self.design_status_list.append(self.
design_status)
            self.optimization_tab_check(self)
        elif (self.steel_cost_per_kg < 0.10) or (self.
effective_area_factor > 1.0) or (self.
effective_area_factor < 0):
            # No suggested range in Description
            logger.warning(
                "The defined value of the effective area
factor in the design preferences tab is out of the
suggested range."
            )
            logger.info("Assuming a default value of 1.0")

            self.steel_cost_per_kg = 50
            self.effective_area_factor = 1

            self.design_status = False
            # self.design_status_list.append(self.
design_status)
        else:
            if self.latex_tension_zone:
                if self.effective_area_factor >= (
                        self.material_property.fy * self.
gamma_m0 / (self.material_property.fu * 0.9 * self.
gamma_m1)):
                    pass
                else:
                    self.latex_tension_zone = True
                    print(f'self.latex_tension_zone: {self.
latex_tension_zone}')
                # self.effective_area_factor = (
                #     self.material_property.fy
                #     * self.gamma_m0
                #     / (self.material_property.fu * 0.9 *
self.gamma_m1)
                # )
                # logger.info(
                #     f"The effect of holes in the tension
flange is considered on the design bending strength. The
ratio of net to gross area of the flange in tension is
considered {self.effective_area_factor}"
                # )

        logger.info("Provided appropriate design preference,
now checking input.")

    def input_modifier(self):
        """Classify the sections based on Table 2 of IS
800:2007"""
        print(f"Inside input_modifier")
        local_flag = True
        self.input_modified = []
        self.input_section_list = []
        self.input_section_classification = {}

        for section in self.sec_list:
            section = section.strip("'")
            self.section_property = self.
section_connect_database(self, section)

            self.Zp_req = self.load.moment * self.gamma_m0 /
self.material_property.fy
```

```
            print('Inside input_modifier not allow_class',
self.allow_class, self.load.moment, self.gamma_m0,
                  self.material_property.fy)
            if self.section_property.plast_sec_mod_z >= self.
Zp_req:
                self.input_modified.append(section)
                # logger.info(
                #     f"Required self.Zp_req = {round(self.
Zp_req * 10**-3,2)} x 10^3 mm^3 and Zp of section {self.
section_property.designation} = {round(self.
section_property.plast_sec_mod_z* 10**-3,2)} x 10^3 mm^3.
Section satisfy Min self.Zp_req value")
            # else:
            # local_flag = False

            # logger.warning(
            #     f"Required self.Zp_req = {round(self.Zp_req
* 10**-3,2)} x 10^3 mm^3 and Zp of section {self.
section_property.designation} = {round(self.
section_property.plast_sec_mod_z* 10**-3,2)} x 10^3 mm^3.
Section dosen't satisfy Min self.Zp_req value")
        print("self.input_modified", self.input_modified)

    def section_connect_database(self, section):
        print(f"section_connect_database{section}")
        print(section)
        # print(self.sec_profile)
        if (
                self.sec_profile == VALUES_SECTYPE[1]
                or self.sec_profile == "I-section"
        ):  # I-section
            self.section_property = ISection(
                designation=section, material_grade=self.
material
```

```
            )
            print(self.section_property)
            self.material_property.
connect_to_database_to_get_fy_fu(
                self.material, max(self.section_property.
flange_thickness, self.section_property.web_thickness)
            )
            print(f"section_connect_database
material_property.fy{self.material_property.fy}")
            self.epsilon = math.sqrt(250 / self.
material_property.fy)
        elif (self.sec_profile == VALUES_SECTYPE[6]):
            print(self.material)
            self.section_property = ISection(
                designation=section, material_grade=self.
material, table= self.sec_profile
            )
            print(self.section_property)
            self.material_property.
connect_to_database_to_get_fy_fu(
                self.material, max(self.section_property.
flange_thickness, self.section_property.web_thickness)
            )
            print(f"section_connect_database
material_property.fy{self.material_property.fy}")
            self.epsilon = math.sqrt(250 / self.
material_property.fy)

        return self.section_property

    def design_beam(self, design_dictionary):
        print(f"Inside design_beam")
        # 1- Based on optimum UR
        self.optimum_section_ur_results = {}
        self.optimum_section_ur = []

        # 2 - Based on optimum cost
        self.optimum_section_cost_results = {}
        self.optimum_section_cost = []

        # 1 - section classification
        self.flag = self.section_classification(self,
design_dictionary)

        print('self.flag:', self.flag)
        if self.effective_area_factor < 1.0:
            logger.warning(
                "Reducing the effective sectional area as per
 the definition in the Design Preferences tab."
            )
        else:
            logger.info(
                "The effective sectional area is taken as
100% of the cross-sectional area [Reference: Cl. 7.3.2, IS
 800:2007]."
            )
        print(
            f"self.effective_length {self.effective_length} \
n self.input_section_classification{self.
input_section_classification} ")

        if self.flag:
            for section in self.input_section_list:
                # initialize lists for updating the results
dictionary
                self.section_property = self.
section_connect_database(self, section)
                if self.section_property.type == 'Rolled':
                    self.effective_depth = (self.
section_property.depth - 2 * (
                            self.section_property.
flange_thickness + self.section_property.root_radius))
                else:
                    self.effective_depth = (self.
section_property.depth - 2 * self.section_property.
flange_thickness)
                print('self.section_property.type:', self.
section_property.type, self.bending_type)

                # Step 1.1 - computing the effective
sectional area
                self.effective_area = self.section_property.
area


                list_result = []
                list_1 = []
                list_result.append(section)
                list_1.append("Designation")
                self.section_class = self.
input_section_classification[section][0]
                self.It = self.input_section_classification[
section][5]
                self.hf = self.input_section_classification[
section][6]
                self.Iw = self.input_section_classification[
section][7]
                # 2.9 - Cost of the section in INR
                self.depth_thickness_ratio = self.
effective_depth / self.section_property.web_thickness
                self.web_buckling_check = IS800_2007.
cl_8_2_1_web_buckling(
                    d=self.effective_depth,
                    tw=self.section_property.web_thickness,
                    e=self.epsilon,
                )

                if self.section_class == KEY_Plastic or self.
section_class == KEY_Compact:
                    self.beta_b_lt = 1.0
                else:
                    self.beta_b_lt = (
                            self.section_property.
elast_sec_mod_z
                            / self.section_property.
plast_sec_mod_z
                    )

                if(not self.web_buckling_check):
                    self.shear_area_zz = self.
section_property.depth * self.section_property.
web_thickness
                    self.shear_area_yy = 2 * self.
section_property.flange_width * self.section_property.
flange_thickness
                    self.buckling_class = 'c'
                    print(f"shear area ZZ is {self.
shear_area_zz}")
                    print(f"shear area YY is {self.
shear_area_yy}")
                    self.V_d_yy = IS800_2007.
cl_8_4_design_shear_strength(
                        self.shear_area_yy,
                        self.material_property.fy
                    )
                    self.V_d_zz = IS800_2007.
cl_8_4_design_shear_strength(
                        self.shear_area_zz,
                        self.material_property.fy
                    )

                    print(f"shear force yy is {self.load.
shear_force_yy}")
                    print(f"shear force zz is {self.load.
shear_force_zz}")

                    if self.load.shear_force_yy<self.V_d_yy
and self.load.shear_force_zz<self.V_d_zz :

                        self.high_shear_check_yy = IS800_2007
.cl_8_2_1_2_high_shear_check(
                            self.load.shear_force_yy,
                            self.V_d_yy
                        )
                        self.high_shear_check_zz = IS800_2007
.cl_8_2_1_2_high_shear_check(
                            self.load.shear_force_zz,
                            self.V_d_zz
                        )
                        print(f"high shear check yy is {self.
high_shear_check_yy}")
                        print(f"high shear check zz is {self.
high_shear_check_zz}")

                        self.M_d_yy = self.
design_bending_strength_purlins(
                            self,
                            self.section_class,
                            self.section_property.
plast_sec_mod_y,
                            self.section_property.
elast_sec_mod_y,
                            self.section_property.fy,
                            self.gamma_m0,
                            self.high_shear_check_yy,
                            'y'
                        )

                        self.M_d_zz = self.
design_bending_strength_purlins(
                            self,
                            self.section_class,
                            self.section_property.
plast_sec_mod_z,
                            self.section_property.
elast_sec_mod_z,
                            self.section_property.fy,
                            self.gamma_m0,
                            self.high_shear_check_zz,
                            'z'
                        )
                        if self.load.moment_yy<self.M_d_yy
and self.load.moment_zz<self.M_d_zz :

                            self.M_d_yy1 = self.
web_resistance_check(
                            self,
                            self.section_property.
mom_inertia_y,
                            self.section_property.
elast_sec_mod_y,
                            self.section_property.
plast_sec_mod_y,
                                        self.section_property.
   rad_of_gy_z,
                                        'y'
                                    )

                                    self.M_d_zz1 = self.
   web_resistance_check(
                                        self,
                                        self.section_property.
   mom_inertia_z,
                                        self.section_property.
   elast_sec_mod_z,
                                        self.section_property.
   plast_sec_mod_z,
                                        self.section_property.
   rad_of_gy_y,
                                        'z'
                                    )

                                    self.ur = max(
                                        self.load.shear_force_yy /
   self.V_d_yy,
                                        self.load.shear_force_zz /
   self.V_d_zz,
                                        self.load.moment_yy / self.
   M_d_yy,
                                        self.load.moment_zz / self.
   M_d_zz ,
                                        (self.load.moment_yy/self.
   M_d_yy1) +
                                        (self.load.moment_zz/self.
   M_d_zz1)
                                    )
```

```
                         self.optimum_section_ur.append(
self.ur)

                         self.web_buckling_check1 = self.
buckling_resistance_check(
                             self,
                             self.load.moment_yy,
                             self.load.moment_zz,
                             self.M_d_yy1,
                             self.M_d_zz1,
                         )

                         if (self.web_buckling_check1):
                             '''
                             Deflection check
                             '''
                             w_y = (self.load.
shear_force_yy**2)/(2*self.load.moment_yy*1.5)
                             w_z = (self.load.
shear_force_zz**2)/(2*self.load.moment_zz*1.5)
                             del_y = self.
serviceability_check(self,w_y,

 self.section_property.elast_sec_mod_y,

 self.section_property.mom_inertia_y

 )
                             del_z = self.
serviceability_check(self,w_z,

 self.section_property.elast_sec_mod_z,

 self.section_property.mom_inertia_z
 )

                              if self.cladding == 'Brittle
Cladding':
                                  del_limit = self.length /
 180
                              else :
                                  del_limit = self.length /
 150

                              if del_z < del_limit and
del_y < del_limit:
                                  self.cost = (
                                          (
                                                 self.
section_property.unit_mass
                                                 * self.
section_property.area
                                                 * 1e-4
                                          )
                                          * self.length
                                          * self.
steel_cost_per_kg
                                  )
                                  self.optimum_section_cost
.append(self.cost)
                                  list_result, list_1 =
self.list_changer(self,
                                                   change=
None,
                                                   check=
True,
                                                            list=
list_result,list_name=list_1)
                                 self.common_checks_1(self
, section, 5, list_result, list_1)
                             else:
                                 list_1.extend(["Section
class", "It", "Iw", "Web.Buckling", "Beta_b"])
                                 list_result.extend(
                                     [self.section_class,
self.It, self.Iw, self.web_buckling_check, self.beta_b_lt
])
                                 self.common_checks_1(self
, section, 5, list_result, list_1)
                         else:
                             list_1.extend(["Section class
", "It", "Iw", "Web.Buckling", "Beta_b"])
                             list_result.extend(
                                 [self.section_class, self
.It, self.Iw, self.web_buckling_check, self.beta_b_lt])
                             self.common_checks_1(self,
section, 5, list_result, list_1)
                     else:
                         list_1.extend(["Section class", "
It", "Iw", "Web.Buckling", "Beta_b"])
                         list_result.extend(
                             [self.section_class, self.It,
 self.Iw, self.web_buckling_check, self.beta_b_lt])
                         self.common_checks_1(self,
section, 5, list_result, list_1)
                 else:
                     list_1.extend(["Section class", "It",
 "Iw", "Web.Buckling", "Beta_b"])
                     list_result.extend(
                                    [self.section_class, self.It,
self.Iw, self.web_buckling_check, self.beta_b_lt])
                              self.common_checks_1(self, section,
5, list_result, list_1)
                  else:
                        list_1.extend(["Section class", "It", "Iw
", "Web.Buckling", "Beta_b"])
                        list_result.extend([self.section_class,
self.It, self.Iw, self.web_buckling_check, self.beta_b_lt
])
                        self.common_checks_1(self, section, 5,
list_result, list_1)



                  '''
                  if self.bearing_length != 'NA':  # and self.
web_crippling
                        print(f"Check for Web Buckling")
                        try:
                              self.bearing_length = float(
design_dictionary[KEY_BEARING_LENGTH])
                              self.web_buckling = True  # WEB
BUCKLING
                              self.I_eff_web = self.bearing_length
* self.section_property.web_thickness ** 3 / 12
                              self.A_eff_web = self.bearing_length
* self.section_property.web_thickness
                              self.r = math.sqrt(self.I_eff_web /
self.A_eff_web)
                              self.slenderness = 0.7 * self.
effective_depth / self.r
                              self.common_checks_1(self, section,
step=3)
                              # step == 4
                        self.common_checks_1(
                            self, section, step=4,
list_result=["Concentric"]
                        )
                        # 2.7 - Capacity of the section for
web_buckling
                        self.section_capacity = (
                                self.
design_compressive_stress * (
                                self.bearing_length + self.
section_property.depth / 2) * self.section_property.
web_thickness
                                * 10 ** -3)  # N
                        print(self.design_compressive_stress,
 self.bearing_length, self.section_property.depth,
                              self.section_property.
web_thickness)

                        print(self.bending_strength_section,
self.shear_strength, self.section_capacity)

                        self.F_wb = (self.bearing_length +
2.5 * (
                                self.section_property.
root_radius + self.section_property.flange_thickness)) *
self.section_property.web_thickness * self.
material_property.fy / (
                                            self.gamma_m0 *
10 ** 3)
                        if self.bending_strength_section >
self.load.moment * 10 ** -6 and self.shear_strength > self
.load.shear_force * 10 ** -3 and self.section_capacity >
self.load.shear_force * 10 ** -3 and self.F_wb > self.load
.shear_force * 10 ** -3:
```

```
                            list_result, list_1 = self.
list_changer(self, change='Web Buckling', check=True,

        list=list_result, list_name=list_1)
                            self.optimum_section_ur.append(
self.ur)
                        else:
                            list_result, list_1 = self.
list_changer(self, change='Web Buckling', check=True,

        list=list_result, list_name=list_1)
                            self.optimum_section_ur.append(
self.ur)
                        # Step 3 - Storing the optimum
results to a list in a descending order
                        self.common_checks_1(self, section,
5, list_result, list_1)
                    except:
                        logger.warning('Bearing length is
invalid.')
                        logger.info('Ignoring web Buckling
and Crippling check')
                        self.bearing_length = 'NA'
                        self.web_buckling = False
                        # 2.8 - UR
                        print(self.bending_strength_section,
self.shear_strength)
                        if self.bending_strength_section >
self.load.moment * 10 ** -6 and self.shear_strength > self
.load.shear_force * 10 ** -3:
                            list_result, list_1 = self.
list_changer(self, change='', check=True, list=list_result
,
        list_name=list_1)
                            self.optimum_section_ur.append(
self.ur)

                            # Step 3 - Storing the optimum
results to a list in a descending order
                            self.common_checks_1(self,
section, 5, list_result, list_1)
                        else:
                            list_result, list_1 = self.
list_changer(self, change='', check=True, list=list_result
,

        list_name=list_1)
                            self.optimum_section_ur.append(
self.ur)
                            # Step 3 - Storing the optimum
results to a list in a descending order
                            self.common_checks_1(self,
section, 5, list_result, list_1)

                else:
                    self.web_buckling = False
                    # 2.8 - UR
                    print(self.bending_strength_section, self
.shear_strength)
                    if self.bending_strength_section > self.
load.moment * 10 ** -6 and self.shear_strength > self.load
.shear_force * 10 ** -3:

                        self.optimum_section_ur.append(self.
ur)
```

```
                        list_result , list_1 = self.
list_changer(self, change=' ', check=True, list=
list_result,

   list_name=list_1)

                        # Step 3 - Storing the optimum
results to a list in a descending order
                        self.common_checks_1(self, section,
5, list_result, list_1)
                    else:
                        self.optimum_section_ur.append(self.
ur)
                        list_result , list_1 = self.
list_changer(self, change=' ', check=True, list=
list_result,

   list_name=list_1)

                        # Step 3 - Storing the optimum
results to a list in a descending order
                        self.common_checks_1(self, section,
5, list_result, list_1)
                '''
             print('self.optimum_section_ur', self.
optimum_section_ur)

 def beam_web_buckling(self):

     print(f"Working web_buckling_check")
     # 3 - web buckling under shear
     self.web_buckling_check = IS800_2007.
cl_8_2_1_web_buckling(
         d=self.effective_depth,
         tw=self.section_property.web_thickness,
         e=self.epsilon,
     )
     print(self.web_buckling_check, self.section_property.
designation)

     if not self.web_buckling_check:
         self.web_not_buckling_steps(self)

 def web_buckling_steps(self):
     print(f"Not using web_buckling_steps")
     # logger.info(f"Considering  {self.
support_cndition_shear_buckling}")
     # 5 - Web Buckling check(when high shear) -If user
wants then only
     # if web_buckling:
     #     b1 = input('Enter bearing')
     #     self.web_buckling_strength = self.
section_property.web_thickness * (b1 + 1.25 * self.
section_property.depth)
     # self.V_d = pass
     # web_buckling_message = 'Thin web'
     if self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[0]:
         self.K_v = IS800_2007.
cl_8_4_2_2_K_v_Simple_postcritical('only support')
         self.plate_girder_strength(self)
         # logger.info('Section = {}, V_cr = {}'.format(
self.section_property.designation, round(self.V_cr,2)))
         self.shear_strength = self.V_cr / self.gamma_m0
         # if self.V_d > self.load.shear_force * 10**-3:
         #
         #     return True
         # else:
```

```
          #     return False
          # self.V_d = IS800_2007.
cl_8_4_2_2_ShearBuckling_Simple_postcritical((self.
section_property.depth - 2 *(self.section_property.
flange_thickness + self.section_property.root_radius),
          #
                        self.section_property.web_thickness,
space,0.3, self.fyw))
      elif self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[1]:
          self.V_p = IS800_2007.
cl_8_4_design_shear_strength(
              self.shear_area,
              self.material_property.fy
          ) / 10 ** 3 * self.gamma_m0
          self.Mfr = IS800_2007.cl_8_4_2_2_Mfr_TensionField
(self.section_property.flange_width,

 self.section_property.flange_thickness, self.fyf,

 self.load.moment / (

          self.section_property.depth - self.
section_property.flange_thickness),

 self.gamma_m0)
          print('MFr', self.Mfr)
          if self.Mfr > 0:
              print('Starting loop', int(round(self.
effective_length * 10 ** 4 / self.effective_depth, -1) /
10))
              # for c_d in range(3,self.effective_length/
self.result_eff_d):
              for c_d in reversed(
                        list(range(3, int(round(self.
effective_length * 1000 / self.effective_depth, -1))))):
                    print('c_d', c_d, 'c/d', self.
effective_length * 1000 / self.effective_depth)
                    c_d = c_d / 10 + 0.1
                    self.c = round(c_d * self.effective_depth
, -1)
                    print('c', self.c)
                    self.K_v = IS800_2007.
cl_8_4_2_2_K_v_Simple_postcritical('many support', self.c,

                   self.effective_depth)
                    self.plate_girder_strength2(self)


                    self.shear_strength = self.V_tf_girder /
self.gamma_m0 * 10 ** -3
                    logger.info(
                        'Intermediate Stiffeners required d
={}, c = {}, Section = {}, V_tf = {}, V_d = {}'.format(
                            self.effective_depth, self.c,
                            self.section_property.designation
,
                            self.V_tf_girder, self.
shear_strength))
                    if self.shear_strength > self.load.
shear_force * 10 ** -3:
                        return
                return
            else:
                self.shear_strength = 0.1


    def web_not_buckling_steps(self):
        print(f"Working web_not_buckling_steps")
        self.V_d = IS800_2007.cl_8_4_design_shear_strength(
            self.shear_area,
            self.material_property.fy
        ) / 10 ** 3
        self.shear_strength = self.V_d
        self.high_shear_check = IS800_2007.
cl_8_2_1_2_high_shear_check(
            self.load.shear_force / 1000, self.V_d
        )
        print(
            f"self.V_d {self.V_d},{self.section_property.
depth * self.section_property.web_thickness}, {self.
material_property.fy}")
        # 4 -  design bending strength
        self.bending_strength_section = self.bending_strength
(self) / 10 ** 6

    def bending_strength(self):
        print('Inside bending_strength ', '\n self.
section_class', self.section_class)
        # 4 - design bending strength

        if self.high_shear_check:
            if self.section_class == KEY_Plastic or self.
section_class == KEY_Compact:
                bending_strength_section = self.
bending_strength_reduction(self, M_d)
            else:
                bending_strength_section = (
                        self.section_property.elast_sec_mod_z
                        * self.material_property.fy
                        / self.gamma_m0
                )
        else:
            bending_strength_section = M_d
        print('Inside bending_strength 1', M_d, self.
high_shear_check, bending_strength_section)
        # self.It = (
        #     2
        #     * self.section_property.flange_width
        #     * self.section_property.flange_thickness**3
        # ) / 3 + (
        #     (self.section_property.depth - self.
section_property.flange_thickness)
        #     * self.section_property.web_thickness**3
        # ) / 3
        # self.hf = self.section_property.depth - self.
section_property.flange_thickness
        # self.Iw = 0.5**2 * self.section_property.
mom_inertia_y * self.hf**2
        # self.M_cr = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_non_slenderness(
        #     self.material_property.modulus_of_elasticity,
        #     0.3,
        #     self.section_property.mom_inertia_y,
        #     self.It,
        #     self.Iw,
        #     self.effective_length * 1e3
        # )
        #
        # if self.section_class == KEY_Plastic or self.
section_class == KEY_Compact:
        #     self.beta_b_lt = 1.0
        # else:
        #     self.beta_b_lt = (
        #         self.section_property.elast_sec_mod_z
        #         / self.section_property.plast_sec_mod_z
        #     )
        if self.section_property.type == "Rolled":
            alpha_lt = 0.21
        else:
            alpha_lt = 0.49
        # lambda_lt = IS800_2007.
cl_8_2_2_1_elastic_buckling_moment(
        #     self.beta_b_lt,
        #     self.section_property.plast_sec_mod_z,
        #     self.section_property.elast_sec_mod_z,
        #     self.material_property.fy,
        #     self.M_cr
        # )
        phi_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_phi_lt(
            alpha_lt, self.lambda_lt
        )
        X_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_stress_reduction_factor(
            phi_lt, self.lambda_lt
        )
        fbd = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_compressive_stress(
            X_lt, self.material_property.fy, self.gamma_m0
        )
        bending_strength_section = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_strength(
            self.section_property.plast_sec_mod_z,
            self.section_property.elast_sec_mod_z,
            fcd=fbd,
            section_class=self.section_class
        )
        # self.beta_b_lt = beta_b
        self.alpha_lt = alpha_lt
        # self.lambda_lt = lambda_lt
        self.phi_lt = phi_lt
        self.X_lt = X_lt
        self.fbd_lt = fbd
        self.lateral_tb = self.M_cr * 10 ** -6
        print('Inside bending_strength 2.1', fbd, self.
    section_property.plast_sec_mod_z)
        if self.high_shear_check:
            if self.section_class == KEY_Plastic or self.
    section_class == KEY_Compact:
                bending_strength_section = self.
    bending_strength_reduction(self, Md=
    bending_strength_section

                 )
            else:
                bending_strength_section = (
                        self.beta_b_lt
                        * self.section_property.
    plast_sec_mod_z
                        * fbd
                )
        print('Inside bending_strength 2', self.It, self.hf,
    self.Iw, self.M_cr, self.beta_b_lt, alpha_lt,
               self.lambda_lt, phi_lt, X_lt, fbd,
    bending_strength_section)
        self.bending_strength_section_reduced =
    bending_strength_section
        return bending_strength_section

    def bending_strength_girder(self):
        print('Inside bending_strength of girder ')
        web_class = IS800_2007.Table2_i(
            (self.section_property.flange_width - self.
    section_property.web_thickness) / 2,
            self.section_property.flange_thickness,
            self.material_property.fy, self.section_property.
type
        )[0]
        flange_class = IS800_2007.Table2_i(
            self.section_property.depth - 2 * self.
section_property.flange_thickness,
            self.section_property.web_thickness,
            self.material_property.fy, self.section_property.
type
        )[0]
        if flange_class == "Slender" or web_class == "Slender
":
            self.section_class_girder = "Slender"
        else:
            if flange_class == KEY_Plastic and web_class ==
KEY_Plastic:
                self.section_class_girder = KEY_Plastic
            elif flange_class == KEY_Plastic and web_class ==
 KEY_Compact:
                self.section_class_girder = KEY_Compact
            elif flange_class == KEY_Plastic and web_class ==
 KEY_SemiCompact:
                self.section_class_girder = KEY_SemiCompact
            elif flange_class == KEY_Compact and web_class ==
 KEY_Plastic:
                self.section_class_girder = KEY_Compact
            elif flange_class == KEY_Compact and web_class ==
 KEY_Compact:
                self.section_class_girder = KEY_Compact
            elif flange_class == KEY_Compact and web_class ==
 KEY_SemiCompact:
                self.section_class_girder = KEY_SemiCompact
            elif flange_class == KEY_SemiCompact and
web_class == KEY_Plastic:
```

```
                self.section_class_girder = KEY_SemiCompact
            elif flange_class == KEY_SemiCompact and
web_class == KEY_Compact:
                self.section_class_girder = KEY_SemiCompact
            elif flange_class == KEY_SemiCompact and
web_class == KEY_SemiCompact:
                self.section_class_girder = KEY_SemiCompact
        # 4 - design bending strength
        I_flange = 2 * (
                    self.section_property.flange_width * self
.section_property.flange_thickness ** 3 / 12 + self.
section_property.flange_width * self.section_property.
flange_thickness * (
                        self.section_property.depth / 2 -
self.section_property.flange_thickness / 2) ** 2)
        Zez_flange = I_flange / self.section_property.depth /
 2
        y_top = (self.section_property.flange_width * self.
section_property.flange_thickness * (
                    self.section_property.depth - self.
section_property.flange_thickness) / 2) / (
                            self.section_property.
flange_width * self.section_property.flange_thickness)
        Zpz_flange = 2 * self.section_property.flange_width *
 self.section_property.flange_thickness * y_top
        M_d = IS800_2007.cl_8_2_1_2_design_bending_strength(
            self.section_class_girder,
            Zpz_flange,
            Zez_flange,
            self.material_property.fy,
            self.gamma_m0,
            self.support,
        )
        if self.section_class_girder == KEY_Plastic or self.
section_class_girder == KEY_Compact:
            self.beta_b_lt = 1
        else:
            self.beta_b_lt = Zez_flange / Zpz_flange
        self.M_d = M_d
        if self.design_type == KEY_DISP_DESIGN_TYPE_FLEXURE:
            if self.high_shear_check:
                if self.section_class_girder == KEY_Plastic
or self.section_class_girder == KEY_Compact:
                    bending_strength_section = self.
bending_strength_reduction(self, M_d)
                else:
                    bending_strength_section = (
                            self.section_property.
elast_sec_mod_z
                            * self.material_property.fy
                            / self.gamma_m0
                    )
            else:
                bending_strength_section = M_d
            print('Inside bending_strength 1', M_d, self.
high_shear_check, bending_strength_section)
        else:
            # self.It = (
            #     2
            #     * self.section_property.flange_width
            #     * self.section_property.flange_thickness**3
            # ) / 3 + (
            #     (self.section_property.depth - self.
section_property.flange_thickness)
            #     * self.section_property.web_thickness**3
            # ) / 3
            self.hf = self.section_property.depth - self.
section_property.flange_thickness
            # self.Iw = 0.5**2 * self.section_property.
mom_inertia_y * self.hf**2
            self.fcrb = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_fcrb(
                self.material_property.modulus_of_elasticity,
                self.effective_length / self.section_property
.rad_of_gy_y,
                self.hf / self.section_property.
flange_thickness
            )

            if self.section_class_girder == KEY_Plastic or
self.section_class_girder == KEY_Compact:
                self.beta_b_lt = 1.0
            else:
                self.beta_b_lt = (
                        self.section_property.elast_sec_mod_z
                        / self.section_property.
plast_sec_mod_z
                )
            if self.section_property.type == "Rolled":
                alpha_lt = 0.21
            else:
                alpha_lt = 0.49
            lambda_lt = IS800_2007.
cl_8_2_2_1_elastic_buckling_moment_fcrb(
                self.material_property.fy, self.fcrb
            )
            phi_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_phi_lt(
                alpha_lt, lambda_lt
            )
            X_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_stress_reduction_factor(
                phi_lt, lambda_lt
            )
            fbd = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_compressive_stress(
                X_lt, self.material_property.fy, self.
gamma_m0
            )
            bending_strength_section = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_strength(
                self.section_property.plast_sec_mod_z,
                self.section_property.elast_sec_mod_z,
                fcd=fbd,
                section_class=self.section_class_girder
            )

            # self.beta_b_lt = beta_b
            self.alpha_lt = alpha_lt
            # self.lambda_lt = lambda_lt
            self.phi_lt = phi_lt
            self.X_lt = X_lt
            self.fbd_lt = fbd
            self.lateral_tb = self.fcrb * 10 ** -6
            print('Inside bending_strength 2.1', fbd, self.
section_property.plast_sec_mod_z)
            if self.high_shear_check:
                if self.section_class_girder == KEY_Plastic
or self.section_class_girder == KEY_Compact:
                    bending_strength_section = self.
bending_strength_reduction(self, Md=
bending_strength_section

                    )
                else:
                    bending_strength_section = (
                                self.beta_b_lt
                                * self.section_property.
plast_sec_mod_z
                                * fbd
                    )
            print('Inside bending_strength 2', self.It, self.
hf, self.Iw, self.fcrb, self.beta_b_lt, alpha_lt,
                  lambda_lt, phi_lt, X_lt, fbd,
bending_strength_section)
        self.bending_strength_section_reduced =
bending_strength_section
        return bending_strength_section

    def bending_strength_reduction(self, Md, axis):
        if axis == 'y':
            Zp = self.section_property.plast_sec_mod_y
            force = self.load.shear_force_yy
            Vd = self.V_d_yy
        else:
            Zp = self.section_property.plast_sec_mod_z
            force = self.load.shear_force_zz
            Vd = self.V_d_zz

        Zfd = ( Zp -
        (self.section_property.depth ** 2 * self.
section_property.web_thickness / 4)
        )
        Mfd = Zfd * self.material_property.fy / self.gamma_m0
        beta = ((2 * force / (Vd * 10 ** 3)) - 1) ** 2
        Mdv = (Md - beta * (Md - Mfd))
        print('Inside bending_strength_reduction', Mdv, Md,
beta, Mfd, Zfd)
```

```
        self.bending_strength_section_reduced_by = Mfd
        self.beta_reduced = beta
        if (
                Mdv
                <= 1.2
                * self.section_property.plast_sec_mod_z
                * self.material_property.fy
                / self.gamma_m0
        ):
            return Mdv
        else:
            return (
                    1.2
                    * self.section_property.plast_sec_mod_z
                    * self.material_property.fy
                    / self.gamma_m0
            )

    def Channels_Classification(self, depth, thickness_web,
f_y):
        epsilon = math.sqrt(250 / int(f_y))
        d_t = depth / thickness_web

        if d_t <= (42 * epsilon) :
            section_class = KEY_Plastic
        elif d_t <= (42 * epsilon) :
            section_class = KEY_Compact
        elif d_t <= (42 * epsilon) :
            section_class = KEY_SemiCompact
        else:
            section_class = 'Slender'

        return section_class
```

```
 def design_bending_strength_purlins(self,section_class,
Zp, Ze, fy, gamma_mo, high_shear_check, axis):
     beta_b = 1.0 if section_class == KEY_Plastic or
KEY_Compact else Ze / Zp
     Md = beta_b * Zp * fy / gamma_mo
     if Md < 1.2 * Ze * fy / gamma_mo:
         M_d =  Md
     else:
          M_d = 1.2 * Ze * fy / gamma_mo

     bending_strength_section = self.
bending_strength_reduction(self, M_d, axis)
     if high_shear_check:
         if self.section_class == KEY_Plastic or self.
section_class == KEY_Compact:
             bending_strength_section =
bending_strength_section
         else:
             bending_strength_section = Ze * fy / gamma_mo
     else:
         bending_strength_section = M_d

     return bending_strength_section

 def beam_M_cr_fcrb( self, E, meu,Iy, It, Iw, Llt,beta_b,
Zp, hf,ry, tf):
     G = E/(2+2*meu)
     fcrb = (1.1 * math.pi**2 * E/(Llt/ry)**2) * math.sqrt
(1 + (((Llt/ry)/(hf/tf))**2) / 20)
     M_cr_candidate1 = math.sqrt((math.pi**2 * E * Iy/Llt
**2)*(G *It + (math.pi**2 * E * Iw/Llt**2) ))
     M_cr_candidate2 = beta_b * Zp * fcrb
     return [min(M_cr_candidate1, M_cr_candidate2), fcrb]
```

```
    def non_slenderness_ratio_purlin(self, betab, Zp, Ze, fy,
     Mcr, fcrb = 0):
        if (betab * Zp * fy / Mcr) ** 0.5 <= (1.2 * Ze * fy /
     Mcr) ** 0.5:
            if (betab * Zp * fy / Mcr) ** 0.5 == math.sqrt(fy
/fcrb):
                return math.sqrt(fy/fcrb)
            else:
                return math.sqrt(fy/fcrb)
        else:
            return math.sqrt(fy/fcrb)

    def X_lt_calc(self, phi_lt, lambda_lt):
        si = 1 / ( phi_lt + (phi_lt **2 - lambda_lt **2) **
0.5)
        if si <= 1.0:
            return si
        else:
            return -1

    def web_resistance_check(self, m_inertia, Ze, Zp, rg,
axis ):
        [M_cr, fcrb] = self.beam_M_cr_fcrb(
            self,
            self.material_property.modulus_of_elasticity,
            0.3,
            m_inertia,
            self.It,
            self.Iw,
            self.effective_length * 1e3, self.beta_b_lt, Zp ,
     self.hf,
            rg, self.section_property.flange_thickness
        )
        if self.section_property.type == "Rolled":
            alpha_lt = 0.21
        else:
            alpha_lt = 0.49
        lambda_lt = self.non_slenderness_ratio_purlin(
            self,
            self.beta_b_lt,
            Zp, Ze,
            self.material_property.fy,
            M_cr,
            fcrb
        )
        phi_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_phi_lt(
            alpha_lt, lambda_lt
        )
        X_lt = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_stress_reduction_factor(
            phi_lt, lambda_lt
        )
        fbd = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_compressive_stress(
            X_lt, self.material_property.fy, self.gamma_m0
        )
        bending_strength_section = IS800_2007.
cl_8_2_2_Unsupported_beam_bending_strength(
            Zp,
            Ze,
            fcd=fbd,
            section_class=self.section_class,
        )
        if axis == 'y':
            self.M_cr_y = M_cr
            self.fcrb_y = fcrb
            self.lambda_lt_y = lambda_lt
            self.phi_lt_y = phi_lt
            self.X_lt_y = X_lt
            self.fbd_y = fbd
            self.imperfection_factor = alpha_lt
        elif axis == 'z':
            self.M_cr_z = M_cr
            self.fcrb_z = fcrb
            self.lambda_lt_z = lambda_lt
            self.phi_lt_z = phi_lt
            self.X_lt_z = X_lt
            self.fbd_z = fbd
            self.imperfection_factor = alpha_lt

        return bending_strength_section

    def buckling_resistance_check(self, M_y, M_z, M_d_y,
M_d_z):
        if (M_y/M_d_y) + M_z/M_d_z <= 1.0:
            return True
        else:
            return False

    def serviceability_check(self,w,Ze,mi):
        deflection = (5 * w * self.length**4) / (384 * Ze *
mi)
        return deflection

    def section_classification(self, design_dictionary,
trial_section=""):
        """Classify the sections based on Table 2 of IS
800:2007"""
        print(f"Inside section_classification")
        local_flag = True
        self.input_modified = []
        self.input_section_list = []
        self.input_section_classification = {}
        lambda_check = False
        for trial_section in self.sec_list:
            trial_section = trial_section.strip("'")
            self.section_property = self.
section_connect_database(self, trial_section)
            print(f"Type of section{self.section_property.
designation}")
            if self.section_property.type == "Rolled":
                self.effective_depth = (self.section_property
.depth - 2 * (
                        self.section_property.
flange_thickness + self.section_property.root_radius))

                web_ratio = self.effective_depth / self.
section_property.web_thickness

                web_class = self.Channels_Classification(
                    self,
                    self.section_property.depth - 2 * (
                            self.section_property.
flange_thickness + self.section_property.root_radius),
                    self.section_property.web_thickness,
                    self.material_property.fy,
                )

                flange_class = IS800_2007.Table2_i(
                    self.section_property.flange_width / 2,
                    self.section_property.flange_thickness,
                    self.material_property.fy, self.
section_property.type
                )[0]
                '''
                web_class = IS800_2007.Table2_iii(
                    self.section_property.depth - 2 * (
                                self.section_property.
flange_thickness + self.section_property.root_radius),
                    self.section_property.web_thickness,
                    self.material_property.fy,
                )
                '''
                flange_ratio = self.section_property.
flange_width / 2 / self.section_property.flange_thickness
            else:
                flange_class = IS800_2007.Table2_i(
                    (
                        (self.section_property.flange_width /
 2)
                        # - (self.section_property.
web_thickness / 2)
                    ),
                    self.section_property.flange_thickness,
                    self.section_property.fy,
                    self.section_property.type,
                )[0]

                web_class = self.Channels_Classification(
                    self,
                    (
                            self.section_property.depth - 2 *
 (
                                self.section_property.
flange_thickness)
                    ),
                    self.section_property.web_thickness,
                    self.material_property.fy,   #
classification_type="Axial compression",
                )
                self.effective_depth = (self.section_property
.depth - 2 * self.section_property.flange_thickness)

                web_ratio = (self.section_property.depth - 2
* self.section_property.flange_thickness) / self.
section_property.web_thickness



                flange_ratio = self.section_property.
flange_width / 2 / self.section_property.flange_thickness
            print(f"\n \n \n flange_class {flange_class} \n
web_class{web_class} \n \n")
            if flange_class == "Slender" or web_class == "
Slender":
                self.section_class = "Slender"
            else:
                if flange_class == KEY_Plastic and web_class
== KEY_Plastic:
                    self.section_class = KEY_Plastic
                elif flange_class == KEY_Plastic and
web_class == KEY_Compact:
                    self.section_class = KEY_Compact
                elif flange_class == KEY_Plastic and
web_class == KEY_SemiCompact:
                    self.section_class = KEY_SemiCompact
                elif flange_class == KEY_Compact and
web_class == KEY_Plastic:
                    self.section_class = KEY_Compact
                elif flange_class == KEY_Compact and
web_class == KEY_Compact:
                    self.section_class = KEY_Compact
                elif flange_class == KEY_Compact and
web_class == KEY_SemiCompact:
                    self.section_class = KEY_SemiCompact
                elif flange_class == KEY_SemiCompact and
web_class == KEY_Plastic:
                    self.section_class = KEY_SemiCompact
                elif flange_class == KEY_SemiCompact and
web_class == KEY_Compact:
                    self.section_class = KEY_SemiCompact
                elif flange_class == KEY_SemiCompact and
web_class == KEY_SemiCompact:
                    self.section_class = KEY_SemiCompact

            print(f"section class is {self.section_class}")
            self.Zp_req = self.load.moment * self.gamma_m0 /
self.material_property.fy

            self.effective_length = self.length

            if self.section_property.plast_sec_mod_z >= self.
Zp_req:

                self.It = self.section_property.It
                self.hf = self.section_property.depth - self.
section_property.flange_thickness
                self.Iw = self.section_property.Iw

                self.input_section_list.append(trial_section)
                self.input_section_classification.update(
                    {trial_section: [self.section_class,
flange_class,
                                     web_class, flange_ratio,
                                     web_ratio, self.It, self
.hf,
                                        self.Iw]})

        if len(self.input_section_list) == 0:
            local_flag = False
            logger.warning('No section passed found')
        else:
            local_flag = True
        return local_flag

    def effective_length_beam(self, design_dictionary, length
):
        print(f"Inside effective_length_beam")
        self.Loading = design_dictionary[KEY_LOAD]  # 'Normal
'or 'Destabilizing'
        # self.Latex_length = design_dictionary[
KEY_LENGTH_OVERWRITE]
        if design_dictionary[KEY_LENGTH_OVERWRITE] == 'NA':
            if self.support == KEY_DISP_SUPPORT1:
                self.Torsional_res = design_dictionary[
KEY_TORSIONAL_RES]
                self.Warping = design_dictionary[
KEY_WARPING_RES]
                self.effective_length = IS800_2007.
cl_8_3_1_EffLen_Simply_Supported(
                    Torsional=self.Torsional_res,
                    Warping=self.Warping,
                    length=length,
                    depth=self.section_property.depth,
                    load=self.Loading,
                )
                print(f"Working 1 {self.effective_length}")
            elif self.support == KEY_DISP_SUPPORT2:
                self.Support = design_dictionary[
KEY_SUPPORT_TYPE]
```

```
                self.Top = design_dictionary[
KEY_SUPPORT_TYPE2]
                self.effective_length = IS800_2007.
cl_8_3_3_EffLen_Cantilever(
                    Support=self.Support,
                    Top=self.Top,
                    length=length,
                    load=self.Loading,
                )
                print(f"Working 2 {self.effective_length}")
        else:
            if self.support == KEY_DISP_SUPPORT1:
                self.Torsional_res = design_dictionary[
KEY_TORSIONAL_RES]
                self.Warping = design_dictionary[
KEY_WARPING_RES]

            elif self.support == KEY_DISP_SUPPORT2:
                self.Support = design_dictionary[
KEY_SUPPORT_TYPE]
                self.Top = design_dictionary[
KEY_SUPPORT_TYPE2]

            try:
                if float(design_dictionary[
KEY_LENGTH_OVERWRITE]) <= 0:
                    design_dictionary[KEY_LENGTH_OVERWRITE] =
 'NA'
                else:
                    length = length * float(design_dictionary
[KEY_LENGTH_OVERWRITE])

                self.effective_length = length
                print(f"Working 3 {self.effective_length}")
            except:
                print(f"Inside effective_length_beam", type(
design_dictionary[KEY_LENGTH_OVERWRITE]))
                logger.warning("Invalid Effective Length
Parameter.")
                logger.info('Effective Length Parameter is
set to default: 1.0')
                design_dictionary[KEY_LENGTH_OVERWRITE] = '
1.0'
                self.effective_length_beam(self,
design_dictionary, length)
                print(f"Working 4 {self.effective_length}")
        print(f"Inside effective_length_beam", self.
effective_length, design_dictionary[KEY_LENGTH_OVERWRITE])

    def lambda_lt_check_member_type(self, Mcr=0, fcrb=0, Zp
=0, f_y=0, Ze=0, beta_b=0):
        lambda_lt_1 = math.sqrt(beta_b * Zp * f_y / Mcr)
        lambda_lt_2 = math.sqrt(f_y / fcrb)
        lambda_lt_check = math.sqrt(1.2 * Ze * f_y / Mcr)
        if lambda_lt_1 == lambda_lt_2:
            if lambda_lt_1 <= lambda_lt_check:
                return lambda_lt_1
        logger.warning(" Issues with the non-dimensional
slenderness ratio Lambda_lt")

    def common_checks_1(self, section, step=1, list_result
=[], list_1=[]):
        if step == 1:
            print(f"Working correct here")
        elif step == 2:
            # reduction of the area based on the connection
requirements (input from design preferences)
            if self.effective_area_factor < 1.0:
                self.effective_area = round(
                    self.effective_area * self.
effective_area_factor, 2
                )



        elif step == 3:
            # 2.1 - Buckling curve classification and
Imperfection factor
            if self.section_property.type == 'Rolled':
                self.buckling_class = 'c'
            self.imperfection_factor = IS800_2007.
cl_7_1_2_1_imperfection_factor(
                buckling_class=self.buckling_class
            )
        elif step == 4:
            # self.slenderness = self.effective_length / min(
self.section_property.rad_of_gy_z, self.section_property.
rad_of_gy_y) * 1000
            print(
                f"\n data sent "
                f" self.material_property.fy {self.
material_property.fy}"
                f"self.gamma_m0 {self.gamma_m0}"
                f"self.slenderness {self.slenderness}"
                f" self.imperfection_factor {self.
imperfection_factor}"
                f"self.section_property.modulus_of_elasticity
 {self.section_property.modulus_of_elasticity}"
            )

            list_cl_7_1_2_1_design_compressisive_stress = (
                IS800_2007.
cl_7_1_2_1_design_compressisive_stress(
                    self.material_property.fy,
                    self.gamma_m0,
                    self.slenderness,
                    self.imperfection_factor,
                    self.section_property.
modulus_of_elasticity,
                    check_type=list_result,
                )
            )
            for x in
list_cl_7_1_2_1_design_compressisive_stress:
                print(f"x {x} ")
            self.euler_buckling_stress =
list_cl_7_1_2_1_design_compressisive_stress[0]
            self.nondimensional_effective_slenderness_ratio =
 (
                list_cl_7_1_2_1_design_compressisive_stress
[1]
            )
            self.phi =
list_cl_7_1_2_1_design_compressisive_stress[2]
            self.stress_reduction_factor =
list_cl_7_1_2_1_design_compressisive_stress[
                3
            ]
            self.design_compressive_stress_fr = (
                list_cl_7_1_2_1_design_compressisive_stress
[4]
            )
            self.design_compressive_stress = (
                list_cl_7_1_2_1_design_compressisive_stress
[5]
            )
            self.design_compressive_stress_max = (
                list_cl_7_1_2_1_design_compressisive_stress
[6]
            )
        elif step == 5:
            # 1- Based on optimum UR
            if not "UR" in list_1:
                self.ur = 0
                self.cost = (
                        (
                                self.section_property.
unit_mass
                                * self.section_property.area
                                * 1e-4
                        )
                        * self.length
                        * self.steel_cost_per_kg
                )
                list_1.extend(["UR","Cost"])
                list_result.extend([self.ur, self.cost])

            self.optimum_section_ur_results[self.ur] = {}
            list_2 = list_result.copy()
            for j in list_1:
                # k = 0
                for k in list_2:
                    self.optimum_section_ur_results[self.ur][
j] = k
                    # k += 1
                    list_2.pop(0)
                    break

            # 2- Based on optimum cost
            self.optimum_section_cost_results[self.cost] = {}
            list_2 = list_result.copy()  # Why?
            for j in list_1:
                for k in list_2:
                    self.optimum_section_cost_results[self.
cost][j] = k
                    list_2.pop(0)
                    break
            print(
                f"\n self.optimum_section_cost_results {self.
optimum_section_cost_results}"
                f"\n self.optimum_section_ur_results {self.
optimum_section_ur_results}"
            )
        elif step == 6:
            self.single_result[self.sec_profile] = {}
            list_2 = list_result.copy()
            for j in list_1:
                # k = 0
                for k in list_2:
                    self.single_result[self.sec_profile][j] =
 k
                    # k += 1
                    list_2.pop(0)
                    break
            print(f"\n self.single_result {self.single_result
}")

    def list_changer(self, change, list, list_name, check=
True):

        list.extend(
            [self.bending_strength_section_reduced_by, self.
beta_reduced, self.M_d_zz, self.M_d_yy])
        list_name.extend([
            "Mfd",
            "Beta_reduced",
            "M_d_zz",
            "M_d_yy"
        ])
        # Latex para also
        list.extend(
            [self.web_buckling_check, self.effective_depth,
self.web_buckling_check1,
             self.section_class, self.effective_area, self.
V_d_yy, self.V_d_zz, self.high_shear_check_yy,
             self.high_shear_check_zz, self.M_d_yy1, self.
M_d_zz1, self.effective_length,
             self.ur, self.cost, self.beta_b_lt, self.
buckling_class])
        list_name.extend([
            'Web.Buckling',
            'Reduced.depth',
            'Buckling.resistance',
            "Section class",
            "Effective area",
            "Shear Strength YY",
            "Shear Strength ZZ",
            "High Shear check YY",
            "High Shear check ZZ",
            "Bending Strength YY",
            "Bending Strength ZZ",
            "Effective_length",
            "UR",
            "Cost",
            "Beta_b",
            "Buckling.class"
        ])
        # Web buckling parameters
```

```
        # if self.web_buckling_check and (self.
    support_cndition_shear_buckling == KEY_DISP_SB_Option[0]
    or self.support_cndition_shear_buckling ==
    KEY_DISP_SB_Option[1] ) :
        #     list.extend(
        #         [self.K_v, self.tau_crc, self.lambda_w,
    self.tau_b,
        #          self.V_cr])
        #     list_name.extend([
        #         'Kv',
        #         'tau_crc',
        #         'lambda_w',
        #         'tau_b',
        #         "V_cr"
        #     ])

        # if self.support_cndition_shear_buckling ==
    KEY_DISP_SB_Option[1] and self.web_buckling_check:
        #     list.extend(
        #         [self.Mfr, self.load.moment / (
        #                 self.section_property.depth - self.
    section_property.flange_thickness)
        #             , self.c, self.phi_girder, self.
    s_girder, self.wtf_girder, self.sai_girder, self.fv_girder
    ,
        #          self.V_p, self.V_tf_girder])
        #     list_name.extend([
        #         'Mfr',#1
        #         'Nf',
        #         'c',
        #         'phi_girder',
        #         "s_girder",
        #         'wtf_girder',
        #         'sai_girder',
        #           'fv_girder',
        #           'V_p',
        #           'V_tf_girder'
        #       ])
        # if change == 'Web Buckling':
        #       list.extend([self.I_eff_web, self.A_eff_web,
    self.r, self.buckling_class,
        #                       self.imperfection_factor,
        #                       self.slenderness,
        #                       self.euler_buckling_stress,
        #                       self.
    nondimensional_effective_slenderness_ratio,
        #                       self.phi,
        #                       self.stress_reduction_factor,
        #                       self.design_compressive_stress_fr,
        #                       self.design_compressive_stress_max
    ,
        #                       self.design_compressive_stress,
        #                       self.section_capacity,
        #                       self.F_wb])
        #
        #       list_name.extend([
        #           "WebBuckling.I_eff",#1
        #           "WebBuckling.A_eff",#1
        #           "WebBuckling.r_eff",#1
        #           "Buckling_class",#0
        #           "IF",#0
        #           "Effective_SR",
        #           "EBS",
        #           "ND_ESR",
        #           "phi",
        #           "SRF",
        #           "FCD_formula",
        #           "FCD_max",
        #         "FCD",
        #         "Capacity",
        #         "Web_crippling"
        #     ])


        list.extend([self.It,
                     self.Iw,
                     self.imperfection_factor,
                     self.lambda_lt_y,
                     self.lambda_lt_z,
                     self.phi_lt_y,
                     self.phi_lt_z,
                     self.X_lt_y,
                     self.X_lt_z,
                     self.fbd_y,
                     self.fbd_z,
                     self.M_cr_y,
                     self.M_cr_z,
                     self.fcrb_y,
                     self.fcrb_z])

        list_name.extend([
            "It",
            "Iw",
            "IF_lt",
            "lambda_lt_yy",
            "lambda_lt_zz",
            "phi_lt_yy",
            "phi_lt_zz",
            "X_lt_yy",
            "X_lt_zz",
            "Fbd_yy",
            "Fbd_zz",
            "Mcr_yy",
            "Mcr_zz",
            "Fcrb_yy",
            "Fcrb_zz"
        ])
        print(list,list_name)
        return list, list_name

 # def plate_girder_design(self, section):
 #     if self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[0]:
 #         self.tau_crc = IS800_2007.
cl_8_4_2_2_tau_crc_Simple_postcritical(self.K_v,
 #
                  self.material_property.
modulus_of_elasticity,
 #
                  0.3,self.effective_depth,
 #
                  self.section_property.web_thickness)
 #         self.lambda_w = IS800_2007.
cl_8_4_2_2_lambda_w_Simple_postcritical(self.fyw,self.
tau_crc)
 #         self.tau_b = IS800_2007.
cl_8_4_2_2_tau_b_Simple_postcritical(self.lambda_w, self.
fyw)
 #         self.V_cr = IS800_2007.
cl_8_4_2_2_Vcr_Simple_postcritical(self.tau_b, self.
effective_depth * self.section_property.web_thickness)
 # d_red = self.section_property.depth - 2*(self.
section_property.flange_thickness + self.section_property.
root_radius)
 # tau_b = self.load.shear_force / (self.effective_depth *
 self.section_property.web_thickness)
```

```
    # if tau_b <= self.fyw / math.sqrt(3):
    #      lambda_w = 0.8
    # else:
    #      lambda_w = min((tau_b*(math.sqrt(3)/self.fyw) -
   1.64) / (-0.8), math.sqrt(tau_b*(math.sqrt(3)/self.fyw)))
    # tau_crc = self.fyw / (math.sqrt(3) * lambda_w ** 2)

    def plate_girder_strength(self):
        self.tau_crc = IS800_2007.
   cl_8_4_2_2_tau_crc_Simple_postcritical(self.K_v,

               self.material_property.modulus_of_elasticity,

               0.3, self.effective_depth,

               self.section_property.web_thickness)
        self.lambda_w = IS800_2007.
   cl_8_4_2_2_lambda_w_Simple_postcritical(self.fyw, self.
   tau_crc)
        self.tau_b = IS800_2007.
   cl_8_4_2_2_tau_b_Simple_postcritical(self.lambda_w, self.
   fyw)
        self.V_cr = IS800_2007.
   cl_8_4_2_2_Vcr_Simple_postcritical(self.tau_b,

        self.effective_depth * self.section_property.
   web_thickness) / 10 ** 3
        print('\n plate_girder_strength', '\n tau_crc', self.
   tau_crc, '\n self.lambda_w', self.lambda_w,
             '\n self.tau_b', self.tau_b, '\n self.V_cr',
   self.V_cr)

    def plate_girder_strength2(self):
        self.plate_girder_strength(self)
        self.phi_girder, self.M_fr_girder, self.s_girder,
self.wtf_girder, self.sai_girder, self.fv_girder, self.
V_tf_girder = IS800_2007.cl_8_4_2_2_TensionField(
            self.c,
            self.effective_depth, self.section_property.
web_thickness,
            self.fyw, self.section_property.flange_width,
            self.section_property.flange_thickness, self.fyf,
            self.load.moment / (self.section_property.depth -
 self.section_property.flange_thickness),
            self.gamma_m0, self.effective_depth * self.
section_property.web_thickness, self.tau_b, self.V_p)

    def results(self, design_dictionary):
        _ = [i for i in self.optimum_section_ur if i < 1.0]
        if len(_) == 1:
            temp = _[0]
        elif len(_) == 0:
            temp = None
        else:
            temp = sorted(_)[0]
        self.failed_design_dict = self.
optimum_section_ur_results[temp] if temp is not None else
None
        print('self.failed_design_dict ', self.
failed_design_dict)

        # sorting results from the dataset
        # if len(self.input_section_list) > 1:
        # results based on UR
        if self.optimization_parameter == "Utilization Ratio"
:
            filter_UR = filter(
                lambda x: x <= min(self.
allowable_utilization_ratio, 1.0),
                self.optimum_section_ur
            )
            self.optimum_section_ur = list(filter_UR)

            self.optimum_section_ur.sort()
            print(
                f"self.optimum_section_ur{self.
optimum_section_ur} \n self.optimum_section_ur_results{
self.optimum_section_ur_results}")
            # print(f"self.result_UR{self.result_UR}")

            # selecting the section with most optimum UR
            if len(self.optimum_section_ur) == 0:  # no
design was successful
                logger.warning(
                    "The sections selected by the solver from
 the defined list of sections did not satisfy the
Utilization Ratio (UR) "
                    "criteria"
                )
                logger.error(
                    "The solver did not find any adequate
section from the defined list."
                )

                self.design_status = False

                if self.failed_design_dict is None:

                    logger.info(
                        "The details for the best section
provided is being shown"
```

```
                    )
                    self.failed_design_dict = self.
optimum_section_ur_results
                    ur_vals = self.optimum_section_ur_results
.keys()
                    self.result_UR = min(ur_vals)
                    self.design_status = True

                    self.common_result(
                        self,
                        list_result = self.
optimum_section_ur_results,
                        result_type = self.result_UR,
                        flag = True
                    )

                elif len(self.failed_design_dict) > 0:
                    logger.info(
                        "The details for the best section
provided is being shown"
                    )
                    self.result_UR = self.failed_design_dict[
'UR']  # temp  TODO @Rutvik
                    self.common_result(
                        self,
                        list_result=self.failed_design_dict,
                        result_type=None,
                    )
                    logger.warning(
                        "Re-define the list of sections or
check the Design Preferences option and re-design."
                    )
                else:
                    logger.warning(
                        "Plastic section modulus of selected
sections is less than required."
                    )
                    return
                # self.design_status_list.append(self.
design_status)

            else:
                self.failed_design_dict = None
                self.result_UR = self.optimum_section_ur[
                    -1
                ]  # optimum section which passes the UR
check
                print(f"self.result_UR{self.result_UR}")
                self.design_status = True
                self.common_result(
                    self,
                    list_result=self.
optimum_section_ur_results,
                    result_type=self.result_UR,
                )

        else:  # results based on cost
            self.optimum_section_cost.sort()

            # selecting the section with most optimum cost
            self.result_cost = self.optimum_section_cost[0]
            self.design_status = True
        # print results
        # if len(self.optimum_section_ur) == 0:
        #     logger.warning(
        #         "The sections selected by the solver from
the defined list of sections did not satisfy the
Utilization Ratio (UR) "
```

```
    #         "criteria"
    #     )
    #     logger.error(
    #         "The solver did not find any adequate
section from the defined list."
    #     )
    #     logger.info(
    #         "Re-define the list of sections or check
the Design Preferences option and re-design."
    #     )
    #     self.design_status = False
    #     self.design_status_list.append(self.
design_status)
    #     pass
    # else:
    #     if self.optimization_parameter == "Utilization
Ratio":
    #         self.common_result(
    #             self,
    #             list_result=self.
optimum_section_ur_results,
    #             result_type=self.result_UR,
    #         )
    #     else:
    #         self.result_UR = self.
optimum_section_cost_results[
    #             self.result_cost
    #         ]["UR"]
    #
    #         # checking if the selected section based on
 cost satisfies the UR
    #         if self.result_UR > min(self.
allowable_utilization_ratio, 1.0):
    #             trial_cost = []
```

```
     #              for cost in self.optimum_section_cost:
     #                  self.result_UR = self.
optimum_section_cost_results[
     #                      cost
     #                  ]["UR"]
     #                  if self.result_UR <= min(
     #                      self.
allowable_utilization_ratio, 1.0
     #                  ):
     #                      trial_cost.append(cost)
     #
     #              trial_cost.sort()
     #
     #              if len(trial_cost) == 0:  # no design
was successful
     #                  logger.warning(
     #                      "The sections selected by the
solver from the defined list of sections did not satisfy
the Utilization Ratio (UR) "
     #                      "criteria"
     #                  )
     #                  logger.error(
     #                      "The solver did not find any
adequate section from the defined list."
     #                  )
     #                  logger.info(
     #                      "Re-define the list of sections
 or check the Design Preferences option and re-design."
     #                  )
     #                  self.design_status = False
     #                  self.design_status_list.append(self
.design_status)
     #                  print(f"design_status_list{self.
design_status} \n")
```

```
        #               else:
        #                   self.result_cost = trial_cost[
        #                       0
        #                   ]  # optimum section based on cost
    which passes the UR check
        #                   self.design_status = True
        #
        #           # results
        #           self.common_result(
        #               self,
        #               list_result=self.
    optimum_section_cost_results,
        #               result_type=self.result_cost,
        #           )
        #
        #           print(f"design_status_list2{self.
    design_status}")
        self.design_status_list.append(self.design_status)
        for status in self.design_status_list:
            print('status list', status)
            if status is False:
                self.design_status = False
                break
            else:
                self.design_status = True

     def common_result(self, list_result, result_type, flag=
    False):
        if flag:
            self.result_designation = list_result[result_type
    ]["Designation"]
            self.result_web_buckling_check = list_result[
    result_type]["Web.Buckling"]
```

```
            self.result_section_class = list_result[
result_type]["Section class"]
            self.result_tc = round(list_result[result_type]["
It"], 2)
            self.result_wc = round(list_result[result_type]["
Iw"], 2)
            self.result_eff_d = 'NA'
            self.result_buckling_resistance = 'NA'
            self.result_effective_area = 'NA'
            self.result_shear_yy = 'NA'
            self.result_shear_zz = 'NA'
            self.result_high_shear_yy = 'NA'
            self.result_high_shear_zz = 'NA'
            self.result_bending_yy = 'NA'
            self.result_bending_zz = 'NA'
            self.result_eff_len = 'NA'
            self.result_cost = list_result[result_type]["Cost
"]
            self.result_betab = list_result[result_type]["
Beta_b"]
            self.result_mcr_yy = 'NA'
            self.result_mcr_zz = 'NA'
            self.result_IF_lt = 'NA'
            self.result_tc = 'NA'
            self.result_wc = 'NA'
            self.result_phi_lt_zz = 'NA'
            self.result_phi_lt_yy = 'NA'
            self.result_lambda_lt_yy = 'NA'
            self.result_lambda_lt_zz = 'NA'
            self.result_X_lt_yy = 'NA'
            self.result_X_lt_zz = 'NA'
            self.result_Fbd_yy = 'NA'
            self.result_Fbd_zz = 'NA'
            self.result_Fcrb_yy = 'NA'
            self.result_Fcrb_zz = 'NA'
            self.result_resistance_bending_yy = 'NA'
            self.result_resistance_bending_zz = 'NA'
            self.result_buckling_class = 'NA'
            return

        try:
            self.result_designation = list_result[result_type
][ 'Designation' ]  # TODO debug
            logger.info(
                "The section is {}. The {} section  has  {}
flange({}) and  {} web({}).  [Reference: Cl 3.7, IS
800:2007].".format(
                    self.input_section_classification[self.
result_designation][0],
                    self.result_designation,
                    self.input_section_classification[self.
result_designation][1],
                    round(self.input_section_classification[
self.result_designation][3], 2),
                    self.input_section_classification[self.
result_designation][2],
                    round(self.input_section_classification[
self.result_designation][4], 2)
                )
            )
            # self.result_latex_tension_zone = list_result[
result_type]["latex.tension_zone"]
            self.result_web_buckling_check = list_result[
result_type]["Web.Buckling"]
            print('self.result_web_buckling_check', self.
result_web_buckling_check)
            self.result_eff_d = list_result[result_type]["
Reduced.depth"]
```

```
            self.result_buckling_resistance = list_result[
result_type]["Buckling.resistance"]

            self.result_section_class = list_result[
result_type]["Section class"]
            self.result_effective_area = round(list_result[
result_type]["Effective area"], 2)
            if self.effective_area_factor < 1.0:
                logger.info(
                    "The actual effective area is {} mm2 and
the reduced effective area is {} mm2 [Reference: Cl.
7.3.2, IS 800:2007]".format(
                        round((self.result_effective_area /
self.effective_area_factor), 2),
                        self.result_effective_area,
                    )
                )

            self.result_shear_yy = round(list_result[
result_type]["Shear Strength YY"], 2)
            self.result_shear_zz = round(list_result[
result_type]["Shear Strength ZZ"], 2)
            self.result_high_shear_yy = list_result[
result_type]["High Shear check YY"]
            self.result_high_shear_zz = list_result[
result_type]["High Shear check ZZ"]
            self.result_bending_yy = round(list_result[
result_type]["M_d_yy"], 2)
            self.result_bending_zz = round(list_result[
result_type]["M_d_zz"], 2)
            self.result_eff_len = round(list_result[
result_type]["Effective_length"], 2)
```

```
            self.result_cost = list_result[result_type]["Cost
"]
            self.result_betab = list_result[result_type]["
Beta_b"]
            self.result_buckling_class = list_result[
result_type]["Buckling.class"]

            if self.result_web_buckling_check:
                logger.warning(
                    "Thin web so take flange to resist moment
 and web to resist shear[Reference: Cl 8.2.1.1, IS
800:2007]")


            self.result_mcr_yy = round(list_result[
result_type]['Mcr_yy'], 2)
            self.result_mcr_zz = round(list_result[
result_type]['Mcr_zz'], 2)
            self.result_IF_lt = round(list_result[result_type
]["IF_lt"], 2)
            self.result_tc = round(list_result[result_type]["
It"], 2)
            self.result_wc = round(list_result[result_type]["
Iw"], 2)
            self.result_phi_lt_zz = round(list_result[
result_type]["phi_lt_zz"], 2)
            self.result_phi_lt_yy = round(list_result[
result_type]["phi_lt_yy"], 2)
            self.result_lambda_lt_yy = round(list_result[
result_type]["lambda_lt_yy"], 2)
            self.result_lambda_lt_zz = round(list_result[
result_type]["lambda_lt_zz"], 2)
            self.result_X_lt_yy = round(list_result[
result_type]["X_lt_yy"], 2)
```

```
            self.result_X_lt_zz = round(list_result[
result_type]["X_lt_zz"], 2)
            self.result_Fbd_yy = round(list_result[
result_type]["Fbd_yy"], 2)
            self.result_Fbd_zz = round(list_result[
result_type]["Fbd_zz"], 2)
            self.result_Fcrb_yy = round(list_result[
result_type]["Fcrb_yy"], 2)
            self.result_Fcrb_zz = round(list_result[
result_type]["Fcrb_zz"], 2)
            self.result_resistance_bending_yy = round(
list_result[result_type]["Bending Strength YY"], 2)
            self.result_resistance_bending_zz = round(
list_result[result_type]["Bending Strength ZZ"], 2)

            # if self.web_buckling:
            #
            #     self.result_bcI_eff = list_result[
result_type]['WebBuckling.I_eff']
            #     self.result_bcA_eff = list_result[
result_type]['WebBuckling.A_eff']
            #     self.result_bcr_eff = list_result[
result_type]['WebBuckling.r_eff']
            #     self.result_bc = list_result[result_type]['
Buckling_class']
            #     self.result_IF = round(list_result[
result_type]["IF"], 2)
            #     self.result_eff_sr = round(list_result[
result_type]["Effective_SR"], 2)
            #     self.result_ebs = round(list_result[
result_type]["EBS"], 2)
            #     self.result_nd_esr = round(list_result[
result_type]["ND_ESR"], 2)
```

```
        #     self.result_phi_zz = round(list_result[
result_type]["phi"], 2)
        #     self.result_srf = round(list_result[
result_type]["SRF"], 2)
        #     self.result_fcd_1_zz = round(list_result[
result_type]["FCD_formula"], 2)
        #     self.result_fcd_2 = round(list_result[
result_type]["FCD_max"], 2)
        #     self.result_fcd = round(list_result[
result_type]["FCD"], 2)
        #     self.result_capacity = round(list_result[
result_type]["Capacity"], 2)
        #     self.result_crippling = round(list_result[
result_type]["Web_crippling"], 2)
        # else:
        #     self.result_bc = 'NA'
        #     self.result_IF = 'NA'
        #     self.result_eff_sr = 'NA'
        #     self.result_lambda_vv = 'NA'
        #     self.result_lambda_psi = 'NA'
        #     self.result_ebs = 'NA'
        #     self.result_nd_esr = 'NA'
        #     self.result_phi_zz = 'NA'
        #     self.result_srf = 'NA'
        #     self.result_fcd_1_zz = 'NA'
        #     self.result_fcd_2 = 'NA'
        #     self.result_fcd = 'NA'
        #     self.result_capacity = 'NA'
        #     self.result_crippling = 'NA'
        # if self.result_high_shear and self.
input_section_classification[self.result_designation][
        #     0] != 'Semi-Compact':
        #     self.result_mfd = list_result[result_type
]["Mfd"]
```

```
            #     self.result_beta_reduced = list_result[
result_type]["Beta_reduced"]
            #     self.result_Md = list_result[result_type]["
M_d"]
        except:
            self.result_designation = list_result["
Designation"]
            # logger.info(
            #     "The section is {}. The {} section  has  {}
 flange({}) and  {} web({}).  [Reference: Cl 3.7, IS
800:2007].".format(
            #         self.input_section_classification[self.
result_designation][0],
            #         self.result_designation,
            #         self.input_section_classification[self.
result_designation][1],
            #         round(self.input_section_classification
[self.result_designation][3], 2),
            #         self.input_section_classification[self.
result_designation][2],
            #         round(self.input_section_classification
[self.result_designation][4], 2)
            #     )
            # )
            self.result_web_buckling_check = list_result["Web
.Buckling"]
            self.result_eff_d = list_result["Reduced.depth"]
            self.result_buckling_crippling = list_result["
Buckling.resistance"]

            self.result_section_class = list_result["Section
class"]
            self.result_effective_area = round(list_result["
Effective area"], 2)
```

```
        if self.effective_area_factor < 1.0:
            logger.info(
                "The actual effective area is {} mm2 and
the reduced effective area is {} mm2 [Reference: Cl.
7.3.2, IS 800:2007]".format(
                    round((self.result_effective_area /
self.effective_area_factor), 2),
                    self.result_effective_area,
                )
            )

        self.result_shear_yy = round(list_result["Shear
Strength YY"], 2)
        self.result_shear_zz = round(list_result["Shear
Strength ZZ"], 2)
        self.result_high_shear_yy = list_result["High
Shear check YY"]
        self.result_high_shear_zz = list_result["High
Shear check ZZ"]
        self.result_bending_yy = round(list_result["
M_d_yy"], 2)
        self.result_bending_zz = round(list_result["
M_d_zz"], 2)
        self.result_eff_len = round(list_result["
Effective_length"], 2)
        self.result_cost = list_result["Cost"]
        self.result_betab = list_result["Beta_b"]
        self.result_buckling_class = list_result["
Buckling.class"]

        if self.result_web_buckling_check:
            logger.warning(
                "Thin web so take flange to resist moment
 and web to resist shear[Reference: Cl 8.2.1.1, IS
800:2007]")

            self.result_mcr_yy = round(list_result['Mcr_yy'],
 2)
            self.result_mcr_zz = round(list_result['Mcr_zz'],
 2)
            self.result_IF_lt = round(list_result["IF_lt"],
2)
            self.result_tc = round(list_result["It"], 2)
            self.result_wc = round(list_result["Iw"], 2)
            self.result_phi_lt_zz = round(list_result["
phi_lt_zz"], 2)
            self.result_phi_lt_yy = round(list_result["
phi_lt_yy"], 2)
            self.result_lambda_lt_yy = round(list_result["
lambda_lt_yy"], 2)
            self.result_lambda_lt_zz = round(list_result["
lambda_lt_zz"], 2)
            self.result_X_lt_yy = round(list_result["X_lt_yy"
], 2)
            self.result_X_lt_zz = round(list_result["X_lt_zz"
], 2)
            self.result_Fbd_yy = round(list_result["Fbd_yy"],
 2)
            self.result_Fbd_zz = round(list_result["Fbd_zz"],
 2)
            self.result_Fcrb_yy = round(list_result["Fcrb_yy"
], 2)
            self.result_Fcrb_zz = round(list_result["Fcrb_zz"
], 2)
            self.result_resistance_bending_yy = round(
list_result["Bending Strength YY"], 2)
            self.result_resistance_bending_zz = round(
list_result["Bending Strength ZZ"], 2)
```

```
            # if self.design_type ==
KEY_DISP_DESIGN_TYPE2_FLEXURE:
            #       self.result_mcr = round(list_result['Mcr'],
 2)
            #       self.result_IF_lt = round(list_result["
IF_lt"], 2)
            #       self.result_tc = round(list_result["It"],
2)
            #       self.result_wc = round(list_result["Iw"],
2)
            #       self.result_nd_esr_lt = round(list_result["
ND_ESR_lt"], 2)
            #       self.result_phi_lt = round(list_result["
phi_lt"], 2)
            #       self.result_srf_lt = round(list_result["
SRF_lt"], 2)
            #       self.result_fcd__lt = round(list_result["
FCD_lt"], 2)
            # else:
            #       self.result_mcr = 'NA'
            #       self.result_IF_lt = 'NA'
            #       self.result_tc = 'NA'
            #       self.result_wc = 'NA'
            #       self.result_nd_esr_lt = 'NA'
            #       self.result_phi_lt = 'NA'
            #       self.result_srf_lt = 'NA'
            #       self.result_fcd__lt = 'NA'
            #
            # if self.web_buckling:
            #
            #       self.result_bcI_eff = list_result['
WebBuckling.I_eff']
```

```
            #       self.result_bcA_eff = list_result['
WebBuckling.A_eff']
            #       self.result_bcr_eff = list_result['
WebBuckling.r_eff']
            #       self.result_bc = list_result['
Buckling_class']
            #       self.result_IF = round(list_result["IF"],
2)
            #       self.result_eff_sr = round(list_result["
Effective_SR"], 2)
            #       self.result_ebs = round(list_result["EBS"],
 2)
            #       self.result_nd_esr = round(list_result["
ND_ESR"], 2)
            #       self.result_phi_zz = round(list_result["phi
"], 2)
            #       self.result_srf = round(list_result["SRF"],
 2)
            #       self.result_fcd_1_zz = round(list_result["
FCD_formula"], 2)
            #       self.result_fcd_2 = round(list_result["
FCD_max"], 2)
            #       self.result_fcd = round(list_result["FCD"],
 2)
            #       self.result_capacity = round(list_result["
Capacity"], 2)
            #       self.result_crippling = round(list_result["
Web_crippling"], 2)
            # else:
            #       self.result_bc = 'NA'
            #       self.result_IF = 'NA'
            #       self.result_eff_sr = 'NA'
            #       self.result_lambda_vv = 'NA'
            #       self.result_lambda_psi = 'NA'
```

```
          #     self.result_ebs = 'NA'
          #     self.result_nd_esr = 'NA'
          #     self.result_phi_zz = 'NA'
          #     self.result_srf = 'NA'
          #     self.result_fcd_1_zz = 'NA'
          #     self.result_fcd_2 = 'NA'
          #     self.result_fcd = 'NA'
          #     self.result_capacity = 'NA'
          #     self.result_crippling = 'NA'
          # if self.result_high_shear and self.
input_section_classification[self.result_designation][
          #     0] != 'Semi-Compact':
          #     self.result_mfd = list_result["Mfd"]
          #     self.result_beta_reduced = list_result["
Beta_reduced"]
          #     self.result_Md = list_result["M_d"]

 ### start writing save_design from here!
 def save_design(self, popup_summary):
     # print('self.design_status', self.design_status,'len
(self.failed_design_dict)', len(self.failed_design_dict))
     if (self.design_status and self.failed_design_dict is
 None) or (
             not self.design_status and len(self.
failed_design_dict) > 0):  # TODO @Rutvik
         self.section_property = self.
section_connect_database(self, self.result_designation)
         if self.sec_profile == 'Columns' or self.
sec_profile == 'Beams' or self.sec_profile ==
VALUES_SECTYPE[1]:
             self.report_column = {KEY_DISP_SEC_PROFILE: "
ISection",
                                   KEY_DISP_SECSIZE: (self
.section_property.designation, self.sec_profile),
                                    KEY_DISP_COLSEC_REPORT:
 self.section_property.designation,
                                    KEY_DISP_MATERIAL: self
.section_property.material,
                                    #
           KEY_DISP_APPLIED_AXIAL_FORCE: self.
section_property.,
                                    KEY_REPORT_MASS: self.
section_property.mass,
                                    KEY_REPORT_AREA: round(
self.section_property.area * 1e-2, 2),
                                    KEY_REPORT_DEPTH: self.
section_property.depth,
                                    KEY_REPORT_WIDTH: self.
section_property.flange_width,
                                    KEY_REPORT_WEB_THK:
self.section_property.web_thickness,
                                    KEY_REPORT_FLANGE_THK:
self.section_property.flange_thickness,

KEY_DISP_FLANGE_S_REPORT: self.section_property.
flange_slope,
                                    KEY_REPORT_R1: self.
section_property.root_radius,
                                    KEY_REPORT_R2: self.
section_property.toe_radius,
                                    KEY_REPORT_IZ: round(
self.section_property.mom_inertia_z * 1e-4, 2),
                                    KEY_REPORT_IY: round(
self.section_property.mom_inertia_y * 1e-4, 2),
                                    KEY_REPORT_RZ: round(
self.section_property.rad_of_gy_z * 1e-1, 2),
                                    KEY_REPORT_RY: round(
self.section_property.rad_of_gy_y * 1e-1, 2),
                                            KEY_REPORT_ZEZ: round(
self.section_property.elast_sec_mod_z * 1e-3, 2),
                                            KEY_REPORT_ZEY: round(
self.section_property.elast_sec_mod_y * 1e-3, 2),
                                            KEY_REPORT_ZPZ: round(
self.section_property.plast_sec_mod_z * 1e-3, 2),
                                            KEY_REPORT_ZPY: round(
self.section_property.plast_sec_mod_y * 1e-3, 2)}

         self.report_input = \
             {  # KEY_MAIN_MODULE: self.mainmodule,
                 KEY_MODULE: self.module,  # "Axial load
on column "
                 KEY_DISP_SHEAR + '*': self.load.
shear_force * 10 ** -3,
                 KEY_DISP_BEAM_MOMENT_Latex + '*': self.
load.moment * 10 ** -6,
                 KEY_DISP_LENGTH_BEAM: self.result_eff_len
,
                 KEY_DISP_SEC_PROFILE: self.sec_profile,
                 KEY_DISP_SECSIZE: str(self.sec_list),
                 KEY_MATERIAL: self.material,
                 "Selected Section Details": self.
report_column,
                 KEY_BEAM_SUPP_TYPE: self.
latex_design_type,
             }

         # if self.latex_design_type ==
VALUES_SUPP_TYPE_temp[0]:
         #     self.report_input.update({
         #         KEY_DISP_BENDING: self.bending_type})
         # elif self.latex_design_type ==
VALUES_SUPP_TYPE_temp[1]:
```

```
            #     self.report_input.update({
            #         KEY_BEAM_SUPP_TYPE_DESIGN: self.support
,
            #         # KEY_DISP_BENDING: self.bending_type,
            #     })
            self.report_input.update({
                KEY_DISP_SUPPORT: self.support,
                KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.
material_property.fu,
                KEY_DISP_YIELD_STRENGTH_REPORT: self.
material_property.fy,
                "End Conditions - " + str(self.support): "
TITLE",
            })
            # if self.Latex_length == 'NA':
            if self.support == KEY_DISP_SUPPORT1:
                self.report_input.update({
                    DISP_TORSIONAL_RES: self.Torsional_res,
                    DISP_WARPING_RES: self.Warping})
            else:
                self.report_input.update({
                    DISP_SUPPORT_RES: self.Support,
                    DISP_TOP_RES: self.Top})
            self.report_input.update({
                "Design Preference": "TITLE",
                KEY_DISP_EFFECTIVE_AREA_PARA: self.
effective_area_factor,
                KEY_DISP_CLASS: self.allow_class,
                KEY_DISP_LOAD: self.Loading,
                KEY_DISPP_LENGTH_OVERWRITE: self.latex_efp,
                KEY_DISP_BEARING_LENGTH + ' (mm)': self.
bearing_length,

            })
            # if self.latex_design_type ==
VALUES_SUPP_TYPE_temp[0] and self.
result_web_buckling_check:
            #     self.report_input.update({
            #         KEY_ShearBuckling: self.
support_cndition_shear_buckling
            #     })
            # self.report_input.update({
            #       # KEY_DISP_SEC_PROFILE: self.sec_profile,
            #       "I Section - Mechanical Properties": "
TITLE",
            #       })
            self.report_input.update()
            self.report_check = []

            t1 = ('Selected', 'Selected Member Data', '|p{5cm
}|p{2cm}|p{2cm}|p{2cm}|p{4cm}|')
            self.report_check.append(t1)

            t1 = ('SubSection', 'Effective Area', '|p{4cm}|p
{1.5cm}|p{9.5cm}|p{1cm}|')
            self.report_check.append(t1)
            t1 = ('Effective Area ($mm^2$)', ' ',
                  sectional_area_change(round(self.
result_effective_area, 2), round(self.section_property.
area, 2),
                                                 self.
effective_area_factor),
                  ' ')
            self.report_check.append(t1)

            # t1 = ('SubSection', 'Section parameters', '|p{4
cm}|p{1.5cm}|p{9.5cm}|p{1cm}|')
            # self.report_check.append(t1)
            # t1 = ('d_{web}', ' ',
            #        sectional_area_change(round(self.
result_effective_area,2), round(self.section_property.area
,2),
            #                            self.
effective_area_factor),
            #        ' ')
            # self.report_check.append(t1)

            t1 = ('SubSection', 'Section Classification', '|p
{3cm}|p{3.5cm}|p{8.5cm}|p{1cm}|')
            self.report_check.append(t1)
            t1 = ('Web Class', 'Neutral Axis at Mid-Depth',
                  cl_3_7_2_section_classification_web(round(
self.result_eff_d, 2),
                                                      round(
self.section_property.web_thickness, 2), round(
                              self.input_section_classification[
self.result_designation][4], 2),
                                                      self.
epsilon, self.section_property.type,
                                                      self.
input_section_classification[self.result_designation][2]),
                  ' ')
            self.report_check.append(t1)
            t1 = ('Flange Class', self.section_property.type,
                  cl_3_7_2_section_classification_flange(
round(self.section_property.flange_width / 2, 2),

round(self.section_property.flange_thickness, 2), round(
                              self.input_section_classification[
self.result_designation][3], 2),
                                                         self
.epsilon,
                                                                    self
.input_section_classification[self.result_designation][1])
,
               ' ')
         self.report_check.append(t1)
         t1 = ('Section Class', ' ',
               cl_3_7_2_section_classification(
                   self.input_section_classification[self.
result_designation][0]),
               ' ')
         self.report_check.append(t1)

         t1 = ('SubSection', 'Web Slenderness Check', '|p
{3cm}|p{4cm}|p{6cm}|p{3 cm}|')
         self.report_check.append(t1)
         t1 = (
         KEY_DISP_Web_Buckling,
cl_8_2_1web_buckling_required(round(self.epsilon, 2),
round(67 * self.epsilon, 2)),
         cl_8_2_1web_buckling_1(self.result_eff_d, self.
section_property.web_thickness,
                                 round(self.result_eff_d /
self.section_property.web_thickness, 2),
                                 self.
result_web_buckling_check),
         get_pass_fail(67 * self.epsilon, round(self.
result_eff_d / self.section_property.web_thickness, 2),
                        relation="Custom"))
         self.report_check.append(t1)
         if self.result_web_buckling_check:
             t1 = ('SubSection', 'Shear Strength Results:
' + self.support_cndition_shear_buckling,
                   '|p{3.5cm}|p{1.5cm}|p{10cm}|p{1cm}|')
             self.report_check.append(t1)
```

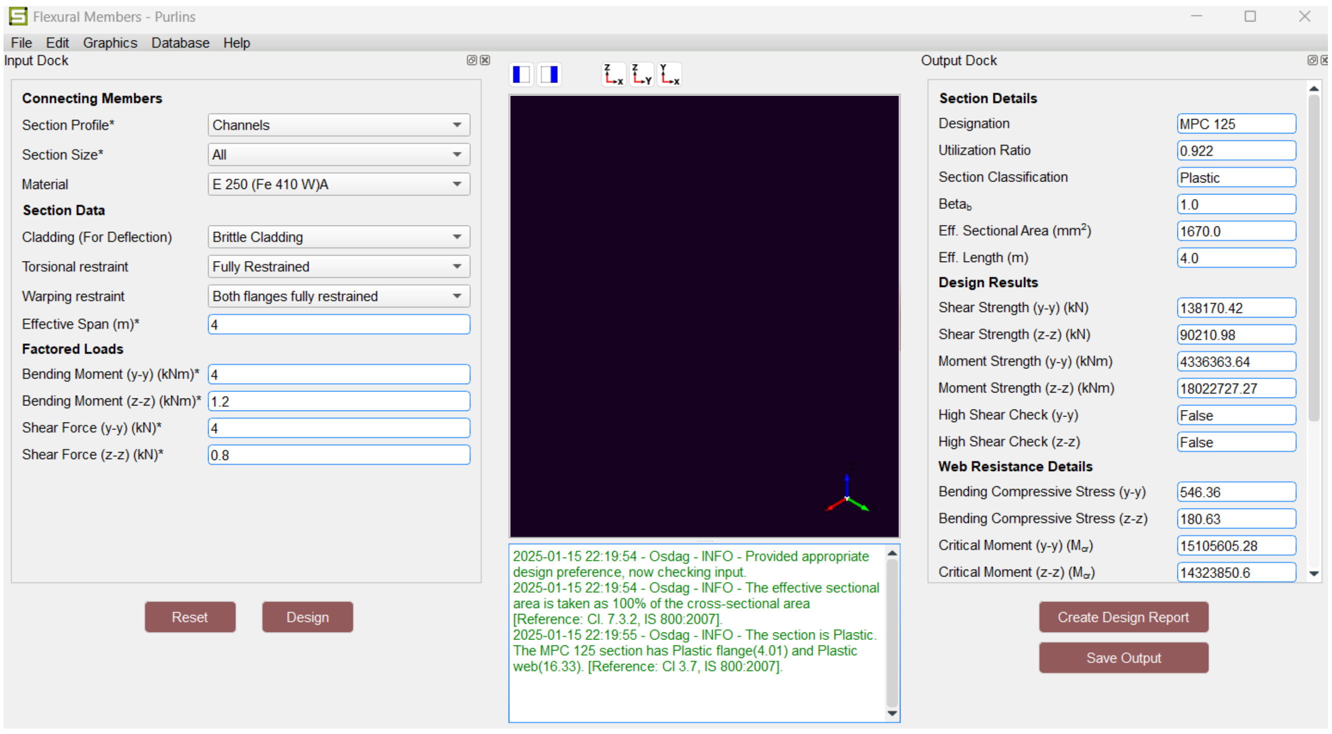
```
                if self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[0]:
                    t1 = (KEY_DISP_K_v_latex, ' ',
                          cl_8_4_2_2_KV(self.
result_web_buckling_simple_kv, self.
support_cndition_shear_buckling),

                          ' ')
                elif self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[1]:
                    t1 = (
KEY_DISP_Transverse_Stiffener_spacing, ' ',

cl_8_4_2_2_Transverse_Stiffener_spacing(self.
result_web_buckling_simple_c),
                          ' ')
                    self.report_check.append(t1)

                    t1 = (KEY_DISP_K_v_latex, ' ',
                          cl_8_4_2_2_KV(self.
result_web_buckling_simple_kv, self.
support_cndition_shear_buckling,
                                        self.
result_web_buckling_simple_c, self.result_eff_d),
                          ' ')
                self.report_check.append(t1)

                t1 = (
KEY_DISP_Elastic_Critical_shear_stress_web, ' ',
                      cl_8_4_2_2_taucrc(self.
result_web_buckling_simple_kv, 2 * 10 ** 5, 0.3,
                                        self.result_eff_d,
                                        self.section_property
.web_thickness,
                                            self.
result_web_buckling_simple_tau_crc),
                        ' ')
                t1 = (KEY_DISP_slenderness_ratio_web, ' ',
                        cl_8_4_2_2_slenderness_ratio(self.fyw,
self.result_web_buckling_simple_lambda_w,
                                                          self.
result_web_buckling_simple_tau_crc),
                        ' ')
                self.report_check.append(t1)

                t1 = (KEY_OUT_DISP_WELD_SHEAR_STRESS, ' ',
                        cl_8_4_2_2_shearstress_web(self.fyw,
self.result_web_buckling_simple_lambda_w,
                                                        self.
result_web_buckling_simple_tau_b),
                        ' ')
                self.report_check.append(t1)

                if self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[0]:
                    t1 = (KEY_DISP_DESIGN_STRENGTH_SHEAR + '(
V_{d})', self.load.shear_force * 10 ** -3,
                            cl_8_4_2_2_shearstrength(self.
result_eff_d, self.section_property.web_thickness,
                                                          self.
result_web_buckling_simple_V_cr,
                                                          self.
result_web_buckling_simple_tau_b, self.result_shear),
                            ' ')
                    self.report_check.append(t1)
                    t1 = (KEY_DISP_ALLOW_SHEAR, ' ',
                          cl_8_2_1_2_shear_check(round(self.
result_shear, 2), round(0.6 * self.result_shear, 2),
                                                 self.
result_high_shear, self.load.shear_force * 10 ** -3),
                          get_pass_fail(self.load.shear_force
 * 10 ** -3, round(0.6 * self.result_shear, 2),
                                        relation="Warn", M1=
self.result_high_shear))
                    self.report_check.append(t1)

                elif self.support_cndition_shear_buckling ==
KEY_DISP_SB_Option[1]:
                    t1 = (KEY_DISP_BUCKLING_STRENGTH + '(V_p)
', ' ',

cl_8_4_1_plastic_shear_resistance_Vp(self.result_eff_d,
self.section_property.web_thickness,

  self.fyw, self.result_web_buckling_simple_V_p_girder

  ),
                          ' ')
                    self.report_check.append(t1)

                    t1 = ('N_f (N)', ' ',
                          cl_8_4_2_2_N_f(self.
section_property.depth,
                                         self.
section_property.flange_thickness,
                                         self.
section_property.depth - self.section_property.
flange_thickness,
                                                                round(self.load.
moment / (
                                                                             self.
section_property.depth - self.section_property.
flange_thickness),
                                                                           2), self.load.
moment
                                                                ),
                                        ' ')
                                 self.report_check.append(t1)

                                 t1 = (KEY_DISP_reduced_moment + '(M_{fr})
', ' ',

cl_8_4_2_2_TensionField_reduced_moment(self.
result_web_buckling_simple_Mfr,

       self.section_property.flange_width,

       self.section_property.flange_thickness,

       self.fyf, round(self.load.moment / (
                                                            self.section_property.
depth - self.section_property.flange_thickness), 2)

       ),
                                        ' ')
                                 self.report_check.append(t1)

                                 t1 = (KEY_DISP_tension_field_incline, ' '
,
                                           cl_8_4_2_2_TensionField_phi(self.
result_web_buckling_simple_phi_girder,
                                                          self.
result_web_buckling_simple_c, self.result_eff_d
                                                          ),
                         ' ')
                    self.report_check.append(t1)

                    t1 = (
KEY_DISP_AnchoragelengthTensionField, ' ',

cl_8_4_2_2_TensionField_anchorage_length(self.
result_web_buckling_simple_s_girder,

      self.result_web_buckling_simple_phi_girder,

      self.result_web_buckling_simple_Mfr, self.fyw,

      self.section_property.web_thickness

      ),
                         ' ')
                    self.report_check.append(t1)

                    t1 = (KEY_DISP_WidthTensionField, ' ',

cl_8_4_2_2_KEY_DISP_WidthTensionField(self.result_eff_d,

   self.result_web_buckling_simple_phi_girder,

   self.result_web_buckling_simple_c,

   self.result_web_buckling_simple_s_girder,

   self.result_web_buckling_simple_wtf_girder
```

```
    ),
                              ' ')
                    self.report_check.append(t1)
                    # t1 = (KEY_DISP_reduced_moment + '(M_{fr
}', ' ',
                    #
cl_8_4_2_2_TensionField_reduced_moment(self.result_eff_d,
                    #
      self.result_web_buckling_simple_phi_girder,
                    #
      self.result_web_buckling_simple_c,
                    #
      self.result_web_buckling_simple_s_girder,self.
result_web_buckling_simple_wtf_girder
                    #
      ),
                    #         ' ')
                    # self.report_check.append(t1)
                    t1 = (
KEY_DISP_Yield_Strength_Tension_field, ' ',

cl_8_4_2_2_Yield_Strength_Tension_field(self.fyw,

     self.result_web_buckling_simple_tau_b,

     self.result_web_buckling_simple_phi_girder,

     self.result_web_buckling_simple_fv_girder

     ),
                              ' ')
                    self.report_check.append(t1)
                    t1 = (KEY_DISP_DESIGN_STRENGTH_SHEAR + '(
V_{d})', self.load.shear_force * 10 ** -3,

cl_8_4_2_2_shearstrength_tensionfield(
                                 self.effective_depth * self.
section_property.web_thickness,
                                 self.
result_web_buckling_simple_tau_b, self.
result_web_buckling_simple_V_p_girder,
                                 self.result_shear, self.
section_property.web_thickness,
                                 self.
result_web_buckling_simple_wtf_girder, self.
result_web_buckling_simple_fv_girder,
                                 self.
result_web_buckling_simple_phi_girder,
                                 round(self.
result_web_buckling_simple_fV_tf_girder * 10 ** -3, 2)),
                            ' ')
                    self.report_check.append(t1)



          else:

               t1 = ('SubSection', 'Shear Strength Results',
   '|p{4cm}|p{5cm}|p{5.5cm}|p{1.5cm}|')
               self.report_check.append(t1)

               t1 = (KEY_DISP_DESIGN_STRENGTH_SHEAR, self.
load.shear_force * 10 ** -3,
                    cl_8_4_shear_yielding_capacity_member_(
self.section_property.depth,

self.section_property.web_thickness,
self.material_property.fy,

self.gamma_m0, round(self.result_shear, 2)),
                   get_pass_fail(self.load.shear_force *
10 ** -3, round(self.result_shear, 2), relation="lesser"))
             self.report_check.append(t1)

             t1 = (KEY_DISP_ALLOW_SHEAR, ' ',
                   cl_8_2_1_2_shear_check(round(self.
result_shear, 2), round(0.6 * self.result_shear, 2),
                                          self.
result_high_shear, self.load.shear_force * 10 ** -3),
                   get_pass_fail(self.load.shear_force *
10 ** -3, round(0.6 * self.result_shear, 2),
                                 relation="Warn", M1=self.
result_high_shear))
             self.report_check.append(t1)

             # t1 = ('SubSection', 'Moment Strength
Results', '|p{4cm}|p{4cm}|p{6.5cm}|p{1.5cm}|')

         t1 = ('SubSection', 'Moment Strength Results', '|
p{4cm}|p{1.5cm}|p{9cm}|p{1.5cm}|')
         self.report_check.append(t1)
         if self.design_type ==
KEY_DISP_DESIGN_TYPE_FLEXURE:
             if self.result_high_shear:
                t1 = (KEY_DISP_Bending_STRENGTH_MOMENT,
self.load.moment * 10 ** -6,

cl_9_2_2_combine_shear_bending_md_init(
                           self.section_property.
elast_sec_mod_z,
                              self.section_property.
plast_sec_mod_z,
                              self.material_property.fy, self
.support,
                              self.gamma_m0, round(self.
result_betab, 2),
                              round(self.result_Md * 10 **
-6, 2), self.result_section_class
                          ),
                          ' ')
                 self.report_check.append(t1)
                 t1 = (KEY_DISP_PLASTIC_STRENGTH_MOMENT, '
 ',
                          cl_9_2_2_combine_shear_bending_mfd(
                              self.section_property.
plast_sec_mod_z,
                              self.section_property.depth,
                              self.section_property.
web_thickness,
                              self.material_property.fy,
                              self.gamma_m0,
                              round(self.result_mfd * 10 **
-6, 2)),
                          ' ')
                 self.report_check.append(t1)

                 # temp =
cl_8_2_1_2_plastic_moment_capacity_member(self.
result_betab,
                 #
   self.section_property.plast_sec_mod_z,
                 #
   self.material_property.fy, self.gamma_m0,
                 #
   round(self.result_bending, 2))
                 # print('tempt',temp)

                 t1 = (KEY_DISP_DESIGN_STRENGTH_MOMENT,
self.load.moment * 10 ** -6,
                          cl_9_2_2_combine_shear_bending(
round(self.result_bending, 2),
                                                          self
.section_property.elast_sec_mod_z,
                                                          self
.material_property.fy, self.result_section_class,
                                                          self
.load.shear_force * 10 ** -3, round(self.result_shear, 2),
                                                          self
.gamma_m0, round(self.result_beta_reduced, 2),

round(self.result_Md * 10 ** -6, 2),

round(self.result_mfd * 10 ** -6, 2)),
                          get_pass_fail(self.load.moment * 10
    ** -6, round(self.result_bending, 2), relation="lesser"))
                 self.report_check.append(t1)

             else:
                 t1 = (KEY_DISP_DESIGN_STRENGTH_MOMENT,
self.load.moment * 10 ** -6,
                          cl_8_2_1_2_moment_capacity_member(
round(self.result_betab, 3),

self.section_property.plast_sec_mod_z,

self.material_property.fy, self.gamma_m0,
round(self.result_bending, 2),

self.section_property.elast_sec_mod_z,

self.result_section_class, self.support),
                           get_pass_fail(self.load.moment * 10
 ** -6, round(self.result_bending, 2), relation="lesser"))
                    self.report_check.append(t1)
          elif self.design_type ==
KEY_DISP_DESIGN_TYPE2_FLEXURE:
               # KEY_DISP_Elastic_CM_latex
               t1 = (KEY_DISP_Elastic_CM_latex, ' ',
                      cl_8_2_2_1_Mcr(
                           self.result_mcr,
                           self.material_property.
modulus_of_elasticity,
                           self.section_property.mom_inertia_y
,
                           self.result_eff_len, self.
material_property.modulus_of_elasticity / (2 * 1.3),
                           self.section_property.It, self.
section_property.Iw
                           #   round(self.result_Md * 10 **
-6, 2), self.result_section_class
                      ),
                      ' ')
               self.report_check.append(t1)

               # t1 = (KEY_DISP_I_eff_latex + '($mm^4$)', '
',
               #         cl_8_7_3_Ieff_web_check(self.
bearing_length, self.section_property.web_thickness,
               #
round(self.result_bcI_eff,2)),
               #        ' ')
               #     self.report_check.append(t1)

               #     t1 = (KEY_DISP_A_eff_latex+ '($mm^2$)',
' ',
               #           cl_8_7_3_Aeff_web_check(self.
bearing_length, self.section_property.web_thickness,
               #
   self.result_bcA_eff),
               #           ' ')
               #     self.report_check.append(t1)

               #     t1 = (KEY_DISP_r_eff_latex+ '(mm)', '
',
               #           cl_8_7_3_reff_web_check(round(
self.result_bcr_eff,2), round(self.result_bcI_eff,2),
               #
   self.result_bcA_eff),
               #           ' ')
               #     self.report_check.append(t1)

               t1 = (KEY_DISP_SLENDER + '($\lambda_{LT}$)',
' ',
                      cl_8_2_2_slenderness(round(self.
result_betab, 2), self.section_property.elast_sec_mod_z,
                                           self.
section_property.plast_sec_mod_z, self.result_mcr,
                                           self.
material_property.fy,
                                           self.
result_nd_esr_lt),
                      ' ')
               self.report_check.append(t1)

               #      # t1 = (KEY_DISP_SLENDER, ' ',
               #      #        cl_8_7_1_5_slenderness(round(
self.result_bcr_eff, 2), round(self.result_eff_d, 2),
               #      #                              self.
result_eff_sr),
               #      #        ' ')
               #      # self.report_check.append(t1)

               #      t1 = (KEY_DISP_BUCKLING_CURVE_ZZ, ' ',
               #            cl_8_7_1_5_buckling_curve(),
               #            ' ')
               #      self.report_check.append(t1)

               t1 = (KEY_DISP_IMPERFECTION_FACTOR_ZZ + r'($\
alpha_{LT}$)', ' ',
                      cl_8_7_1_5_imperfection_factor(self.
result_IF_lt),
                      ' ')
               self.report_check.append(t1)

               #      t1 = (KEY_DISP_EULER_BUCKLING_STRESS_ZZ
, ' ',
               #            cl_8_7_1_5_buckling_stress(self.
section_property.modulus_of_elasticity,self.result_eff_sr,
self.result_ebs),
               #            ' ')
               #      self.report_check.append(t1)

               t1 = ('$\phi_{LT}$', ' ',
                      cl_8_2_2_phi(self.result_IF_lt, self.
result_nd_esr_lt, self.result_phi_lt),
                      ' ')
                self.report_check.append(t1)

                t1 = ('Bending Compressive stress($N/mm^2$)',
 ' ',
                      cl_8_2_2_Bending_Compressive(self.
material_property.fy, self.gamma_m0, self.result_nd_esr_lt
,
                                                            self.
result_phi_lt, self.result_fcd__lt),
                      ' ')
                self.report_check.append(t1)

                #     t1 = (KEY_DISP_BUCKLING_STRENGTH, self.
load.shear_force * 10 ** -3,
                #
cl_7_1_2_design_compressive_strength(self.result_capacity,
round((
                #                     self.bearing_length +
self.section_property.depth / 2) * self.section_property.
web_thickness,2), self.result_fcd,self.load.shear_force *
10 ** -3),
                #            get_pass_fail(self.load.
shear_force * 10 ** -3, round(self.result_capacity, 2),
relation="leq"))
                #     self.report_check.append(t1)

                if self.result_high_shear:
                    t1 = (
KEY_DISP_LTB_Bending_STRENGTH_MOMENT, self.load.moment *
10 ** -6,

cl_9_2_2_combine_shear_bending_md_init(
                                  self.section_property.
elast_sec_mod_z,
                            self.section_property.
plast_sec_mod_z,
                            self.material_property.fy, self
.support,
                            self.gamma_m0, round(self.
result_betab, 2),
                            round(self.result_Md * 10 **
-6, 2), self.result_section_class
                        ),
                        ' ')
                self.report_check.append(t1)
                t1 = (KEY_DISP_PLASTIC_STRENGTH_MOMENT, '
 ',
                        cl_9_2_2_combine_shear_bending_mfd(
                            self.section_property.
plast_sec_mod_z,
                            self.section_property.depth,
                            self.section_property.
web_thickness,
                            self.material_property.fy,
                            self.gamma_m0,
                            round(self.result_mfd * 10 **
-6, 2)),
                        ' ')
                self.report_check.append(t1)

                # temp =
cl_8_2_1_2_plastic_moment_capacity_member(self.
result_betab,
                #
   self.section_property.plast_sec_mod_z,
                #
   self.material_property.fy, self.gamma_m0,
                    #
    round(self.result_bending, 2))
                    # print('tempt',temp)
                    t1 = (KEY_DISP_REDUCE_STRENGTH_MOMENT,
self.load.moment * 10 ** -6,
                          cl_9_2_2_combine_shear_bending(
round(self.result_bending, 2),
                                                        self
.section_property.elast_sec_mod_z,
                                                        self
.material_property.fy, self.result_section_class,
                                                        self
.load.shear_force * 10 ** -3, round(self.result_shear, 2),
                                                        self
.gamma_m0, round(self.result_betab, 2),

round(self.result_Md * 10 ** -6, 2),

round(self.result_mfd * 10 ** -6, 2)),
                          get_pass_fail(self.load.moment * 10
 ** -6, round(self.result_bending, 2), relation="lesser"))
                    self.report_check.append(t1)

                else:
                    t1 = ('Moment Strength (kNm)', self.load.
moment * 10 ** -6,
                          cl_8_2_2_moment_capacity_member(
round(self.result_betab, 2),

self.section_property.plast_sec_mod_z,

self.material_property.fy, self.gamma_m0,

round(self.result_bending, 2),
self.section_property.elast_sec_mod_z,

self.result_section_class, self.support),
                        get_pass_fail(self.load.moment * 10
 ** -6, round(self.result_bending, 2), relation="lesser"))
                  self.report_check.append(t1)

         if self.result_buckling_crippling:
             t1 = ('SubSection', 'Web Buckling Checks', '|
p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
             self.report_check.append(t1)

             t1 = (KEY_DISP_I_eff_latex + '($mm^4$)', ' ',
                   cl_8_7_3_Ieff_web_check(self.
bearing_length, self.section_property.web_thickness,
                                           round(self.
result_bcI_eff, 2)),
                   ' ')
             self.report_check.append(t1)

             t1 = (KEY_DISP_A_eff_latex + '($mm^2$)', ' ',
                   cl_8_7_3_Aeff_web_check(self.
bearing_length, self.section_property.web_thickness,
                                           self.
result_bcA_eff),
                   ' ')
             self.report_check.append(t1)

             t1 = (KEY_DISP_r_eff_latex + '(mm)', ' ',
                   cl_8_7_3_reff_web_check(round(self.
result_bcr_eff, 2), round(self.result_bcI_eff, 2),
                                           self.
result_bcA_eff),
                    ' ')
            self.report_check.append(t1)

            t1 = (KEY_DISP_SLENDER + '($\lambda$)', ' ',
                    cl_8_7_1_5_slenderness(round(self.
result_bcr_eff, 2), round(self.result_eff_d, 2),
                                                self.
result_eff_sr),
                    ' ')
            self.report_check.append(t1)

            # t1 = (KEY_DISP_SLENDER, ' ',
            #       cl_8_7_1_5_slenderness(round(self.
result_bcr_eff, 2), round(self.result_eff_d, 2),
            #                                   self.
result_eff_sr),
            #       ' ')
            # self.report_check.append(t1)

            t1 = (KEY_DISP_BUCKLING_CURVE_ZZ, ' ',
                    cl_8_7_1_5_buckling_curve(),
                    ' ')
            self.report_check.append(t1)

            t1 = (KEY_DISP_IMPERFECTION_FACTOR_ZZ + r'($\
alpha$)', ' ',
                    cl_8_7_1_5_imperfection_factor(self.
result_IF),
                    ' ')
            self.report_check.append(t1)

            t1 = (KEY_DISP_EULER_BUCKLING_STRESS_ZZ, ' ',
                    cl_8_7_1_5_buckling_stress(self.
section_property.modulus_of_elasticity, self.result_eff_sr
,
                                                         self.
result_ebs),
                       ' ')
                self.report_check.append(t1)

                t1 = ('$\phi$', ' ',
                       cl_8_7_1_5_phi(0.49, self.result_eff_sr
, self.result_phi_zz),
                       ' ')
                self.report_check.append(t1)

                t1 = ('Buckling stress($N/mm^2$)', ' ',
                       cl_8_7_1_5_Buckling(self.
material_property.fy, self.gamma_m0, self.result_eff_sr,
                                                  self.result_phi_zz,
 self.result_fcd_2, self.result_fcd),
                       ' ')
                self.report_check.append(t1)

                t1 = (KEY_DISP_BUCKLING_STRENGTH, self.load.
shear_force * 10 ** -3,
                       cl_7_1_2_design_compressive_strength(
self.result_capacity, round((

                                             self.bearing_length +
self.section_property.depth / 2) * self.section_property.
web_thickness,

                                     2), self.result_fcd,

self.load.shear_force * 10 ** -3),
                       get_pass_fail(self.load.shear_force *
10 ** -3, round(self.result_capacity, 2), relation="leq"))
```

```
                    self.report_check.append(t1)

                    t1 = ('SubSection', 'Web Bearing Checks', '|p
{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
                    self.report_check.append(t1)

                    t1 = ('Bearing Strength(kN)', self.load.
shear_force * 10 ** -3,
                          cl_8_7_4_Bearing_stiffener_check(self.
bearing_length, round(2.5 * (
                                  self.section_property.
root_radius + self.section_property.flange_thickness), 2),
                                                          self.
section_property.web_thickness,
                                                          self.
material_property.fy, self.gamma_m0,
                                                          round(
self.result_crippling, 2),
                                                          self.
section_property.root_radius,
                                                          self.
section_property.flange_thickness),
                          get_pass_fail(self.load.shear_force *
10 ** -3, round(self.result_crippling, 2), relation="leq")
)

                    self.report_check.append(t1)

                t1 = ('SubSection', 'Utilization', '|p{4cm}|p{2
cm}|p{7cm}|p{3 cm}|')
                self.report_check.append(t1)
                # TODO
                if self.result_buckling_crippling:
                    t1 = (KEY_DISP_Utilization_Ratio, 1.0,
                    Utilization_Ratio_Latex(self.load.
shear_force * 10 ** -3, round(self.result_shear, 2),
                                                self.load.
moment * 10 ** -6, round(self.result_bending, 2),
                                                self.result_UR,
 type=2, Pd=self.result_capacity,
                                                fw=self.
result_crippling),
                    get_pass_fail(1.0, self.result_UR,
relation="geq"))
         else:
             t1 = (KEY_DISP_Utilization_Ratio, 1.0,
                   Utilization_Ratio_Latex(self.load.
shear_force * 10 ** -3, round(self.result_shear, 2),
                                                self.load.
moment * 10 ** -6, round(self.result_bending, 2),
                                                self.result_UR)
,
                   get_pass_fail(1.0, self.result_UR,
relation="geq"))
         self.report_check.append(t1)
         # if self.design_type ==
KEY_DISP_DESIGN_TYPE2_FLEXURE:
         #     t1 = ('SubSection', 'Lateral Torsional
Buckling Checks', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
         #     self.report_check.append(t1)
         # t1 = (KEY_DISP_A_eff_latex + '(mm^2)', ' ',
         #        cl_8_7_3_Aeff_web_check(self.
bearing_length, self.section_property.web_thickness,
         #                                self.
result_bcA_eff),
         #        ' ')
         # self.report_check.append(t1)
         # if self.latex_tension_zone == True :
```

```
        #     t1 = (KEY_DISP_TENSION_HOLES, ' ',
        #           sectional_area_change(self.
result_effective_area, self.section_property.area,
        #                                      self.
effective_area_factor),
        #           ' ')
        #     self.report_check.append(t1)

    # else:
    #     t1 = (KEY_DISP_ALLOW_SHEAR, self.load.
shear_force,
    #           allow_shear_capacity(round(self.
result_shear, 2), round(0.6 * self.result_shear, 2)),
    #           get_pass_fail(self.load.shear_force))
    # self.report_check.append(t1)

    # self.h = (self.beam_D - (2 * self.beam_tf))
    #
    # 1.1 Input sections display
    # t1 = ('SubSection', 'List of Input Sections',self.
sec_list),
    # self.report_check.append(t1)
    #
    # # 2.2 CHECK: Buckling Class - Compatibility Check
    # t1 = ('SubSection', 'Buckling Class - Compatibility
 Check', '|p{4cm}|p{3.5cm}|p{6.5cm}|p{2cm}|')
    # self.report_check.append(t1)
    #
    # t1 = ("Section Class ",
comp_column_class_section_check_required(self.
result_section_class, self.h, self.bf),
    #       comp_column_class_section_check_provided(self
.bucklingclass, self.h, self.bf, self.tf, self.var_h_bf),
    #       'Compatible')  # if self.
bc_compatibility_status is True else 'Not compatible')
    # self.report_check.append(t1)

    # t1 = ("h/bf , tf ",
comp_column_class_section_check_required(self.
bucklingclass, self.h, self.bf),
    #       comp_column_class_section_check_provided(self
.bucklingclass, self.h, self.bf, self.tf, self.var_h_bf),
    #       'Compatible')  # if self.
bc_compatibility_status is True else 'Not compatible')
    # self.report_check.append(t1)
    #
    # # 2.3 CHECK: Cross-section classification
    # t1 = ('SubSection', 'Cross-section classification',
'|p{4.5cm}|p{3cm}|p{6.5cm}|p{1.5cm}|')
    # self.report_check.append(t1)
    #
    # t1 = ("b/tf and d/tw ",
cross_section_classification_required(self.section),
    #       cross_section_classification_provided(self.tf
, self.b1, self.epsilon, self.section, self.b1_tf,
    #                                             self.
d1_tw, self.ep1, self.ep2, self.ep3, self.ep4),
    #       'b = bf / 2,d = h    2 ( T + R1),   = (250 /
 Fy )^0.5,Compatible')  # if self.bc_compatibility_status
is True else 'Not compatible')
    # self.report_check.append(t1)
    #
    # # 2.4 CHECK : Member Check
    # t1 = ("Slenderness", cl_7_2_2_slenderness_required(
self.KL, self.ry, self.lamba),
    #       cl_7_2_2_slenderness_provided(self.KL, self.
ry, self.lamba), 'PASS')
```

```
        # self.report_check.append(t1)
        #
        # t1 = (
        # "Design Compressive stress (fcd)",
    cl_7_1_2_1_fcd_check_required(self.gamma_mo, self.f_y,
    self.f_y_gamma_mo),
        # cl_7_1_2_1_fcd_check_provided(self.facd), 'PASS')
        # self.report_check.append(t1)
        #
        # t1 = ("Design Compressive strength (Pd)",
    cl_7_1_2_design_comp_strength_required(self.axial),
        #        cl_7_1_2_design_comp_strength_provided(self.
    Aeff, self.facd, self.A_eff_facd), "PASS")
        # self.report_check.append(t1)
        #
        # t1 = ('', '', '', '')
        # self.report_check.append(t1)
        else:
            self.report_input = \
                {  # KEY_MAIN_MODULE: self.mainmodule,
                    KEY_MODULE: self.module,  # "Axial load
    on column "
                    KEY_DISP_SHEAR + '*': self.load.
    shear_force * 10 ** -3,
                    KEY_DISP_BEAM_MOMENT_Latex + '*': self.
    load.moment * 10 ** -6,
                    KEY_DISP_LENGTH_BEAM: self.length,
                    KEY_DISP_SEC_PROFILE: self.sec_profile,
                    KEY_DISP_SECSIZE: str(self.sec_list),
                    KEY_MATERIAL: self.material,
                    # "Failed Section Details": self.
    report_column,
                    KEY_BEAM_SUPP_TYPE: self.
    latex_design_type,
                }
            self.report_input.update({
                KEY_DISP_SUPPORT: self.support,
                KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.
    material_property.fu,
                KEY_DISP_YIELD_STRENGTH_REPORT: self.
    material_property.fy,
                "End Conditions - " + str(self.support): "
    TITLE",
            })
            # if self.Latex_length == 'NA':
            if self.support == KEY_DISP_SUPPORT1:
                self.report_input.update({
                    DISP_TORSIONAL_RES: self.Torsional_res,
                    DISP_WARPING_RES: self.Warping})
            else:
                self.report_input.update({
                    DISP_SUPPORT_RES: self.Support,
                    DISP_TOP_RES: self.Top})
            self.report_input.update({
                "Design Preference": "TITLE",
                KEY_DISP_EFFECTIVE_AREA_PARA: self.
    effective_area_factor,
                KEY_DISP_CLASS: self.allow_class,
                KEY_DISP_LOAD: self.Loading,
                KEY_DISPP_LENGTH_OVERWRITE: self.latex_efp,
                KEY_DISP_BEARING_LENGTH + ' (mm)': self.
    bearing_length,

            })
            # if self.latex_design_type ==
    VALUES_SUPP_TYPE_temp[0] and self.
    result_web_buckling_check:
            #     self.report_input.update({
            #          KEY_ShearBuckling: self.
support_cndition_shear_buckling
            #     })
            # self.report_input.update({
            #       # KEY_DISP_SEC_PROFILE: self.sec_profile,
            #       "I Section - Mechanical Properties": "
TITLE",
            #       })
            self.report_input.update()
            self.report_check = []

            t1 = ('Selected', 'All Members Failed', '|p{5cm}|
p{2cm}|p{2cm}|p{2cm}|p{4cm}|')
            self.report_check.append(t1)

            t1 = ('SubSection', 'Plastic Section Modulus', '|
p{4cm}|p{1.5cm}|p{2.5cm}|p{8cm}|')
            self.report_check.append(t1)
            t1 = ('Plastic Section Modulus($mm^3$)', round(
self.Zp_req, 2),
                  ' ',
                  'Select Sections with atleast required
Plastic Section Modulus ')
            self.report_check.append(t1)
        print(sys.path[0])
        rel_path = str(sys.path[0])
        rel_path = rel_path.replace("\\", "/")
        fname_no_ext = popup_summary['filename']
        CreateLatex.save_latex(CreateLatex(), self.
report_input, self.report_check, popup_summary,
fname_no_ext,
                              rel_path, [], '', module=self.
module)  #
```



Connecting Members

Section Profile*	Channels
Section Size*	All
Material	E 250 (Fe 410 W)A

Section Data

Cladding (For Deflection)	Brittle Cladding
Torsional restraint	Fully Restrained
Warping restraint	Both flanges fully restrained
Effective Span (m)*	4

Factored Loads

Bending Moment (y-y) (kNm)*	4
Bending Moment (z-z) (kNm)*	1.2
Shear Force (y-y) (kN)*	4
Shear Force (z-z) (kN)*	0.8

```
2025-01-15 22:19:54 - Osdag - INFO - Provided appropriate
design preference, now checking input.
2025-01-15 22:19:54 - Osdag - INFO - The effective sectional
area is taken as 100% of the cross-sectional area
[Reference: Cl. 7.3.2, IS 800:2007].
2025-01-15 22:19:55 - Osdag - INFO - The section is Plastic.
The MPC 125 section has Plastic flange(4.01) and Plastic
web(16.33). [Reference: Cl 3.7, IS 800:2007].
```

Section Details

Designation	MPC 125
Utilization Ratio	0.922
Section Classification	Plastic
Beta$_b$	1.0
Eff. Sectional Area (mm^2)	1670.0
Eff. Length (m)	4.0

Design Results

Shear Strength (y-y) (kN)	138170.42
Shear Strength (z-z) (kN)	90210.98
Moment Strength (y-y) (kNm)	4336363.64
Moment Strength (z-z) (kNm)	18022727.27
High Shear Check (y-y)	False
High Shear Check (z-z)	False

Web Resistance Details

Bending Compressive Stress (y-y)	546.36
Bending Compressive Stress (z-z)	180.63
Critical Moment (y-y) (M$_{cr}$)	15105605.28
Critical Moment (z-z) (M$_{cr}$)	14323850.6

Figure 3.2: Purlin Module

Chapter 4

Conclusions

4.1 Tasks Accomplished

During the internship period, I was able to complete the entire Purlin module and make all associated changes in other files to integrate the module into Osdag.

4.2 Skills Developed

During the internship, I acquired basic knowledge of PyQt5 and how it used to create GUI's. I also improved my skills and Python and going through a large codebase that I am unfamiliar with and understanding it. I was also able to improve my skills in Latex. Additionally, I honed teamwork and coordination skills, which have been essential for finishing the project.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:	**Debayan Ghosh**		
Project:	**Osdag**		
Internship:	**FOSSEE Winter Fellowship 2024**		
Date	Day	Task	Hours Worked
12/11/2024	Tuesday	Installed Osdag on Windows 11 | Went through the installation docs	4
13/11/2024	Wednesday	Went through Osdag docs and videos	4
14/11/2024	Thursday	Received documents regarding timelines and work to be done	4
15/11/2024	Friday	Meeting on Purlins | Went rhough the DDCL	4
16/11/2024	Saturday	WEEKLY HOLIDAY	0
17/11/2024	Sunday	WEEKLY HOLIDAY	0
18/11/2024	Monday	Discussions on Purlin Algorithms	3
19/11/2024	Tuesday	Discussions on Purlin Algorithms	4
20/11/2024	Wednesday	Started going through codebase of flexural members, purlins	4
21/11/2024	Thursday	Going through codebase, Got all doubts regarding codebase cleared	3
22/11/2024	Friday	Going thorugh codebase	4
23/11/2024	Saturday	WEEKLY HOLIDAY	0
24/11/2024	Sunday	WEEKLY HOLIDAY	0
25/11/2024	Monday	Started work on Purlins code	3
26/11/2024	Tuesday	Meeting, fixed errors in codebase	4
27/11/2024	Wednesday	Finished work on Input dock for Purlins	4
28/11/2024	Thursday	Finished coding section classfication	4
29/11/2024	Friday	Fixed bugs in the purlin code	4
30/11/2024	Saturday	Worked on implemeting high and low shear checks	3
1/12/2024	Sunday	Final meet before END SEM BREAK	4
2/12/2024	Monday	END SEM EXAM BREAK	0
3/12/2024	Tuesday	END SEM EXAM BREAK	0
4/12/2024	Wednesday	END SEM EXAM BREAK	0
5/12/2024	Thursday	END SEM EXAM BREAK	0
6/12/2024	Friday	END SEM EXAM BREAK	0

7/12/2024	Saturday	END SEM EXAM BREAK	0
8/12/2024	Sunday	END SEM EXAM BREAK	0
9/12/2024	Monday	END SEM EXAM BREAK	0
10/12/2024	Tuesday	END SEM EXAM BREAK	0
11/12/2024	Wednesday	END SEM EXAM BREAK	0
12/12/2024	Thursday	END SEM EXAM BREAK	0
13/12/2024	Friday	END SEM EXAM BREAK	0
14/12/2024	Saturday	END SEM EXAM BREAK	0
15/12/2024	Sunday	END SEM EXAM BREAK	0
16/12/2024	Monday	Went through the work that was remaining, attended a meet	5
17/12/2024	Tuesday	Fixed logical errors made in section classification code	5
18/12/2024	Wednesday	Wrote the code for buckling check	5
19/12/2024	Thursday	Corrected logic for high shear check	5
20/12/2024	Friday	Wrote code for the moment and shear force checks	5
21/12/2024	Saturday	Wrote code for resistance checks	5
22/12/2024	Sunday	Fixed errors and bugs in the code	5
23/12/2024	Monday	Fixed all the logic in design_beams	5
24/12/2024	Tuesday	Wrote servicibility checks	5
25/12/2024	Wednesday	Wrote code for results to store value of section with highest utilisation ratio	5
26/12/2024	Thursday	Updated code to work with cases with no section passing all the checks	5
27/12/2024	Friday	Updated the output dock and fixed all errors	5
28/12/2024	Saturday	Filed a PR with updated code	5
29/12/2024	Sunday	Started work on the report	5
30/12/2024	Monday	Finished report and wrapped up everything	5

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.