FOSSEE Semester Long Internship Report

On

New Module Development for Osdag using advanced Optimization Techniques

Submitted by

Aman Agarwal

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

Manipal Institute of Technology

Manipal, Karnataka

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

July 1, 2025

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

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.

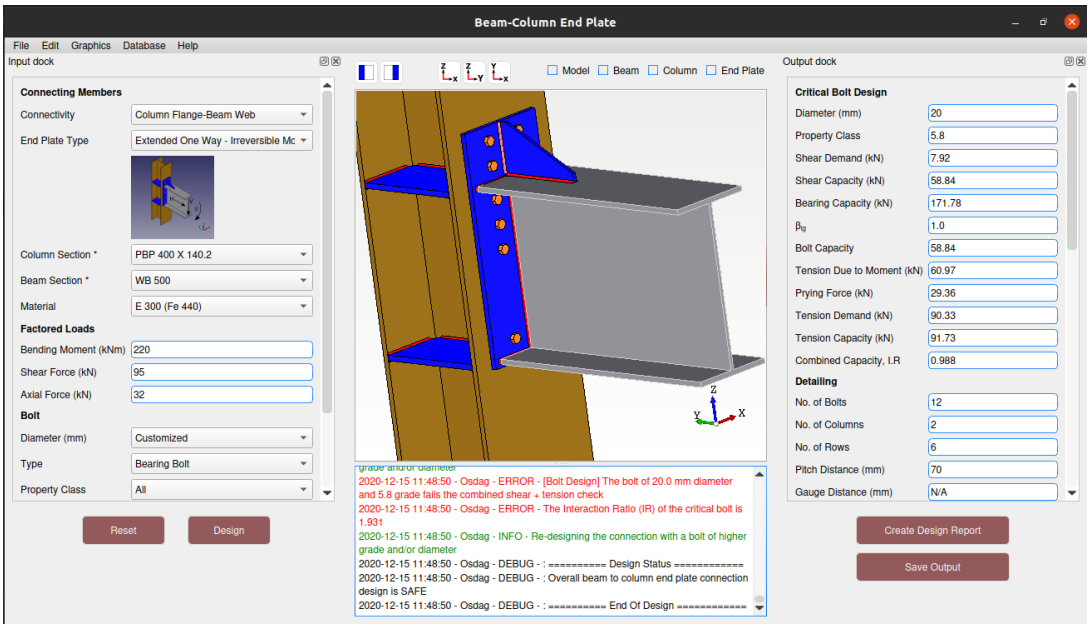

Figure 1.2: Osdag GUI

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

A desktop application designed to compare the life cycle costs of steel and concrete bridges using an SQLite database and PyQt5.

2.2 Tasks Done

As part of the screening task, I developed a Bridge Cost Comparison Tool which is a desktop application to analyze and compare the life cycle costs of steel and concrete bridges. This application is developed in Python with an embedded SQLite database for efficient storage and retrieval of cost data. The tool allows the user to specify the important bridge parameters such as span length, bridge width, expected traffic numbers, and design life. Then it calculates and compares the long-term costs of constructing steel and concrete bridge over its projected life cycle.

The application has a simple graphical interface for the user to input the necessary values, and the results are displayed interactively. There is a built-in bar plot visual, meaning that the user can clearly and graphically compare the calculated costs so that its easier for decision-makers to interpret and benchmark. Users can export the plot visualizations as PNG images for reporting or documentation as needed.

I organized the project while developing, into separate modules: entry point in main.py, graphical interface in gui.py, database work in database.py, cost computation logic in calculations.py, and visualizations in plot.py. I created a requirements.txt file so

that I could simplify the installation of dependencies, and a README.txt file to give a brief overview of the project. The whole tool was open-sourced under the MIT License. This experience not only improved my skills in designing software and graphical user interfaces, but also increased my knowledge of analysing infrastructure costs.

Chapter 3

Internship Task 1 Lap Joint Bolted Module

3.1 Task 1: Problem Statement

Module Development: Simple-Connection Lap-Joint-Bolted module

3.2 Task 1: Tasks Done

I contributed mainly to the Lap-Joint-Bolted sub-module in the Simple Connection module. This involved stemming the user interface to allow the user to add parameters in a smooth, user-friendly way to add relevant parameters. The other aspect I completed was the development of the main calculation and design check functions, which perform checks on the structure depending on the user input and applicable code. I also worked on formatting the output and illustrating the final output results in a clear way that the user could follow to make sure the design was followed, as well as easily outlined and check it aligned with the design. Overall completing the development from the user interface to the logic and output helped the user be sure that the sub-module worked from end to end.

3.3 Task 1: Python Code

This section showcases a Python script developed for designing a Lap-Joint-Bolted connection using the Osdag framework. The script allows for automation of the structural design process through the various calculations required based on user-defined parameters, such as plate thickness, load, and bolt combinations. Furthermore, incorporated within the script are design checks which were based on Indian Standards (IS) codes that provided safety assurance of the final connection design. Additionally, we determined the most applicable bolt diameter and level of grade for bolts through many options, which undoubtedly saves time in the designing process.

3.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user specifies the connecting member parameters, factored loads, and bolt properties.
- **Design Calculations**: The program computes the number of bolts required, their arrangement, and verifies the adequacy of the plate and bolt group.
- **Output**: Results include the number of bolts, their layout, and adequacy checks for the connection components.

3.3.2 Python Code

The Python script is shown below.

Listing 3.1: Lap-Joint-Bolted Connection Design in Osdag

```
1 %--------------------begin code-------------
2 # Imports
3 from .moment_connection import MomentConnection
4 from ...utils.common.component import *
5 from ...utils.common.is800_2007 import *
6 from ...Common import *
7 from ...design_report.reportGenerator_latex import CreateLatex
8 from ...Report_functions import *
```

```
from ...utils.common.load import Load
import logging
import math

# Functions for input, output, and design preferences
def input_values(self):
def output_values(self, flag)
def tab_list(self)

# Calculation Functions
def func_for_validation(self, design_dictionary):

    all_errors = []
    "check valid inputs and empty inputs in input dock"
    self.design_status = False
    flag = False
    flag1 = False
    flag2 = False

    option_list = self.input_values(self)
    missing_fields_list = []

    # print(f'\n func_for_validation option list = {option_list}'
    #       f'\n  design_dictionary {design_dictionary}')

    for option in option_list:
        if option[2] == TYPE_TEXTBOX:
            if design_dictionary[option[0]] == '':

                missing_fields_list.append(option[1])
            else:
                if option[2] == TYPE_TEXTBOX and option[0] ==
                   KEY_PLATE_WIDTH:

                    if float(design_dictionary[option[0]]) <= 0.0:
                        error = "Input value(s) cannot be equal or less
                            than zero."
                        all_errors.append(error)
                    else:
```

```
46                          flag1 = True
47
48                  if option[2] == TYPE_TEXTBOX and option[0] ==
                        KEY_TENSILE_FORCE:
49
50                      if float(design_dictionary[option[0]]) <= 0.0:
51                          error = "Input value(s) cannot be equal or less
                                than zero."
52                          all_errors.append(error)
53                      else:
54                          flag2 = True
55          else:
56              pass
57
58
59      if len(missing_fields_list) > 0:
60          error = self.generate_missing_fields_error_string(self,
                missing_fields_list)
61          all_errors.append(error)
62      else:
63          flag = True
64      if flag  and flag1 and flag2:
65          self.set_input_values(self, design_dictionary)
66      else:
67          return all_errors
68
69  def set_input_values(self, design_dictionary):
70
71      "initialisation of components required to design a tension member
            along with connection"
72
73      self.module = design_dictionary[KEY_MODULE]
74      self.mainmodule = "Lap Joint Bolted Connection"
75      self.main_material = design_dictionary[KEY_MATERIAL]
76      self.tensile_force = design_dictionary[KEY_TENSILE_FORCE]
77      self.width = design_dictionary[KEY_PLATE_WIDTH]
78      self.plate1 = Plate(thickness=[design_dictionary[
            KEY_PLATE1_THICKNESS]],
79                          material_grade=design_dictionary[KEY_MATERIAL],
```

```
                        width=design_dictionary[KEY_PLATE_WIDTH])
    self.plate2 = Plate(thickness=[design_dictionary[
        KEY_PLATE2_THICKNESS]],
                        material_grade=design_dictionary[KEY_MATERIAL],
                            width=design_dictionary[KEY_PLATE_WIDTH])
    self.bolt = Bolt(grade=design_dictionary[KEY_GRD], diameter=
        design_dictionary[KEY_D],
                     bolt_type=design_dictionary[KEY_TYP],
                     bolt_hole_type=design_dictionary[
                        KEY_DP_BOLT_HOLE_TYPE],
                     edge_type=design_dictionary[
                        KEY_DP_DETAILING_EDGE_TYPE],
                     mu_f=design_dictionary.get(KEY_DP_BOLT_SLIP_FACTOR,
                         None),
                     )
    self.planes = 1
    self.count = 0
    self.slip_res = None
    self.yield_stress = None
    # self.number_bolts = 0
    self.cap_red = False
    self.bolt_dia_grade_status = False
    self.dia_available = False
    self.final_pitch = 0
    self.final_end_dist = 0
    self.final_edge_dist = 0
    self.final_gauge = 0
    self.rows = 0
    self.cols = 0
    self.len_conn = 0
    self.max_gauge_round = 0
    self.max_pitch_round = 0
    self.utilization_ratio = 0
    self.bij = 0
    self.blg = 0
    self.select_bolt_dia_and_grade(self,design_dictionary)

def select_bolt_dia_and_grade(self,design_dictionary):
    self.dia_available = False
```

```
112     self.bolt_dia_grade_status = False
113
114     if isinstance(self.plate1.thickness, list):
115         self.plate1thk = self.plate1.thickness[0]
116
117     if isinstance(self.plate2.thickness, list):
118         self.plate2thk = self.plate2.thickness[0]
119
120     self.bolt_conn_plates_t_fu_fy = []
121     self.bolt_conn_plates_t_fu_fy.append((float(self.plate1thk), self.
            plate1.fu, self.plate1.fy))
122     self.bolt_conn_plates_t_fu_fy.append((float(self.plate2thk), self.
            plate2.fu, self.plate2.fy))
123
124     if float(self.plate1thk) < float(self.plate2thk):
125         self.plate = self.plate1
126         self.pltthk = float(self.plate1thk)
127         self.yield_stress = self.plate1.fy
128     else:
129         self.plate = self.plate2
130         self.pltthk = float(self.plate2thk)
131         self.yield_stress = self.plate2.fy
132
133     for self.bolt.bolt_diameter_provided in self.bolt.bolt_diameter:
134         if 8 * float(self.bolt.bolt_diameter_provided) > (float(self.
                plate1thk) + float(self.plate2thk)):
135             self.dia_available = True
136
137             for self.bolt.bolt_grade_provided in self.bolt.bolt_grade:
138
139                 self.bolt.calculate_bolt_spacing_limits(
                        bolt_diameter_provided=float(self.bolt.
                        bolt_diameter_provided),
140                                                         conn_plates_t_fu_fy
                                                            =self.
                                                            bolt_conn_plates_t_fu_fy
                                                            ,n=self.planes)
141
142                 # self.max_pitch_round = self.max_gauge_round =
```

```
143                 # self.bolt.calculate_bolt_capacity(
                        bolt_diameter_provided=float(self.bolt.
                        bolt_diameter_provided),
144                 #                                   bolt_grade_provided=float(
                        self.bolt.bolt_grade_provided),
145                 #                                   conn_plates_t_fu_fy=self.
                        bolt_conn_plates_t_fu_fy,
146                 #                                   n_planes=self.planes, e=
                        float(self.bolt.min_end_dist_round),
147                 #                                   p=float(self.bolt.
                        min_pitch_round))
148                 # self.bolt.calculate_bolt_tension_capacity(
                        bolt_diameter_provided=self.bolt.
                        bolt_diameter_provided,
149                 #
                        bolt_grade_provided=self.bolt.bolt_grade_provided)
150                 # print("fnafnafan",self.bolt.bolt_capacity)
151                 self.bolt.min_pitch_round = min(self.bolt.
                        min_pitch_round, 2.5 * float(self.bolt.
                        bolt_diameter_provided))
152                 self.bolt.min_gauge_round = min(self.bolt.
                        min_gauge_round, 2.5 * float(self.bolt.
                        bolt_diameter_provided))
153
154                 if design_dictionary[KEY_DP_DETAILING_EDGE_TYPE] == '
                        Sheared or hand flame cut':
155                     self.bolt.min_edge_dist_round = round(max(1.7 *
                            float(self.bolt.bolt_diameter_provided),self.
                            bolt.min_edge_dist_round),0)
156                     self.bolt.min_end_dist_round = round(max(1.7 *
                            float(self.bolt.bolt_diameter_provided),self.
                            bolt.min_end_dist_round),0)
157                 else:
158                     self.bolt.min_edge_dist_round = round(max(1.5 *
                            float(self.bolt.bolt_diameter_provided),self.
                            bolt.min_edge_dist_round),0)
159                     self.bolt.min_end_dist_round = round(max(1.5 *
                            float(self.bolt.bolt_diameter_provided),self.
                            bolt.min_end_dist_round),0)
```

```
160
161                 self.max_pitch_round = self.max_gauge_round = min(32 *
                        self.pltthk , 300)
162
163                 self.bolt.max_edge_dist_round = self.bolt.
                        max_end_dist_round = round(min(self.bolt.
                        max_edge_dist_round , 12 * self.pltthk * ((250 /
                        self.yield_stress)** 0.5 )),0)
164                 self.bolt.calculate_bolt_capacity(
                        bolt_diameter_provided=float(self.bolt.
                        bolt_diameter_provided),
165                                         bolt_grade_provided=float(
                                                self.bolt.
                                                bolt_grade_provided),
166                                         conn_plates_t_fu_fy=self.
                                                bolt_conn_plates_t_fu_fy,
167                                         n_planes=self.planes, e=float
                                                (self.bolt.
                                                min_end_dist_round),
168                                         p=float(self.bolt.
                                                min_pitch_round))
169                 num_bolts = float(self.tensile_force) / ( self.bolt.
                        bolt_capacity / 1000)
170                 # print("num_bolts",num_bolts)
171
172                 if num_bolts <= 2:
173                     self.bolt_dia_grade_status = True
174                     break
175
176
177             if self.bolt_dia_grade_status == True:
178                 break
179
180     if self.dia_available == False:
181         self.design_status = False
182         logger.warning(" : The combined thickness ({} mm) exceeds the
                allowable large grip limit check (of {} mm) for the minimum
                available "
183                        "bolt diameter of {} mm [Ref. Cl.10.3.3.2, IS
```

```
                                800:2007]."
184                           .format((float(self.plate1thk) + float(self.
                                plate2thk)),(8*self.bolt.bolt_diameter[-1]),
                                self.bolt.bolt_diameter[-1]))
185             logger.error(": Design is not safe. \n ")
186             logger.info(" :==========End Of design===========")
187
188         # elif self.dia_available == True and self.bolt_dia_grade_status ==
                False:
189         #     self.design_status = True
190         #     if self.bolt.bolt_type == 'Bearing Bolt':
191         #         self.bolt.bolt_bearing_capacity = round(float(self.bolt.
                    bolt_bearing_capacity),2)
192         #     self.bolt.bolt_shear_capacity = round(float(self.bolt.
                    bolt_shear_capacity),2)
193         #     self.bolt.bolt_capacity = round(float(self.bolt.bolt_capacity
                    ),2)
194         #     print(self.bolt)
195         #     self.number_r_c_bolts(self, design_dictionary)
196
197
198         else:
199             self.design_status = True
200             if self.bolt.bolt_type == 'Bearing Bolt':
201                 self.bolt.bolt_bearing_capacity = round(float(self.bolt.
                        bolt_bearing_capacity),2)
202             self.bolt.bolt_shear_capacity = round(float(self.bolt.
                    bolt_shear_capacity),2)
203             self.bolt.bolt_capacity = round(float(self.bolt.bolt_capacity)
                    ,2)
204             logger.info(" : Bolt diameter and grade selected are {} mm and
                    {} respectively.".format(self.bolt.bolt_diameter_provided,
                    self.bolt.bolt_grade_provided))
205             # print(self.bolt)
206             self.number_r_c_bolts(self, design_dictionary,0,0)
207
208
209 def number_r_c_bolts(self,design_dictionary,count=0,hit=0):
210
```

```
211     bolt_cap = self.bolt.bolt_capacity
212     if self.bolt.bolt_type == 'Bearing Bolt':
213         self.slip_res = 'N/A'
214     else:
215         self.slip_res = self.bolt.bolt_capacity
216         self.bolt.bolt_bearing_capacity = 'N/A'
217         self.bolt.bolt_shear_capacity = 'N/A'
218 
219     # print("fafafa",bolt_cap)
220 
221     if hit == 0:
222         self.number_bolts = float(self.tensile_force) /( bolt_cap /
                1000)
223     else:
224         self.number_bolts += 1
225 
226     print("Hit",hit,self.number_bolts)
227 
228     self.number_bolts = math.ceil(self.number_bolts)
229     if self.number_bolts < 2:
230         self.number_bolts = 2
231 
232     def check_no_cols(numbolts):  #in function for recursive call
233         if (2 * self.bolt.min_end_dist_round) + ((numbolts - 1 )*self.
                bolt.min_pitch_round) >= float(self.width):
234             return True
235         else:
236             return False
237 
238     self.cols = 1
239     self.rows = self.number_bolts
240     temp_rows = self.rows
241     while True:
242         if check_no_cols(temp_rows):
243             temp_rows = math.ceil(self.rows/(self.cols + 1))
244             self.cols += 1
245         else:
246             break
247     self.rows = math.ceil(self.rows/self.cols)
```

```
248     if self.rows == 1:
249         self.rows = 2
250 
251     if self.cols>1:
252         self.len_conn = (self.cols - 1)*self.bolt.min_pitch_round + 2*
                self.bolt.min_end_dist_round
253 
254     else:
255         self.len_conn = self.bolt.min_pitch_round + 2*self.bolt.
                min_end_dist_round
256     if self.number_bolts >= 2 and count == 0:
257         self.design_status = True
258         # print("Num bolts leaving",self.number_bolts)
259         self.check_capacity_reduction_1(self, design_dictionary)
260     elif self.number_bolts>=2 and count == 1:
261         self.design_status = True
262         self.final_formatting(self,design_dictionary)
263     else:
264         self.design_status = False
265         logger.error(": Number of min bolts not satisfied. \n ")
266         logger.info(" :==========End Of design===========")
267 
268 
269 def check_capacity_reduction_1(self,design_dictionary):
270     # print("Capacity red check 1")
271     if self.number_bolts > 2:
272         lg = (self.rows - 1)*self.bolt.min_pitch_round
273         if  lg > 15 * self.bolt.bolt_diameter_provided:
274             self.bij = 1.075 - (lg / (200 * self.bolt.
                    bolt_diameter_provided))
275     if self.bij >= 0.75 and self.bij <= 1.0:
276         self.cap_red = True
277         # print("1 cap red")
278         self.bolt.bolt_shear_capacity = self.bolt.bolt_shear_capacity *
                self.bij
279         if self.bolt.bolt_type == 'Bearing Bolt':
280             self.bolt.bolt_capacity = min(self.bolt.bolt_shear_capacity
                    , self.bolt.bolt_bearing_capacity)
281         else:
```

```
282                 self.slip_res = self.bolt.bolt_shear_capacity
283                 self.bolt.bolt_capacity = self.slip_res
284
285
286     self.design_status = True
287     self.check_capacity_reduction_2(self,design_dictionary)
288
289 def check_capacity_reduction_2(self,design_dictionary):
290     self.cap_red = False
291     # print("Capacity red check 2")
292     if self.plate1thk + self.plate2thk > 5 * self.bolt.
            bolt_diameter_provided:
293         self.blg = 8 / (3 + (self.plate1thk + self.plate2thk / self.
                bolt.bolt_diameter_provided))
294     if self.blg < self.bij and self.blg != 0:
295         self.cap_red = True
296         # print("blg",self.blg)
297         # print("2 cap red")
298         self.bolt.bolt_shear_capacity = self.bolt.bolt_shear_capacity *
                self.blg
299         if self.bolt.bolt_type == 'Bearing Bolt':
300             self.bolt.bolt_capacity = min(self.bolt.bolt_shear_capacity
                    , self.bolt.bolt_bearing_capacity)
301         else:
302             self.slip_res = self.bolt.bolt_shear_capacity
303             self.bolt.bolt_capacity = self.slip_res
304
305         self.number_r_c_bolts(self,design_dictionary,1,0)
306
307     if self.cap_red == False:
308         self.design_status = True
309         # print("Going to formatting")
310         # print("After checks 2 numbolts",self.number_bolts)
311         self.final_formatting(self,design_dictionary)
312
313 # Output results
314 def final_formatting(self,design_dictionary):
315     # print("I am herefa fafjafjafjafjajfjafajf")
316     # print(self.bolt)
```

```
317
318     gauge_dist = (float(self.width) - 2*self.bolt.min_end_dist_round)/(
            self.rows - 1)
319
320     if gauge_dist > self.max_gauge_round:
321         self.final_gauge = self.max_gauge_round
322         self.final_pitch = self.bolt.min_pitch_round
323
324         enddist = (float(self.width) - ((self.rows - 1)*self.
                final_gauge))/2
325         if enddist > self.bolt.max_end_dist_round:
326             self.design_status = False
327             self.number_r_c_bolts(self,design_dictionary,0,1)
328             # print("okay")
329
330             # self.design_status = True
331         else:
332             self.final_end_dist = enddist
333             self.final_edge_dist = enddist
334             self.design_status = True
335     else:
336         self.final_gauge = gauge_dist
337         self.final_pitch = self.bolt.min_pitch_round
338         enddist = (float(self.width) - ((self.rows - 1)*self.
                final_gauge))/2
339         if enddist > self.bolt.max_end_dist_round:
340             # self.loop_helper_func(self,design_dictionary)
341             # print("okay")
342             # self.design_status = False
343             self.design_status = False
344             self.number_r_c_bolts(self,design_dictionary,0,1)
345
346
347         else:
348             self.final_end_dist = enddist
349             self.final_edge_dist = enddist
350             self.design_status = True
351     # print("I got here")
352     if self.bolt.bolt_type == 'Bearing Bolt':
```

```
353             self.bolt.bolt_shear_capacity = self.bolt.bolt_shear_capacity/
                    1000
354             self.bolt.bolt_bearing_capacity = self.bolt.
                    bolt_bearing_capacity / 1000
355             self.bolt.bolt_bearing_capacity = round(self.bolt.
                    bolt_bearing_capacity, 2)
356             self.bolt.bolt_shear_capacity = round(self.bolt.
                    bolt_shear_capacity, 2)
357             self.bolt.bolt_capacity = self.bolt.bolt_capacity / 1000
358             self.bolt.bolt_capacity = round(self.bolt.bolt_capacity, 2)
359         else:
360             self.slip_res = self.slip_res / 1000
361             self.slip_res = round(self.slip_res, 2)
362             self.bolt.bolt_capacity = self.bolt.bolt_capacity / 1000
363             self.bolt.bolt_capacity = round(self.bolt.bolt_capacity, 2)
364 
365         # print("Going for util ratio")
366         # print("Still here")
367         # print("Numbolts",self.number_bolts)
368         bltcap = self.bolt.bolt_capacity
369         if bltcap < 1:
370             bltcap = 1
371         self.utilization_ratio = float(self.tensile_force) / (bltcap * self
                .number_bolts)
372         self.utilization_ratio = round(self.utilization_ratio, 2)
373 
374         self.final_gauge = round(self.final_gauge,0)
375         self.final_pitch = round(self.final_pitch,0)
376         # print("fafafafafa",self.final_edge_dist, self.final_end_dist,
                self.final_pitch, self.final_gauge)
377         print("FINAL FINAL",self.bolt)
378         print("Final Edge/End/Gauge/Pitch",self.final_edge_dist,self.
                final_end_dist,self.final_gauge,self.final_pitch)
379         logger.info("Design is successful. \n")
380         # print(self)
381         # print("faahfnafanfaf")
382         print("Max and min end edge dist ",self.bolt.max_end_dist_round,
                self.bolt.min_end_dist_round, self.bolt.max_edge_dist_round,
                self.bolt.min_edge_dist_round)
```

```
383     print("Max min gauge pitch dist",self.max_gauge_round,self.bolt.
            min_gauge_round, self.max_pitch_round, self.bolt.min_pitch_round
            )
384     # self.design_status = True
385
386 %-------------------- end code ---------------
```

3.3.3 Explanation of the Code

- **Line 1-11**: Imports necessary libraries and Osdag modules for design calculations and setup.
- **Line 14-16**: Defines the input parameters, output parameters, and design preference inputs.
- **Line 19-108**: Validates the user inputs and checks if all the mandatory inputs boxes are filled, raises an error if not. Also initializes calculation variables and populates them with user assigned values.
- **Line 110-206**: Iterate through the provided bolt grades and diameters, perform the necessary checks, select the best combination, and calculate the various capacities accordingly.
- **Line 209-266**: Calculates the number of bolts required and determines their arrangement in terms of rows and columns.
- **Line 269-311**: Capacity reduction checks to make the design optimized based on given conditions.
- **Line 314-384**: Format the final results to the desired number of decimal places and present the output.

3.4 Task 1: Documentation

The **LapJointBolted** module is a part of *Osdag*, a structural design automation software. This module automates the design of bolted lap joints, commonly used to connect two steel plates using bolts. These joints are designed to carry forces like tension, making

their proper arrangement crucial for structural safety and efficiency in applications such as bridges, towers, and buildings.

This module follows an object-oriented programming approach and inherits from a base class called `MomentConnection`, ensuring integration with other types of connections within Osdag.

1. User Preferences and Design Input Handling

The module starts by allowing users to select design preferences and enter required input values. These are handled through the following functions:

- **tab_list**: Controls visible input tabs in the UI (e.g., "Bolt", "Detailing").
- **input_dictionary_design_pref** and **input_values**: Define necessary user inputs such as material grade, plate thickness, bolt diameter, and tensile force.

Inputs are validated using drop-down menus and text boxes to ensure proper formatting and units. Default values are provided for ease of use. Users can choose bolt pre-tensioning, bolt hole type (standard or oversized), and plate edge preparation method (sheared, sawn, etc.).

2. Input Validation and Setup

Before design computations:

- **func_for_validation** ensures mandatory fields are filled and values like plate width or tensile force are positive.
- **set_input_values** stores validated inputs and sets up plate and bolt objects accordingly.

This prepares the software to understand the materials and components involved.

3. Selecting Bolt Diameter and Grade

A critical step is selecting an appropriate bolt diameter and grade:

- **select_bolt_dia_and_grade** tests different bolt diameter and grade combinations.

- It ensures bolts satisfy both geometric spacing requirements and strength demands (based on IS 800:2007).
- The smallest diameter and lowest grade that satisfy all checks are selected for economic design.

If no suitable combination is found, a warning is logged, and the design is marked unsafe.

4. Calculating Number of Bolts and Their Layout

Once a bolt is selected:

- **number_r_c_bolts** calculates the minimum number of bolts needed.
- Bolts are arranged in rows and columns to fit within plate width and satisfy spacing rules.
- Recursive adjustments are made if an arrangement is not feasible.
- A minimum of two bolts is enforced for safety.

5. Capacity Reduction Checks

To prevent overestimating connection strength, two checks are applied:

- **check_capacity_reduction_1**: Applies reduction factor b_{ij} for long bolt lines.
- **check_capacity_reduction_2**: Applies reduction factor b_{lg} if plate thickness is high.

If reductions are applied, bolt numbers and layouts are recalculated.

6. Final Formatting and Utilization Calculation

After a valid bolt layout is achieved:

- **final_formatting** computes final values for pitch, gauge, and end/edge distances.
- Adjustments are made if any distances violate IS code standards.

- Bolt capacities are recalculated and converted to kilonewtons.
- Utilization ratio is computed, indicating how much of the bolt strength is used (ideal: close to 1.0).

7. Generating Output

output_values produces a structured output including:

- Bolt diameter, grade, type
- Shear, bearing, and total bolt capacities
- Number of bolts, rows, and columns
- Bolt spacing, layout, and utilization ratio

These outputs are displayed in the UI and may be added to the design report.

8. 3D Visualization Support

To improve user interaction and clarity:

- Functions such as **call_3DColumn**, **call_3DPlate**, and **get_3d_components** are used.
- These render 3D views of bolts and plates using a PyQt interface.
- Components are color-coded and positioned for better understanding.

9. Additional Features and Logging

- **warn_text** alerts users about outdated section profiles.
- **save_design** logs when a design is saved, which is helpful for version control and professional tracking.

Conclusion

The **LapJointBolted** module provides a robust, automated solution for designing safe and code-compliant bolted lap joints in steel structures. It guides the user through:

- Input validation
- Optimal bolt selection
- Bolt layout and spacing calculations
- Capacity safety checks
- Output formatting and visualization

Its modular design, code compliance, and intuitive interface make it a valuable part of the Osdag suite for structural engineers.

Chapter 4

Internship Task 2 Plate Girder Module

4.1 Task 2: Problem Statement

Module Development: Welded-Plate-Girder module

4.2 Task 2: Tasks Done

My role mainly involved work on the Plate-Girder module in Osdag. The first part of my work was to create a good user interface, through which a user could easily input relevant design parameters and check relevant design codes. I was also responsible for ensuring that all calculation logic and design checks were executed correctly, in accordance with user inputs and the relevant design codes. Furthermore, my responsibilities also included ensuring the output results were displayed formatted in a usable and interpretable way for users, so they would not only be able to check that their design met the compliance levels stated, but also to check each step of their design. Working through the complete development process; from user interface, through to the backend logic and then the final result visual, I was able to ensure that the sub-module could be operated reliably end-to-end.

4.3 Task 2: Python Code

his section presents a Python script used for designing a Plate Girder using the Osdag environment. It facilitates the structural design process by using Particle Swarm Optimization (PSO) to arrive at an optimized set of girder parameters based on user-defined specifications (flange width and thickness, web thickness, girder length, type of stiffener and applied shear and moment). This prompts optimized designs with respect to both usability and material efficiency. In addition, it performs required design checks with respect to shear capacity and moment capacity so that given configuration fulfills the necessary fire resistance properties and performance to maintain the material for the user's specification. The checks are performed according to the Indian Standards (IS) codes to ensure that the final design is structurally sound and that code compliance has been taken into consideration.

4.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user specifies the dimensions of flanges and web, the stiffener details and other factored loads.
- **Design Calculations**: The program performs various design checks based on Indian Standards for shear and moment capacities, as well as stiffener design. After passing all the checks, it selects the most economical parameters to generate the final design.
- **Output**: The results include the various design parameters selected by the optimizer, the final stiffener details, load values, and the corresponding CAD design.

4.3.2 Python Code

The Python script is shown below.

Listing 4.1: Plate-Girder Design in Osdag

```
1 %--------------------begin code-------------
2 # Imports
```

```
import logging
import math
import pyswarm
import numpy as np
from pyswarms.single.global_best import GlobalBestPSO
from pyswarm import pso
from ...Common import *
# from ..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 ..member import Member
from ...Report_functions import *
from ...design_report.reportGenerator_latex import CreateLatex
from ...utils.common.common_calculation import *
from ..tension_member import *
from ...utils.common.Section_Properties_Calculator import
    BBAngle_Properties
from ...utils.common import is800_2007
from ...utils.common.component import *
from osdag.cad.items.plate import Plate
from ...utils.common.Unsymmetrical_Section_Properties import
    Unsymmetrical_I_Section_Properties

#GUI TO SELECT CUSTOM IN DESIGN PREFERENCES
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QDialog, QListWidget, QListWidgetItem
from PyQt5.QtWidgets import (
    QDialog, QLabel, QLineEdit, QPushButton, QFormLayout,
    QApplication, QMessageBox
)
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt
import re
import sys

# Input and output values, and design preferences
def tab_list(self)
```

```
40 def input_values(self)
41 def output_values(self, flag)
42
43 #Calculations and Logic
44 def func_for_validation(self, design_dictionary):
45     # print("DESIGN",design_dictionary)
46     # print(f"func_for_validation here")
47     all_errors = []
48     self.design_status = False
49     flag = False
50     self.output_values(self, flag)
51     flag1 = False
52     flag2 = False
53     flag3 = False
54     option_list = self.input_values(self)
55     missing_fields_list = []
56     # print(f'func_for_validation option_list {option_list}'
57     #     f"\n  design_dictionary {design_dictionary}"
58         #   )
59     for option in option_list:
60         if option[2] == TYPE_TEXTBOX or option[0] == KEY_LENGTH or
               option[0] == KEY_SHEAR or option[0] == KEY_MOMENT:
61             try:
62                 if design_dictionary[option[0]] == '':
63                     if design_dictionary['Total.Design_Type'] == '
                           Optimized':
64                         if design_dictionary[KEY_OVERALL_DEPTH_PG] == '
                               ' or design_dictionary[KEY_TOP_Bflange_PG]
                               == '' or design_dictionary[
                               KEY_BOTTOM_Bflange_PG] == '':
65                             pass
66                         else:
67                             missing_fields_list.append(option[1])
68                             continue
69
70                     else:
71                         missing_fields_list.append(option[1])
72                         continue
73                 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:
                    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_MOMENT:
                    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
            except:
                    error = "Input value(s) are not valid"
                    all_errors.append(error)

    if len(missing_fields_list) > 0:
        error = self.generate_missing_fields_error_string(self, missing_fields_list)
        all_errors.append(error)
    else:
        flag = True
    # print(f"flag {flag} flag1 {flag1} flag2 {flag2} flag3 {flag3}")
    if flag and flag1 and flag2 and flag3:
```

```
106             # print(f"\n design_dictionary{design_dictionary}")
107             self.set_input_values(self, design_dictionary)
108             # print("WORKING VALIDATION")
109             # if self.design_status ==False and len(self.failed_design_dict
                    )>0:
110             #     logger.error(
111             #         "Design Failed, Check Design Report"
112             #     )
113             #     return # ['Design Failed, Check Design Report'] @TODO
114             # elif self.design_status:
115             #     pass
116             # else:
117             #     logger.error(
118             #         "Design Failed. Selender Sections Selected"
119             #     )
120             #     return # ['Design Failed. Selender Sections Selected']
121         else:
122             return all_errors
123     def set_input_values(self, design_dictionary):
124 
125         self.module = design_dictionary[KEY_MODULE]
126         self.design_type = design_dictionary[KEY_OVERALL_DEPTH_PG_TYPE]
127         # print('design_type', design_dictionary[KEY_OVERALL_DEPTH_PG_TYPE
                ])
128         self.section_class = None
129         if self.design_type == 'Optimized':
130             # print('Optimized Design')
131             self.total_depth = 1
132             self.web_thickness_list = design_dictionary[
                    KEY_WEB_THICKNESS_PG]
133             self.top_flange_width = 1
134             self.top_flange_thickness_list = design_dictionary[
                    KEY_TOP_FLANGE_THICKNESS_PG]
135             self.bottom_flange_width = 1
136             self.bottom_flange_thickness_list = design_dictionary[
                    KEY_BOTTOM_FLANGE_THICKNESS_PG]
137             #optimize the initialization for list outputs
138             self.web_thickness = float(design_dictionary[
                    KEY_WEB_THICKNESS_PG][0])
```

```
139         self.top_flange_thickness = float(design_dictionary[
                KEY_TOP_FLANGE_THICKNESS_PG][0])
140         self.bottom_flange_thickness = float(design_dictionary[
                KEY_BOTTOM_FLANGE_THICKNESS_PG][0])
141
142     else:
143         # print('Cus Design')
144         self.total_depth = float(design_dictionary[KEY_OVERALL_DEPTH_PG
                ])
145         self.web_thickness_list = design_dictionary[
                KEY_WEB_THICKNESS_PG]
146         self.web_thickness = float(design_dictionary[
                KEY_WEB_THICKNESS_PG][0])
147         self.top_flange_width = float(design_dictionary[
                KEY_TOP_Bflange_PG])
148         self.top_flange_thickness = float(design_dictionary[
                KEY_TOP_FLANGE_THICKNESS_PG][0])
149         self.top_flange_thickness_list = design_dictionary[
                KEY_TOP_FLANGE_THICKNESS_PG]
150         self.bottom_flange_width = float(design_dictionary[
                KEY_BOTTOM_Bflange_PG])
151         self.bottom_flange_thickness = float(design_dictionary[
                KEY_BOTTOM_FLANGE_THICKNESS_PG][0])
152         self.bottom_flange_thickness_list = design_dictionary[
                KEY_BOTTOM_FLANGE_THICKNESS_PG]
153
154         #3 list loops for V inp<V_d and M inp < Md criteria for not
                considering thickness (3)
155         # self.total_depth = float(design_dictionary[
                KEY_OVERALL_DEPTH_PG])
156         # self.web_thickness_list = float(design_dictionary[
                KEY_WEB_THICKNESS_PG])
157         # self.top_flange_width = float(design_dictionary[
                KEY_TOP_Bflange_PG])
158         # self.top_flange_thickness_list = float(design_dictionary[
                KEY_TOP_FLANGE_THICKNESS_PG])
159         # self.bottom_flange_width = float(design_dictionary[
                KEY_BOTTOM_Bflange_PG])
160         # self.bottom_flange_thickness_list = float(design_dictionary[
```

```
            KEY_BOTTOM_FLANGE_THICKNESS_PG])
161
162
163     ########## - modify when the thickness becomes a list
164     thickness_for_mat = max(self.web_thickness,self.
            top_flange_thickness, self.bottom_flange_thickness)
165     #print('total_depth', self.total_depth, 'top_flange_thickness',
            self.top_flange_thickness, 'bottom_flange_thickness', self.
            bottom_flange_thickness)
166     self.eff_depth = self.total_depth - self.top_flange_thickness -
            self.bottom_flange_thickness
167
168     #print('eff_depth', self.eff_depth)
169     self.IntStiffnerwidth = min(self.top_flange_width,self.
            bottom_flange_width) - self.web_thickness/2 - 10
170     self.material = Material(design_dictionary[KEY_MATERIAL],
            thickness_for_mat)
171     self.eff_width_longitudnal = min(self.top_flange_width,self.
            bottom_flange_width) - self.web_thickness/2 - 10
172
173     #
            ------------------------------------------------------------------------
174     # if design_dictionary[KEY_IntermediateStiffener_thickness] == 'All
            ':
175     if design_dictionary[KEY_IntermediateStiffener_thickness] == '
            Customized':
176         design_dictionary[KEY_IntermediateStiffener_thickness_val] =
                PlateGirderWelded.int_thicklist
177     else:
178         design_dictionary[KEY_IntermediateStiffener_thickness_val] =
                VALUES_STIFFENER_THICKNESS
179
180     self.int_thickness_list = design_dictionary[
            KEY_IntermediateStiffener_thickness_val]
181
182     if design_dictionary[KEY_LongitudnalStiffener_thickness] == '
            Customized':
183         design_dictionary[KEY_LongitudnalStiffener_thickness_val] =
```

```
            PlateGirderWelded.long_thicklist
    else:
        design_dictionary[KEY_LongitudnalStiffener_thickness_val] =
            VALUES_STIFFENER_THICKNESS

    self.long_thickness_list = design_dictionary[
        KEY_LongitudnalStiffener_thickness_val]   #float conv required
    self.deflection_criteria= design_dictionary[KEY_MAX_DEFL]
    self.support_condition = 'Simply Supported'
    self.loading_case = design_dictionary[KEY_BENDING_MOMENT_SHAPE]
    self.shear_type = None
    self.support_type = design_dictionary[KEY_DESIGN_TYPE_FLEXURE]
    self.loading_condition = design_dictionary[KEY_LOAD]
    self.torsional_res = design_dictionary[KEY_TORSIONAL_RES]
    self.warping = design_dictionary[KEY_WARPING_RES]
    self.length = float(design_dictionary[KEY_LENGTH])

    self.effective_length = None
    self.allow_class = design_dictionary[KEY_ALLOW_CLASS]
    self.loading_case = design_dictionary[KEY_BENDING_MOMENT_SHAPE]
    self.beta_b_lt = None
    self.web_philosophy = design_dictionary[KEY_WEB_PHILOSOPHY]
    self.epsilon = math.sqrt(250 / self.material.fy)
    self.b1 = float(design_dictionary[KEY_SUPPORT_WIDTH])
    self.c = design_dictionary[KEY_IntermediateStiffener_spacing]
    self.Is = None
    self.IntStiffThickness = float(self.int_thickness_list[0])
    self.LongStiffThickness = float(self.long_thickness_list[0])
    self.x1= 0
    self.x2 = 0
    self.V_cr = None
    self.V_d = None
    self.V_tf = None
    self.long_Stiffner = design_dictionary[KEY_LongitudnalStiffener]
    self.load = Load(shear_force=design_dictionary[KEY_SHEAR],
        axial_force="",moment=design_dictionary[KEY_MOMENT],unit_kNm=
        True,)
    self.alpha_lt = 0.49
    self.phi_lt = None
```

```
218     self.gamma_m0 = IS800_2007.cl_5_4_1_Table_5["gamma_m0"]["yielding"]
219     self.X_lt = None
220     self.fbd_lt = None
221     self.Md = None
222     if self.support_type == 'Major Laterally Supported':
223         self.lefactor = 0.7
224     else:
225         self.lefactor = 1
226     self.M_cr = None
227     self.F_q = None
228     self.Critical_buckling_load = None
229     self.shear_ratio = 0
230     self.endshear_ratio = 0
231     self.moment_ratio = 0
232     self.deflection_ratio = 0
233     self.It = None
234     self.Iw = None
235     self.torsion_cnst = None
236     self.warping_cnst = None
237     self.critical_moment = None
238     self.fcd = None
239     self.end_stiffthickness = 0
240     self.stiffener_type = None
241     self.end_panel_stiffener_thickness = None
242     self.end_stiffwidth = min(self.top_flange_width,self.
            bottom_flange_width)/2 - self.web_thickness/2 - 10
243
244 def section_classification(self,design_dictionary):
245     self.design_status = False
246
247     #print("THICKNESS VALUES INT STIFFNER", self.int_thickness_list)
248     #print("THICKNESS VALUE LONG STIFFENER",self.long_thickness_list)
249     flange_class_top = IS800_2007.Table2_i(((self.top_flange_width / 2)
            ),self.top_flange_thickness,self.material.fy,'Welded')[0]
250     flange_class_bottom = IS800_2007.Table2_i(((self.
            bottom_flange_width / 2)),self.bottom_flange_thickness,self.
            material.fy,'Welded')[0]
251     web_class = IS800_2007.Table2_iii((self.total_depth - self.
            top_flange_thickness - self.bottom_flange_thickness),self.
```

```
        web_thickness,self.material.fy)
252     web_ratio = (self.total_depth - (self.top_flange_thickness + self.
        bottom_flange_thickness)) / self.web_thickness
253     flange_ratio_top = self.top_flange_width / (2 *self.
        top_flange_thickness)
254     flange_ratio_bottom = self.bottom_flange_width / (2 *self.
        bottom_flange_thickness)
255     # print("Section classification- top flange, bottom flange, Web",
        flange_class_top, flange_class_bottom,web_class,web_ratio,
        flange_ratio_top,flange_ratio_bottom)
256     if flange_class_bottom == "Slender" or web_class == "Slender" or
        flange_class_top == 'Slender':
257             self.section_class = "Slender"
258     else:
259         if flange_class_top == KEY_Plastic:
260             if web_class == KEY_Plastic:
261                 if flange_class_bottom == KEY_Plastic:
262                     self.section_class = KEY_Plastic
263                 elif flange_class_bottom == KEY_Compact:
264                     self.section_class = KEY_Compact
265                 else:  # SemiCompact
266                     self.section_class = KEY_SemiCompact
267             elif web_class == KEY_Compact:
268                 if flange_class_bottom in [KEY_Plastic, KEY_Compact]:
269                     self.section_class = KEY_Compact
270                 else:  # SemiCompact
271                     self.section_class = KEY_SemiCompact
272             else:  # web SemiCompact
273                 self.section_class = KEY_SemiCompact
274
275         elif flange_class_top == KEY_Compact:
276             if web_class == KEY_Plastic:
277                 if flange_class_bottom in [KEY_Plastic, KEY_Compact]:
278                     self.section_class = KEY_Compact
279                 else:  # SemiCompact
280                     self.section_class = KEY_SemiCompact
281             elif web_class == KEY_Compact:
282                 if flange_class_bottom in [KEY_Plastic, KEY_Compact]:
283                     self.section_class = KEY_Compact
```

```
                else:  # SemiCompact
                    self.section_class = KEY_SemiCompact
            else:  # web SemiCompact
                self.section_class = KEY_SemiCompact

        else:  # flange_class_top == SemiCompact
            self.section_class = KEY_SemiCompact
    # print("overall section class", self.section_class)
    self.Zp_req = self.load.moment * self.gamma_m0 / self.material.fy
    self.effective_length_beam(self, design_dictionary, self.length)

    # print( 'self.allow_class', self.allow_class)

    if self.section_class == 'Slender' and self.web_philosophy == '
        Thick Web without ITS':
        return False
    else:
        return True

def beta_value(self,design_dictionary,section_class):
    self.plast_sec_mod_z = Unsymmetrical_I_Section_Properties.
        calc_PlasticModulusZ(self,self.total_depth,self.top_flange_width
        ,self.bottom_flange_width,
                                                    self.web_thickness,self
                                                        .
                                                        top_flange_thickness
                                                        ,self.
                                                        bottom_flange_thickness
                                                        ,self.epsilon)
    self.elast_sec_mod_z =Unsymmetrical_I_Section_Properties.
        calc_ElasticModulusZz(self,self.total_depth,self.
        top_flange_width,self.bottom_flange_width,
                                                    self.web_thickness,self
                                                        .
                                                        top_flange_thickness
                                                        ,self.
                                                        bottom_flange_thickness
                                                        )
    # print('total_depth',self.total_depth)
```

```
308         # print('top_flange_thickness',self.top_flange_thickness)
309         # print('bottom_flange_thickness',self.bottom_flange_thickness)
310         # print('eff_depth',self.eff_depth)
311         # print("self.plast_sec_mod_z",self.plast_sec_mod_z)
312         # print("self.elast_sec_mod_z",self.elast_sec_mod_z)
313         self.Zp_req = self.load.moment * self.gamma_m0 / self.material.fy
314         if self.plast_sec_mod_z >= self.Zp_req:
315             pass
316             # logger.info( 'self.section_property.plast_sec_mod_z More than
                Requires',self.plast_sec_mod_z,self.Zp_req)
317         if section_class == KEY_Plastic or section_class == KEY_Compact:
318             self.beta_b_lt = 1.0
319         else:
320             self.beta_b_lt = (self.elast_sec_mod_z/ self.plast_sec_mod_z)
321         # print("Beta value",self.beta_b_lt)
322
323 def effective_length_beam(self, design_dictionary, length):
324     if design_dictionary[KEY_LENGTH_OVERWRITE] == 'NA':
325         self.effective_length = IS800_2007.
              cl_8_3_1_EffLen_Simply_Supported(Torsional=self.
              torsional_res,Warping=self.warping,
326                                                                  length
                                                                     =
                                                                     leng
                                                                     ,
                                                                     dept
                                                                     =(
                                                                     self
                                                                     .
                                                                     tota
                                                                     /100
                                                                     ,
                                                                     load
                                                                     =
                                                                     self
                                                                     .
                                                                     load
                                                                     )
```

```
327             # print(f"Working 1 {self.effective_length}")
328         else:
329             try:
330                 if float(design_dictionary[KEY_LENGTH_OVERWRITE]) <= 0:
331                     design_dictionary[KEY_LENGTH_OVERWRITE] = 'NA'
332                 else:
333                     length = length * float(design_dictionary[
                            KEY_LENGTH_OVERWRITE])
334 
335                 self.effective_length = length
336                 # print(f"Working 2 {self.effective_length}")
337             except:
338                 # print(f"Inside effective_length_beam",type(
                        design_dictionary[KEY_LENGTH_OVERWRITE]))
339                 logger.warning("Invalid Effective Length Parameter.")
340                 logger.info('Effective Length Parameter is set to default:
                        1.0')
341                 design_dictionary[KEY_LENGTH_OVERWRITE] = '1.0'
342                 self.effective_length_beam(self, design_dictionary, length)
343                 # print(f"Working 3 {self.effective_length}")
344         # print(f"Inside effective_length_beam",self.effective_length,
                design_dictionary[KEY_LENGTH_OVERWRITE])
345 
346 def optimized_method(self, design_dictionary, is_thick_web,
            is_symmetric):
347 
348 
349         variable_list = self.build_variable_structure(self,is_thick_web,
                is_symmetric)
350         lb, ub = self.get_bounds(self,variable_list)
351         lb = np.array(lb)
352         ub = np.array(ub)
353 
354         # 1) Compute normalized bounds [0  1 ]
355         lb_norm = np.zeros_like(lb)
356         ub_norm = np.ones_like(ub)
357 
358         # 2) Denormalize helper: map  u  [0,1]^n      x  [lb,ub]
359         def denormalize(u):
```

```
360         return lb + u * (ub - lb)
361
362     # 3) Wrap your existing constraint to accept u
363     def cons_norm(u):
364         x = denormalize(u)
365         return constraint(x)  # calls your original constraint
366
367     # 4) Wrap your existing obj_fn_single to accept u
368     def obj_norm(u):
369         x = denormalize(u)
370         return obj_fn_single(x)
371
372     # 5) Generate & normalize your initial swarm
373     init_real = self.generate_first_particle(
374         self, self.length, self.load.moment, self.material.fy,
375         is_thick_web, is_symmetric
376     )
377     init_norm = (init_real - lb) / (ub - lb)
378
379     #                                          END ADD
380
381
382     def constraint(particle):
383         sec = Section()
384         self.assign_particle_to_section(self,particle, variable_list,
                sec)
385
386         # 2) Rebuild design inputs so checks use this particles
                geometry
387         design = dict(design_dictionary)
388         for var in variable_list:
389             design[var] = getattr(sec, var)
390
391         # 3) Run your optimized capacity/deflection/slenderness check
392         max_ratio, slender_ok, thickness_ok = self.
                design_check_optimized_version(self,design)
393
394         # 4) Grab the other ratios set as attributes
```

```
395
396             _shear   = getattr(self, 'shear_ratio', float('inf'))
397             _moment   = getattr(self, 'moment_ratio', float('inf'))
398             _defl   = getattr(self, 'deflection_ratio', float('inf'))
399
400         # 5) Compute     = sqrt(E/Fy) for IS800:2007
401         E, Fy = self.material.modulus_of_elasticity, self.material.fy
402             = math.sqrt(E / Fy)
403         # 6) Extract the 4 key geometric values
404         depth = sec.D
405
406         tw = sec.tw
407         bf_top = sec.bf_top
408         tf_top = sec.tf_top
409         bf_bot = sec.bf_bot
410         tf_bot = sec.tf_bot
411         eff_depth = sec.D - sec.tf_top - sec.tf_bot
412
413         # 7) Compute semi-compact margins:
414         #     flange:    b_f/t_f <= 13.6            (13.6    t_f        b_f)
                0
415         #     web:       d/t_w <= 126              (126    t_w        d)
            0
416
417         m_web = (126.0 *    ) * tw - depth
418         m_fl_top = max((13.6 *    ) * tf_top - bf_top, 3 * m_web)
419         m_fl_bot = max((13.6 *    ) * tf_bot - bf_bot, 3 * m_web)
420
421         # 8) Build a fixed-length list of margins (negative = violate)
422         margins = [
423             1.0 - max_ratio,  # moment capacity
424             m_fl_top,  # top-flange slenderness
425             m_fl_bot,  # bot-flange slenderness
426             m_web,  # web slenderness
427             (1.0 if thickness_ok else -1.0),  # plate thickness limits
428             1.0 -   _shear  ,  # shear ratio
429             1.0 -   _moment  ,  # moment ratio
430             1.0 -   _defl    # deflection ratio
431         ]
```

```
432
433         # (Optional) debug print to see which constraint is worst
434         worst = min(range(len(margins)), key=lambda i: margins[i])
435         # print(f"  constraint: worst margin #{worst} = {margins[worst
                ]:.3f}")
436
437         return margins
438
439     def obj_fn(x):
440         results = []
441         for particle in x:
442             sec = Section()
443             self.assign_particle_to_section(self,particle,
                    variable_list, sec)
444             area = ((self.top_flange_thickness * self.top_flange_width)
                     +
445                     (self.bottom_flange_thickness * self.
                        bottom_flange_width) +
446                     (self.web_thickness * (self.total_depth - self.
                        top_flange_thickness - self.
                        bottom_flange_thickness)))
447             volume = area * self.length  # mm
448             mass = volume * 7.85e-6  # kg
449             results.append(mass)
450         return np.array(results)
451
452     # def obj_fn_single(x):
453     #     sec = Section()
454     #     self.assign_particle_to_section(self,x, variable_list, sec)
455     #     area = ((self.top_flange_thickness * self.top_flange_width) +
456     #             (self.bottom_flange_thickness * self.
            bottom_flange_width) +
457     #             (self.web_thickness * (self.total_depth - self.
            top_flange_thickness - self.bottom_flange_thickness)))
458     #     volume = area * self.length  # mm
459     #     mass = volume * 7.85e-6  # kg
460     #
461     #     if self.c is None or self.t_stiff is None:
462     #         mass_stiff = 0
```

```
463     #     else:
464     #         # Number of stiffeners = length/c minus one at each end
465     #         n_stiff = max(self.length / sec.c - 1, 0)
466     #         # Volume of all stiffeners (mm ): thickness    width
        effective depth    count
467     #         vol_stiff = n_stiff * 2 * (min(self.top_flange_width,self
        .bottom_flange_width) - self.web_thickness/2 - 10) * self.
        t_stiff * (self.total_depth - self.top_flange_thickness - self.
        bottom_flange_thickness)
468     #         mass_stiff = vol_stiff * 7.85e-6
469     #     mass += mass_stiff
470     #
471     #     return mass
472     def obj_fn_single(x):
473         sec = Section()
474         self.assign_particle_to_section(self,x, variable_list, sec)
475         area = (sec.bf_top * sec.tf_top +
476             sec.bf_bot * sec.tf_bot +
477             sec.tw   * (sec.D - sec.tf_top - sec.tf_bot))
478
479         volume = area * self.length  # mm
480         mass = volume * 7.85e-6  # kg
481
482         if sec.c is None or sec.t_stiff is None:
483             mass_stiff = 0.0
484         else:
485             # how many stiffeners fit (minus one at each end)
486             n_stiff = max(self.length / sec.c - 1, 0)
487
488             # approximate stiffener cross-section:
489             #   2 stiffeners per bay, width as the smaller flange minus
                    half the web minus 10 mm clearance,
490             #   height as the clear web depth
491             width_stiff = min(sec.bf_top, sec.bf_bot) - sec.tw / 2 - 10
492             height_stiff = sec.D - sec.tf_top - sec.tf_bot
493
494             vol_stiff = n_stiff * 2 * width_stiff * sec.t_stiff *
                height_stiff  # mm
495             mass_stiff = vol_stiff * 7.85e-6  # kg
```

```
496
497             # 4) Return total
498         return mass + mass_stiff
499
500     results = []
501     for run in range(10):
502         # Call PSO as usual
503         best_u, best_cost = pso(
504             obj_norm,
505             lb_norm, ub_norm,
506             f_ieqcons=cons_norm,
507             swarmsize=20,
508             maxiter=80,
509             debug=False   # (set True if you want debug output
510         )
511         best_pos = denormalize(best_u)
512         results.append((best_pos, best_cost))
513
514     best_pos, best_cost = min(results, key=lambda x: x[1])
515     #best_pos, best_cost = pso(obj_fn_single, lb, ub, f_ieqcons=
        constraint, swarmsize=100, maxiter=500, debug=True, init_pos=
        self.generate_first_particle(self,self.length, self.load.moment,
         self.material.fy, is_thick_web, is_symmetric))
516     margins = constraint(best_pos)
517     best_section = Section()
518     self.assign_particle_to_section(self,best_pos, variable_list,
        best_section)
519     logger.info("PSO calculation successfully completed")
520     # print("Best cost:", best_cost)
521     best_design_var = dict(zip(variable_list, best_pos))
522     # print("Best design variables:", best_design_var)
523     def ceil_to_nearest(x, multiple):
524         return float(math.ceil(x / multiple) * multiple)
525     if is_symmetric:
526         self.bottom_flange_thickness = self.top_flange_thickness =
            float(best_design_var['tf'])
527         for i in self.bottom_flange_thickness_list:
528             if float(i) > self.bottom_flange_thickness:
529                 self.bottom_flange_thickness = float(i)
```

```
                self.top_flange_thickness = float(i)
                break
        self.web_thickness = float(best_design_var['tw'])
        for i in self.web_thickness_list:
            if float(i) > self.web_thickness:
                self.web_thickness = float(i)
                break

        self.top_flange_width = self.bottom_flange_width = round(float(
            best_design_var['bf']),0)
        self.top_flange_width = self.bottom_flange_width =
            ceil_to_nearest(self.top_flange_width,10)
        self.total_depth = round(float(best_design_var['D']),0)
        self.total_depth =  ceil_to_nearest(self.total_depth,25)


        # self.IntStiffThickness = float(best_design_var[''])
        # for i in self.int_thickness_list:
        #     if float(i) > se
    else:
        self.bottom_flange_thickness = float(best_design_var['tf_bot'])
        for i in self.bottom_flange_thickness_list:
            if float(i) > self.bottom_flange_thickness:
                self.bottom_flange_thickness = float(i)
                break
        self.top_flange_thickness = float(best_design_var['tf_top'])
        for i in self.top_flange_thickness_list:
            if float(i) > self.top_flange_thickness:
                self.top_flange_thickness = float(i)
                break
        self.web_thickness = float(best_design_var['tw'])
        for i in self.web_thickness_list:
            if float(i) > self.web_thickness:
                self.web_thickness = float(i)
                break

        self.bottom_flange_width = round(float(best_design_var['bf_bot'
            ]),0)
        self.bottom_flange_width = ceil_to_nearest(self.
```

```
            bottom_flange_width,10)
566         self.top_flange_width = round(float(best_design_var['bf_top'])
            ,0)
567         self.top_flange_width = ceil_to_nearest(self.top_flange_width
            ,10)
568         self.total_depth = round(float(best_design_var['D']),0)
569         self.total_depth =  ceil_to_nearest(self.total_depth,25)
570
571
572     if not is_thick_web:
573         self.IntStiffThickness = float(best_design_var['t_stiff'])
574         for i in self.int_thickness_list:
575             if float(i) > self.IntStiffThickness:
576                 self.IntStiffThickness = float(i)
577                 break
578
579         self.c = round(float(best_design_var['c']),0)
580         self.c = ceil_to_nearest(self.c,10)
581
582     # logger.info("NEW PSO RUNNING")
583     logger.info(f"Optimized values : Flange width top and bottom {self.
            top_flange_width} {self.bottom_flange_width} flange thickness
            top and bottom {self.top_flange_thickness} { self.
            bottom_flange_thickness} web_thickness  {self.web_thickness}
            total depth { self.total_depth} C value {self.c} thickness
            stiffener { self.IntStiffThickness}")
584     self.design_check(self,design_dictionary)
585     # self.final_format(self,design_dictionary)
586     self.design_status = True
587
588 # OUTPUT
589 def final_format(self,design_dictionary):
590
591     self.result_designation = (str(int(self.total_depth)) + " x " +str(
            int(self.web_thickness)) + " x " +str(int(self.
            bottom_flange_width)) + " x " +str(int(self.
            bottom_flange_thickness)) + " x " +str(int(self.top_flange_width
            )) + " x "  +str(int(self.top_flange_thickness)))
592     if self.moment_ratio == None:
```

```
593             self.moment_ratio = 0
594         if self.shear_ratio == None:
595             self.shear_ratio = 0
596         # if self.deflection_ratio == None:
597         #     self.deflection_ratio = 0
598         # if self.web_buckling_ratio == None:
599         #     self.web_buckling_ratio = 0
600         # if self.web_crippling_ratio == None:
601         #     self.web_crippling_ratio = 0
602         # print("RATIOS",'moment ratio', self.moment_ratio, 'shear ratio',
                self.shear_ratio, 'deflection ratio', self.deflection_ratio)
603         # print(self.moment_ratio, self.shear_ratio)
604         self.result_UR = max(self.moment_ratio,self.shear_ratio, self.
                deflection_ratio)
605         self.section_classification_val = self.section_class
606         if self.beta_b_lt == None:
607             self.beta_b_lt = 0
608         self.betab = round(self.beta_b_lt,2)
609         self.effectivearea = Unsymmetrical_I_Section_Properties.calc_area(
                self,self.total_depth, self.top_flange_width, self.
                bottom_flange_width, self.web_thickness, self.
                top_flange_thickness, self.bottom_flange_thickness)
610         if self.Md == None:
611             self.Md = 0
612 
613         if self.M_cr == None:
614             self.M_cr = 0
615         if self.V_cr == None:
616             self.V_cr = 0
617         if self.It == None:
618             self.It = 0
619         if self.Iw == None:
620             self.Iw = 0
621 
622         if self.shear_type == 'Low':
623             self.design_moment = round(self.Md/1000000,1)
624         else:
625             self.design_moment = round(self.Md/1000000,1)
626         if self.support_type == 'Major Laterally Unsupported':
```

```
627             self.critical_moment = round(self.M_cr/1000000,1)
628             self.torsion_cnst = round(self.It/10000,1)
629             self.warping_cnst = round(self.Iw/1000000,1)
630         self.intstiffener_thk = self.IntStiffThickness
631         self.longstiffener_thk = self.LongStiffThickness
632         self.longstiffener_no = 0
633         if self.long_Stiffner == 'Yes and 1 stiffener':
634             self.longstiffener_no = 1
635         elif self.long_Stiffner == 'Yes and 2 stiffeners':
636             self.longstiffener_no = 2
637         self.intstiffener_spacing = self.c
638         self.end_panel_stiffener_thickness = self.end_stiffthickness
639         self.atop= 0
640         self.abot= 0
641         self.weld_stiff= None
642         self.atop, self.abot= self.design_welds_with_strength_web_to_flange
                (self, self.load.shear_force, self.top_flange_width, self.
                top_flange_thickness, self.bottom_flange_width, self.
                bottom_flange_thickness, self.web_thickness, self.eff_depth, [
                self.material.fu])
643         self.weld_stiff = self.weld_for_end_stiffener(self, self.
                end_stiffthickness, self.end_stiffwidth, self.load.shear_force,
                self.V_d, self.total_depth, self.top_flange_thickness, self.
                bottom_flange_thickness, self.web_thickness, [self.material.fu])
644         self.design_status = True
645 %-------------------- end code ---------------
```

4.3.3 Explanation of the Code

- **Line 3-36**: Imports essential libraries and modules from Osdag, including GUI elements, structural component definitions, and IS 800:2007 utilities required for the plate girder design.
- **Line 39-41**: Defines the input parameters, output parameters, and design preference inputs.
- **Line 44-242**: Validates the user inputs and checks if all the mandatory inputs boxes are filled, raises an error if not. Also initializes calculation variables and

populates them with user assigned values.

- **Line 244-344**: Implements the section classification procedure as per IS 800:2007 and initializes structural analysis parameters like plastic modulus, effective length, and beta factor.

- **Line 346-586**: Performs main calculations using either a customized check routine or an optimization algorithm like PSO for minimum weight or maximum efficiency.

- **Line 589-644**: Format the final results to the desired number of decimal places and present the output.

4.4 Task 2: Documentation

This code is part of a structural design tool built for designing **welded plate girders**—a specific type of steel beam widely used in bridges and buildings. The software provides a **Graphical User Interface (GUI)** that enables engineers to enter design parameters, select options, and view calculated results in compliance with Indian Standards (IS 800:2007 and IS 808:1989).

Rather than manually solving design equations, the tool automates decisions like selecting dimensions, verifying strength, and optimizing material usage. It supports both *standard* and *customized* design workflows.

Main Components of the Code

1. **User Input Dialogs**

   The software provides several GUI elements to help users input design data:

   - **RangeInputDialog:** Allows input of a numeric range (start, end, step). Useful for optimizing variables like plate thickness.
   - **PopupDialog:** Enables users to move items between *Available* and *Selected* lists. This helps customize options like stiffener thicknesses.
   - **My_ListWidget & My_ListWidgetItem:** Custom list box classes that sort and manage list entries for better user interaction.

2. **The Core Class:** `PlateGirderWelded`

   This is the main class that drives the welded plate girder design process. It inherits from a base class `Member` and contains all logic needed for computations and GUI interactions.

How the Design Process is Organized

- **A. Tabs for Input:**

  The GUI organizes inputs into multiple tabs, each focused on a specific design area:

  - *Girder Section*: For entering geometry, material, and dimensions.
  - *Stiffeners*: For specifying longitudinal and intermediate stiffeners.
  - *Optimization*: For setting parameters if automated optimization is used.
  - *Deflection*: For specifying maximum allowable deflection.

  Tabs are defined using the `tab_list()` function.

- **B. Design Preferences:**

  These store the user's selections for use throughout the design process.

  - `input_dictionary_design_pref()`: Saves data from design preference tabs.
  - `get_values_for_design_pref()`: Provides default values for parameters if the user hasn't entered any.

- **C. Dynamic Interface Updates:**

  Some GUI fields are reactive:

  - Changing structural type updates the deflection limit.
  - Selecting `Customized` opens range input dialogs.

  This behavior is managed via `tab_value_changed()` and `input_value_changed()` functions.

Design Logic

1. **Setting Input Values:**

   The function `set_input_values()` captures and stores all values entered in the GUI for use in calculations.

2. **Validation:**

   Before starting design computations, `func_for_validation()` checks for missing or invalid data, such as empty fields or non-numeric input.

3. **Design Execution:**

   - **Optimized Design:** Uses a Particle Swarm Optimization (PSO) algorithm to find the most efficient girder dimensions.
   - **Customized Design:** Verifies user-defined sections against IS code criteria for strength and stability.

Output and Feedback

After processing the design, the software calculates and displays:

- Section classification (Plastic, Compact, Slender)
- Utilization Ratio (a measure of design efficiency)
- Required stiffener thicknesses and positions
- Weld sizes
- Deflection and moment capacity

The `output_values()` function is responsible for collecting and organizing this information for display in the GUI.

3D Model and Logging Support

- `get_3d_components()`: Enables integration with 3D modeling tools.
- `set_osdaglogger()`: Logs events and warnings during the design process for debugging and tracking.

In Summary

This code is a specialized module for designing steel plate girders, combining engineering principles with user-friendly software features. It offers:

- A clear GUI for complex engineering inputs
- Smart optimization and customization features
- Automated code compliance checks
- Structured, readable outputs for easy interpretation

Every function is designed to reduce manual effort, minimize errors, and ensure efficient, standards-compliant design of welded plate girders.

Chapter 5

Conclusions

5.1 Tasks Accomplished

Successfully developed 2 sub-modules (Lap-Joint-Bolted and Plate-Girder) in Osdag as a part of the Coding team .

5.2 Skills Developed

In the course of the fellowship, I received an all-around operational experience that enhanced my technical and professional skills. This experience has improved my programming capabilities, as well as my ability to work with others. Technically, I improved my understanding of modular programming, which is breaking down code into smaller, reusable components and keeping those components functional for both maintainability and scalability. When we worked through error handling and debugging, I especially improved in working with structural design modules. I learned the steps to thoroughly identify, trace, and work through resolving bugs, and I learned to validate the inputs to the extent that they pertain to structural engineering. Working through the real-world engineering logic and thinking improved my logical reasoning and analytical reasoning, as I would take complicated structural calculations and provide an efficient translation into code. I learned to use some of the best practices of software, such as writing clean, organized, and documented code that would work in a larger system.

As a professional, this experience enhance my teamwork and collaboration skills. I learned to connect with peers and iterate together on shared codebases. I also learned

to participate in code reviews and give productive feedback. Communication was crucial for talking through technical requirements, but it was equally important when discussing implementation. The project also provided me with an opportunity to cultivate resilience, tenacity, and grit - I learned to work through problems initiated by flaws in the code and process, to adjust to changing requirements, and finally to deliver projects on short timelines. Lastly, I learned how to efficiently manage time and tasks while balancing development, testing and documenting throughout the fellowship.

Chapter A

Appendix

A.1 Work Reports

Name - Aman Agarwal	Mentor - Parth		
Date	**Day**	**TASK**	**Hours**
10th Feb 2025	Monday	Onboarding meeting - learned about the communities, had an icebreaker session to introduce everyone | Installed the osdag software and learned about the method of installation and the dependencies required | Surfed through the videos to get to know about the software from the perspective of a user, gained insights about the various features of the application	4
11th Feb 2025	Tuesday	Learned about the Code Architechture and Coding norms | Went through the github repository familiarizing myself with the code structure | Dived into the codes of some of the features which I had seen yesterday to figure out how they work in the backend and learned a little about the methods being used to optimize the designs and generate apt results.	4
12th Feb 2025	Wednesday	Got a quick introduction session about the installation from Mehendi and about how the CAD designs are made from Aryan. Delved into the code base to familiarize myself on the structure and learn about the dependencies.	4
13th Feb 2025	Thursday	Started on the column column cover plate bolted task and went though the import statements chain to dive into which module is doing what work in brief and learned about the common modules and listed them for future refernce.	4
14th Feb 2025	Friday	Delved into the common module functions to know what they are importing and why also looked into the parameters and calculations of the modules in the functions of the import chain. Started compiling the report.	4
15th Feb 2025	Saturday	Delved into the cad and report generation and learned about the parameters calculated bascically continuing prev day's work. Compiled the full report in detail.	5.5
16th Feb 2025	Sunday	WEEKLY LEAVE - Submission deadline for the report	0
17th Feb 2025	Monday	Had a meeting regarding the discussion on the report created, insights received - my report was more code oriented and the structure for future reports will be shared by the mentors. Also assigned to the developer manual team.	4
18th Feb 2025	Tuesday	Task assigned to generate the UI of the Lap Joint bolted. Created the Placeholder page of Simple Connections and its launcher funciton for 4 sub modules. Created a modified trace_wrapper file to show unique function calls and their location to help everyone track the flow of the code more easily.	4
19th Feb 2025	Wednesday	Had a session on GITHUB and learned about the git ethics for OSDAG commits. Also got assigned the task to study the code structure of imports of tension member bolted connection.	4
20th Feb 2025	Thursday	Started on the LapJointBolted module, first understood different modules its related to, this helped me recreate the code structure which is followed in the codebase.	4

21th Feb 2025	Friday	Started on the sub module, created the Input and the Output Dock of the sub module. encounterd some bugs in the preference details and the output dock. Had a discussion with the mentor and discussed about the Output dock structure.	5
22th Feb 2025	Saturday	Mitigated the bugs and continued on the completetion of the module, just some finishing touched remained.	6
23th Feb 2025	Sunday	Finished the last remaining details.	1
24th Feb 2025	Monday	started the buttjoint bolted module, worked on the similarity parts as lap joint bolted	4
25th Feb 2025	Tuesday	Continued with the Butt Joint Module.	4
26th Feb 2025	Wednesday	Finished some finishing touches on the working Module.	4
27th Feb 2025	Thursday	Made some minor changes in the home page and fixed some issues in the Lap Joint Bolted module	4
28th Feb 2025	Friday	Changes made to Butt Joint Bolted, UI was finalized and branch is commited.	4
1st march - 9th march		MID SEM EXAM BREAK	
10th March 2025	Monday	Started on the calculations part, got the Tension bolted DDCL and learned some calculations of the Lap Joint Bolted	5
11th March 2025 12th 13th 14th	Tuesday Wednesday thurs Fri	Went through the DDCL for Tension bolted and also through all the calculation functions for the module in depth to understand the data and function flow	4
15th Mar	Sat	Completed the understanding for the Tension Bolted module calculation and created a rough functional flow for myself to understand it and track the functions better.	4
16th Mar	Sun	WEEKLY LEAVE	0
17th - 22th Mar	Mon - Sat	Worked on the Lap Joint Bolted Calculation module, created all the functions checked for correct outputs.	4
23th Mar	Sun	WEEKLY LEAVE	0
24 - 27 Mar	Mon - Thurs	Completed the calculations made changes to the prev functions structure getting output in the output docks. Meeting on plate girder module calc functions, need to look into them as soon as this module is finished.	4
28th Mar	Friday	Integrated meeting for CAD design in the Module.	4
29th Mar	Sat	Final module code after making some functional modifications.	3
30th mar	Sun	WEEKLY LEAVE	0
31mar - 5Apr	Mon - Sat	Modified the final edge, end, pitch and gauge calculations in the simple connection lap joint bolted and also fixed cad integration related bugs.	4
6Apr	Sun	WEEKLY LEAVE	0

7Apr - 17Apr	Mon - Thurs	Started with the Plate Girder module, created the ui components in the ui and all the docks with labels. Started with the check functions and tried to find solutions based on ui in the Design Preferences.	4
18Apr, 19Apr	Fri - Sat	Worked on adding a customized Stiffener thickenss in the Plate Girder Design Preference, made PyQt edits to incorporate a customized box like the one existing for input dock in the UI.	5-6
20Apr	Sun	WEEKLY LEAVE	0
21Apr - 23Apr	Mon - wed	Final works in solving calculation and edge cases bugs in the Simple Connection Lap Joint Bolted module. The final module s pull request is raised.	4
24Apr - 1May	Thrus - Thurs	Worked on Plate girder checks and all the calculation functions required for checks, based on various cases selected across variables.	4
1May - 15May		END SEM EXAMS BREAK	
16 May - 22May	Fri-Thurs	Worked on the PSO part for the Optimized method where all the parameter values are automatically calculated. Also woked on design check constraints that help PSO determine the best particle.	4
23May - 31May	Fri - Sat	Worked on migrating the entire section classification and various parametric checks developed in the module into modularity for better usage of functions. Also fixed some bugs regarding PSO which were throwing errors in Thin plate cases. Overall optimized the modularity of the PSO code too. Also worked on adding the CAD design. The integration worked but was shown in a seperate window due to error from the cad code. The cad team person was tasked to fix it.	4
2June - 5June	Mon - Thurs	Worked on adding various UI changes to change the way the customized and optimized parameters are inputted, failed attemps in creating an Input button, so we switched to the dropdown select method with all textboxes for the customized option.	5
6June - 7June	Fri - Sat	Worked on improving the PSO to generate more accurate results. Started on changing the PSO weight calculation and constraints. This attempt failed and led to less accurate results.	4
9June - 13June	Mon - Fri	Reverted back to the old PSO code and changed the number of times the whole PSO ran. Increasing the PSO runs led to efficient and accurate results both for Thin and Thick web.	4.5
14June - 15June	Sat - Sun	Minor bug fixes and commiting the final version to github.	3.5

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.