Semester-Long Internship Report

On

Development of Compression Member Design Module, Debugging, Testing, and creation of Design report in Osdag

Submitted by

Rutvik Nitin Joshi

Under the guidance of

Prof. Sidhartha Ghosh
Civil Engineering Department IIT Bombay

Under the Mentorship of

Danish Ansari
Assistant Project Manager
Osdag, FOSSEE
IIT Bombay

August 23, 2021

Acknowledgment

I would like to thank FOSSEE for providing me a platform to work on and also thank them for developing open-source tools for the country. The internship has improved my knowledge and application. I am thankful to everyone who was involved in the selection process based on screening tasks. I am grateful to be a part of the team which promotes open-source software.

I am grateful that I got a chance to work under **Prof. Siddhartha Ghosh**, who took the time to mentor us and monitored individual contributions as well. I thank **Mr. Danish Ansari** who made us feel welcome and planned all the tasks meticulously during this period. I would like to thank **N Swaroop** who has been my teammate in the Internship and has helped in design of compression members and testing. Finally, I thank profusely to all the faculty members for their kind help and the members who helped directly and indirectly.

Table of Contents

TITLE **PAGE NO**

Chapter 1 Introduction

1.1 Osdag Internship

Osdag internship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free/Libre and Open-Source Software) tools to improve the quality of education in our country. FOSSEE encourages the use of FOSS tools through various activities to ensure the availability of competent free software equivalent to commercial (paid) software.

The FOSSEE project is a part of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education, Government of India.

Osdag is one such open-source software that comes under the FOSSEE project. Osdag internship is provided through the FOSSEE project. Any UG/PG/Ph.D. holder can apply for this internship. And the selection will be based on a screening task.

1.2 What is Osdag?

Osdag is Free/Libre and Open-Source Software being developed for the design of steel structures following IS 800:2007 and other relevant design codes. OSDAG helps users in designing steel connections, members and systems using interactive Graphical User Interface (GUI).

The source code is written in Python, 3D CAD images are developed using PythonOCC. GitHub is used to ensure smooth workflow between different modules and team members. It is in a path where people from around the world would be able to contribute to its development. FOSSEE's "Share alike" policy would improve the standard of the software when the source code is further modified based on the industrial and educational needs across the country.

Design and Detailing Checklist (DDCL) for different connections, members and structure designs is one of the main products of this project. It would create a repository and design guidebook for steel construction based on

Indian Standard codes and best industry practices.

Home page of OSDAG

1.3 Who can use Osdag?

Osdag is primarily created for use in academia for students and teachers but industry professionals also find it useful. As Osdag is currently funded by MHRD, the Osdag team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject.

Osdag can be used by anyone starting from novice to professionals. Its simple user interface makes it flexible and attractive than other software. Video tutorials are available to help get started. The video tutorials of Osdag can be accessed here.

- The video tutorials of OSDAG can be easily accessed from https://osdag.fossee.in/resources/videos or YouTube.
- The sample design problems for different modules can be viewed from https://osdag.fossee.in/resources/sample-design.
- One can view the user tools used for the development of OSDAG from https://osdag.fossee.in/resources/user-tools.

OSDAG can be downloaded from https://osdag.fossee.in/resources/downloads.

Chapter 2 Axial Load on Columns

Axial loading on columns is due to loads from floors, walls and roofs which are transmitted to the column through beams and also because of its self-weight. Such types of columns only take axial load i.e., no eccentricity is present and the net end moments are zero. In this case, the column is subjected to Axial compressive load only. Here the user is provided with two options for designing.

Case 1:

A design code is developed which runs for two cases selected as per by the user. The first case is when the user wants to design a column section concerning compression. The user is provided with series of Indian Standard steel sections as per IS code. The user enters axial load to be subjected by the section and also selects desired list of sections. After pressing the design tab, the code gives output stating the section from the provided list which is optimized for the given load and thus increasing efficiency.

Case 2:

In the second case, the user is required to pick a section from the available IS code database and after running the program the software provides the maximum compression load-carrying capacity of the selected section. This helps the user to check compatibility and operationality for the selected section

The user is provided with various information about the preferred section such as buckling class, type of section, sectional properties to facilitate the user to make an informed decision. The code for both cases is combined in a single file to increase efficiency by reusing multiple functions which help to reduce run time and saves space. The stand-alone program code is available here. For the Osdag module for axial load click here

Chapter 3 Axial and Bending Load on Columns

(Beam-Column)

Typically, columns encounter moments about one or both axes in addition to the axial load. Columns that are subjected to both axial load and bending moment are referred to as beam-column. In steel structures, beam-column are often subjected. to a biaxial moment that acts in two different planes. In this case, the column is subjected to both Axial and Bending load and shows behavior similar to a Beam in bending and is referred to here as Beam-Column. Here also the user is provided with two options for designing.

Case 1:

The first case is when the user wants to design a column section concerning compression and bending. The user is provided with series of Indian Standard steel sections as per IS code. The user enters axial load, bending moments to be subjected by the section, and also selects the desired list of sections. After pressing the design tab, the code gives output stating the section from the provided list which is optimized for the given load and thus increasing efficiency.

Case 2:

In the second case, the user is required to pick a section from the available IS code database and after running the program the software provides the maximum compression load, critical bending moment carrying capacity of the selected section. This helps the user to check compatibility and operationality for the selected section.

Both the cases are compiled in a single python file using multiple classes, functions and loops thus increasing the usability of code while reducing space needed. For both modules (Axial load, Axial and bending load) the logger shows messages regarding ongoing compilation. It also shows the number of checks cleared by a respective section which helps the user in further deciding section selection.

Chapter 4 IS 800:2007 code additions to the Osdag database

Functions developed to determine design compressive strength of a member, the effective length of prismatic compression member, design compressive stress, buckling class of cross-sections referring to IS 800:2007 which can be accessed from here

Chapter 5 Documentation and Technical report

Generation of technical reports to easily understand the design steps.

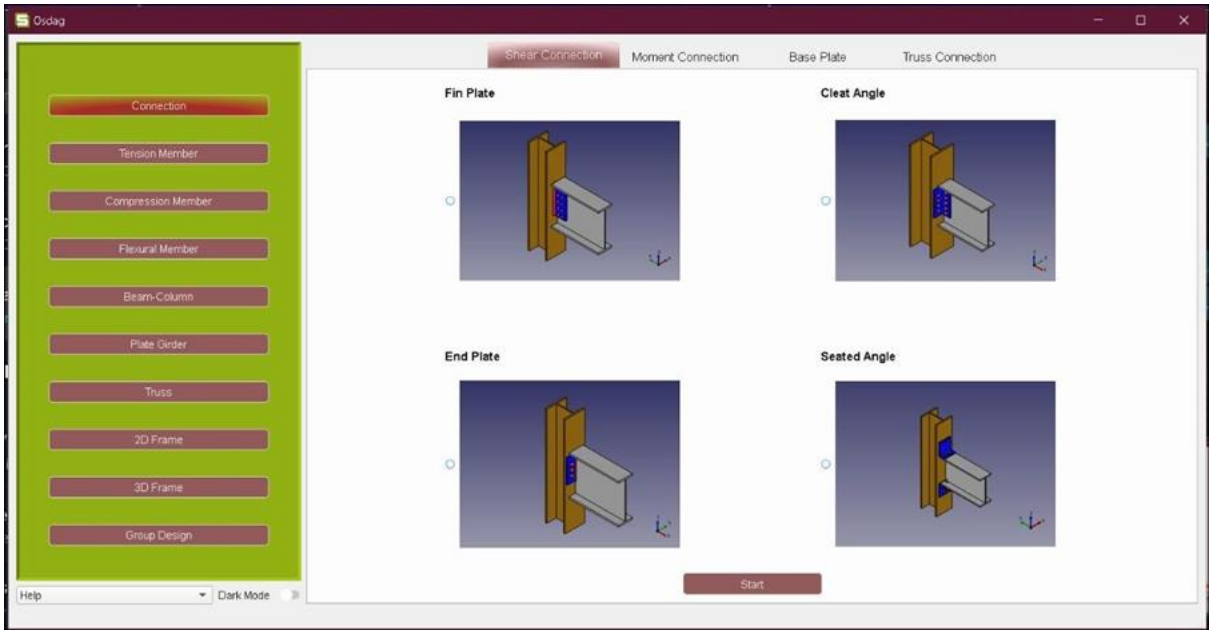
LaTeX – A document preparation system

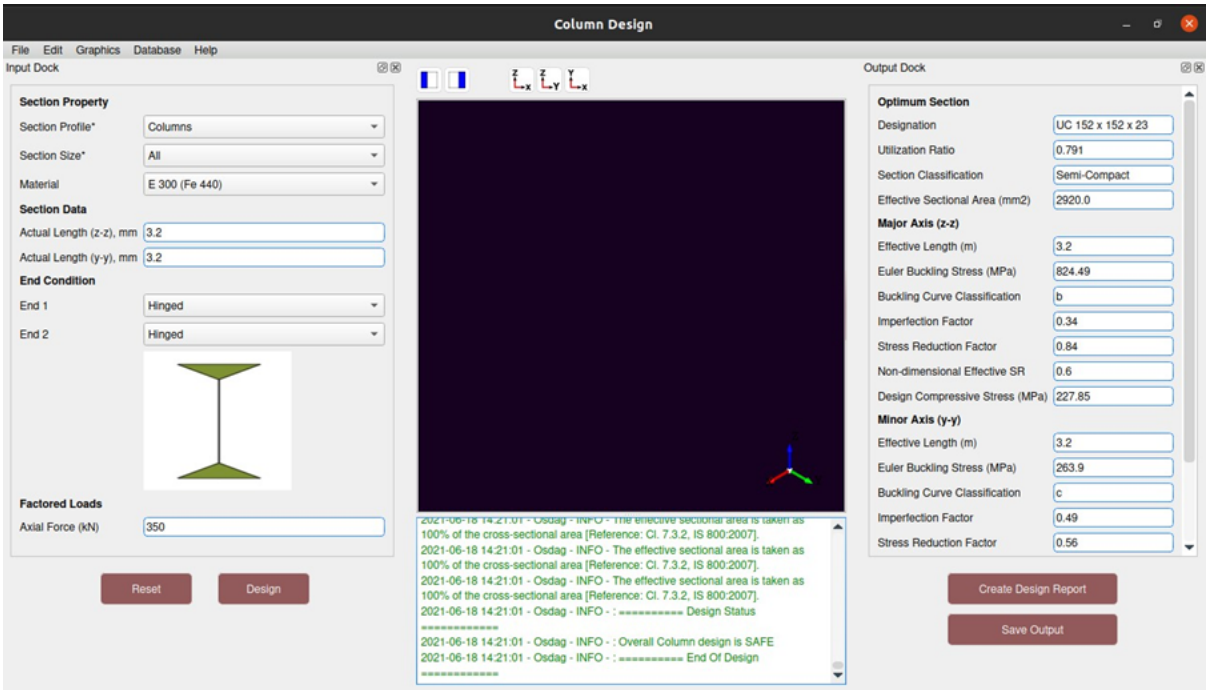
LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents. LaTeX is available as free software. All these provide LaTeX as the most suitable medium to create a technical report.

For the technical report to be used as a dynamic medium with python new functions pertaining to the column module were added to the Report_functions.py file to automatically create a design report with LaTeX as the output. The code for the same can be found here

Chapter 6 GRAPHICAL USER INTERFACE

GUI plays an important role in any software development enriching the aesthetic looks and providing an easier interface to the user in order to carry out his works. GUI in OSDAG has been developed in such a way that it can be easily understood while performing design. All the input and output parameters have been presented in simple words so that there won't be any confusion to anyone who is using OSDAG





Appendix

Code for Axial Load on Columns

```
"""
Main module: Design of Compression Member
Sub-module:  Design of column (loaded axially)

@author: Rutvik Joshi, N Swaroop, Adnan Abdullah (Project interns, 2021)
         Danish Ansari

Reference:
            1) IS 800: 2007 General construction in steel - Code of
practice (Third revision)

"""
import logging
import math
import numpy as np
from Common import *
from design_type.connection.moment_connection import MomentConnection
from utils.common.material import Material
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


class ColumnDesign(Member):
    def __init__(self):
        super(ColumnDesign, 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_column_design)
        tabs.append(t2)

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

        return tabs

    def tab_value_changed(self):
        """

        :return: This function is used to update the values of the keys in
design preferences,
         which are dependent on other inputs.
         It returns list of tuple which contains, tab name, keys whose
values will be changed,
         function to change the values and arguments for the function.

         [Tab Name, [Argument list], [list of keys to be updated], input
widget type of keys, change_function]

         Here Argument list should have only one element.
         Changing of this element,(either changing index or text depending
on widget type),
         will update the list of keys (this can be more than one).
         TODO: input widget type of keys (3rd element) is no longer
required. needs to be removed

         """
        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, ['Label_8',
KEY_SEC_MATERIAL])
        t1 = (KEY_DISP_COLSEC, TYPE_COMBOBOX, [KEY_SEC_MATERIAL])
        design_input.append(t1)

        # t1 = (KEY_DISP_COLSEC, TYPE_TEXTBOX, [KEY_SEC_FU, KEY_SEC_FY,
'Label_21'])
        t1 = (KEY_DISP_COLSEC, TYPE_TEXTBOX, [KEY_SEC_FU, KEY_SEC_FY])
        design_input.append(t1)

        t2 = ("Optimization", TYPE_TEXTBOX, [KEY_ALLOW_UR,
KEY_EFFECTIVE_AREA_PARA, KEY_STEEL_COST])
        design_input.append(t2)

        t2 = ("Optimization", TYPE_COMBOBOX, [KEY_OPTIMIZATION_PARA,
KEY_ALLOW_CLASS1, KEY_ALLOW_CLASS2, KEY_ALLOW_CLASS3, KEY_ALLOW_CLASS4])
        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):
        """

        :return: This function is used to choose values of design
preferences to be saved to
        design dictionary if design preference is never opened by user. It
sets are design preference values to default.
        If any design preference value needs to be set to input dock value,
tuple shall be written as:

        (Key of input dock, [List of Keys from design prefernce], 'Input
Dock')
```

```
        If the values needs to be set to default,

        (None, [List of Design Prefernce Keys], '')

         """
        design_input = []

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

        t2 = (None, [KEY_ALLOW_UR, KEY_EFFECTIVE_AREA_PARA,
KEY_OPTIMIZATION_PARA, KEY_ALLOW_CLASS1, KEY_ALLOW_CLASS2,
KEY_ALLOW_CLASS3,
                     KEY_ALLOW_CLASS4, KEY_STEEL_COST,
KEY_DP_DESIGN_METHOD], '')
        design_input.append(t2)

        return design_input

    def refresh_input_dock(self):
        """

        :return: This function returns list of tuples which has keys that
needs to be updated,
         on changing Keys in design preference (ex: adding a new section to
database should reflect in input dock)

         [(Tab Name,  Input Dock Key, Input Dock Key type, design
preference key, Master key, Value, Database Table Name)]
        """
        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_UR: '1.0',
            KEY_EFFECTIVE_AREA_PARA: '1.0',
            KEY_OPTIMIZATION_PARA: 'Utilization Ratio',
            KEY_STEEL_COST: '50',
            KEY_ALLOW_CLASS1: 'Yes',
            KEY_ALLOW_CLASS2: 'Yes',
            KEY_ALLOW_CLASS3: 'Yes',
            KEY_ALLOW_CLASS4: 'Yes',
            KEY_DP_DESIGN_METHOD: "Limit State Design",
        }[key]
```

```
        return val

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

    def module_name(self):
        return KEY_DISP_COMPRESSION_COLUMN

    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)"""

        options_list = []

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

        t1 = (KEY_MODULE, KEY_DISP_COMPRESSION_COLUMN, TYPE_MODULE, None,
True, 'No Validator')
        options_list.append(t1)
```

```
        t2 = (KEY_SEC_PROFILE, KEY_DISP_SEC_PROFILE, TYPE_COMBOBOX,
VALUES_SEC_PROFILE, True, 'No Validator')
        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)

        t5 = (KEY_ACTUAL_LEN_ZZ, KEY_DISP_ACTUAL_LEN_ZZ, TYPE_TEXTBOX,
None, True, 'Int Validator')
        options_list.append(t5)

        t6 = (KEY_ACTUAL_LEN_YY, KEY_DISP_ACTUAL_LEN_YY, TYPE_TEXTBOX,
None, True, 'Int Validator')
        options_list.append(t6)

        t9 = (None, KEY_DISP_END_CONDITION, TYPE_TITLE, None, True, 'No
Validator')
        options_list.append(t9)

        t10 = (KEY_END1, KEY_DISP_END1, TYPE_COMBOBOX, VALUES_END1, True,
'No Validator')
        options_list.append(t10)

        t11 = (KEY_END2, KEY_DISP_END2, TYPE_COMBOBOX, VALUES_END2, True,
'No Validator')
        options_list.append(t11)

        t12 = (KEY_IMAGE, None, TYPE_IMAGE_COMPRESSION,
"./ResourceFiles/images/6.RRRR.PNG", True, 'No Validator')
        options_list.append(t12)

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

        t8 = (KEY_AXIAL, KEY_DISP_AXIAL, TYPE_TEXTBOX, None, True, 'Int
Validator')
        options_list.append(t8)

        return options_list

    def fn_profile_section(self):

        profile = self[0]
        if profile == 'Beams':
            return connectdb("Beams", call_type="popup")
        elif profile == 'Columns':
            return connectdb("Columns", call_type="popup")
        elif profile == 'RHS':
            return connectdb("RHS", call_type="popup")
        elif profile == 'SHS':
            return connectdb("SHS", call_type="popup")
        elif profile == 'CHS':
```

```
            return connectdb("CHS", call_type="popup")

    def fn_end1_end2(self):

        end1 = self[0]
        if end1 == 'Fixed':
            return VALUES_END2
        elif end1 == 'Free':
            return ['Fixed']
        elif end1 == 'Hinged':
            return ['Fixed', 'Hinged', 'Roller']
        elif end1 == 'Roller':
            return ['Fixed', 'Hinged']

    def fn_end1_image(self):

        if self == 'Fixed':
            return "./ResourceFiles/images/6.RRRR.PNG"
        elif self == 'Free':
            return "./ResourceFiles/images/1.RRFF.PNG"
        elif self == 'Hinged':
            return "./ResourceFiles/images/5.RRRF.PNG"
        elif self == 'Roller':
            return "./ResourceFiles/images/4.RRFR.PNG"

    def fn_end2_image(self):

        end1 = self[0]
        end2 = self[1]

        if end1 == 'Fixed':
            if end2 == 'Fixed':
                return "./ResourceFiles/images/6.RRRR.PNG"
            elif end2 == 'Free':
                return "./ResourceFiles/images/1.RRFF_rotated.PNG"
            elif end2 == 'Hinged':
                return "./ResourceFiles/images/5.RRRF_rotated.PNG"
            elif end2 == 'Roller':
                return "./ResourceFiles/images/4.RRFR_rotated.PNG"
        elif end1 == 'Free':
            return "./ResourceFiles/images/1.RRFF.PNG"
        elif end1 == 'Hinged':
            if end2 == 'Fixed':
                return "./ResourceFiles/images/5.RRRF.PNG"
            elif end2 == 'Hinged':
                return "./ResourceFiles/images/3.RFRF.PNG"
            elif end2 == 'Roller':
                return "./ResourceFiles/images/2.FRFR_rotated.PNG"
        elif end1 == 'Roller':
            if end2 == 'Fixed':
                return "./ResourceFiles/images/4.RRFR.PNG"
            elif end2 == 'Hinged':
                return "./ResourceFiles/images/2.FRFR.PNG"

    def input_value_changed(self):

        lst = []

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

```
        t2 = ([KEY_END1], KEY_END2, TYPE_COMBOBOX, self.fn_end1_end2)
        lst.append(t2)

        t3 = ([KEY_END1, KEY_END2], KEY_IMAGE, TYPE_IMAGE,
self.fn_end2_image)
        lst.append(t3)

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

        # t4 = (KEY_END2, KEY_IMAGE, TYPE_IMAGE, self.fn_end2_image)
        # lst.append(t4)

        return lst

    def output_values(self, flag):

        out_list = []

        t1 = (None, DISP_TITLE_OPTIMUM_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, self.result_UR 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_EFF_SEC_AREA_ZZ, KEY_DISP_EFF_SEC_AREA_ZZ, TYPE_TEXTBOX,
round(self.result_effective_area, 2) if flag else '', True)
        out_list.append(t2)

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

        t2 = (KEY_EFF_LEN_ZZ, KEY_DISP_EFF_LEN_ZZ, TYPE_TEXTBOX,
round(self.result_eff_len_zz * 1e-3, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_EULER_BUCKLING_STRESS_ZZ,
KEY_DISP_EULER_BUCKLING_STRESS_ZZ, TYPE_TEXTBOX, round(self.result_ebs_zz,
2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_BUCKLING_CURVE_ZZ, KEY_DISP_BUCKLING_CURVE_ZZ,
TYPE_TEXTBOX, self.result_bc_zz if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_IMPERFECTION_FACTOR_ZZ, KEY_DISP_IMPERFECTION_FACTOR_ZZ,
TYPE_TEXTBOX, round(self.result_IF_zz, 2) if flag else '', True)
        out_list.append(t2)
```

```
        t2 = (KEY_SR_FACTOR_ZZ, KEY_DISP_SR_FACTOR_ZZ, TYPE_TEXTBOX,
round(self.result_srf_zz, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_NON_DIM_ESR_ZZ, KEY_DISP_NON_DIM_ESR_ZZ, TYPE_TEXTBOX,
round(self.result_nd_esr_zz, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_COMP_STRESS_ZZ, KEY_DISP_COMP_STRESS_ZZ, TYPE_TEXTBOX,
round(self.result_fcd_zz, 2) if flag else '', True)
        out_list.append(t2)

        t10 = (None, DISP_TITLE_YY, TYPE_TITLE, None, True)
        out_list.append(t10)

        t2 = (KEY_EFF_LEN_YY, KEY_DISP_EFF_LEN_YY, TYPE_TEXTBOX,
round(self.result_eff_len_yy * 1e-3, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_EULER_BUCKLING_STRESS_YY,
KEY_DISP_EULER_BUCKLING_STRESS_YY, TYPE_TEXTBOX, round(self.result_ebs_yy,
2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_BUCKLING_CURVE_YY, KEY_DISP_BUCKLING_CURVE_YY,
TYPE_TEXTBOX, self.result_bc_yy if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_IMPERFECTION_FACTOR_YY, KEY_DISP_IMPERFECTION_FACTOR_YY,
TYPE_TEXTBOX, round(self.result_IF_yy, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_SR_FACTOR_YY, KEY_DISP_SR_FACTOR_YY, TYPE_TEXTBOX,
round(self.result_srf_yy, 2) if flag else '', True)
        out_list.append(t2)

        t2 = (KEY_NON_DIM_ESR_YY, KEY_DISP_NON_DIM_ESR_YY, TYPE_TEXTBOX,
round(self.result_nd_esr_yy, 2) if flag else '', True)
        out_list.append(t2)

        # t2 = (KEY_EFF_SEC_AREA_YY, KEY_DISP_EFF_SEC_AREA_YY,
TYPE_TEXTBOX, round(self.effective_area, 2) if flag else '', True)
        # out_list.append(t2)

        t2 = (KEY_COMP_STRESS_YY, KEY_DISP_COMP_STRESS_YY, TYPE_TEXTBOX,
round(self.result_fcd_yy, 2) if flag else '', True)
        out_list.append(t2)

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

        t1 = (KEY_DESIGN_STRENGTH_COMPRESSION,
KEY_DISP_DESIGN_STRENGTH_COMPRESSION, TYPE_TEXTBOX,
round(self.result_capacity * 1e-3, 2) if flag else
        '', True)
        out_list.append(t1)

        return out_list

    def func_for_validation(self, design_dictionary):
```

```
        all_errors = []
        self.design_status = False
        flag = False
        option_list = self.input_values(self)
        missing_fields_list = []

        for option in option_list:
            if option[2] == TYPE_TEXTBOX:
                if design_dictionary[option[0]] == '':
                    missing_fields_list.append(option[1])
            elif option[2] == TYPE_COMBOBOX and option[0] not in
[KEY_SEC_PROFILE, KEY_END1, KEY_END2]:
                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)
            # flag = False
        else:
            flag = True

        if flag:
            self.set_input_values(self, design_dictionary)
            print(design_dictionary)
        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):
        super(ColumnDesign, self).set_input_values(self, design_dictionary)

        # section properties
        self.module = design_dictionary[KEY_MODULE]
        self.mainmodule = 'Member'
```

```
        self.sec_profile = design_dictionary[KEY_SEC_PROFILE]
        self.sec_list = design_dictionary[KEY_SECSIZE]
        self.material = design_dictionary[KEY_SEC_MATERIAL]

        # section data
        self.length_zz = float(design_dictionary[KEY_ACTUAL_LEN_ZZ])
        self.length_yy = float(design_dictionary[KEY_ACTUAL_LEN_YY])

        # end condition
        self.end_1 = design_dictionary[KEY_END1]
        self.end_2 = design_dictionary[KEY_END2]

        # factored loads
        self.load = Load(axial_force=design_dictionary[KEY_AXIAL],
shear_force="", moment="", moment_minor="", unit_kNm=True)

        # design preferences
        self.allowable_utilization_ratio =
float(design_dictionary[KEY_ALLOW_UR])
        self.effective_area_factor =
float(design_dictionary[KEY_EFFECTIVE_AREA_PARA])
        self.optimization_parameter =
design_dictionary[KEY_OPTIMIZATION_PARA]
        self.allow_class1 = design_dictionary[KEY_ALLOW_CLASS1]
        self.allow_class2 = design_dictionary[KEY_ALLOW_CLASS2]
        self.allow_class3 = design_dictionary[KEY_ALLOW_CLASS3]
        self.allow_class4 = design_dictionary[KEY_ALLOW_CLASS4]
        self.steel_cost_per_kg = float(design_dictionary[KEY_STEEL_COST])

        self.allowed_sections = []

        if self.allow_class1 == "Yes":
            self.allowed_sections.append('Plastic')
        if self.allow_class2 == "Yes":
            self.allowed_sections.append('Compact')
        if self.allow_class3 == "Yes":
            self.allowed_sections.append('Semi-Compact')
        if self.allow_class4 == "Yes":
            self.allowed_sections.append('Slender')

        print(self.allowed_sections)

        print("==================")
        print(self.module)
        print(self.sec_list)
        print(self.sec_profile)
        print(self.material)
        print(self.length_yy)
        print(self.length_zz)
        print(self.load)
        print(self.end_1, self.end_2)
        print("==================")

        # safety factors
        self.gamma_m0 = IS800_2007.cl_5_4_1_Table_5["gamma_m0"]["yielding"]

        # material property
        self.material_property = Material(material_grade=self.material,
thickness=0)

        # initialize the design status
```

```
        self.design_status_list = []
        self.design_status = False

        self.section_classification(self)
        self.design_column(self)
        self.results(self)

    # Simulation starts here
    def section_classification(self):
        """ Classify the sections based on Table 2 of IS 800:2007 """
        self.input_section_list = []
        self.input_section_classification = {}

        for section in self.sec_list:
            trial_section = section.strip("'")

            # fetching the section properties
            if self.sec_profile == VALUES_SEC_PROFILE[0]:  # Beams
                self.section_property = Beam(designation=trial_section,
material_grade=self.material)
            elif self.sec_profile == VALUES_SEC_PROFILE[1]:  # Columns
                self.section_property = Column(designation=trial_section,
material_grade=self.material)
            elif self.sec_profile == VALUES_SEC_PROFILE[2]:  # RHS
                self.section_property = RHS(designation=trial_section,
material_grade=self.material)
            elif self.sec_profile == VALUES_SEC_PROFILE[3]:  # SHS
                self.section_property = SHS(designation=trial_section,
material_grade=self.material)
            elif self.sec_profile == VALUES_SEC_PROFILE[4]:  # CHS
                self.section_property = CHS(designation=trial_section,
material_grade=self.material)
            else:
                self.section_property = Column(designation=trial_section,
material_grade=self.material)

            # updating the material property based on thickness of the
thickest element

self.material_property.connect_to_database_to_get_fy_fu(self.material,

max(self.section_property.flange_thickness,
self.section_property.web_thickness))

            # section classification
            if (self.sec_profile == VALUES_SEC_PROFILE[0]) or
(self.sec_profile == VALUES_SEC_PROFILE[1]):  # Beams or Columns

                if self.section_property.type == 'Rolled':
                    self.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]
                else:
                    self.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]

                self.web_class =
IS800_2007.Table2_iii((self.section_property.depth - (2 *
self.section_property.flange_thickness)),

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

classification_type='Axial compression')

            elif (self.sec_profile == VALUES_SEC_PROFILE[2]) or
(self.sec_profile == VALUES_SEC_PROFILE[3]):  # RHS or SHS
                self.flange_class =
IS800_2007.Table2_iii((self.section_property.depth - (2 *
self.section_property.flange_thickness)),

self.section_property.flange_thickness, self.material_property.fy,

classification_type='Axial compression')
                self.web_class = self.flange_class

            elif self.sec_profile == VALUES_SEC_PROFILE[4]:  # CHS
                self.flange_class =
IS800_2007.Table2_x(self.section_property.out_diameter,
self.section_property.flange_thickness,

self.material_property.fy, load_type='axial compression')
                self.web_class = self.flange_class

            if self.flange_class == 'Slender' or self.web_class ==
'Slender':
                self.section_class = 'Slender'
            else:
                if self.flange_class == 'Plastic' and self.web_class ==
'Plastic':
                    self.section_class = 'Plastic'
                elif self.flange_class == 'Plastic' and self.web_class ==
'Compact':
                    self.section_class = 'Compact'
                elif self.flange_class == 'Plastic' and self.web_class ==
'Semi-Compact':
                    self.section_class = 'Semi-Compact'
                elif self.flange_class == 'Compact' and self.web_class ==
'Plastic':
                    self.section_class = 'Compact'
                elif self.flange_class == 'Compact' and self.web_class ==
'Compact':
                    self.section_class = 'Compact'
                elif self.flange_class == 'Compact' and self.web_class ==
'Semi-Compact':
                    self.section_class = 'Semi-Compact'
                elif self.flange_class == 'Semi-Compact' and self.web_class
== 'Plastic':
                    self.section_class = 'Semi-Compact'
                elif self.flange_class == 'Semi-Compact' and self.web_class
== 'Compact':
                    self.section_class = 'Semi-Compact'
                elif self.flange_class == 'Semi-Compact' and self.web_class
== 'Semi-Compact':
                    self.section_class = 'Semi-Compact'
```

```
            logger.info("The flange of the trial section ({}) is {} and web
is {}. The section is {} [Reference: Cl 3.7, IS 800:2007].".
                        format(trial_section, self.flange_class,
self.web_class, self.section_class))

            if len(self.allowed_sections) == 0:
                logger.warning("Select at-least one type of section in the
design preferences tab.")
                logger.error("Cannot compute. Selected section
classification type is Null.")
                self.design_status = False
                self.design_status_list.append(self.design_status)

            if self.section_class in self.allowed_sections:
                self.input_section_list.append(trial_section)
                self.input_section_classification.update({trial_section:
self.section_class})

    def design_column(self):
        """ Perform design of column """
        # checking DP inputs
        if (self.allowable_utilization_ratio <= 0.10) or
(self.allowable_utilization_ratio > 1.0):
            logger.warning("The defined value of Utilization Ratio in the
design preferences tab is out of the suggested range.")
            logger.info("Provide an appropriate input and re-design.")
            logger.info("Assuming a default value of 1.0.")
            self.allowable_utilization_ratio = 1.0
            self.design_status = False
            self.design_status_list.append(self.design_status)

        if (self.effective_area_factor <= 0.10) or
(self.effective_area_factor > 1.0):
            logger.warning("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.info("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)

        if (self.steel_cost_per_kg == 0.10) or (self.effective_area_factor
> 1.0):
            logger.warning("The defined value of the cost of steel (in INR)
in the design preferences tab is out of the suggested range.")
            logger.info("Provide an appropriate input and re-design.")
            logger.info("Assuming a default rate of 50 (INR/kg).")
            self.steel_cost_per_kg = 50
            self.design_status = False
            self.design_status_list.append(self.design_status)

        if len(self.input_section_list) > 0:

            # initializing lists to store the optimum results based on
optimum UR and cost

            # 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 = []

            i = 1
            for section in self.input_section_list:  # iterating the design
over each section to find the most optimum section

                # fetching the section properties of the selected section
                if self.sec_profile == VALUES_SEC_PROFILE[0]:  # Beams
                    self.section_property = Beam(designation=section,
material_grade=self.material)
                elif self.sec_profile == VALUES_SEC_PROFILE[1]:  # Columns
                    self.section_property = Column(designation=section,
material_grade=self.material)
                elif self.sec_profile == VALUES_SEC_PROFILE[2]:  # RHS
                    self.section_property = RHS(designation=section,
material_grade=self.material)
                elif self.sec_profile == VALUES_SEC_PROFILE[3]:  # SHS
                    self.section_property = SHS(designation=section,
material_grade=self.material)
                elif self.sec_profile == VALUES_SEC_PROFILE[4]:  # CHS
                    self.section_property = CHS(designation=section,
material_grade=self.material)
                else:
                    self.section_property = Column(designation=section,
material_grade=self.material)


self.material_property.connect_to_database_to_get_fy_fu(self.material,
max(self.section_property.flange_thickness,

self.section_property.web_thickness))

                self.epsilon = math.sqrt(250 / self.material_property.fy)

                # initialize lists for updating the results dictionary
                list_zz = []
                list_yy = []

                list_zz.append(section)
                list_yy.append(section)

                # Step 1 - computing the effective sectional area
                self.section_class =
self.input_section_classification[section]

                if self.section_class == 'Slender':
                    logger.warning("The trial section ({}) is Slender.
Computing the Effective Sectional Area as per Sec. 9.7.2, "
                                   "Fig. 2 (B & C) of The National Building
Code of India (NBC), 2016.".format(section))

                    if (self.sec_profile == VALUES_SEC_PROFILE[0]) or
(self.sec_profile == VALUES_SEC_PROFILE[1]):  # Beams or Columns
                        self.effective_area = (2 * ((31.4 * self.epsilon *
self.section_property.flange_thickness) *

self.section_property.flange_thickness)) + \
```

```
                                                (2 * ((21 * self.epsilon *
self.section_property.web_thickness) *
self.section_property.web_thickness))
                    elif (self.sec_profile == VALUES_SEC_PROFILE[2]) or
(self.sec_profile == VALUES_SEC_PROFILE[3]):
                        self.effective_area = (2 * 21 * self.epsilon *
self.section_property.flange_thickness) * 2
                else:
                    self.effective_area = self.section_property.area  # mm2

                # 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)

                    logger.warning("Reducing the effective sectional area
as per the definition in the Design Preferences tab.")
                    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.effective_area /
self.effective_area_factor), 2), self.effective_area))
                else:
                    if self.section_class != 'Slender':
                        logger.info("The effective sectional area is taken
as 100% of the cross-sectional area [Reference: Cl. 7.3.2, IS 800:2007].")

                list_zz.append(self.section_class)
                list_yy.append(self.section_class)

                list_zz.append(self.effective_area)
                list_yy.append(self.effective_area)

                # Step 2 - computing the design compressive stress

                # 2.1 - Buckling curve classification and Imperfection
factor
                if (self.sec_profile == VALUES_SEC_PROFILE[0]) or
(self.sec_profile == VALUES_SEC_PROFILE[1]):  # Beams or Columns

                    if self.section_property.type == 'Rolled':
                        self.buckling_class_zz =
IS800_2007.cl_7_1_2_2_buckling_class_of_crosssections(self.section_property
.flange_width,

self.section_property.depth,

self.section_property.flange_thickness,

cross_section='Rolled I-sections',

section_type='Hot rolled')['z-z']
                        self.buckling_class_yy =
IS800_2007.cl_7_1_2_2_buckling_class_of_crosssections(self.section_property
.flange_width,

self.section_property.depth,

self.section_property.flange_thickness,

cross_section='Rolled I-sections',
```

```
section_type='Hot rolled')['y-y']
                    else:
                        self.buckling_class_zz =
IS800_2007.cl_7_1_2_2_buckling_class_of_crosssections(self.section_property
.flange_width,

self.section_property.depth,

self.section_property.flange_thickness,

cross_section='Welded I-section',

section_type='Hot rolled')['z-z']
                        self.buckling_class_yy =
IS800_2007.cl_7_1_2_2_buckling_class_of_crosssections(self.section_property
.flange_width,

self.section_property.depth,

self.section_property.flange_thickness,

cross_section='Welded I-section',

section_type='Hot rolled')['y-y']
                else:
                    self.buckling_class_zz = 'a'
                    self.buckling_class_yy = 'a'

                self.imperfection_factor_zz =
IS800_2007.cl_7_1_2_1_imperfection_factor(buckling_class=self.buckling_clas
s_zz)
                self.imperfection_factor_yy =
IS800_2007.cl_7_1_2_1_imperfection_factor(buckling_class=self.buckling_clas
s_yy)

                list_zz.append(self.buckling_class_zz)
                list_yy.append(self.buckling_class_yy)

                list_zz.append(self.imperfection_factor_zz)
                list_yy.append(self.imperfection_factor_yy)

                # 2.2 - Effective length
                self.effective_length_zz =
IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(self.
length_zz * 1e3,

end_1=self.end_1,

end_2=self.end_2)  # mm
                self.effective_length_yy =
IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(self.
length_yy * 1e3,

end_1=self.end_1,

end_2=self.end_2)  # mm

                list_zz.append(self.effective_length_zz)
                list_yy.append(self.effective_length_yy)
```

```
                # 2.3 - Effective slenderness ratio
                self.effective_sr_zz = self.effective_length_zz /
self.section_property.rad_of_gy_z
                self.effective_sr_yy = self.effective_length_yy /
self.section_property.rad_of_gy_y

                list_zz.append(self.effective_sr_zz)
                list_yy.append(self.effective_sr_yy)

                # 2.4 - Euler buckling stress
                self.euler_bs_zz = (math.pi ** 2 *
self.section_property.modulus_of_elasticity) / self.effective_sr_zz ** 2
                self.euler_bs_yy = (math.pi ** 2 *
self.section_property.modulus_of_elasticity) / self.effective_sr_yy ** 2

                list_zz.append(self.euler_bs_zz)
                list_yy.append(self.euler_bs_yy)

                # 2.5 - Non-dimensional effective slenderness ratio
                self.non_dim_eff_sr_zz =
math.sqrt(self.material_property.fy / self.euler_bs_zz)
                self.non_dim_eff_sr_yy =
math.sqrt(self.material_property.fy / self.euler_bs_yy)

                list_zz.append(self.non_dim_eff_sr_zz)
                list_yy.append(self.non_dim_eff_sr_yy)

                # 2.5 - phi
                self.phi_zz = 0.5 * (1 + (self.imperfection_factor_zz *
(self.non_dim_eff_sr_zz - 0.2)) + self.non_dim_eff_sr_zz ** 2)
                self.phi_yy = 0.5 * (1 + (self.imperfection_factor_zz *
(self.non_dim_eff_sr_yy - 0.2)) + self.non_dim_eff_sr_yy ** 2)

                list_zz.append(self.phi_zz)
                list_yy.append(self.phi_yy)

                # 2.6 - Design compressive stress
                self.stress_reduction_factor_zz = 1 / (self.phi_zz +
(self.phi_zz ** 2 - self.non_dim_eff_sr_zz ** 2) ** 0.5)
                self.stress_reduction_factor_yy = 1 / (self.phi_yy +
(self.phi_yy ** 2 - self.non_dim_eff_sr_yy ** 2) ** 0.5)

                list_zz.append(self.stress_reduction_factor_zz)
                list_yy.append(self.stress_reduction_factor_yy)

                self.f_cd_1_zz = (self.stress_reduction_factor_zz *
self.material_property.fy) / self.gamma_m0
                self.f_cd_1_yy = (self.stress_reduction_factor_yy *
self.material_property.fy) / self.gamma_m0
                self.f_cd_2 = self.material_property.fy / self.gamma_m0

                self.f_cd_zz = min(self.f_cd_1_zz, self.f_cd_2)
                self.f_cd_yy = min(self.f_cd_1_yy, self.f_cd_2)

                self.f_cd = min(self.f_cd_zz, self.f_cd_yy)

                list_zz.append(self.f_cd_1_zz)
                list_yy.append(self.f_cd_1_yy)

                list_zz.append(self.f_cd_2)
                list_yy.append(self.f_cd_2)
```

```
                list_zz.append(self.f_cd_zz)
                list_yy.append(self.f_cd_yy)

                list_zz.append(self.f_cd)
                list_yy.append(self.f_cd)

                # 2.7 - Capacity of the section
                self.section_capacity = self.f_cd * self.effective_area  #
N

                list_zz.append(self.section_capacity)
                list_yy.append(self.section_capacity)

                # 2.8 - UR
                self.ur = round(self.load.axial_force /
self.section_capacity, 3)

                list_zz.append(self.ur)
                list_yy.append(self.ur)
                self.optimum_section_ur.append(self.ur)

                # 2.9 - Cost of the section in INR
                self.cost = (self.section_property.unit_mass *
self.section_property.area * 1e-4) * min(self.length_zz, self.length_yy) *
\
                             self.steel_cost_per_kg

                list_zz.append(self.cost)
                list_yy.append(self.cost)
                self.optimum_section_cost.append(self.cost)

                # Step 3 - Storing the optimum results to a list in a
descending order

                list_1 = ['Designation', 'Section class', 'Effective area',
'Buckling_curve_zz', 'IF_zz', 'Effective_length_zz', 'Effective_SR_zz',
                          'EBS_zz', 'ND_ESR_zz', 'phi_zz', 'SRF_zz',
'FCD_1_zz', 'FCD_2', 'FCD_zz', 'FCD', 'Capacity', 'UR', 'Cost',
'Designation',
                          'Section class', 'Effective area',
'Buckling_curve_yy', 'IF_yy', 'Effective_length_yy', 'Effective_SR_yy',
'EBS_yy',
                          'ND_ESR_yy', 'phi_yy', 'SRF_yy', 'FCD_1_yy',
'FCD_2', 'FCD_yy', 'FCD', 'Capacity', 'UR', 'Cost']

                # 1- Based on optimum UR
                self.optimum_section_ur_results[self.ur] = {}

                list_2 = list_zz + list_yy
                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_zz + list_yy
                for j in list_1:
                    for k in list_2:
                        self.optimum_section_cost_results[self.cost][j] = k
                        list_2.pop(0)
                        break
        else:
            logger.warning("The section(s) defined for performing the
column design is/are not selected based on the selected Inputs and/or "
                           "Design Preferences")
            logger.error("Cannot compute!")
            logger.info("Change the inputs provided and re-design.")
            self.design_status = False
            self.design_status_list.append(self.design_status)

    def results(self):
        """ """
        # sorting results from the dataset

        # 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()

            # 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.")
                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)
            else:
                self.result_UR = self.optimum_section_ur[-1]  # optimum
section which passes the UR check
                self.design_status = True

        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]

        # print results
        if self.optimization_parameter == 'Utilization Ratio':
            self.result_designation =
self.optimum_section_ur_results[self.result_UR]['Designation']
            self.result_section_class =
self.optimum_section_ur_results[self.result_UR]['Section class']
            self.result_effective_area =
self.optimum_section_ur_results[self.result_UR]['Effective area']

            self.result_bc_zz =
self.optimum_section_ur_results[self.result_UR]['Buckling_curve_zz']
```

```
            self.result_bc_yy =
self.optimum_section_ur_results[self.result_UR]['Buckling_curve_yy']

            self.result_IF_zz =
self.optimum_section_ur_results[self.result_UR]['IF_zz']
            self.result_IF_yy =
self.optimum_section_ur_results[self.result_UR]['IF_yy']

            self.result_eff_len_zz =
self.optimum_section_ur_results[self.result_UR]['Effective_length_zz']
            self.result_eff_len_yy =
self.optimum_section_ur_results[self.result_UR]['Effective_length_yy']

            self.result_eff_sr_zz =
self.optimum_section_ur_results[self.result_UR]['Effective_SR_zz']
            self.result_eff_sr_yy =
self.optimum_section_ur_results[self.result_UR]['Effective_SR_yy']

            self.result_ebs_zz =
self.optimum_section_ur_results[self.result_UR]['EBS_zz']
            self.result_ebs_yy =
self.optimum_section_ur_results[self.result_UR]['EBS_yy']

            self.result_nd_esr_zz =
self.optimum_section_ur_results[self.result_UR]['ND_ESR_zz']
            self.result_nd_esr_yy =
self.optimum_section_ur_results[self.result_UR]['ND_ESR_yy']

            self.result_phi_zz =
self.optimum_section_ur_results[self.result_UR]['phi_zz']
            self.result_phi_yy =
self.optimum_section_ur_results[self.result_UR]['phi_yy']

            self.result_srf_zz =
self.optimum_section_ur_results[self.result_UR]['SRF_zz']
            self.result_srf_yy =
self.optimum_section_ur_results[self.result_UR]['SRF_yy']

            self.result_fcd_1_zz =
self.optimum_section_ur_results[self.result_UR]['FCD_1_zz']
            self.result_fcd_1_yy =
self.optimum_section_ur_results[self.result_UR]['FCD_1_yy']

            self.result_fcd_2 =
self.optimum_section_ur_results[self.result_UR]['FCD_2']

            self.result_fcd_zz =
self.optimum_section_ur_results[self.result_UR]['FCD_zz']
            self.result_fcd_yy =
self.optimum_section_ur_results[self.result_UR]['FCD_yy']

            self.result_fcd =
self.optimum_section_ur_results[self.result_UR]['FCD']
            self.result_capacity =
self.optimum_section_ur_results[self.result_UR]['Capacity']
            self.result_cost =
self.optimum_section_ur_results[self.result_UR]['Cost']
        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)
                else:
                    self.result_cost = trial_cost[0]  # optimum section
based on cost which passes the UR check
                    self.design_status = True

            # results
            self.result_designation =
self.optimum_section_cost_results[self.result_cost]['Designation']
            self.result_section_class =
self.optimum_section_cost_results[self.result_cost]['Section class']
            self.result_effective_area =
self.optimum_section_cost_results[self.result_cost]['Effective area']

            self.result_bc_zz =
self.optimum_section_cost_results[self.result_cost]['Buckling_curve_zz']
            self.result_bc_yy =
self.optimum_section_cost_results[self.result_cost]['Buckling_curve_yy']

            self.result_IF_zz =
self.optimum_section_cost_results[self.result_cost]['IF_zz']
            self.result_IF_yy =
self.optimum_section_cost_results[self.result_cost]['IF_yy']

            self.result_eff_len_zz =
self.optimum_section_cost_results[self.result_cost]['Effective_length_zz']
            self.result_eff_len_yy =
self.optimum_section_cost_results[self.result_cost]['Effective_length_yy']

            self.result_eff_sr_zz =
self.optimum_section_cost_results[self.result_cost]['Effective_SR_zz']
            self.result_eff_sr_yy =
self.optimum_section_cost_results[self.result_cost]['Effective_SR_yy']

            self.result_ebs_zz =
self.optimum_section_cost_results[self.result_cost]['EBS_zz']
```

```
            self.result_ebs_yy =
self.optimum_section_cost_results[self.result_cost]['EBS_yy']

            self.result_nd_esr_zz =
self.optimum_section_cost_results[self.result_cost]['ND_ESR_zz']
            self.result_nd_esr_yy =
self.optimum_section_cost_results[self.result_cost]['ND_ESR_yy']

            self.result_phi_zz =
self.optimum_section_cost_results[self.result_cost]['phi_zz']
            self.result_phi_yy =
self.optimum_section_cost_results[self.result_cost]['phi_yy']

            self.result_srf_zz =
self.optimum_section_cost_results[self.result_cost]['SRF_zz']
            self.result_srf_yy =
self.optimum_section_cost_results[self.result_cost]['SRF_yy']

            self.result_fcd_1_zz =
self.optimum_section_cost_results[self.result_cost]['FCD_1_zz']
            self.result_fcd_1_yy =
self.optimum_section_cost_results[self.result_cost]['FCD_1_yy']

            self.result_fcd_2 =
self.optimum_section_cost_results[self.result_cost]['FCD_2']

            self.result_fcd_zz =
self.optimum_section_cost_results[self.result_cost]['FCD_zz']
            self.result_fcd_yy =
self.optimum_section_cost_results[self.result_cost]['FCD_yy']

            self.result_fcd =
self.optimum_section_cost_results[self.result_cost]['FCD']
            self.result_capacity =
self.optimum_section_cost_results[self.result_cost]['Capacity']

        # end of the design simulation
        # overall design status
        for status in self.design_status_list:
            if status is False:
                self.design_status = False
                break
            else:
                self.design_status = True

        if self.design_status:
            logger.info(": ========== Design Status ============")
            logger.info(": Overall Column design is SAFE")
            logger.info(": ========== End Of Design ============")
        else:
            logger.info(": ========== Design Status ============")
            logger.info(": Overall Column design is UNSAFE")
            logger.info(": ========== End Of Design ============")

    ### start writing save_design from here!
    def save_design(self, popup_summary):

        if self.connectivity == 'Hollow/Tubular Column Base':
            if self.dp_column_designation[1:4] == 'SHS':
                select_section_img = 'SHS'
            elif self.dp_column_designation[1:4] == 'RHS':
```

```
                select_section_img = 'RHS'
            else:
                select_section_img = 'CHS'
        else:
            if self.column_properties.flange_slope != 90:
                select_section_img = "Slope_Beam"
            else:
                select_section_img = "Parallel_Beam"

            # column section properties
        if self.connectivity == 'Hollow/Tubular Column Base':
            if self.dp_column_designation[1:4] == 'SHS':
                section_type = 'Square Hollow Section (SHS)'
            elif self.dp_column_designation[1:4] == 'RHS':
                section_type = 'Rectangular Hollow Section (RHS)'
            else:
                section_type = 'Circular Hollow Section (CHS)'
        else:
            section_type = 'I Section'


        if self.section_property=='Columns' or
self.section_property=='Beams':
            self.report_column = {KEY_DISP_SEC_PROFILE: "ISection",
                                  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)}
        else:
            self.report_column = {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}


        self.report_input = \
            {KEY_MAIN_MODULE: self.mainmodule,
             KEY_MODULE: self.module, #"Axial load on column "
             KEY_DISP_SECTION_PROFILE: self.sec_profile,
             KEY_MATERIAL: self.material,
             KEY_DISP_ACTUAL_LEN_ZZ: self.length_zz,
             KEY_DISP_ACTUAL_LEN_YY: self.length_yy,
             KEY_DISP_END1: self.end_1,
             KEY_DISP_END2: self.end_2,
             KEY_DISP_AXIAL: self.load,
             KEY_DISP_SEC_PROFILE: self.sec_profile,
             KEY_DISP_SECSIZE: self.result_section_class,
             KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.euler_bs_yy,
             KEY_DISP_YIELD_STRENGTH_REPORT: self.result_bc_yy,


             "Column Section - Mechanical Properties": "TITLE",
             "Section Details": self.report_column,
             }

        self.report_check = []

        self.h = (self.beam_D - (2 * self.beam_tf))

        #1.1 Input sections display
        t1 = ('SubSection', 'List of Input
Sections',self.input_section_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 = ("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),ε = (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)
        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)
```

IS 800:2007 code additions to the Osdag database

```
# cl. 7.1, Design strength
@staticmethod
def cl_7_1_2_design_compressisive_strength_member( effective_area ,
design_compressive_stress , axial_load ):
    """
    Args:
        effective_area:effective sectional area as defined in 7.3.2
(float)
        design_compressive_stress:design compressive stress, obtained as
per 7.1.2.1            (float)
        axial_load:Load acting on column
(float)

    Returns:
        Design compressive strength
       'Pass', if the section qualifies as the required section_class,
'Fail' if it does not

    Note:
        Reference: IS 800 pg34
        @author:Rutvik Joshi
    """
    design_compressive_strength= effective_area * design_compressive_stress
#area in mm2,stress in kN/mm2
    if axial_load < design_compressive_strength:
#kN
        check = 'pass'
    else:
        check = 'fail'
    return check
#str

# cl. 7.2.2 Effective Length of Prismatic Compression Members
@staticmethod
def cl_7_2_2_effective_length_of_prismatic_compression_members( V1 , theta1
, V2 , theta2 , l , ry ):
    """
    Args:
        V1:Translation At One End
(str)
        theta1:Rotation At One End
(str)
        V2:Translation At the Other End
(str)
        theta2:Rotation At the Other End
(str)
        l: Complete length of member
(float)
        ry: section detail ry
(float)

    Returns:
        effective_length
(float)

    Note:
        Reference: IS 800 pg45
```

```
        @author:Rutvik Joshi
    """
    if V1 == 'r' and theta1 == 'r' and V2 == 'r' and theta2 == 'r':
        KL = 0.65 * l
# l in mm
    elif V1 == 'r' and theta1 == 'r' and V2 == 'r' and theta2 == 'f':
        KL = 0.8 * l
    elif V1 == 'r' and theta1 == 'r' and V2 == 'f' and theta2 == 'r':
        KL = 1.2 * l
    elif V1 == 'r' and theta1 == 'f' and V2 == 'r' and theta2 == 'f':
        KL = 1 * l
    elif V1 == 'f' and theta1 == 'r' and V2 == 'f' and theta2 == 'r':
        KL = 2 * l
    elif V1 == 'r' and theta1 == 'r' and V2 == 'f' and theta2 == 'f':
        KL = 2 * l
    lamba = KL / (ry * 10)
# ry in cm
    if lamba <= 180:
        lamba = lamba
    else:
        lamba = 180
    return lamba
#lamba in mm
# cl. 7.1.2.1, Design stress
@staticmethod
def cl_7_1_2_1_design_compressisive_stress(f_y,  lamba , buklingclass ):
    """
    Args:
        f_y:Yield stress
(float)
        lamba:Effective length of member
(float)
        buklingclass:Euler buckling class
(float)

    Returns:
        Design compressive stress
    Note:
        Reference: IS 800 pg34
        @author:Rutvik Joshi
    """
    Esteel= 2*(10**5)
    gamma_mo=1.1
#table 5 of IS 800:2007
    fcc = (3.14 ** 2 * Esteel) / lamba ** 2
    lambda1 = (f_y / fcc) ** 0.5
    phi = 0.5 * (1 + buklingclass * (lambda1 - 0.2) + lambda1 ** 2)
    facd = (f_y / gamma_mo) / (phi + ((phi ** 2 - lambda1 ** 2) ** 0.5))
    return facd
#kN/cm2

# cl. 7.1.2.1, Buckling Class of Cross-Sections
@staticmethod
def cl_7_1_2_2_buckling_class_of_crosssections( h , bf , tf ):
    """
    Args:
        h:Depth of section(mm)
(float)
        bf: Breadth of section(mm)
(float)
```

```
        tf:Thickness of flange
(float)

    Returns:
            alphaxx
(float)
            alphayy
(float)
            buklingclass
(float)

    Note:
        Reference: IS 800 pg44
        @author:Rutvik Joshi
    """
    #for rolled I section only
    if h / bf > 1.2:
        if tf <= 40:
            alphaxx = 0.21
            alphayy = 0.34
            buklingclass = 0.34
        if 40 <= tf <= 100:
            alphaxx = 0.34
            alphayy = 0.49
            buklingclass = 0.49
    if h / bf <= 1.2:
        if tf <= 100:
            alphaxx = 0.34
            alphayy = 0.49
            buklingclass = 0.49
        if tf > 100:
            alphaxx = 0.76
            alphayy = 0.76
            buklingclass = 0.76
    return(alphaxx , alphayy , buklingclass)
```

For Report function

```
def comp_column_class_section_check_required( bucklingclass , h , bf ):
    """
    Args:
        h:Depth of section(mm)
(float)
        bf: Breadth of section(mm)
(float)
        bucklingclass: buckling class
(float)
    Returns:
            bucklingclass_eq
    Note:
        Reference: IS 800 Cl.7.1.2.2
        @author:Rutvik Joshi
    """
    bucklingclass_eq=Math(inline=True)
    if bucklingclass==0.34:
        bucklingclass_eq.append(NoEscape(r'\begin{aligned}
frac{h}{b_\text{f}}>1.2\\'))
        bucklingclass_eq.append(NoEscape(r'
t_\text{f}<=40\\'))
        bucklingclass_eq.append(NoEscape(r'
\end(aligned)'))
    elif bucklingclass==0.49:
        if h/bf>1.2:
            bucklingclass_eq.append(NoEscape(r'\begin{aligned}
frac{h}{b_\text{f}}>1.2\\'))
            bucklingclass_eq.append(NoEscape(r'40 <= t_\text{f} <= 100'))
            bucklingclass_eq.append(NoEscape(r'
\end(aligned)'))
        else:
            bucklingclass_eq.append(NoEscape(r'\begin{aligned}
frac{h}{b_\text{f}}<=1.2\\'))
            bucklingclass_eq.append(NoEscape(r' t_\text{f} <= 100'))
            bucklingclass_eq.append(NoEscape(r'
\end(aligned)'))
    elif bucklingclass==0.76:
        bucklingclass_eq.append(NoEscape(r'\begin{aligned}
frac{h}{b_\text{f}}<=1.2\\'))
        bucklingclass_eq.append(NoEscape(r' t_\text{f} > 100'))
        bucklingclass_eq.append(NoEscape(r'
\end(aligned)'))
    return bucklingclass_eq


def comp_column_class_section_check_provided( bucklingclass , h , bf , tf ,
var_h_bf ):
    """
    Args:
        h:Depth of section(mm)
(float)
        bf: Breadth of section(mm)
(float)
        bucklingclass: buckling class
(float)
    Returns:
            bucklingclass_eq
    Note:
```

```
        Reference: IS 800 Cl.7.1.2.2
        @author:Rutvik Joshi
    """
    bucklingclass_eq=Math(inline=True)
    h=str(h)
    bf=str(bf)
    tf=str(tf)
    var_h_bf=str(var_h_bf)

    bucklingclass_eq.append(NoEscape(r'\begin{aligned}
frac{h}{b_\text{f}}&= frac{'+ h +r'}{' + bf + r'}\\'))
    bucklingclass_eq.append(NoEscape(r'
&='+ var_h_bf +r'\\'))
    bucklingclass_eq.append(NoEscape(r' t_f =' + tf + r' \\'))
    bucklingclass_eq.append(NoEscape(r'&[\text{Ref.
IS\:800:2007,\:Cl.7.1.2.2}]\end{aligned}'))
    return bucklingclass_eq


def cross_section_classification_required( section ):
    """
    Args:
        section:type of section
    Returns:
            cross_section_classification_required_eq
    Note:
        Reference: IS 800 Cl.7.1.2.2
        @author:Rutvik Joshi
    """
    section=str(section)
    cross_section_classification_required_eq=Math(inline=True)
    if section=='plastic':

cross_section_classification_required_eq.append(NoEscape(r'\begin{aligned}
frac{b}{t_f} & < 9.4 \times {epsilon} and frac{d}{tw}<=42 \times {epsilon}
\\'))
    elif section=='compact':

cross_section_classification_required_eq.append(NoEscape(r'\begin{aligned}
9.4 \times {epsilon} < frac{b}{t_f} <10.5 \times {epsilon} and
frac{d}{tw}<=42 \times {epsilon} \\'))
    elif section=='semi-compact':

cross_section_classification_required_eq.append(NoEscape(r'\begin{aligned}
10.5 \times {epsilon} < frac{b}{t_f} <15.7 \times {epsilon} and
frac{d}{tw}<=42 \times {epsilon} \\'))
    cross_section_classification_required_eq.append(NoEscape(r'
\end{aligned}'))
    return cross_section_classification_required_eq


def cross_section_classification_provided( tf , b1 , epsilon , section ,
b1_tf , d1_tw , ep1 , ep2 , ep3 , ep4 ):
    """
    Args:
        tf:Thickness of flange(mm)
(float)
        tw:Thickness of web(mm)
(float)
        b1,d1,epsilon,b1_tf,d1_tw,ep1,ep2,ep3,ep4..... refer 3.7.2 ,
3.7.4                                                          (float)
```

```
    Returns:
            cross_section_classification_required_eq
    Note:
        Reference: IS 800 Cl:3.7.2 & 3.7.4
        @author:Rutvik Joshi
    """
    tf = str(tf)
    b1 = str(b1)
    epsilon = str(epsilon)
    section=str(section)
    b1_tf=str(b1_tf)
    d1_tw=str(d1_tw)
    ep1=str(ep1)
    ep2=str(ep2)
    ep3=str(ep3)
    ep4=str(ep4)
    cross_section_classification_required_eq = Math(inline=True)

cross_section_classification_required_eq.append(NoEscape(r'\begin{aligned}
frac{b}{t_f} & =frac{' + b1 + r'}{' + tf + r'} \\'))
    cross_section_classification_required_eq.append(NoEscape(r'\'
& =' + b1_tf + r'\\'))
    if section=='plastic':
        cross_section_classification_required_eq.append(NoEscape(r' 9.4
\times{ ' + epsilon + '} &='+ ep1 + r'\\'))
    elif section=='compact':
        cross_section_classification_required_eq.append(NoEscape(r' 9.4
\times{' + epsilon + '} &='+ ep1 + r'\text{and} 10.5 \times {' + epsilon +
'} &='+ ep2 + r'\\'))
    elif section=='semi_compact':
        cross_section_classification_required_eq.append(NoEscape(r'  10.5
\times {' + epsilon + '} &='+ ep2 + r'\text{and} 15.7 \times{' + epsilon +
'} &=' + ep3 + r'\\'))
    cross_section_classification_required_eq.append(NoEscape(r'
frac{d}{t_w}&=' + d1_tw + r'\\'))
    cross_section_classification_required_eq.append(NoEscape(r' 42 \times '
+ epsilon + ' &='+ ep4 + r'\\'))
    cross_section_classification_required_eq.append(NoEscape(r'
\text{Therefore section is }'+ section + r'\end{aligned}'))
    return cross_section_classification_required_eq


def cl_7_2_2_slenderness_required( KL , ry , lamba ):
    """
        Args:
            KL,ry,lamba:refer cl:7.2.2
        Returns:
                slenderness_eq
        Note:
            Reference: IS 800 Cl:7.2.2
            @author:Rutvik Joshi
        """
    slenderness_eq=Math(inline=True)
    slenderness_eq.append(NoEscape(r'\begin{aligned} \text{slenderness
ratio(\lambda)} =frac{KL}{ry}<=180 \end{aligned}'))


def cl_7_2_2_slenderness_provided( KL , ry , lamba ):
    """
        Args:
            KL,ry,lamba:refer cl:7.2.2
```

```
        Returns:
                slenderness_eq
        Note:
            Reference: IS 800 Cl:7.2.2
            @author:Rutvik Joshi
        """
    KL = str(KL)
    ry = str(ry)
    lamba = str(lamba)
    slenderness_eq=Math(inline=True)
    slenderness_eq.append(NoEscape(r'\begin{aligned} \lambda =frac{'+ KL
+r'}{'+ ry +r'\\'))
    slenderness_eq.append(NoEscape(r'    &= '+ lamba + r'\end{aligned}'))


def cl_7_1_2_1_fcd_check_required( gamma_mo , f_y , f_y_gamma_mo ):
    """
            Args:
                facd:design compressive stress
                gamma_mo:1.1
                f_y:yield strength
                f_y_gamma_mo:f_y/gamma_mo
                axial:axial load
                Aeff:Aeff area determined by code
                A_eff_facd:A_eff*facd
            Returns:
                    facd_check_required_eq
            Note:
                Reference: IS 800 Cl:7.1.2.1
                @author:Rutvik Joshi
            """
    f_y= str(f_y)
    gamma_mo=str(gamma_mo)
    f_y_gamma_mo=str(f_y_gamma_mo)
    facd_check_required_eq=Math(inline=True)
    facd_check_required_eq.append(NoEscape(r' \begin{aligned} f_\text{cd} &
<= frac{d}{\gamma_\text{mo}}'))
    facd_check_required_eq.append(NoEscape(r'             & =frac{'+ f_y +
r'}{'+ gamma_mo + r'}'))
    facd_check_required_eq.append(NoEscape(r'             & = '+
f_y_gamma_mo + r'\\'))


def cl_7_1_2_1_fcd_check_provided( facd  ):
    """
                Args:
                    facd:design compressive stress
                Returns:
                        facd_check_required_eq
                Note:
                    Reference: IS 800 Cl:7.1.2.1
                    @author:Rutvik Joshi
                """
    facd = str(facd)
    facd_check_required_eq=Math(inline=True)
    facd_check_required_eq.append(NoEscape(r'\begin{aligned} f_\text{cd} &
='+ facd + r'\end{aligned}'))


def cl_7_1_2_design_comp_strength_required( axial ):
    """
```

```
                Args:
                    axial:axial load
                Returns:
                        facd_check_required_eq
                Note:
                    Reference: IS 800 Cl:7.1.2
                    @author:Rutvik Joshi
                """
    axial=str(axial)
    facd_check_required_eq=Math(inline=True)
    facd_check_required_eq.append(NoEscape(r' \begin{aligned} Compressive
designed strength & = A_\text{eff} \times {f_\text{cd}} '))
    facd_check_required_eq.append(NoEscape(r' A_\text{eff} \times
{f_\text{cd}} & > axial end{aligned}'))
    facd_check_required_eq.append(NoEscape(r' axial=' + axial +
r'\end{aligned}'))


def cl_7_1_2_design_comp_strength_provided( Aeff , facd , A_eff_facd ):
    """
                Args:
                    facd:design compressive stress
                    Aeff:Aeff area determined by code
                    A_eff_facd:A_eff*facd
                Returns:
                        facd_check_required_eq
                Note:
                    Reference: IS 800 Cl:7.1.2
                    @author:Rutvik Joshi
                """
    Aeff=str(Aeff)
    facd=str(facd)
    A_eff_facd=str(A_eff_facd)
    facd_check_required_eq=Math(inline=True)
    facd_check_required_eq.append(NoEscape(r' \begin{aglined} A_\text{eff}
\times {f_\text{cd}} &=' + Aeff + ' \times ' + facd + r'\\'))
    facd_check_required_eq.append(NoEscape(r' A_\text{eff_\text{provided}}
\times {f_\text{cd}} & =' + A_eff_facd + r'\end{aligned}'))
```

REFERENCES

Ghosh S. et al. (2021) Osdag: A Software for Structural Steel Design Using IS 800:2007. In: Adhikari S., Dutta A., Choudhury S. (eds) Advances in Structural Technologies. Lecture Notes in Civil Engineering, vol 81. Springer, Singapore.
https://link.springer.com/chapter/10.1007/978-981-15-5235-9_17
(https://doi.org/10.1007/978-981-15-5235-9_17)

IS 800:2007, General Construction in Steel-Code of Practice, Third Revision, Bureau of Indian Standards (BIS), New Delhi

Design of Steel Structures (2013), N. Subramanian, 12th Impression, Oxford University Press

Design of Steel Structures, S. Ramamrutham, Dhanpat Rai Publishing Company