PROGRAMSKO RJEŠENJE PROBLEMA 2D GREDE U KOMBINACIJI S AKSIJALNIM OPTEREĆENJEM

Bakotin, Antonio

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Strojarstvo

ANTONIO BAKOTIN

Z A V R Š N I R A D

Programsko rješenje problema 2D grede u kombinaciji s aksijalnim opterećenjem

Split, rujan 2024.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Strojarstvo

Predmet: Python u strojarstvu

Z A V R Š N I R A D

Kandidat: Antonio Bakotin

Naslov rada: Programsko rješenje problema 2D grede u kombinaciji s aksijalnim opterećenjem

Mentor: Igor Gabrić, v. pred.

Split, rujan 2024.

REPUBLIKA HRVATSKA
SVEUČILIŠTE U SPLITU
Sveučilišni odjel za stručne studije

Studij: Strojarstvo
Predmet: Python u strojarstvu
Nastavnik: Igor Gabrić, v. pred.

ZADATAK

Kandidat: Antonio Bakotin

Zadatak: Programsko rješenje problema 2D grede u kombinaciji s aksijalnim opterećenjem

U radu je potrebno:

- uvodno dati kratki teorijski osvrt na rješavanje problema 2D grede primjenom metode konačnih elemenata
- detaljno opisati postupak rješavanja općenitih problema 2D greda u kombinaciji s aksijalnim opterećenjem primjenom metode konačnih elemenata
- primijeniti proceduralni način programiranja korištenjem Python programskog jezika
- osigurati i organizirati unos ulaznih podataka primjenom vanjske tablične datoteke (xlsx)
- program mora imati mogućnost određivanja reakcija u osloncima, pomaka i zakreta u čvorovima, veličine i lokacije maksimalnog momenta, crtanja dijagrama poprečnih sila, momenata te deformirane elastične linije grede
- dobivene rezultate (brojčane i slike) program treba organizirati i ispisati na terminalu te pohraniti u vanjsku datoteku (xlsx).
- programski kod treba biti podijeljen na funkcionalne cjeline (paket, moduli, funkcije) uz detaljna tekstualna pojašnjenja svih funkcionalnosti, varijabli ...
- testirati izrađeni program na barem jednom primjeru koji će biti riješen primjenom izrađenog programskog koda i na alternativan način - analitički, pokusom ili primjenom renomiranih računalnih programa i sl. (način i rezultate testiranja prikazati u radu)
- iznijeti zaključke
- navesti literaturu i izvore podataka.

Sažetak

Ovaj rad se bavi razvojem i primjenom Python programa za rješavanje problema 2D grede pod kombiniranim opterećenjem koristeći metodu konačnih elemenata (MKE). MKE omogućuje detaljnu analizu unutarnjih sila, naprezanja i deformacija unutar strukturnih elemenata, što je ključno za osiguranje sigurnosti i stabilnosti struktura. Razvijeni program omogućuje unos podataka iz Excel datoteke, a zatim putem zadatih jednadžbi generira sveobuhvatne rezultate, uključujući reakcije u osloncima, pomake i zakrete u čvorovima, dijagrame sila i momenata te deformiranu liniju grede. Testiranje na primjeru IPE 300 grede pokazalo je da su rezultati programa u skladu s analitičkim metodama i drugim softverskim alatima, čime je potvrđena njegova točnost i pouzdanost.

Ključne riječi: metoda konačnih elemenata, Python, 2D greda, kombinirano opterećenje, inženjerska analiza

Summary (Software solution to the problem of a 2D beam in combination with an axial load)

This paper focuses on the development and application of a Python program for solving the problem of a 2D beam under axial load using the finite element method (FEM). FEM enables detailed analysis of internal forces, stresses, and deformations within structural elements, which is crucial for ensuring the safety and stability of constructions. The developed program allows data input from an Excel file and then, through the use of specified equations, generates comprehensive results, including support reactions, displacements and rotations at nodes, force and moment diagrams, and the deformed shape of the beam. Testing on the example of an IPE 300 beam showed that the program's results are consistent with analytical methods and other software tools, confirming its accuracy and reliability.

Keywords: finite element method, Python, 2D beam, combined load, engineering analysis

SADRŽAJ

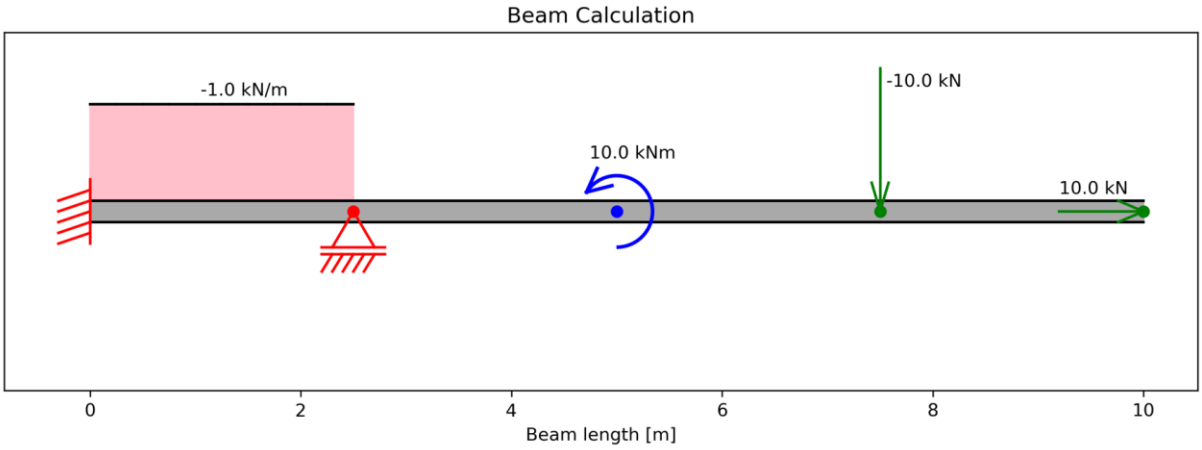
Popis slika

1. Uvod

Razvoj numeričkih metoda, poput metode konačnih elemenata (MKE), doveo je do revolucije u inženjerskoj analizi, omogućujući inženjerima da analiziraju i rješavaju kompleksne probleme s većom preciznošću nego ikada prije. MKE je postala ključni alat u analizi konstrukcija, posebno kada se radi o strukturnim elementima poput greda koje su izložene različitim vrstama opterećenja. Ova metoda omogućava detaljno ispitivanje unutarnjih sila, naprezanja i deformacija unutar konstrukcijskih elemenata, što je ključno za osiguranje sigurnosti i stabilnosti građevinskih objekata.

Greda, kao jedan od osnovnih elemenata u konstrukcijama, često se koristi za prenošenje opterećenja na potpore. Analiza ponašanja grede pod opterećenjem može biti vrlo složena, pogotovo kada se uzimaju u obzir različiti uvjeti opterećenja, materijala i geometrije. Tradicionalne metode analize, iako korisne, često nisu dovoljne za preciznu procjenu naprezanja i deformacija u složenijim scenarijima. Upravo tu MKE dolazi do izražaja, omogućujući podjelu grede u manji broj konačnih elemenata, za koje se mogu precizno izračunati odgovarajuće fizikalne veličine.

U ovom radu prikazan je problem na sljedećoj slici (*Slika 1.1*), koji je riješen korištenjem Python [1] programa, kao i drugog softvera koji potvrđuje podatke dobivene putem Pythona.



Slika 1.1 – Zadani problem

Python, sa svojim bogatim setom biblioteka i jednostavnom sintaksom, nudi snažan alat za numeričku analizu. U kombinaciji s drugim softverskim alatima, Python omogućava provjeru i validaciju rezultata, čime se osigurava visoka točnost i pouzdanost analize. Kroz detaljnu

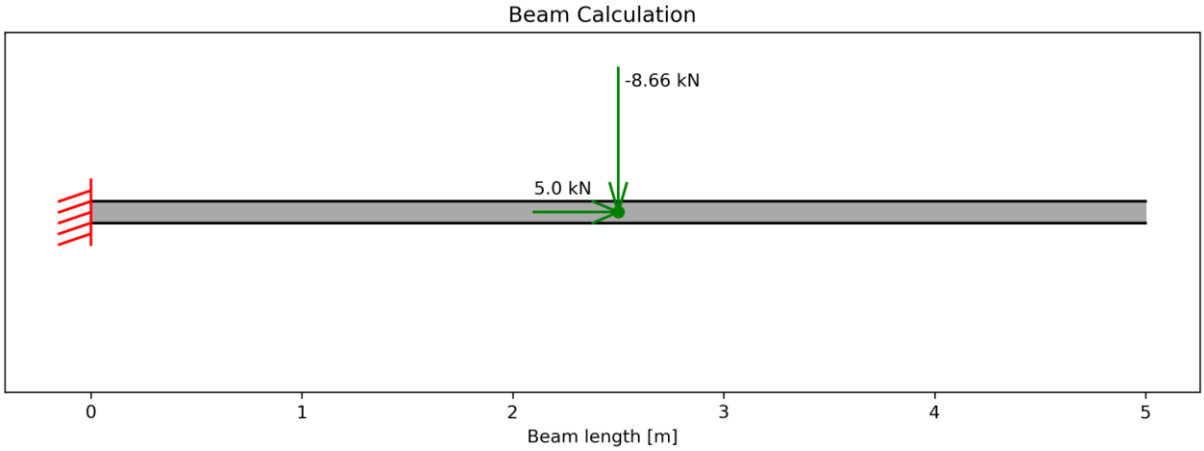
implementaciju metode konačnih elemenata za rješavanje 2D problema grede, cilj je prikazati sve ključne korake u procesu analize: od formulacije matematičkog modela, preko podijele grede, do rješavanja sustava linearnih jednadžbi pomoću kojih se dobivaju pomaci, naprezanja i unutarnje sile u svakom konačnom elementu.

U nastavku rada, detaljno će se obraditi sve faze procesa, uz objašnjenje teoretske podloge, implementacijskih detalja i interpretacije rezultata. Ovaj integrirani pristup osigurava visoku razinu preciznosti i pouzdanosti u analizi, čineći ga vrijednim alatom u svakodnevnoj inženjerskoj praksi.

2. Rješavanje 2D problema grede primjenom metode konačnih elemenata

Postupak rješavanja [2]:

Metoda konačnih elemenata (MKE) koristi se za analizu struktura poput greda, okvira i drugih inženjerskih konstrukcija. Ovaj postupak započinje definiranjem geometrije problema, postavljanjem rubnih uvjeta i raspodjelom opterećenja. U ovom primjeru koristi se konzolno ukliještena greda (*Slika 2.1*).



Slika 2.1 – Primjer konzolne grede

2.1.1. Podjela grede na elemente

Prvi korak u MKE analizi je podjela grede na željeni broj konačnih elemenata. Na primjer, ako se greda podijeli na dva elementa, svaki element modela će se analizirati zasebno. Cilj je postići dovoljno precizne rezultate uz što manji broj elemenata.

2.1.2. Postavljanje rubnih uvjeta

Rubni uvjeti definiraju kako je greda učvršćena i kako se može pomicati. U slučaju konzolno ukliještene grede, na mjestu uklještenja svi pomaci i zakreti bit će jednaki nuli. To znači da se svi stupci i redci koji odgovaraju tim stupnjevima slobode (DOF) u matričnoj formulaciji eliminiraju ili postave na nulu.

2.1.3. Matrica krutosti elementa

Nakon definiranja rubnih uvjeta, izrađuje se matrica krutosti za svaki pojedini element. Pretpostavlja se da oba elementa grede imaju iste materijalne i geometrijske karakteristike, pa će njihove matrice krutosti biti identične. Matrica krutosti opisuje otpor elementa prema deformaciji (*Slika 2.2*).

$$k_{element} = \begin{bmatrix} \frac{12 \cdot E \cdot I}{L^3} & 0 & \frac{6 \cdot E \cdot I}{L^2} & -\frac{12 \cdot E \cdot I}{L^3} & 0 & \frac{6 \cdot E \cdot I}{L^2} \\ 0 & \frac{A \cdot E}{L} & 0 & 0 & -\frac{A \cdot E}{L} & 0 \\ \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{4 \cdot E \cdot I}{L} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} \\ -\frac{12 \cdot E \cdot I}{L^3} & 0 & -\frac{6 \cdot E \cdot I}{L^2} & \frac{12 \cdot E \cdot I}{L^3} & 0 & -\frac{6 \cdot E \cdot I}{L^2} \\ 0 & -\frac{A \cdot E}{L} & 0 & 0 & \frac{A \cdot E}{L} & 0 \\ \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{4 \cdot E \cdot I}{L} \end{bmatrix}$$

Slika 2.2 – Matrica krutosti elementa

2.1.4. Matrica krutosti sustava

Globalna matrica krutosti dobiva se kombiniranjem pojedinačnih matrica krutosti svakog elementa. Ova matrica predstavlja cjelokupni sustav grede (*Slika 2.3*) u odnosu na sve stupnjeve slobode. Prilikom sastavljanja globalne matrice, potrebno je voditi računa o kompatibilnosti deformacija između elemenata.

$$k_{global} = \begin{bmatrix} \frac{12 \cdot E \cdot I}{L^3} & 0 & \frac{6 \cdot E \cdot I}{L^2} & -\frac{12 \cdot E \cdot I}{L^3} & 0 & \frac{6 \cdot E \cdot I}{L^2} & 0 & 0 & 0 \\ 0 & \frac{A \cdot E}{L} & 0 & 0 & -\frac{A \cdot E}{L} & 0 & 0 & 0 & 0 \\ \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{4 \cdot E \cdot I}{L} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} & 0 & 0 & 0 \\ -\frac{12 \cdot E \cdot I}{L^3} & 0 & -\frac{6 \cdot E \cdot I}{L^2} & \frac{24 \cdot E \cdot I}{L^3} & 0 & 0 & -\frac{12 \cdot E \cdot I}{L^3} & 0 & \frac{6 \cdot E \cdot I}{L^2} \\ 0 & -\frac{A \cdot E}{L} & 0 & 0 & \frac{2 \cdot A \cdot E}{L} & 0 & 0 & -\frac{A \cdot E}{L} & 0 \\ \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} & 0 & 0 & \frac{4 \cdot E \cdot I}{L} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} \\ 0 & 0 & 0 & -\frac{12 \cdot E \cdot I}{L^3} & 0 & -\frac{6 \cdot E \cdot I}{L^2} & \frac{12 \cdot E \cdot I}{L^3} & 0 & -\frac{6 \cdot E \cdot I}{L^2} \\ 0 & 0 & 0 & 0 & -\frac{A \cdot E}{L} & 0 & 0 & \frac{A \cdot E}{L} & 0 \\ 0 & 0 & 0 & \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{2 \cdot E \cdot I}{L} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{4 \cdot E \cdot I}{L} \end{bmatrix}$$

Matrica krutosti elementa 1

Matrica krutosti elementa 2

Slika 2.3 – Matrica krutosti sustava

2.1.5. Izrada matrica ekvivalentnih opterećenja

Sljedeći korak je izrada matrice ekvivalentnih opterećenja, koja predstavlja sva opterećenja primijenjena na gredi, uključujući kontinuirana opterećenja, koncentrirane sile i momente. Ova matrica kombinira sve vanjske sile i predstavlja ih u obliku koji je kompatibilan s matričnom formulacijom MKE.

Ako se uzme u obzir sve stupnjeve slobode (pomak u x, y smjeru i rotaciju), vektor ekvivalentnih opterećenja uslijed kontinuiranog opterećenja ima oblik (*Slika 2.4*).

$$f_{equ\_q} = \begin{Bmatrix} F_{equ\_v1} \\ F_{equ\_h1} \\ M_{equ\_1} \\ F_{equ\_v2} \\ F_{equ\_h2} \\ M_{equ\_2} \\ \vdots \end{Bmatrix} = \begin{Bmatrix} \frac{q \cdot L}{2} \\ 0 \\ \frac{q \cdot L^2}{12} \\ \frac{q \cdot L}{2} \\ 0 \\ -\frac{q \cdot L^2}{12} \\ \vdots \end{Bmatrix}$$

Slika 2.4 – Matrica ekvivalentnih opterećenja uslijed kontinuiranog opterećenja

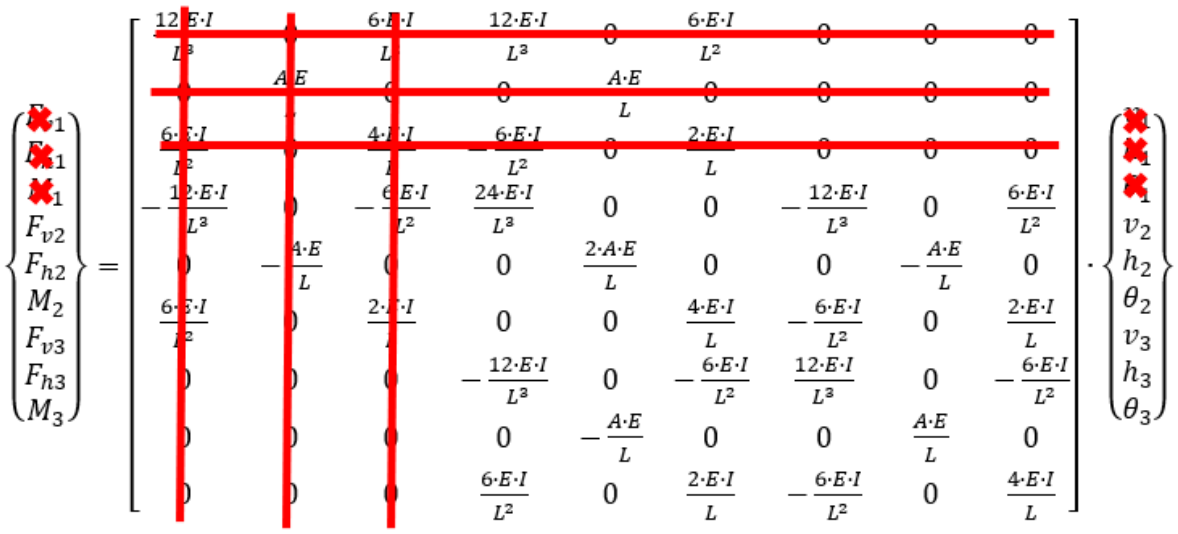
Da bi se dobila ukupna sila u čvorovima, potrebno je zbrojiti opterećenja uzrokovana koncentriranim i kontinuiranim opterećenjima. To se može vidjeti na sljedećoj slici (*Slika 2.5*).

$$F_{ekv} = f_{set} + f_{equ\_q} = \begin{Bmatrix} F_{set\_v1} \\ F_{set\_h1} \\ M_{set\_1} \\ F_{set\_v2} \\ F_{set\_h2} \\ M_{set\_2} \\ \vdots \end{Bmatrix} + \begin{Bmatrix} \frac{q \cdot L}{2} \\ 0 \\ \frac{q \cdot L^2}{12} \\ \frac{q \cdot L}{2} \\ 0 \\ -\frac{q \cdot L^2}{12} \\ \vdots \end{Bmatrix}$$

Slika 2.5 – Matrica ekvivalentnih opterećenja

2.1.6. Redukcija globalne matrice krutosti i matrice opterećenja

Globalna matrica krutosti i matrica ekvivalentnih opterećenja zatim se reduciraju. Ova redukcija uključuje brisanje redaka i stupaca iz globalne matrice krutosti (*Slika 2.6*) koji odgovaraju stupnjevima slobode gdje su pomaci jednaki nuli (npr. na mjestima gdje je greda ukliještena).



Slika 2.6 – Reducirana globalna matrica krutosti

2.1.7. Izračun pomaka i zakreta

Reducirana globalna matrica krutosti zatim se invertira i množi s reduciranim vektorom opterećenja (jednadžba (2.1)). Rezultat ovog izračuna su pomaci i zakreti u čvorovima grede. Ovi rezultati pružaju informacije o tome kako će se greda deformirati pod utjecajem zadanih opterećenja.

$$\{F\} = [k] \cdot \{v\} \tag{2.1}$$

Množenjem s inverznom matricom [k]-1 s lijeve strane oba člana jednakosti dobije se izraz pomoću kojega se može odrediti pomak slobodnih čvorova, izraz prikazan (jednadžbe (2.2), (2.3), (2.4)):

$$[k]^{-1} \cdot \{F\} = [k]^{-1} \cdot [k] \cdot \{v\} \tag{2.2}$$

$$[k]^{-1} \cdot \{F\} = [I] \cdot \{v\} \tag{2.3}$$

$$\{v\} = [k]^{-1} \cdot \{F\} \tag{2.4}$$

2.1.8. Izračun efektivnih sila u čvorovima

Iz dobivenih pomaka i globalne matrice krutosti, množenjem tih vrijednosti, dobivaju se efektivne sile u čvorovima grede. Ove sile predstavljaju unutarnje reakcije u gredi uslijed vanjskih opterećenja.

2.1.9. Izračun stvarnih sila u čvorovima

Kako bi se dobile stvarne sile u čvorovima grede, potrebno je efektivne sile korigirati oduzimanjem ekvivalentnih sila. Na taj način dobiva se točna distribucija sila unutar grede, koja je ključna za daljnju analizu stabilnosti i sigurnosti konstrukcije (jednadžba (2.5)).

$$\{F\} = \{F_{ef}\} - \{F_{ekv}\} \tag{2.5}$$

3. Python program za rješavanje 2D problema grede uporabom metode konačnih elemenata

Korištenje Python programa za rješavanje 2D problema grede uporabom metode konačnih elemenata (FEM) donosi brojne prednosti u usporedbi s tradicionalnim pristupima ili drugim softverskim alatima. Evo nekoliko ključnih razloga zašto je Python bolji izbor:

1. Fleksibilnost i prilagodljivost

   Python je izuzetno fleksibilan jezik, što omogućuje lako prilagođavanje programa specifičnim potrebama projekta. Korisnici mogu jednostavno proširiti ili modificirati Python skripte za različite scenarije analize, dodavanje novih elemenata, materijala ili složenih opterećenja, bez potrebe za prekomjernim prilagodbama koda.
2. Širok spektar dostupnih biblioteka

   Python ima bogat ekosustav biblioteka kao što su NumPy, SciPy, i Matplotlib, koje omogućuju jednostavno izvođenje naprednih matematičkih operacija, efikasnu manipulaciju podacima i vizualizaciju rezultata. Korištenjem ovih biblioteka, Python programi mogu implementirati složene FEM algoritme i odmah prikazati rezultate na intuitivan način.
3. Automatizacija i integracija

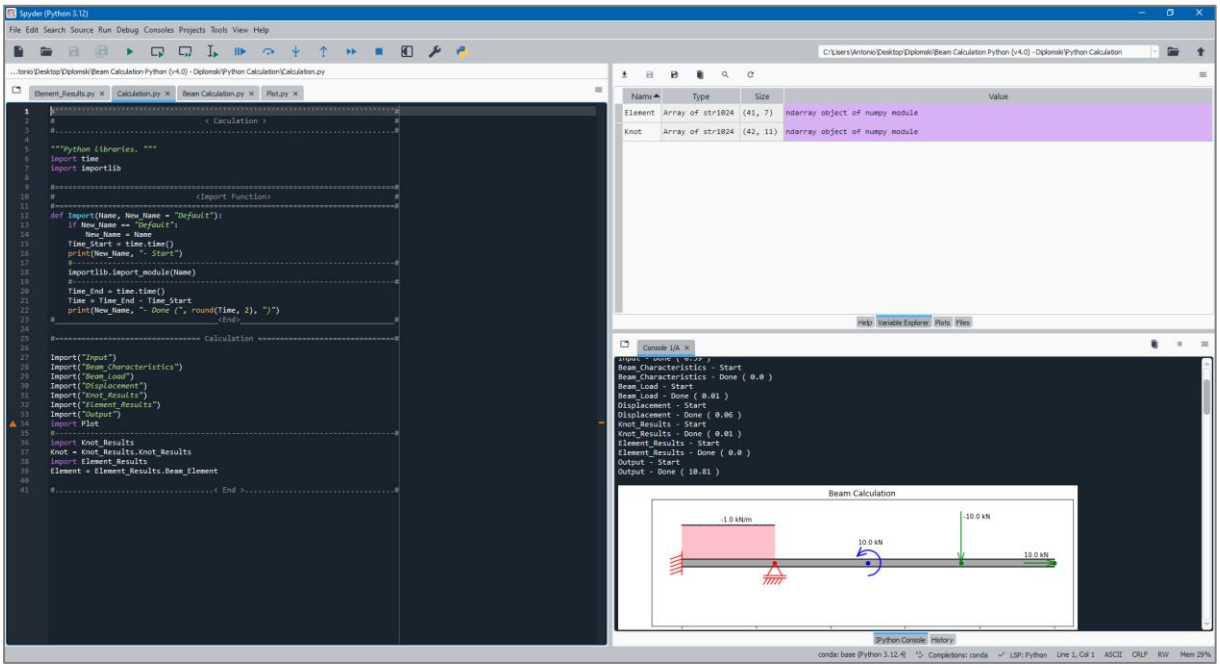
   Python omogućuje visoki stupanj automatizacije procesa, od unosa podataka do generiranja konačnih izvještaja i grafova. Također, lako se integrira s drugim alatima i platformama, omogućujući stvaranje potpunih rješenja koja obuhvaćaju sve faze analize i projektiranja, od simulacije do konačne prezentacije rezultata.
4. Otvorenost i pristupačnost

   Kao open-source jezik, Python je besplatan i široko dostupan. To omogućava svima, od studenata do profesionalaca, da koriste Python bez potrebe za skupim licencama. Uz to, veliki broj online resursa, uputa i foruma pruža podršku korisnicima svih razina iskustva.
5. Ponovna upotreba i proširivost koda

   Python omogućuje jednostavno pisanje modula i funkcija koje se mogu ponovno koristiti u različitim projektima. Ovo značajno smanjuje vrijeme potrebno za razvoj novih rješenja i povećava efikasnost. Također, programi napisani u Pythonu lako se mogu proširiti kako bi podržali nove funkcionalnosti, što ih čini dugoročno održivim.

6. Transparentnost i kontrola nad procesom

   Korištenjem Python programa, inženjeri imaju potpunu kontrolu nad svakim aspektom analize, od definicije modela do izvođenja proračuna i provjere rezultata. To povećava povjerenje u točnost i pouzdanost rezultata, jer su svi koraci jasno vidljivi i mogu se detaljno pregledati (*Slika 3.1*).



Slika 3.1 – Spyder sučelje

7. Brza iteracija i testiranje

   Python omogućuje brzu iteraciju i testiranje različitih scenarija, što je ključno u inženjerskim projektima gdje se parametri često mijenjaju tijekom procesa dizajna. Inženjeri mogu brzo prilagoditi modele i odmah vidjeti kako te promjene utječu na rezultate, što doprinosi bržem donošenju odluka.

4. Prikaz rada programskog koda

Program za rješavanje 2D problema grede podijeljen je u nekoliko sekcija unutar direktorija „Python Calculation“ (*Slika 4.1*). Svaka sekcija predstavlja jedan modul koji obavlja određeni dio proračuna, poput modula za ulazne podatke, modula za kalkulaciju pomaka čvorova, modula za izračun rezultata u čvorovima, i slično.

Osim toga, direktorij sadrži i nekoliko paketa s različitim funkcijama:

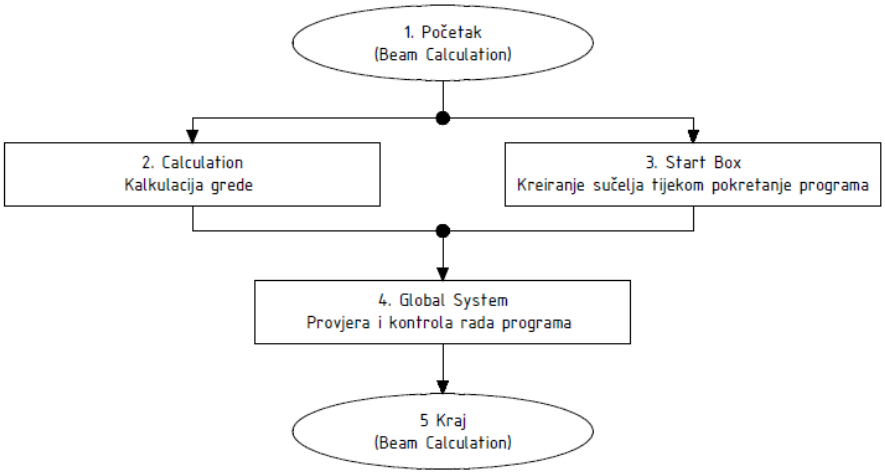
1. Paket „Excel Data“: služi za manipulaciju Excel datotekama i njihovim podacima.
2. Paket „Function“: omogućuje transformaciju i manipulaciju podacima, uključujući rad s matricama, različitim tipovima podataka, i slično.
3. Paket „Calculation“: sadrži module za kalkulacije poput matrica krutosti, momenta inercije, i drugih ključnih parametara.
4. Paket „Results“: obuhvaća module za dobivanje rezultata poput pomaka, sila u čvorovima, i drugih relevantnih informacija.

U sljedećim poglavljima detaljno su opisani svi korišteni moduli.

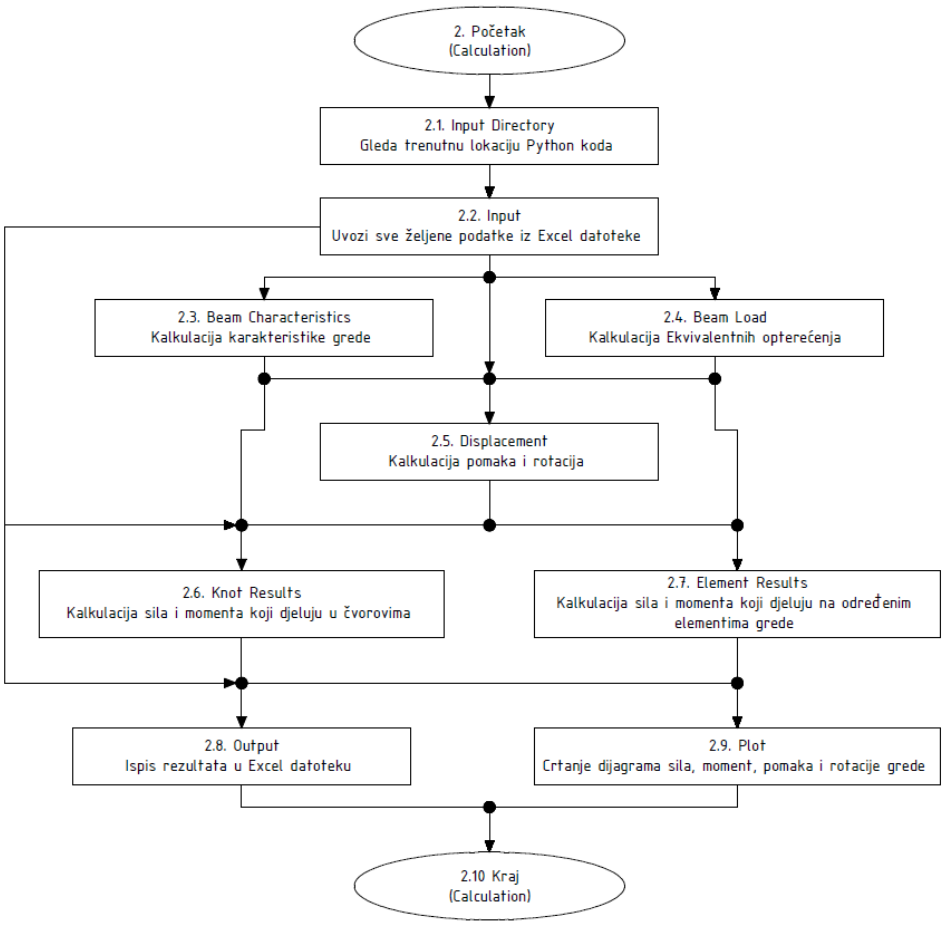
__pycache__	2.9.2024. 11:14	File folder	
_1_Excel_Data	27.8.2024. 11:08	File folder	
_2_Function	27.8.2024. 11:08	File folder	
_3_Calculation	27.8.2024. 11:08	File folder	
_4_Results	27.8.2024. 11:08	File folder	
Beam Calculation.py	27.8.2024. 11:08	Python File	1 KB
Beam_Characteristics.py	27.8.2024. 11:08	Python File	1 KB
Beam_Load.py	27.8.2024. 11:08	Python File	1 KB
Calculation.py	27.8.2024. 14:01	Python File	2 KB
Displacement.py	27.8.2024. 11:08	Python File	1 KB
Element_Results.py	27.8.2024. 11:08	Python File	4 KB
Global_System.py	27.8.2024. 11:08	Python File	4 KB
Input.py	1.9.2024. 10:47	Python File	3 KB
Input_Directory.py	27.8.2024. 11:08	Python File	1 KB
Knot_Results.py	1.9.2024. 11:39	Python File	3 KB
Output.py	27.8.2024. 11:08	Python File	2 KB
Plot.py	2.9.2024. 11:14	Python File	15 KB
Start_Box.py	27.8.2024. 11:08	Python File	4 KB

Slika 4.1 – Sadržaj „Python Calculation“ mape

Na sljedećim slikama prikazani su dijagrami rada glavnog koda (*Slika 4.2*, *Slika 4.3*). Važno je istaknuti da dio koda koji se odnosi na kalkulacije (*Slika 4.3*) koristi pakete navedene ranije. Ovi paketi skraćuju kod i olakšavaju njegovo praćenje.



Slika 4.2 – Dijagram slijeda „Beam Calculation“



Slika 4.3 – Dijagram slijeda „Calculation“

4.1. Excel data

„Excel Data“ paket je skupina modula koji su dizajnirani za rad s Excel datotekama i obradu podataka unutar njih. Paket sadrži četiri modula, od kojih svaki ima specifičnu ulogu u upravljanju Excel podacima, te jedan „__init__.py“ modul (*Slika 4.4*) koji služi za uvoz svih ovih modula u paket, olakšavajući njihovu upotrebu u glavnom kodu.

Prednosti ovakve strukture:

- Ovakva organizacija omogućuje modularnost i ponovnu upotrebu koda. Uvozom svih potrebnih funkcionalnosti putem „__init__.py“ modula, glavnom kodu se olakšava rad s Excel podacima, jer sve funkcionalnosti postaju dostupne kroz jednostavne pozive, bez potrebe za višestrukim uvozima i kompliciranim putanjama do pojedinih modula.
- Ovaj pristup ne samo da pojednostavljuje korištenje koda već također poboljšava njegovu čitljivost i održavanje. Ako je potrebno napraviti promjene u radu s Excel datotekama, te promjene mogu se centralizirano upravljati unutar odgovarajućeg modula unutar paketa.

```
#=============================================================================#
#                                    __init__                                 #
#                                 _1_Excel_Data                               #
#=============================================================================#
from .Directory import Input_Directory as Input_Directory
from .Directory import Output_Directory as Output_Directory
from .Excel_Raw_Data import Excel_Raw_Data as Raw_Data
from .Excel_Data import Excel_Data as Data
from .Output_Data import Output_Data as Output
#_____________________________________End_____________________________________#
```

Slika 4.4 – „__init__.py“ modul, Excel Data

4.1.1. „Directory“ modul

Modul „Directory“ (*Slika 4.5*) služi za određivanje putanje do direktorija na računalu kako bi Python program uvijek imao ispravnu adresu direktorija i datoteka unutar njega koje su ključne za pravilno izvršavanje koda, poput „Input“ i „Output“ datoteka. Modul sadrži dvije funkcije: „Input Directory“, koja preuzima lokaciju Excel datoteke s ulaznim podacima („Input“ datoteka), i „Output Directory“, koja preuzima lokaciju Excel datoteke za pohranu rezultata („Output“ datoteka). U ovom slučaju obje funkcije koriste isti slijed koda. Prilikom pozivanja funkcija potrebno je navesti jedan argument, naziv ulazne ili izlazne datoteke.

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Directory >                                  #
#.............................................................................#

"""Python libraries. """
import os

#=============================================================================#
#                             <Input Directory>                               #
#=============================================================================#
def Input_Directory(Input):
    Input_directory_syntax = os.getcwd()
    Input_directory=Input_directory_syntax[:-18] + Input
    #-------------------------------------------------------------------------#
    return Input_directory
#____________________________________<End>____________________________________#


#=============================================================================#
#                            <Output Directory>                              #
#=============================================================================#
def Output_Directory(Output):

    Output_directory_syntax = os.getcwd()
    Output_directory=Output_directory_syntax[:-18] + Output
    #-------------------------------------------------------------------------#
    return Output_directory
#____________________________________<End>____________________________________#

#...................................< End >.....-.............................#
```

Slika 4.5 – „Directory“ modul

4.1.2. „Excel Raw Data“ modul

Modul „Excel Raw Data“ (*Slika 4.8*) služi za kreiranje matrice sirovih podataka koji su zadani putem Excel datoteke.

Funkcija prihvaća jedan argument, „file name“, koji definira lokaciju Excel datoteke na računalu kako bi mogla izvući potrebne podatke.

Način rada funkcije je takav da uzima podatke s dva lista unutar Excel datoteke te kreira dvije matrice: matricu „Knot“, koja sadrži sve potrebne podatke za čvorove, i matricu „Element“, koja sadrži sve potrebne podatke za zadane elemente. Svi podaci također se pretvaraju u mjerne jedinice u kojima će se vršiti proračuni. Na sljedećim slikama može se vidjeti izgled podataka u Excel datoteci te u Spyder [3] editoru (*Slika 4.6*, *Slika 4.7*).

Element No.	E	Lenght	q	Number of elements	Inertia	Cross section
1	210 [GPa]	2,50 [m]	-1,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**
2	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**
3	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**
4	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**

Slika 4.6 – Ulazni podaci u Excel datoteci

E	L	q	Num	I	A
210000000000	2.5	-1000	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381

Slika 4.7 – Ulazni podaci u Spyder editoru

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Excel Raw Data >                             #
#.............................................................................#

"""Python libraries. """
import pandas as pd
import numpy as np

#=============================================================================#
#                              <Excel Raw Data>                               #
#=============================================================================#
def Excel_Raw_Data (filename):
    Knots = pd.read_excel (filename, sheet_name='Knots')
    Elements = pd.read_excel (filename, sheet_name='Elements')
    #-------------------------------------------------------------------------#
    v_b = Knots['Vertical boundary'].values[:]
    v_b = np.hstack((["v_b", v_b]))
    h_b = Knots['Horizontal boundary'].values[:]
    h_b = np.hstack((["h_b", h_b]))
    r_b = Knots['Rotation boundary'].values[:]
    r_b = np.hstack((["r_b", r_b]))
    F_v = Knots['Vertical force'].values[:]*1000    # [from kN to N]
    F_v = np.hstack((["F_v", F_v]))
    F_h = Knots['Horizontal force'].values[:]*1000  # [from kN to N]
    F_h = np.hstack((["F_h", F_h]))
    M = Knots['Moment'].values[:]*1000              # [fron kNm to Nm]
    M = np.hstack((["M", M]))
    #-------------------------------------------------------------------------#
    E = Elements['E'].values[:]*10**9               # [from GPa to Pa]
    E = np.hstack((["E", E]))
    L = Elements['Lenght'].values[:]                # [m]
    L = np.hstack((["L", L]))
    q = Elements['q'].values[:]*1000                # [from kN/m to N/m]
    q = np.hstack((["q", q]))
    Num = Elements['Number of elements'].values[:]
    Num = np.hstack((["Num", Num]))
    I = Elements['Inertia'].values[:]*10**-8        #[cm4 to m4]
    I = np.hstack((["I", I]))
    A = Elements['Cross section'].values[:]*10**-4    #[cm2 to m2]
    A = np.hstack((["A", A]))
    H = Elements['Height'].values[:]                # [mm]
    H = np.hstack((["H", H]))
    W = Elements['Width'].values[:]                 # [mm]
    W = np.hstack((["W", W]))
    t = Elements['Thickness'].values[:]             # [mm]
    t = np.hstack((["t", t]))
    #-------------------------------------------------------------------------#
    Raw_Input_Knots = np.vstack((v_b, h_b, r_b, F_v, F_h, M))
    Raw_Input_Elements = np.vstack((E, L, q, Num, I, A, H, W, t))
    Raw_Input_Knots = Raw_Input_Knots.transpose()
    Raw_Input_Elements = Raw_Input_Elements.transpose()
    #-------------------------------------------------------------------------#
    return Raw_Input_Knots, Raw_Input_Elements
#____________________________________<End>____________________________________#

#..................................< End >.....-..............................#
```

Slika 4.8 – „Excel Raw Data“ modul

4.1.3. „Excel Data“ modul

Modul „Excel Data“ (*Slika 4.11*, *Slika 4.12*) služi za generiranje pod-elemenata i u skladu s tim, novih čvorova kako bi program zadovoljio ulazne parametre.

Funkcija ima dva argumenta: jedan obavezni i jedan opcionalni. Argument „Matrix“ je obavezni argument funkcije i u njemu se nalaze svi podaci o čvorovima i elementima. Argument „Matrix_Type“ je opcionalni argument koji određuje izlaznu matricu ove funkcije. „Matrix_Type“ može imati dvije vrijednosti: „Element“ ili „Knot“. Ako argumentu „Matrix_Type“ dodijelimo vrijednost „Element“, funkcija će generirati novu matricu elemenata koja zadovoljava ulazne parametre. Ako dodijelimo vrijednost „Knot“, funkcija će generirati novu matricu čvorova koja također zadovoljava ulazne parametre. U slučaju da argument „Matrix_Type“ nije definiran, funkcija će po zadanim postavkama ispisati matricu čvorova.

Način rada funkcije može se vidjeti na sljedećim slikama (*Slika 4.9*, *Slika 4.10*), koje prikazuju početnu matricu elemenata te matricu nakon izvršenja funkcije. Funkcija, neovisno o opcijskom argumentu, analizira koliko svaki element ima pod-elemenata (stupac „Num“), te na temelju tog opcijskog argumenta generira pod-elemente za određeni element ili čvorove koji odgovaraju zadanim pod-elementima. Pod-elemente generira za svaki element na način da im dodjeljuje karakteristike elementa od kojeg su nastali, poput momenta inercije, poprečnog presjeka, modula elastičnosti i kontinuiranog opterećenja tog elementa. Jedina iznimka je dužina, koja se proračunava dijeljenjem ukupne dužine elementa grede na zadani broj pod-elemenata. Kod čvorova se generiraju novi čvorovi koji nemaju nikakve oslonce i nisu opterećeni vanjskim opterećenjima.

E	L	q	Num	I	A
210000000000	2.5	-1000	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381
210000000000	2.5	0	10	8.356e-05	0.005381

Slika 4.9 – Matrica ulaznih podataka prije izvršenja funkcije „Excel Data“

Element	E	L	q	I	A
1.1	210000000000	0.25	-1000	8.356e-05	0.005381
1.2	210000000000	0.25	-1000	8.356e-05	0.005381
1.3	210000000000	0.25	-1000	8.356e-05	0.005381
1.4	210000000000	0.25	-1000	8.356e-05	0.005381
1.5	210000000000	0.25	-1000	8.356e-05	0.005381
1.6	210000000000	0.25	-1000	8.356e-05	0.005381
1.7	210000000000	0.25	-1000	8.356e-05	0.005381
1.8	210000000000	0.25	-1000	8.356e-05	0.005381
1.9	210000000000	0.25	-1000	8.356e-05	0.005381
1.10	210000000000	0.25	-1000	8.356e-05	0.005381
2.1	210000000000	0.25	0	8.356e-05	0.005381
2.2	210000000000	0.25	0	8.356e-05	0.005381
2.3	210000000000	0.25	0	8.356e-05	0.005381

Slika 4.10 – Matrica ulaznih podataka nakon izvršenja funkcije „Excel Data“

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Excel Data >                                  #
#.............................................................................#

"""Python libraries. """
import numpy as np

#=============================================================================#
#                               <Excel Data>                                   #
#=============================================================================#
def Excel_Data(Matrix, Matrix_Type = "Knot"):
    if Matrix_Type == "Element":
        Type = 1
    else:
        Type = 0
    #-------------------------------------------------------------------------#
    for Num_variable in range(len(Matrix[1][0])):
        if Matrix[1][0][Num_variable] == "Num":
            Num = Matrix[1][1:, Num_variable]
    #-------------------------------------------------------------------------#
    Sum_Row = len(Matrix[Type])
    #-------------------------------------------------------------------------#
    New_Matrix = []
#-----------------------------------------------------------------------------#
    for Row in range(Sum_Row):
        #=====================================================================#
        #                   First_Row_in_Matrix_(Column_Name)                  #
        #=====================================================================#
        if Row == 0:
            New_Matrix = np.append(New_Matrix, Matrix[Type][Row])
        #=====================================================================#
        #              Last_Row_in_Matrix_(Only_in_Element_Matrix)             #
        #=====================================================================#
        elif (Type == 0 and Row == Sum_Row - 1):
            New_Matrix = np.vstack((New_Matrix, Matrix[Type][Row]))
        #=====================================================================#
        #                          Insert_Row_in_Matrix                        #
        #=====================================================================#
        else:
            for Added in range(int(Num[Row-1])):
                if  Type == 0:
                    #=========================================================#
                    #                   Row_for_Knot_Matrix                    #
                    #=========================================================#
                    if Added == 0:
                        New_Matrix = np.vstack((New_Matrix, Matrix[0][Row]))
                    else:
                        New_Row = []
                        #-------------------------v_b-------------------------#
                        New_Row = np.append(New_Row, 0)
                        #-------------------------h_b-------------------------#
                        New_Row = np.append(New_Row, 0)
                        #-------------------------r_b-------------------------#
                        New_Row = np.append(New_Row, 0)
                        #-------------------------F_v-------------------------#
                        New_Row = np.append(New_Row, 0)
                        #-------------------------F_h-------------------------#
                        New_Row = np.append(New_Row, 0)
                        #--------------------------M--------------------------#
                        New_Row = np.append(New_Row, 0)
                        New_Matrix = np.vstack((New_Matrix, New_Row))
```

Slika 4.11 – „Excel Data“ modul (1)

```
            #_____________________________End_____________________________#
        else:
            #=========================================================#
            #                    Row_for_Element_Matrix                #
            #=========================================================#
            New_Row = []
            #----------------------------E----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][0])
            #----------------------------L----------------------------#
            Old_Lenght_Data = float(Matrix[1][Row][1])
            Length_Devider = float(Num[Row-1])
            New_Length_Data = Old_Lenght_Data/Length_Devider
            New_Row = np.append(New_Row, New_Length_Data)
            #----------------------------q----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][2])
            #----------------------------Num----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][3])
            #----------------------------I----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][4])
            #----------------------------A----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][5])
            #----------------------------H----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][6])
            #----------------------------W----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][7])
            #----------------------------t----------------------------#
            New_Row = np.append(New_Row, Matrix[1][Row][8])
            #-------------------------------------------------------------#
            New_Matrix = np.vstack((New_Matrix, New_Row))
            #_____________________________End_____________________________#

    return New_Matrix
#_____________________________________<End>_____________________________________#

#.....................................< End >.....-.............................#
```

Slika 4.12 – „Excel Data“ modul (2)

4.1.4. „Outout Data“ modul

Modul „Output Dana“ (*Slika 4.16*) služi za ispis rezultata proračuna u Excel datoteku.

Funkcija ima sedam argumenata, od kojih su četiri obavezna, a tri opcijska.

Obavezni argumenti su:

1. „Output“ - određuje lokaciju Excel datoteke u koju će se podaci ispisati.
2. „Sheet“ - specificira na kojem listu će se podaci ispisivati.
3. „Dana“ - predstavlja matricu podataka koja se treba ispisati u Excel datoteku.
4. „Unit“ - definira ili mijenja mjernu jedinicu iz one korištene u kalkulacijama u jedinicu definiranu u Excelu za ispis rezultata.

Opcijski argumenti su:

1. „Row_Start“ - određuje prvi redak od kojeg će se početi ispisivati rezultati.
2. „Column_Start“ - definira stupac od kojeg će se početi ispisivati rezultati.
3. „Process“ - određuje format ispisivanja. Po zadanoj postavci, prvi stupac se ispisuje bez ikakvih uređivanja, što je korisno kada taj stupac opisuje sadržaj pojedinog retka. Ako se zada neka specifična vrijednost, svi podaci će biti formatirani prema toj vrijednosti.

Način rada funkcije uključuje pozivanje funkcije s odgovarajućim argumentima, nakon čega Python program provjerava zadanu lokaciju za ispis podataka, briše postojeće podatke (ako ih ima), te upisuje nove podatke. Primjer rada funkcije može se vidjeti na sljedećim slikama (*Slika 4.13*, *Slika 4.14*, *Slika 4.15*).

```
Excel.Output(Input.Output_Excel, "Knot Results", Knot.Knot_Results[:,5:],
                0.001, Column_Start = 6, Process = "No")
```

Slika 4.13 – Način pozivanje „Output Data" funkcije

Knot	L_x	v	h	r	F_v	F_h	M	SR-F_v	SR-F_h	SR-M
1.1	0.0	0.0	0.0	0.0	-22437.49999…	-9999.999999…	-19218.74999…	-22437.49999…	-9999.999999…	-19218.74999…
1.2	0.25	3.0886969585…	2.2123698020…	0.0002337021…	1.1641532182…	0.0	-1.455191522…	N/A	N/A	N/A
1.3	0.5	0.0001101171…	4.4247396040…	0.0003865972…	0.0	0.0	6.6938810050…	N/A	N/A	N/A
1.4	0.75	0.0002173774…	6.6371094061…	0.0004577949…	9.3132257461…	0.0	-1.455191522…	N/A	N/A	N/A
1.5	1.0	0.0003321323…	8.8494792081…	0.0004464048…	0.0	0.0	-4.656612873…	N/A	N/A	N/A
1.6	1.25	0.0004336233…	1.1061849010…	0.0003515363…	0.0	0.0	9.0221874415…	N/A	N/A	N/A
1.7	1.5	0.0005008697…	1.3274218812…	0.0001722992…	0.0	0.0	3.0559021979…	N/A	N/A	N/A

Slika 4.14 – Matrica rezultata unutar Spyder editora

Knots	Length	Vertical displacement	Horizontal displacement	Rotation displacement	Vertical force	Horizontal force	Moment	Vertical support reaction	Horizontal support reaction	Moment support reaction
1.1	0,00 [m]	0,000 [mm]	0,000 [mm]	0,000 [deg]	-22,44 [kN]	-10,00 [kN]	-19,22 [kNm]	-22,44 [kN]	-10,00 [kN]	-19,22 [kNm]
1.2	0,25 [m]	0,031 [mm]	0,002 [mm]	0,013 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.3	0,50 [m]	0,110 [mm]	0,004 [mm]	0,022 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.4	0,75 [m]	0,217 [mm]	0,007 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.5	1,00 [m]	0,332 [mm]	0,009 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.6	1,25 [m]	0,434 [mm]	0,011 [mm]	0,020 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.7	1,50 [m]	0,501 [mm]	0,013 [mm]	0,010 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A

Slika 4.15 – Izgled rezultata unutar Excel datoteke

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                            < Output Data >                                  #
#.............................................................................#

"""Python libraries. """
import openpyxl as pyxl
import pandas as pd

#=============================================================================#
#                        <Output Data Function>                               #
#=============================================================================#
def Output_Data(Output, Sheet, Data,  Unit, Row_Start=2, Column_Start=1, Process = "All"):
    filename = Output
    workbook = pyxl.load_workbook(filename)
    sheet = workbook.get_sheet_by_name(Sheet)
    List = pd.read_excel (filename, sheet_name=Sheet)
    #-------------------------------------------------------------------------#
    #                        Delete existing data                             #
    #-------------------------------------------------------------------------#
    for col_del in range(len(Data[0])):
        for row_del in range(len(List.values[:])):
            cr = sheet.cell(row = row_del + Row_Start,
                            column = col_del + Column_Start)
            cr.value = ""
    #-------------------------------------------------------------------------#
    #                             Write data                                  #
    #-------------------------------------------------------------------------#
    Data = Data[1:]
    Sum_Row = len(Data)
    Sum_Col = len(Data[0])
    for Col in range(Sum_Col):
        for Row in range(Sum_Row):
            cw = sheet.cell(row=Row + Row_Start, column=Col + Column_Start)

            if (Col == 0 and Process == "All"):
                try:
                    cw.value = Data[Row, Col]
                except:
                    cw.value = Data[Row, Col]
            else:
                try:
                    cw.value = float(Data[Row, Col])*Unit
                except:
                    cw.value = Data[Row, Col]

    workbook.save(filename)
#____________________________________<End>____________________________________#

#..................................< End >.....-..............................#
```

Slika 4.16 – „Output Data“ modul

4.2. Function

„Function“ paket je zbirka modula osmišljenih za pružanje pomoćnih funkcionalnosti tijekom kodiranja, posebno u zadacima vezanim uz manipulaciju matricama i obradu podataka. Ovaj paket sadrži različite module koji olakšavaju specifične operacije, čineći kodiranje učinkovitijim i organiziranijim. Također, kao i u svakom paketu, unutar njega se nalazi datoteka „__init__.py“ (*Slika 4.17*), koja grupira sve module unutar paketa te ih čini lakše dostupnim.

```
#=============================================================================#
#                                    __init__                                 #
#                                  _2_Function                                #
#=============================================================================#
from .Input_Variable import Input_Variable as Input_Variable
from .Max_Value import Max_Value as Max_Value
from .Delete_Column import Delete_Column as Del_Col
from .Numbering import Numbering as Numbering
from .List import List as List
from .deList import deList as deList
from .Title import Title as Title
from .Create_Matrix import Create_Matrix as Create
#_____________________________________End_____________________________________#
```

Slika 4.17 – „__init__.py“ modul, Function

4.2.1. „Input Variable“ modul

Modul „Input Variable“ (*Slika 4.20*) služi za izvlačenje jedne varijable iz matrice podataka.

Funkcija zahtijeva minimalno dva argumenta: argument „x“, koji definira traženi stupac (varijablu), i najmanje jednu matricu u kojoj će se ta vrijednost pretraživati.

Funkcija radi tako da pretražuje prvi redak prve matrice i traži zadanu varijablu. Ako je pronađe, prekida sve operacije i taj stupac ispisuje kao izlaznu matricu (vektor), izostavljajući prvi redak koji služi za definiranje varijable, poput sila, momenata, oslonaca itd. Ako varijablu ne pronađe u prvoj matrici, proces se ponavlja za drugu i sve ostale matrice, ako postoje. Na sljedećim slikama može se vidjeti kako funkcija radi (*Slika 4.18, Slika 4.19*).

Element	E	L	q	I	A
1.1	210000000000	0.25	-1000	8.356e-05	0.005381
1.2	210000000000	0.25	-1000	8.356e-05	0.005381
1.3	210000000000	0.25	-1000	8.356e-05	0.005381
1.4	210000000000	0.25	-1000	8.356e-05	0.005381
1.5	210000000000	0.25	-1000	8.356e-05	0.005381
1.6	210000000000	0.25	-1000	8.356e-05	0.005381
1.7	210000000000	0.25	-1000	8.356e-05	0.005381

Slika 4.18 – Traženi podaci unutar matrice prije izvršenja „Input Variable“ funkcije

-1000
-1000
-1000
-1000
-1000
-1000
-1000

Slika 4.19 – Izgled matrice nakon izvršenja „Input Variable“ funkcije

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                           < Input Variable >                              #
#...........................................................................#

"""Python libraries. """
import numpy as np

#===========================================================================#
#                           <Input Variable>                                #
#===========================================================================#
def Input_Variable(x, *argv):
    Data = argv
    for Data_Num in Data:
        n = len(Data_Num[0])
        for Col in range(n):
            if Data_Num[0][Col] == x:
                List = Data_Num[1:,Col]
                break
    #-----------------------------------------------------------------------#
    Variable = []
    Sum_List = len(List)
    for Row in range(Sum_List):
        try:
            Matrix = float(List[Row])
        except:
            Matrix = List[Row]
        Variable = np.append(Variable, Matrix)
    Variable = np.reshape(Variable,(Sum_List,1))
    #-----------------------------------------------------------------------#
    return Variable
#_________________________________<End>_____________________________________#

#..................................< End >..................................#
```

Slika 4.20 – „Input Variable“ modul

4.2.2. „Max Value“ modul

Modul „Max Value“ (*Slika 4.23*) služi za određivanje maksimalne vrijednosti, bilo da se radi o negativnoj ili pozitivnoj vrijednosti.

Funkcija ima dva obavezna argumenta i jedan opcijski. Argument „Text“ definira naziv varijable koja će se ispisivati (npr. „Shear force“, „Bending moment“, itd.). Argument „x“ služi za unos liste ili matrice na kojoj će se izvršavati funkcija. Opcijski argument „Operation“ definira koju će operaciju funkcija izvršavati. Po zadanim postavkama (default), funkcija će tražiti maksimalnu vrijednost, bilo da je ona negativna ili pozitivna. Ostale opcije uključuju „min“ ili „max“, koje vraćaju maksimalne negativne, odnosno pozitivne vrijednosti.

Način rada funkcije je takav da, pozivanjem funkcije, naziv maksimalne vrijednosti bude definiran kroz „Text“, dok se obrada podataka (matrice) vrši prema zadanoj operaciji. Funkcija ima tri načina rada, ovisno o potrebama. Na primjer, ako je maksimalna vrijednost 0, funkcija će vratiti „N/A“, što znači da taj podatak nije značajan. To se može dogoditi i pri traženju minimalne vrijednosti. Kada se koristi zadana postavka, funkcija će uvijek vraćati brojčanu vrijednost. Sljedeće slike prikazuju način rada funkcije (*Slika 4.21*, *Slika 4.22*).

0	-22.4375	-22.4375
-22.6875	-22.6875	5.82077e-14
-22.9375	-22.9375	0
-23.1875	-23.1875	0
-23.4375	-23.4375	4.65661e-13
-23.6875	-23.6875	0
-23.9375	-23.9375	0
-24.1875	-24.1875	-9.31323e-13
-24.4375	-24.4375	-9.31323e-13
-24.6875	-24.6875	0
-24.9375	10	34.9375

Slika 4.21 – Matrica na kojoj se izvršava „Max Value“ funkcija

Shear Force	34.937499999…

Slika 4.22 – Izgled matrice nakon izvršavanja „Max Value“ funkcije

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Max Value >                                 #
#............................................................................#

"""Python libraries. """
import numpy as np

#============================================================================#
#                              <Max Value>                                   #
#============================================================================#
def Max_Value(Text, x, Operation = "default"):
    Process = 1
    x = np.reshape(x, len(x))
    if Operation == "min":
        x_max = min(np.round(x,8))
        x_mod = np.round(x,8)
        x_calc = x
        if x_max == 0:
            Process = 0
    elif Operation == "max":
        x_max = max(np.round(x,8))
        x_mod = np.round(x,8)
        x_calc = x
        if x_max == 0:
            Process = 0
    else:
        x_max = max(abs(x))
        x_mod = abs(x)
        x_calc = x
    #------------------------------------------------------------------------#
    if Process == 1:
        for idx, i in enumerate(x_mod):
            if i == x_max:
                Result = x_calc[idx]
                break
        #--------------------------------------------------------------------#
        Result = np.hstack(([Text], [Result]))
        Result = np.reshape(np.array(Result), (1,2))
    #------------------------------------------------------------------------#
    else:
        Result = np.reshape(np.array([Text, "N/A"]), (1,2))

    return Result
#____________________________________<End>___________________________________#

#.................................< End >....................................#
```

Slika 4.23 – „Max Value“ modul

4.2.3. „Delete Column“ modul

Modul „Delete Column“ (*Slika 4.27*) služi za jednostavno brisanje stupaca ovisno o vrijednosti prvog retka tog stupca.

Funkcija ima jedan obavezni argument, kojim se zadaje matrica koja će se obrađivati, i minimalno još jedan argument koji definira koji će stupac biti izbrisan.

Način rada funkcije je takav da, nakon upisivanja prvog argumenta (matrice), korisnik unosi željeni broj argumenata koji predstavljaju vrijednosti prvog retka stupaca koji se žele izbrisati. Na primjer, ako matrica ima stupac u kojem je prvi redak označen vrijednošću „H“ (visina), „H“ se upisuje kao drugi argument. Funkcija tada obrađuje početnu matricu i vraća matricu bez stupca s oznakom „H“. Funkciji nije važno gdje se stupac nalazi unutar matrice, a lako je pratiti koji su stupci izbačeni jer pozivanjem funkcije korisnik može vidjeti koji su stupci izbrisani. Na sljedećim slikama može se vidjeti primjer kako funkcija radi (*Slika 4.24*, *Slika 4.25*, *Slika 4.26*).

```
Element = Fun.Del_Col(Element, "Num", "I", "A", "H", "W", "t")
```

Slika 4.24 – Način pozivanja „Delete Column“ funkcije

E	L	q	Num	I	A	H	W	t
210000000000	0.25	-1000	10	8.356e-05	0.005381	400	180	8.6
210000000000	0.25	-1000	10	8.356e-05	0.005381	400	180	8.6
210000000000	0.25	-1000	10	8.356e-05	0.005381	400	180	8.6
210000000000	0.25	-1000	10	8.356e-05	0.005381	400	180	8.6

Slika 4.25 – Matrica koja se obrađuje „Delete Column“ funkcijom

E	L	q
210000000000	0.25	-1000
210000000000	0.25	-1000
210000000000	0.25	-1000
210000000000	0.25	-1000

Slika 4.26 – Izgled matrice nakon izvršavanja „Delete Column“ funkcije

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Delete Column >                             #
#...........................................................................#

"""Python libraries. """
import numpy as np

#===========================================================================#
#                               <Delete Column>                              #
#===========================================================================#
def Delete_Column(Matrix, *args):
    Delete_Col = []
    for Col in range(len(Matrix[0])):
        for Col_Name in args:
            if Col_Name == Matrix[0][Col]:
                Delete_Col = np.append(Delete_Col, Col)
    Delete_Col = Delete_Col[::-1]
    #-----------------------------------------------------------------------#
    for Del in Delete_Col:
        Matrix = np.delete(Matrix, int(Del), axis=1)
    return Matrix
#_____________________________________<End>_________________________________#

#..................................< End >..................................#
```

Slika 4.27 – „Delete Column“ modul

4.2.4. „Numbering“ modul

Modul „Numbering“ (*Slika 4.29*) služi za numeriranje elemenata, pod-elemenata i čvorova.

Funkcija ima jedan obavezni argument, „Num“, koji predstavlja matricu s podacima o količini pod-elemenata za određeni element, i jedan opcijski argument, „variable“, koji određuje hoće li funkcija numeraciju raditi za čvorove ili elemente. Ako nije zadana nijedna vrijednost (npr. „Knot“ ili „Element“), funkcija će automatski generirati numeraciju za tablicu elemenata.

Način rada funkcije je takav da za svaki element generira početnu vrijednost. Na primjer, prvom elementu dodjeljuje vrijednost „1“, a potom točkom odvaja i numerira pod-elemente redoslijedom kojim se pojavljuju. Tako će prvi pod-element prvog elementa dobiti vrijednost „1.1“, drugi „1.2“ itd. *Slika 4.28* prikazuje izgled tablice (matrice) nakon izvršene numeracije.

Element	E	L	q	I	A
1.1	210000000000	0.25	-1000	8.356e-05	0.005381
1.2	210000000000	0.25	-1000	8.356e-05	0.005381
1.3	210000000000	0.25	-1000	8.356e-05	0.005381
1.4	210000000000	0.25	-1000	8.356e-05	0.005381
1.5	210000000000	0.25	-1000	8.356e-05	0.005381
1.6	210000000000	0.25	-1000	8.356e-05	0.005381
1.7	210000000000	0.25	-1000	8.356e-05	0.005381
1.8	210000000000	0.25	-1000	8.356e-05	0.005381
1.9	210000000000	0.25	-1000	8.356e-05	0.005381
1.10	210000000000	0.25	-1000	8.356e-05	0.005381
2.1	210000000000	0.25	0	8.356e-05	0.005381
2.2	210000000000	0.25	0	8.356e-05	0.005381
2.3	210000000000	0.25	0	8.356e-05	0.005381

Slika 4.28 – Matrica podataka nakon „Numbering“ funkcije

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Numbering >                              #
#.........................................................................#

"""Python libraries. """
import numpy as np

#=========================================================================#
#                               <Numbering>                               #
#=========================================================================#
def Numbering(Num, variable = "Element"):
    n = len(Num)
    Numbering = []
    Numbering.append(variable)
    #---------------------------------------------------------------------#
    for i in range(n):
        if int(Num[i]) > 1:
            for j in range(int(Num[i])):
                X = 1+i
                Y = (j+1)
                Z = str(X) + "." + str(Y)
                Numbering.append(Z)
        else:
            Numbering.append(i+1)
    #---------------------------------------------------------------------#
    if variable == "Knot":
        if int(Num[0]) > 1:
            Numbering.append(str(n+1)+ "." + str(1))
        else:
            Numbering.append(n+1)
    #---------------------------------------------------------------------#
    Numbering = np.array(Numbering)
    Numbering = np.reshape(Numbering,(len(Numbering),1))
    #---------------------------------------------------------------------#
    return Numbering
#__________________________________<End>__________________________________#

#..................................< End >................................#
```

Slika 4.29 – „Numbering“ modul

4.2.5. „List“ modul

Modul „List“ (*Slika 4.32*) je jednostavan modul koji služi za ubrzavanje izrade liste od tri varijable.

Funkcija ima tri argumenta: „x1“, „x2“ i „x3“, koji definiraju tri matrice (vektore) koje se žele spojiti u jednu matricu (vektor).

Način rada funkcije je sljedeći: prvi redak prve matrice (argument „x1“) pridodaje se novoj matrici. Isti se proces ponavlja za druge dvije matrice (argumenti „x2“ i „x3“), i to za sve retke svih matrica. Nakon što su svi redci iz svih matrica pridodani novoj matrici, ona se vraća kao izlazna matrica funkcije. Na sljedećim slikama može se vidjeti izgled matrica prije i poslije izvršavanja funkcije (*Slika 4.30*, *Slika 4.31*).

v_b	h_b	r_b
1	1	1
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0

Slika 4.30 – Matrice prije pozivanja „List“ funkcije

1
1
1
0
0
0
0
0
0
0
0

Slika 4.31 – Matrica nakon pozivanja „List“ funkcije

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                  < List >                                  #
#............................................................................#

"""Python libraries. """
import numpy as np

#============================================================================#
#                                   <List>                                   #
#============================================================================#
def List (x1, x2, x3):
    Sum_Row = len(x1)
    Sum_X = []
    for Row in range(Sum_Row):
        Sum_X = np.append(Sum_X, x1[Row])
        Sum_X = np.append(Sum_X, x2[Row])
        Sum_X = np.append(Sum_X, x3[Row])
    return Sum_X
#___________________________________<End>____________________________________#

#..................................< End >...................................#
```

Slika 4.32 – „List“ modul

4.2.6. „deList“ modul

Modul „deList“ (*Slika 4.35*) služi za razdvajanje jedne matrice (vektora) na tri matrice (vektora).

Funkcija ima samo jedan argument, „List“, koji predstavlja matricu (vektor) koju treba razdvojiti.

Način rada funkcije je sličan kao i kod prethodne funkcije „List“, ali djeluje obrnuto, tako da od jedne matrice (vektora) stvara tri zasebne matrice (vektore). Na sljedećim slikama može se vidjeti način rada funkcije (*Slika 4.33*, *Slika 4.34*).

0
0
0
3.0887e-05
2.21237e-06
0.000233702
0.000110117
4.42474e-06
0.000386597

Slika 4.33 – Matrica prije pozivanja „deList“ funkcije

0	0	0
3.0887e-05	2.21237e-06	0.000233702
0.000110117	4.42474e-06	0.000386597

Slika 4.34 – Matrica nakon pozivanja „deList“ funkcije

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                  < deList >                                  #
#............................................................................#

"""Python libraries. """
import numpy as np

#============================================================================#
#                                   <deList>                                   #
#============================================================================#
def deList (List):
    Sum_Row = int(len(List)/3)
    First_Column = []
    Secound_Column = []
    Third_Column = []
    for Row in range(Sum_Row):
        First_Column =  np.append(First_Column, List[Row+2*Row])
        Secound_Column = np.append(Secound_Column, List[Row+2*Row+1])
        Third_Column = np.append(Third_Column, List[Row+2*Row+2])
    deList = np.vstack((First_Column, Secound_Column, Third_Column) )
    deList = np.transpose(deList)
    #------------------------------------------------------------------------#
    return deList
#____________________________________<End>___________________________________#

#..................................< End >..................................#
```

Slika 4.35 – „deList“ modul

4.2.7. „Title“ modul

Modul „Title“ (*Slika 4.38*) služi za generiranje matrice s tekstom koji će biti korišten kao prvi redak u matricama čvorova i elemenata.

Funkcija prihvaća neodređeni broj argumenata, tj. funkcija će generirati matricu s brojem stupaca koji odgovara količini zadanih argumenata.

Način rada funkcije je takav da, prilikom unosa argumenata, generira stupac sa vrijednošću tog argumenta. Na primjer, unosom argumenata „A“ i „B“, funkcija će generirati dva stupca, gdje će prvi redak prvog stupca imati vrijednost „A“, a drugi stupac vrijednost „B“. Na sljedećim slikama može se vidjeti način rada funkcije (*Slika 4.36*, *Slika 4.37*).

```
Fun.Title("L_x","v","h","r","F_v","F_h","M","SR-F_v",
          "SR-F_h", "SR-M"),
```

Slika 4.36 – Način pozivanja „Title“ funkcije

Knot	L_x	v	h	r	F_v	F_h	M	SR-F_v	SR-F_h	SR-M
1.1	0.0	0.0	0.0	0.0	-22437.49999…	-9999.999999…	-19218.74999…	-22437.49999…	-9999.999999…	-19218.74999…
1.2	0.25	3.0886969585…	2.2123698020…	0.0002337021…	1.1641532182…	0.0	-1.455191522…	N/A	N/A	N/A

Slika 4.37 – Izgled matrice nakon izvršavanja funkcije

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Title >                                 #
#.......................................................................#

"""Python libraries. """
import numpy as np

#=======================================================================#
#                                Title                                   #
#=======================================================================#
def Title(*args):
    Title = []
    for Title_Name in args:
        Title = np.append(Title, Title_Name)
    return Title
#_________________________________<End>_________________________________#

#................................< End >................................#
```

Slika 4.38 – „Title“ modul

4.2.8. „Create Matrix“ modul

Modul „Create Matrix“ (*Slika 4.41*) služi za jednostavniju izradu matrice.

Funkcija ima tri osnovna argumenta. Argument „Title“ zadaje prvi redak matrice (funkcija je usko povezana s prethodnom funkcijom „Title“), drugi argument „Number“ zadaje prvi stupac matrice (povezana je s funkcijom „Numbering“), a treći argument „Main_Matrix“ opisuje glavnu matricu s podacima. Funkcija ima opciju dodavanja dodatnih matrica nakon glavne matrice, a broj dodatnih matrica može biti neograničen, pod uvjetom da imaju isti broj redaka kao i glavna matrica.

Način rada funkcije može se vidjeti na sljedećim slikama (*Slika 4.39*, *Slika 4.40*).

```
Knot_Results = Fun.Create(Fun.Title("L_x","v","h","r","F_v","F_h","M","SR-F_v",
                                    "SR-F_h", "SR-M"),
                          Input.Num_Knot,
                          L_x,
                          Fun.deList(Dis.Displacement),
                          Fun.deList(Load_Knots),
                          Fun.deList(Support))
```

Slika 4.39 – Način pozivanja „Create Matrix“ funkcije

Knot	L_x	v	h	r	F_v	F_h	M	SR-F_v	SR-F_h	SR-M
1.1	0.0	0.0	0.0	0.0	-22437.49999…	-9999.999999…	-19218.74999…	-22437.49999…	-9999.999999…	-19218.74999…
1.2	0.25	3.0886969585…	2.2123698020…	0.0002337021…	1.1641532182…	0.0	-1.455191522…	N/A	N/A	N/A
1.3	0.5	0.0001101171…	4.4247396040…	0.0003865972…	0.0	0.0	6.6938810050…	N/A	N/A	N/A
1.4	0.75	0.0002173774…	6.6371094061…	0.0004577949…	9.3132257461…	0.0	-1.455191522…	N/A	N/A	N/A
1.5	1.0	0.0003321323…	8.8494792081…	0.0004464048…	0.0	0.0	-4.656612873…	N/A	N/A	N/A
1.6	1.25	0.0004336233…	1.1061849010…	0.0003515363…	0.0	0.0	9.0221874415…	N/A	N/A	N/A

Slika 4.40 – Izgled matrice podatka nakon izvršavanja „Create Matrix“ funkcije

```
#.........................................................................#
#                            < Create Matrix >                            #
#.........................................................................#

"""Python libraries. """
import numpy as np

#=========================================================================#
#                             <Create Matrix>                             #
#=========================================================================#
def  Create_Matrix(Title , Number, Main_Matrix, *args):
    New_Matrix = Main_Matrix
    for Matrix in args:
        New_Matrix = np.hstack((New_Matrix, Matrix))
    New_Matrix = np.vstack((Title, New_Matrix))
    New_Matrix = np.hstack((Number, New_Matrix))
    return New_Matrix
#_____________________________________<End>_______________________________#

#.................................< End >.................................#
```

Slika 4.41 – „Create Matrix“ modul

4.3. Calculation

Paket „Calculation“ sadrži sve module potrebne za proračune, uključujući dobivanje matrica krutosti elemenata, globalne matrice krutosti, ekvivalentnih opterećenja te proračune poprečnog presjeka i momenta inercije. Datoteka __init__.py unutar paketa „Calculation“ grupira sve ove module, omogućujući njihovu jednostavniju upotrebu unutar glavnog koda. Primjer kako izgleda „__init__.py“ datoteka za ovaj paket može se vidjeti na sljedećoj slici (*Slika 4.42*).

```
#=============================================================================#
#                                    __init__                                 #
#                                  _3_Calculation                             #
#=============================================================================#
from .Element_Stiffness import Element_Stiffness as Element_k
from .Global_Stiffness import Global_Stiffness as Global_k
from .Equivalent_Distributed_Load import Equivalent_Distributed_Load as ED_Load
from .Beam_Characteristics import Beam_Characteristics as Beam_Characteristics
#_____________________________________End_____________________________________#
```

Slika 4.42 – „__init__.py“ modul, Calculation

4.3.1. „Element Stiffness“ modul

Modul „Element Stiffness“ (*Slika 4.44*) služi za generiranje matrice krutosti elemenata (ili pod-elemenata, ako ih ima).

Funkcija ima četiri osnovna argumenta: „E_ele“ (modul elastičnosti), „I_ele“ (moment inercije), „L_ele“ (dužina elementa/pod-elementa) i „A_ele“ (poprečni presjek). Ovi argumenti pružaju sve osnovne ulazne veličine potrebne za izračunavanje matrice krutosti elemenata (ili pod-elemenata).

Način rada funkcije je takav da za svaki element (ili pod-element) izračunava njegovu matricu krutosti te je pohranjuje u listu matrica, koja se kasnije vraća kao izlazna lista funkcije. *Slika 4.43* prikazuje izgled matrice krutosti za jedan element.

1.34766e+10	0	1.68457e+09	-1.34766e+10	0	1.68457e+09
0	4.52004e+09	0	0	-4.52004e+09	0
1.68457e+09	0	2.80762e+08	-1.68457e+09	0	1.40381e+08
-1.34766e+10	0	-1.68457e+09	1.34766e+10	0	-1.68457e+09
0	-4.52004e+09	0	0	4.52004e+09	0
1.68457e+09	0	1.40381e+08	-1.68457e+09	0	2.80762e+08

Slika 4.43 – Izgled matrice krutosti elementa unutar Spyder editora

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Element Stiffness >                          #
#.............................................................................#

"""Python libraries. """
import numpy as np
# import time

#=============================================================================#
#                              <Element Stiffness>                            #
#=============================================================================#
def Element_Stiffness(E_ele, I_ele, L_ele, A_ele):
    n=len(E_ele)
    Mat = []
    #-------------------------------------------------------------------------#
    for i in range(n):
        E,I,L,A = E_ele[i][0],I_ele[i][0],L_ele[i][0],A_ele[i][0]
        Mat_Temp = [[ 12*E*I/L**3, 0, 6*E*I/L**2, -12*E*I/L**3, 0, 6*E*I/L**2],
                    [0, A*E/L, 0, 0, -A*E/L, 0],
                    [ 6*E*I/L**2, 0, 4*E*I/L, -6*E*I/L**2, 0, 2*E*I/L],
                    [-12*E*I/L**3, 0, -6*E*I/L**2, 12*E*I/L**3, 0, -6*E*I/L**2],
                    [0, -A*E/L, 0, 0, A*E/L, 0],
                    [ 6*E*I/L**2, 0, 2*E*I/L, -6*E*I/L**2, 0, 4*E*I/L]]
        Mat.append(Mat_Temp)
        #---------------------------------------------------------------------#
    #     if i % (n/10) == 0:
    #         print('\r', "Element - (",  int(i+round((n/10))), ")", end='')
    #         time.sleep(0.5)
    # print('\r', end='')
    #-------------------------------------------------------------------------#
    Mat = np.array(Mat)
    #-------------------------------------------------------------------------#
    return Mat
#_____________________________________<End>___________________________________#

#..................................< End >....................................#
```

Slika 4.44 – „Element Stiffness“ modul

4.3.2. „Global Stiffness“ modul

Modul „Global Stiffness“ (*Slika 4.46*) služi za generiranje globalne matrice krutosti.

Funkcija ima jedan obavezni argument, „Element_Stiffness“, koji predstavlja listu matrica krutosti elemenata (ili pod-elemenata).

Način rada funkcije je sljedeći: prvo se generira globalna nul-matrica čija veličina odgovara ukupnom broju elemenata (ili pod-elemenata). Zatim se zbraja globalna matrica krutosti s matricom krutosti svakog elementa kako bi se dobila globalna matrica krutosti grede. Na sljedećoj slici može se vidjeti način zbrajanja matrica.

Slika 4.45 – Izgled dijela globalne matrice krutosti unutar Spyder editora

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Global Stiffness >                             #
#...............................................................................#

"""Python libraries. """
import numpy as np
# import time

#===============================================================================#
#                               <Global Stiffness>                              #
#===============================================================================#
def Global_Stiffness(Element_Stiffness):
    n = len(Element_Stiffness)
    x = int(3*(n+1))
    Sum_Raw = len(Element_Stiffness[0])
    Sum_Column = len(Element_Stiffness[0][0])
    New_Matrix = np.zeros((x,x))
    #---------------------------------------------------------------------------#
    for Element in range(len(Element_Stiffness)):
        Temp_Matrix = Element_Stiffness[Element]
        for Raw in range(Sum_Raw):
            for Column in range(Sum_Column):
                New_Matrix[Raw+Element*3][Column+Element*3] += Temp_Matrix[Raw][Column]
        #-----------------------------------------------------------------------#
    #     if Element % (n/10) == 0:
    #         print('\r', "Element Added - (",  int(Element+round((n/10),0)), ")", end='')
    #         time.sleep(0.5)
    # print('\r', end='')
    #---------------------------------------------------------------------------#
    return New_Matrix
#_____________________________________<End>_____________________________________#

#..................................< End >......................................#
```

Slika 4.46 – „Global Stiffness“ modul

4.3.3. „Equivalent Distributed Load“ modul

Modul „Equivalent Distributed Load“ (*Slika 4.49*) služi za generiranje ekvivalentnog opterećenja u slučajevima kada se između dva čvora nalazi kontinuirano ili koncentrirano opterećenje. Cilj je odrediti poprečne sile i momente na krajevima elementa (čvorovima) koji imaju isti učinak kao i opterećenje koje djeluje na promatrani element.

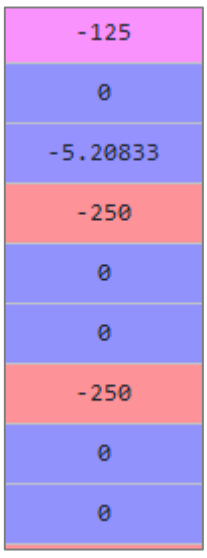
Funkcija ima dva obavezna argumenta: „q“, koji predstavlja kontinuirano opterećenje, i „L“, koji označava duljinu elementa. Također, ima jedan opcijski argument „variable“ koji definira izlaznu varijablu funkcije. Opcije za „variable“ su:

1. „Fd“ (ispisuje ekvivalentnu poprečnu silu),
2. „Md“ (ispisuje ekvivalentni moment),
3. „Qd“ (ispisuje ukupno ekvivalentno naprezanje).

Ako opcijski argument „variable“ nije zadan, funkcija će po postavkama će (defaultu) ispisivati ukupno ekvivalentno naprezanje. Primjer rada funkcije može se vidjeti na sljedećim slikama (*Slika 4.47*, *Slika 4.48*).

```
Qd = Calc.ED_Load(Input.q, Input.L)
```

Slika 4.47 – Način pozivanja „Equivalent Distributed Load“ funkcije



Slika 4.48 – Izgled matrice ekvivalentnih opterećenja

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                       < Equivalent Distributed Load >                         #
#.............................................................................#

"""Python libraries. """
import numpy as np

#=============================================================================#
#                        <Equivalent Distributed Load>                          #
#=============================================================================#
def Equivalent_Distributed_Load(q, L, variable = "Default"):
    n=len(L)
    Fd=np.zeros((n,2))
    Md=np.zeros((n,2))
    Qd=np.array([])
    #-------------------------------------------------------------------------#
    for i in range(n):

        Fd[i][0], Fd[i][1]=(q[i]*L[i])/2, (q[i]*L[i]/2)
        Md[i][0], Md[i][1]=(q[i]*L[i]**2/12), (-q[i]*L[i]**2/12)
    #-------------------------------------------------------------------------#
    for i in range(n):
        if (i==0 and n>1):
            Qd=np.append(Qd,Fd[0][0])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[0][0])
            Qd=np.append(Qd,Fd[0][1]+Fd[1][0])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[0][1]+Md[1][0])
        elif (i==0 and n==1):
            Qd=np.append(Qd,Fd[0][0])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[0][0])
            Qd=np.append(Qd,Fd[0][1])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[0][1])
        elif (i < n-1):
            Qd=np.append(Qd,Fd[i][1]+Fd[i+1][0])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[i][1]+Md[i+1][0])
        else:
            Qd=np.append(Qd,Fd[i][1])
            Qd=np.append(Qd, 0)
            Qd=np.append(Qd,Md[i][1])
    #-------------------------------------------------------------------------#
    if variable == "Fd":
        Output = Fd
    elif variable == "Md":
        Output = Md
    else:
        Output = Qd
    #-------------------------------------------------------------------------#
    return Output
#_____________________________________<End>___________________________________#

#..................................< End >....................................#
```

Slika 4.49 – „Equivalent Distributed Load“ modul

4.3.4. „Beam Characteristics“ modul

Modul „Beam Characteristics“ (*Slika 4.52*) služi za provjeru ulaznih podataka kao što su moment inercije i poprečni presjek. Ako ti podaci nisu zadani, funkcija ih izračunava na temelju visine, širine i debljine poprečnog presjeka.

Funkcija koristi jedan argument, „Matrix“, koji sadrži sve podatke za svaki element.

Način rada funkcije je sljedeći: funkcija pregledava svaki element, odnosno pod-element ako ih ima, i provjerava vrijednosti momenta inercije i poprečnog presjeka. Ako te vrijednosti nisu unesene ili su postavljene na 0, poprečni presjek i moment inercije se računaju na temelju drugih varijabli kao što su visina, širina i debljina kvadratnog presjeka. Način rada funkcije može se vidjeti na sljedećim slikama (*Slika 4.50*, *Slika 4.51*).

E	L	q	Num	I	A	H	W	t
210000000000	0.25	-1000	10	nan	nan	400	180	8.6
210000000000	0.25	-1000	10	nan	nan	400	180	8.6
210000000000	0.25	-1000	10	nan	nan	400	180	8.6
210000000000	0.25	-1000	10	nan	nan	400	180	8.6

Slika 4.50 – Izgled matrice prije izvršavanja „Beam Calculation“ funkcije

Element	E	L	q	I	A
1.1	210000000000	0.25	-1000	0.0001989924…	0.0096801599…
1.2	210000000000	0.25	-1000	0.0001989924…	0.0096801599…
1.3	210000000000	0.25	-1000	0.0001989924…	0.0096801599…
1.4	210000000000	0.25	-1000	0.0001989924…	0.0096801599…

Slika 4.51 – Izgled matrice nakon izvršavanja „Beam Calculation“ funkcije

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                         < Beam Characteristics >                            #
#.............................................................................#

"""Python libraries. """
import numpy as np

#=============================================================================#
#                          <Beam Characteristics>                             #
#=============================================================================#
def Beam_Characteristics (Matrix):
    Sum_Col = len(Matrix[0])
    for Col in range(Sum_Col):
        if Matrix[0][Col] == "I":
            I = Matrix[1:,[Col]]
        elif Matrix[0][Col] == "A":
            A = Matrix[1:,[Col]]
        elif Matrix[0][Col] == "H":
            H = Matrix[1:,[Col]]
        elif Matrix[0][Col] == "W":
            W = Matrix[1:,[Col]]
        elif Matrix[0][Col] == "t":
            t = Matrix[1:,[Col]]
    #-------------------------------------------------------------------------#
    I_calc = []
    I_calc = np.append(I_calc, "I")
    A_calc = []
    A_calc = np.append(A_calc, "A")
    I_Sum = len(I)
    #-------------------------------------------------------------------------#
    for Row in range(I_Sum):
        if float(I[Row]) > 0:
           I_calc = np.append(I_calc, I[Row])
           A_calc = np.append(A_calc, A[Row])
        else:
            h = float(H[Row]) - 2 * float(t[Row])
            w = float(W[Row]) - 2 * float(t[Row])
            I_calc_mm = (float(W[Row]) * float(H[Row])**3 - w * h**3)/12
            I_calc_m = I_calc_mm / (10**12)
            I_calc = np.append(I_calc, I_calc_m)

            A_calc_mm = float(W[Row]) * float(H[Row]) - w * h
            A_calc_m = A_calc_mm / (10**6)
            A_calc = np.append(A_calc, A_calc_m)
    #-------------------------------------------------------------------------#
    I_calc = np.reshape(I_calc,(len(I_calc),1))
    A_calc = np.reshape(A_calc,(len(A_calc),1))
    #-------------------------------------------------------------------------#
    return I_calc, A_calc
#____________________________________<End>____________________________________#

#..................................< End >....................................#
```

Slika 4.52 – „Beam Characteristics" modul (function)

4.4. Results

Paket „Results“ sadrži sve module potrebne za izračun i prikaz rezultata pomaka, kao i rezultate za oslonce i elemente grede. Datoteka __init__.py unutar ovog paketa grupira sve module zajedno, omogućujući njihovo jednostavno korištenje u glavnom kodu. Na sljedećoj slici može se vidjeti primjer __init__.py modula za paket „Results“ (*Slika 4.53*).

```
#=============================================================================#
#                                    __init__                                 #
#                                   _4_Results                                #
#=============================================================================#
from .Displacement_Results import Displacement_Results as Displacement
from .Support_Reaction import Support_Reaction as Support
from .Element_Load_Results import Element_Load_Results as Element
#_____________________________________End_____________________________________#
```

Slika 4.53 – „__init__.py“ modul, Results

4.4.1. „Displacement Results“ modul

Modul „Displacement Results“ (*Slika 4.54*) služi za kalkulaciju pomaka čvorova.

Funkcija koristi tri obavezna argumenta:

1. „Boundary_Condition“: matrica rubnih uvjeta
2. „Global_Stiffness“: matrica globalne krutosti
3. „Equivalent_Load“: ekvivalentno opterećenje grede uslijed svih opterećenja.

Način rada funkcije je sljedeći: globalna matrica krutosti i ekvivalentna opterećenja reducira se pomoću rubnih uvjeta. Kada su obje matrice reducirane, one se međusobno množe kako bi se dobili rezultati pomaka čvorova.

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                          < Displacement Results >                           #
#.............................................................................#

"""Python libraries. """
import numpy as np

#=============================================================================#
#                           <Displacement Results>                            #
#=============================================================================#
def Displacement_Results(Boundary_Condition, Global_Stiffness, Equivalent_Load):
    index = np.where(Boundary_Condition == 1)[0]
    #-------------------------------------------------------------------------#
    Reduced_Global_Stiffness_x = np.delete(Global_Stiffness, index, axis=0)
    Reduced_Global_Stiffness_y = np.delete(Reduced_Global_Stiffness_x , index, axis=1)
    Reduced_Global_Stiffness = Reduced_Global_Stiffness_y
    #-------------------------------------------------------------------------#
    Reduced_Equivalent_Load = np.delete(Equivalent_Load, index, axis=0)
    #-------------------------------------------------------------------------#
    Displacement = np.linalg.solve(Reduced_Global_Stiffness,  Reduced_Equivalent_Load)
    #-------------------------------------------------------------------------#
    for i in index:
        Displacement=np.insert(Displacement, i, 0)
    #-------------------------------------------------------------------------#
    return Displacement

#____________________________________<End>___________________________________#

#..................................< End >....................................#
```

Slika 4.54 – „Displacement Results“ modul

4.4.2. „Support Reaction“ modul

Modul „Support Reaction“ (*Slika 4.55*) služi za kalkulaciju naprezanja u osloncima.

Funkcija koristi tri obavezna argumenta:

1. „Load_Knots“: matrica koja daje naprezanja u čvorovima
2. „C_Load“: matrica koja sadrži koncentrirana opterećenja zadana ulaznim podacima
3. „Boundary_Condition“: matrica koja označava rubne uvjete grede i mjesta gdje se nalaze oslonci.

Način rada funkcije je sljedeći: funkcija identificira mjesta gdje se nalaze oslonci prema rubnim uvjetima i izračunava reakcije u tim osloncima. Reakcije se dobivaju kao razlika između kalkuliranog naprezanja u čvorovima i ulaznih vrijednosti na tim čvorovima.

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                          < Support Reaction >                               #
#.............................................................................#

#=============================================================================#
#                           <Support Reaction>                                #
#=============================================================================#
def Support_Reaction(Load_Knots, C_Load, Boundary_Condition):
    Boundery = Boundary_Condition
    Load = Load_Knots
    Load_input = C_Load
    Support = []
    #-------------------------------------------------------------------------#
    for i in range(len(Boundery)):
        if Boundery[i] == 1:
            x = Load[i] - Load_input[i]
            Support.append(x)
        else:
            Support.append("N/A")
    #-------------------------------------------------------------------------#
    return Support

#____________________________________<End>____________________________________#

#...................................< End >...................................#
```

Slika 4.55 – „Support Reaction“ modul

4.4.3. „Element Load Results“ modul

Modul „Element Load Results“ (*Slika 4.56*) kalkulira opterećenja za svaki element grede.

Funkcija koristi četiri argumenta:

1. „Element_Stiffness“: lista matrica krutosti svih elemenata grede
2. „Displacement“: matrica koja daje pomake grede uslijed svih naprezanja
3. „Fd“: matrica koja sadrži ekvivalentna posmična naprezanja u čvorovima
4. „Md“: matrica koja daje ekvivalentne momente u čvorovima

Funkcija izračunava posmična, aksijalna opterećenja i momente za svaki element. Način rada funkcije uključuje analizu opterećenja na početku i kraju svakog elementa kako bi se dobili traženi rezultati.

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                         < Element Load Results >                            #
#............................................................................#

"""Python libraries. """
import numpy as np

#============================================================================#
#                          <Element Load Results>                             #
#============================================================================#
def Element_Load_Results(Element_stiffness, Displacement, Fd, Md):
    n=len(Fd)
    Element_Load = []
    #------------------------------------------------------------------------#
    for i in range(n):
        Element_Load_0 = np.array(Element_stiffness[i]).dot(Displacement[3*i:3*i+6])
        for j in range(6):
            if j%3 == 0:
                Element_Load_0[j] -= Fd[i][int(j/3)]

            elif (j + 1)%3  == 0:
                Element_Load_0[j] -= Md[i][int(j/3)]

            elif (j + 2)%3  == 0:
                Element_Load_0[j] = Element_Load_0[j]
        #--------------------------------------------------------------------#
        Element_Load_0[1] = -Element_Load_0[1]
        Element_Load_0[2] = -Element_Load_0[2]
        Element_Load.append(Element_Load_0)
    #------------------------------------------------------------------------#
    return Element_Load

#_________________________________<End>______________________________________#

#..................................< End >...................................#
```

Slika 4.56 – „Element Load Results“ modul

4.5. „Program section“ module

4.5.1. „Input Directory“ module

Modul „Input Directory“ (*Slika 4.57*) služi za definiranje naziva ulazne i izlazne Excel datoteke. Ovaj modul koristi funkcije „Input_Directory“ i „Output_Directory“, koje su uvezene iz paketa „_1_Excel_Data“. Paket „_1_Excel_Data“ je namijenjen za manipuliranje Excel datotekama i upravljanje njihovim ulaznim i izlaznim varijablama.

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Input_Directory >                              #
#...............................................................................#

"""User Function. """
import _1_Excel_Data as Excel

#================================= Calculation ================================#

Input = "Input.xlsm"
Output = "Output.xlsx"
#------------------------------------------------------------------------------#
Input_Excel = Excel.Input_Directory(Input)
Output_Excel = Excel.Output_Directory(Output)

#...................................< End >....................................#
```

Slika 4.57 – „Input Directory“ modul

4.5.2. „Input“ modul

Modul „Input“ (*Slika 4.59*) služi za definiranje svih ulaznih varijabli kako bi ostali moduli kasnije mogli lako uvoziti željene varijable. Modul koristi tri paketa i jedan modul iz iste mape.

Modul poziva funkcije za dobivanje sirovih podataka, a zatim, koristeći funkcije iz određenih paketa, transformira te sirove podatke u format koji će biti potreban za daljnju obradu i analizu. Ovaj proces osigurava da su svi ulazni podaci pravilno formatirani i dostupni za upotrebu u drugim modulima. Primjer rezultata se može vidjeti na sljedećoj slici (*Slika 4.58*).

A	Array of float64	(40, 1)	[[0.00968016] [0.00968016]
E	Array of float64	(40, 1)	[[2.1e+11] [2.1e+11]
F_h	Array of float64	(41, 1)	[[0.] [0.]
F_v	Array of float64	(41, 1)	[[0.] [0.]
h_b	Array of float64	(41, 1)	[[1.] [0.]
I	Array of float64	(40, 1)	[[0.00019898]
L	Array of float64	(40, 1)	[[0.25] [0.25]
M	Array of float64	(41, 1)	[[0.] [0.]
Num	Array of float64	(4, 1)	[[10.] [10.]
q	Array of float64	(40, 1)	[[-1000.] [-1000.]
r_b	Array of float64	(41, 1)	[[1.] [0.]
v_b	Array of float64	(41, 1)	[[1.] [0.]
Num_Knot	Array of str128	(42, 1)	ndarray object of numpy module
Num_Element	Array of str224	(41, 1)	ndarray object of numpy module
Raw_Knot	Array of str672	(6, 6)	ndarray object of numpy module
Element	Array of str1024	(41, 6)	ndarray object of numpy module
Knot	Array of str1024	(42, 6)	ndarray object of numpy module
Raw_Element	Array of str1024	(5, 9)	ndarray object of numpy module
Raw_Input_Data	tuple	2	(Numpy array, Numpy array)

Samostalne varijable spremne za pozivanje

Slika 4.58 – Prikaz varijabli unutar Spyder editora

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                  < Input >                                  #
#.............................................................................#

"""Python libraries. """
import warnings
import numpy as np

#=============================================================================#

"""User Function. """
import _1_Excel_Data as Excel
import _2_Function as Fun
import _3_Calculation as Calc
#-----------------------------------------------------------------------------#
warnings.simplefilter(action='ignore', category=UserWarning)

#=============================================================================#

"""Code sections. """
import Input_Directory as Input

#================================= Calculation ================================#

Raw_Input_Data = Excel.Raw_Data(Input.Input_Excel)
Raw_Element = Raw_Input_Data[1]
Raw_Knot = Raw_Input_Data[0]
Num = Fun.Input_Variable("Num", Raw_Knot, Raw_Element)
#-----------------------------------------------------------------------------#
Knot = Excel.Data(Raw_Input_Data, "Knot")
Num_Knot = Fun.Numbering(Num, "Knot")
#-----------------------------------------------------------------------------#
Element = Excel.Data(Raw_Input_Data, "Element")
I, A = Calc.Beam_Characteristics(Element)
Element = Fun.Del_Col(Element, "Num", "I", "A", "H", "W", "t")
Num_Element = Fun.Numbering(Num, "Element")
Element = np.hstack((Num_Element, Element, I, A))
#-----------------------------------------------------------------------------#
v_b = Fun.Input_Variable("v_b", Knot, Element)
h_b = Fun.Input_Variable("h_b", Knot, Element)
r_b = Fun.Input_Variable("r_b", Knot, Element)
#-----------------------------------------------------------------------------#
F_v = Fun.Input_Variable("F_v", Knot, Element)
F_h = Fun.Input_Variable("F_h", Knot, Element)
M = Fun.Input_Variable("M", Knot, Element)
#-----------------------------------------------------------------------------#
E = Fun.Input_Variable("E", Knot, Element)
I = Fun.Input_Variable("I", Knot, Element)
A = Fun.Input_Variable("A", Knot, Element)
L = Fun.Input_Variable("L", Knot, Element)
q = Fun.Input_Variable("q", Knot, Element)

#..................................< End >....................................#
```

Slika 4.59 – „Input“ modul

4.5.3. „Beam Characteristics“ modul

Modul „Beam Characteristics“ (*Slika 4.60*) služi za dobivanje karakteristika grede, uključujući rubne uvjete, matricu krutosti elemenata te globalnu matricu krutosti grede. Modul koristi ulazne podatke iz modula „Input“ i pomoću ranije definiranih funkcija generira potrebne podatke.

Način rada modula uključuje:

- Preuzimanje ulaznih podataka iz modula „Input“.
- Korištenje funkcija za izračunavanje karakteristika grede, uključujući sve relevantne matrice i uvjete.
- Dobivanje rezultata kao što su rubni uvjeti, matrica krutosti elemenata i globalna matrica krutosti grede, spremnih za daljnju analizu i obradu.

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                       < Beam Characteristics >                             #
#............................................................................#

"""User Function. """
import _2_Function as Fun
import _3_Calculation as Calc

#============================================================================#

"""Code sections. """
import Input

#================================ Calculation ===============================#

Boundary_Condition = Fun.List(Input.v_b, Input.h_b, Input.r_b)
Element_Stiffness = Calc.Element_k(Input.E, Input.I, Input.L, Input.A)
Global_Stiffness = Calc.Global_k(Element_Stiffness)

#.................................< End >....................................#
```

Slika 4.60 – „Beam Characteristics“ modul

4.5.4. *„Beam Load" modul*

Modul „Beam Load" (*Slika 4.61*) služi za određivanje svih vanjskih opterećenja na gredi, uključujući koncentrirana i kontinuirana opterećenja. Ulazne podatke preuzima iz modula „Input", kao što je slučaj s prijašnjim modulima. Rezultat rada ovog modula su:

- Koncentrirana opterećenja: Izračunava se raspodjela koncentriranih sila na gredi.
- Ukupna ekvivalentna opterećenja uslijed kontinuiranog opterećenja.
- Ekvivalentna posmična opterećenja uslijed kontinuiranog opterećenja: Određuju se posmične sile koje ekvivalentno djeluju na čvorovima uslijed kontinuiranog opterećenja.
- Ekvivalentni momenti uslijed kontinuiranog opterećenja: Računaju se momenti koji djeluju na krajevima elemenata kao rezultat kontinuiranog opterećenja.
- Ukupno ekvivalentno opterećenje: Kombinira se ukupni učinak svih prethodnih opterećenja kako bi se dobila cjelokupna slika opterećenja koja djeluju na gredu.

Ovaj modul omogućava sveobuhvatan prikaz vanjskih opterećenja koja djeluju na gredu, pružajući ključne podatke za daljnju analizu.

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                             < Beam Load >                                  #
#............................................................................#

"""User Function. """
import _2_Function as Fun
import _3_Calculation as Calc

#============================================================================#

"""Code sections. """
import Input

#================================ Calculation ===============================#

Concetrated_Load = Fun.List(Input.F_v, Input.F_h, Input.M)
Qd = Calc.ED_Load(Input.q, Input.L)
Fd = Calc.ED_Load(Input.q, Input.L, "Fd")
Md = Calc.ED_Load(Input.q, Input.L, "Md")
Equivalent_Load = Concetrated_Load + Qd

#..................................< End >...................................#
```

Slika 4.61 – „Beam Load" modul

4.5.5. *„Displacement“ modul*

Modul „Displacement“ (*Slika 4.62*) služi za izračunavanje pomaka u čvorovima grede. Modul koristi rezultate dobivene iz prethodna dva modula: „Beam Characteristics“ i „Beam Load“. Korištenjem funkcije „Displacement“ iz paketa „_4_Results“, izračunavaju se pomaci u svim čvorovima grede.

Modul uzima u obzir karakteristike grede, uključujući globalnu matricu krutosti, te vanjska opterećenja, kako bi precizno izračunao pomake u svim relevantnim čvorovima grede.

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Displacement >                              #
#............................................................................#

"""User Function. """
import _4_Results as Result

#============================================================================#

"""Code sections. """
import Beam_Characteristics as Char
import Beam_Load as Load

#================================ Calculation ===============================#

Displacement = Result.Displacement(Char.Boundary_Condition,
                                  Char.Global_Stiffness,
                                  Load.Equivalent_Load)

#..................................< End >...................................#
```

Slika 4.62 – „Displacement“ modul

4.5.6. „Knot Results“ modul

Modul „Knot Results“ (*Slika 4.64*) služi za izračunavanje i prikaz rezultata u svim čvorovima grede, uključujući njihove pomake, opterećenja, te maksimalne vrijednosti. Osim toga, modul generira rezultate za sve oslonce. Svi dobiveni podaci grupiraju se u jednu cjelovitu matricu (tablicu), koja daje pregled svih relevantnih informacija o čvorovima i osloncima. Primjer matrice sa svim rezultatima može se vidjeti na sljedećoj slici (*Slika 4.63*).

Knot	L_x	v	h	r	F_v	F_h	M	SR-F_v	SR-F_h	SR-M
1.1	0.0	0.0	0.0	0.0	-22437.50000…	-10000.00000…	-19218.75000…	-22437.50000…	-10000.00000…	-19218.75000…
1.2	0.25	1.2969912383…	1.2298104478…	9.8135108836…	0.0	0.0	-9.313225746…	N/A	N/A	N/A
1.3	0.5	4.6239879826…	2.4596208956…	0.0001623381…	-4.656612873…	0.0	2.0372681319…	N/A	N/A	N/A
1.4	0.75	9.1280140129…	3.6894313435…	0.0001922351…	0.0	0.0	-1.571606844…	N/A	N/A	N/A
1.5	1.0	0.0001394674…	4.9192417913…	0.0001874522…	0.0	0.0	-1.455191522…	N/A	N/A	N/A
1.6	1.25	0.0001820851…	6.1490522391…	0.0001476155…	0.0	0.0	-3.201421350…	N/A	N/A	N/A
1.7	1.5	0.0002103229…	7.3788626870…	7.2351078811…	1.8626451492…	0.0	-3.346940502…	N/A	N/A	N/A
1.8	1.75	0.0002152771…	8.6086731348…	-3.871499372…	1.8626451492…	-2.910383045…	-2.546585164…	N/A	N/A	N/A
1.9	2.0	0.0001879507…	9.8384835827…	-0.000185956…	0.0	2.9103830456…	0.0	N/A	N/A	N/A
1.10	2.25	0.0001192530…	1.1068294030…	-0.000369747…	0.0	0.0	-5.820766091…	N/A	N/A	N/A
2.1	2.5	0.0	1.2298104478…	-0.000590462…	34937.500001…	0.0	7.6058013220…	34937.500001…	N/A	N/A
2.2	2.75	-0.000176904…	1.3527914926…	-0.000822284…	0.0	-2.910383045…	0.0	N/A	N/A	N/A
2.3	3.0	-0.000409895…	1.4757725374…	-0.001039150…	0.0	0.0	-2.328306436…	N/A	N/A	N/A

Slika 4.63 – Izgled matrice rezultata čvorova unutar Spyder editora

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Knot Results >                              #
#...........................................................................#

"""Python libraries. """
import numpy as np

#===========================================================================#

"""User Function. """
import _2_Function as Fun
import _4_Results as Result

#===========================================================================#

"""Code sections. """
import Input
import Displacement as Dis
import Beam_Load
import Beam_Characteristics as Char

#================================ Calculation ===============================#

Load = Char.Global_Stiffness.dot(Dis.Displacement)
Load_Knots = Load - Beam_Load.Qd
Support = Result.Support(Load_Knots,
                          Beam_Load.Concetrated_Load,
                          Char.Boundary_Condition)
#---------------------------------------------------------------------------#
L_x = [0]
for i in range(len(Input.L)):
    L_x = np.append(L_x, L_x[i] + Input.L[i])
L_x = np.reshape(L_x,(len(L_x),1))
#---------------------------------------------------------------------------#
Knot_Results = Fun.Create(Fun.Title("L_x","v","h","r","F_v","F_h","M","SR-F_v",
                                    "SR-F_h", "SR-M"),
                          Input.Num_Knot,
                          L_x,
                          Fun.deList(Dis.Displacement),
                          Fun.deList(Load_Knots),
                          Fun.deList(Support))
#---------------------------------------------------------------------------#
v_max = Fun.Input_Variable("v", Knot_Results)*1000
v_max = Fun.Max_Value("Vertical Displacement", v_max)
h_max = Fun.Input_Variable("h", Knot_Results)*1000
h_max = Fun.Max_Value("Horizontal Displacement", h_max)
r_max = Fun.Input_Variable("r", Knot_Results)*(180/np.pi)
r_max = Fun.Max_Value("Horizontal Displacement", r_max)
Max_Table = np.vstack((v_max, h_max, r_max))

#.................................< End >...................................#
```

Slika 4.64 – „Knot Results“ modul

4.5.7. „Element Results“ modul

Modul „Element Results“ (*Slika 4.66*, *Slika 4.67*) služi za prikaz svih rezultata elemenata grede, uključujući rezultate na početku i kraju svakog elementa (ili pod-elemenata), kao i maksimalne vrijednosti svih elemenata. Varijabla „Beam_Element“ organizira podatke u obliku matrice, gdje svaki redak predstavlja jedan element. Prva tri stupca matrice sadrže podatke o opterećenju na početku elementa (lijeva strana elementa), dok posljednja tri stupca prikazuju podatke o opterećenju na kraju elementa (desna strana elementa). Na sljedećoj slici je prikazan primjer rezultata elemenata (*Slika 4.65*).

Element	F_v1	F_h1	M_1	F_v2	F_h2	M_2
1.1	-22437.49999…	9999.9999999…	19218.749999…	22687.499999…	9999.9999999…	13578.124999…
1.2	-22687.49999…	9999.9999999…	13578.124999…	22937.499999…	9999.9999999…	7874.9999999…
1.3	-22937.49999…	9999.9999999…	7874.9999999…	23187.499999…	9999.9999999…	2109.3749999…
1.4	-23187.49999…	9999.9999999…	2109.3749999…	23437.499999…	9999.9999999…	-3718.749999…
1.5	-23437.49999…	9999.9999999…	-3718.749999…	23687.499999…	9999.9999999…	-9609.374999…
1.6	-23687.49999…	9999.9999999…	-9609.374999…	23937.499999…	9999.9999999…	-15562.49999…
1.7	-23937.49999…	9999.9999999…	-15562.49999…	24187.499999…	9999.9999999…	-21578.12499…
1.8	-24187.49999…	10000.0	-21578.12499…	24437.499999…	10000.0	-27656.24999…
1.9	-24437.49999…	9999.9999999…	-27656.24999…	24687.499999…	9999.9999999…	-33796.87499…
1.10	-24687.49999…	10000.000000…	-33796.87499…	24937.499999…	10000.000000…	-39999.99999…
2.1	9999.9999999…	10000.0	-39999.99999…	-9999.999999…	10000.0	-37499.99999…
2.2	9999.9999999…	9999.9999999…	-37499.99999…	-9999.999999…	9999.9999999…	-34999.99999…
2.3	9999.9999999…	10000.0	-34999.99999…	-9999.999999…	10000.0	-32499.99999…

Slika 4.65 – Izgled matrice rezultata unutar Spyder editora

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                            < Element Results >                             #
#............................................................................#

"""Python libraries. """
import numpy as np

#============================================================================#

"""User Function. """
import _2_Function as Fun
import _4_Results as Result

#============================================================================#

"""Code sections. """
import Input
import Beam_Load as Load
import Beam_Characteristics as Char
import Displacement as Dis

#================================== Calculation ================================#

Beam_Element_List = Result.Element(Char.Element_Stiffness,
                          Dis.Displacement,
                          Load.Fd,
                          Load.Md)
#----------------------------------------------------------------------------#
Beam_Element = [Fun.Title("F_v1", "F_h1", "M_1", "F_v2", "F_h2", "M_2")]
#----------------------------------------------------------------------------#
for Element in range(len(Beam_Element_List)):
    Beam_Element = np.vstack((Beam_Element, Beam_Element_List[Element]))
#----------------------------------------------------------------------------#
Beam_Element = np.hstack((Input.Num_Element, Beam_Element))
#----------------------------------------------------------------------------#

#============================================================================#
#                              <Function>                                    #
#============================================================================#
def Data_Function(x, y, Process = "Standard"):
    x1 = Fun.Input_Variable(x, Beam_Element)

    if Process == "Shear Force":
        x2 = -Fun.Input_Variable(y, Beam_Element)
    else:
        x2 = Fun.Input_Variable(y, Beam_Element)
    n = len(x1)+1
    New = np.zeros((n,3))
    for i in range(n):
        if i == 0:
            New[i,0] = 0
            New[i,1] = x1[i,0]
            New[i,2] = New[i,1] - New[i,0]
        elif i==len(x1):
            New[i,0] = x2[i-1,0]
            New[i,1] = 0
            New[i,2] = New[i,1] - New[i,0]
        else:
            New[i,0] = x2[i-1,0]
            New[i,1] = x1[i,0]
            New[i,2] = New[i,1] - New[i,0]
```

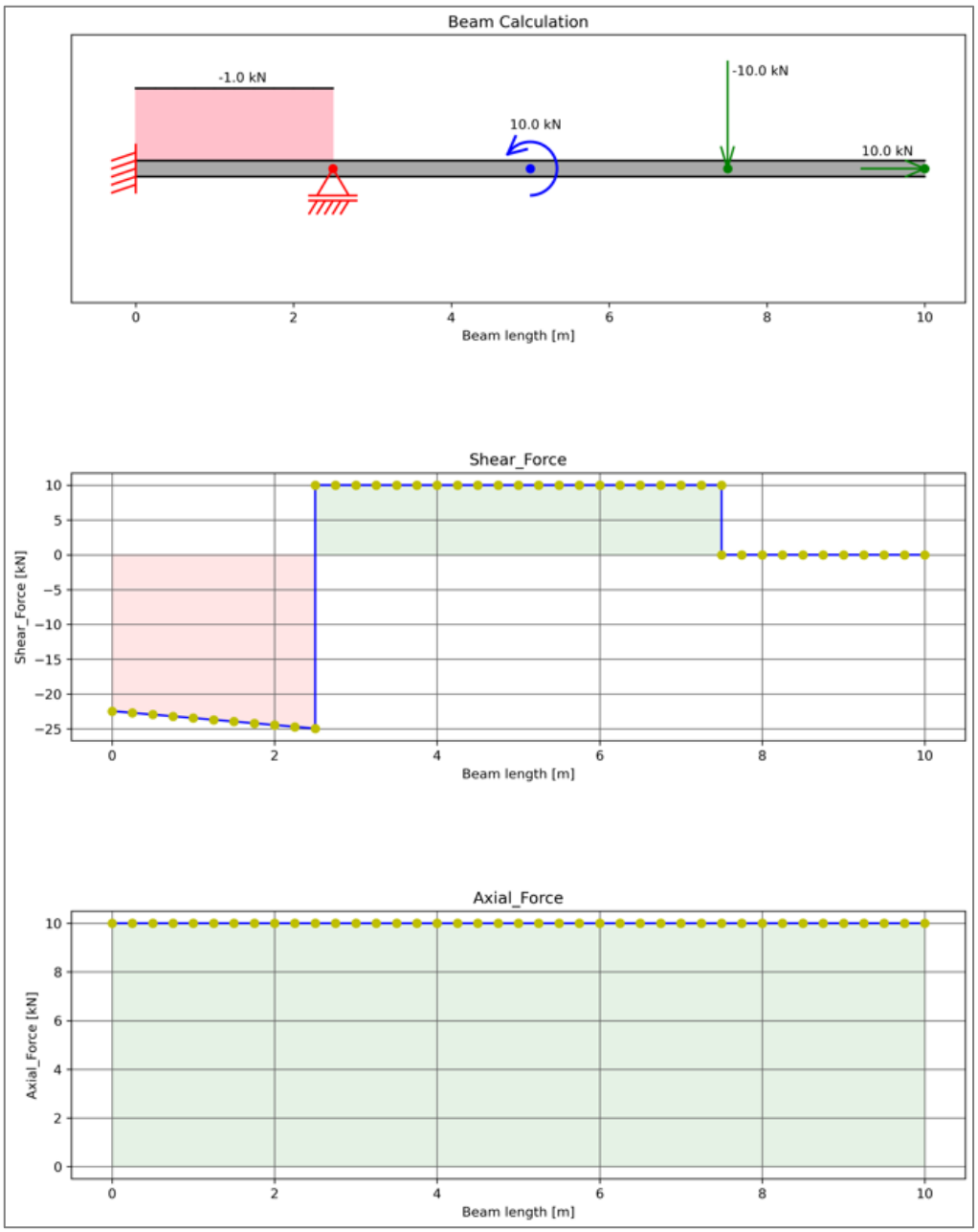
Slika 4.66 – „Element Results“ modul (1)

```
    return New
#_____________________________________<End>_____________________________________#

F_v = Data_Function("F_v1", "F_v2", Process="Shear Force")/1000
F_h = Data_Function("F_h1", "F_h2", Process="Axial Force")/1000
M = Data_Function("M_1", "M_2", Process="Bending Moment")/1000
#-------------------------------------------------------------------------------#
Max_Shear_Force = Fun.Max_Value("Shear Force",
                        np.vstack((F_v[:,0:1],F_v[:,1:2],F_v[:,2:3])))
Max_Tension_Force = Fun.Max_Value("Tension Force",
                        np.vstack((F_h[:,0:1],F_h[:,1:2])), Operation="max")
Max_Compression_Force = Fun.Max_Value("Compression Force",
                        np.vstack((F_h[:,0:1],F_h[:,1:2])), Operation="min")
Max_Bending_Moment = Fun.Max_Value("Bending Moment",
                        np.vstack((M[:,0:1],M[:,1:2],M[:,2:3])))
Max_Table = Max_Table = np.vstack((Max_Shear_Force,
                                   Max_Tension_Force,
                                   Max_Compression_Force,
                                   Max_Bending_Moment))

#.....................................< End >.....................................#
```

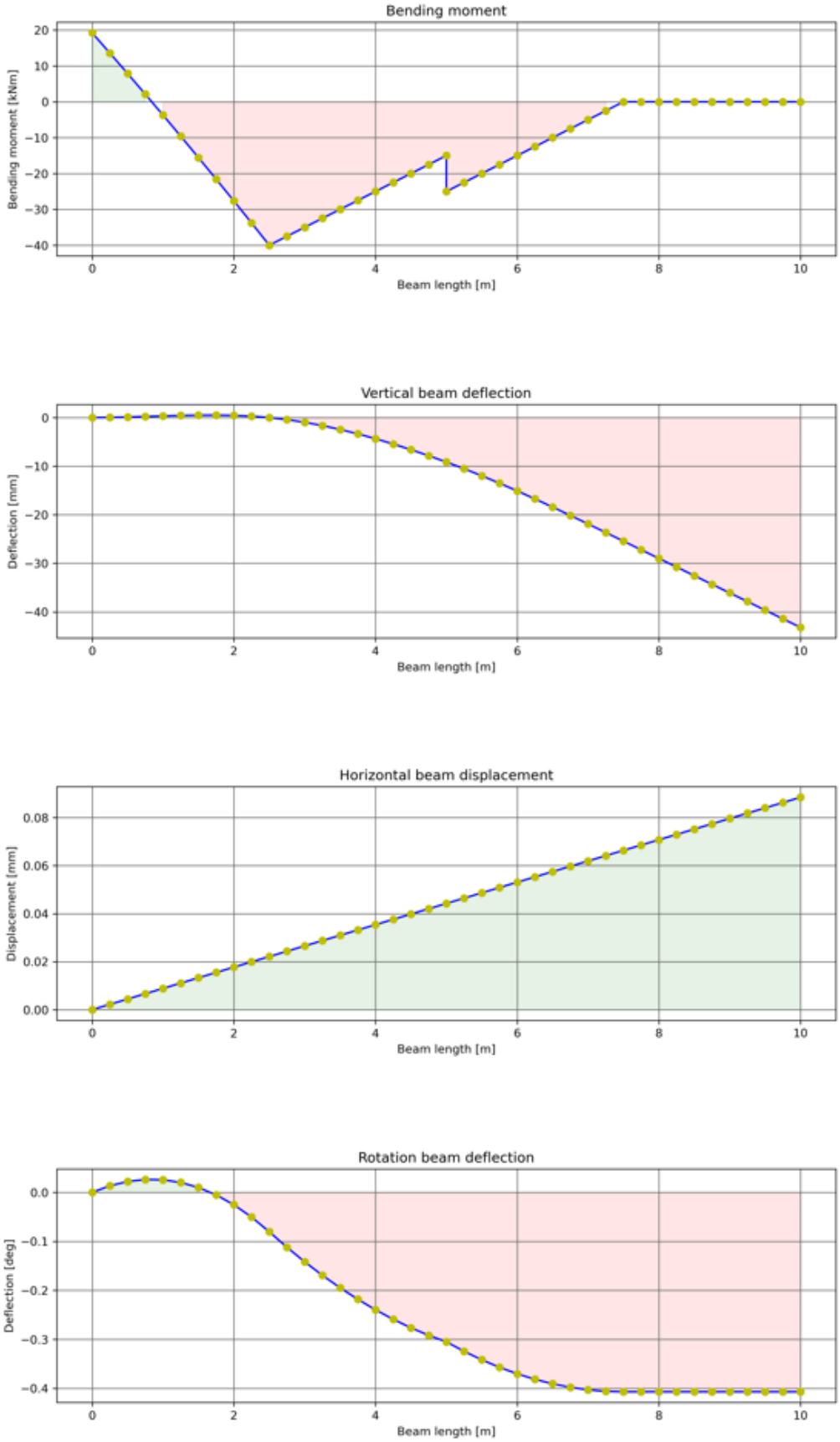
Slika 4.67 – „Element Results“ modul (2)

4.5.8. „Plot“ modul

Modul „Plot“ (*Slika 4.70*, *Slika 4.71*, *Slika 4.72*, *Slika 4.73*, *Slika 4.74*, *Slika 4.75*, *Slika 4.76*) služi za grafički prikaz rezultata u obliku grafova. Modul koristi niz funkcija i biblioteka kako bi nacrtao zadanu gredu i prikazao njene grafove opterećenja i pomaka, uključujući vertikalni i horizontalni pomak, rotacijski pomak, posmične sile, aksijalne sile, te momente koji djeluju na gredu. Na sljedećim slikama možete vidjeti izgled grafova i pripadajuće grede (*Slika 4.68*, *Slika 4.69*).



Slika 4.68 – Dijagram nakon izvršavanja „Plot“ modula (1)



Slika 4.69 – Dijagram nakon izvršavanja „Plot“ modula (2)

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                   < Plot >                                   #
#.............................................................................#

"""Python libraries. """
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np

#=============================================================================#

"""User Function. """
import _1_Excel_Data as Data
import _2_Function as Fun

#=============================================================================#

"""Code sections. """
import Input
import Knot_Results as Knot
import Element_Results as Element

#================================= Calculation ===============================#

Number_of_Plot = 7
Height = Number_of_Plot * 3 - 1
#-----------------------------------------------------------------------------#
plt.figure(figsize=(12, Height*2))
G = gridspec.GridSpec(Height, 1)
G_Last = 0

#=============================================================================#
#                                 <Functions>                                  #
#=============================================================================#
def Deflection(Name, Name_Symbol, Unit, Title):
    global G_Last
    axes = plt.subplot(G[0+G_Last:2+G_Last, :])
    G_Last = G_Last + 3
    L_x = Fun.Input_Variable("L_x", Knot.Knot_Results)
    v = Fun.Input_Variable(Name_Symbol, Knot.Knot_Results)
#-----------------------------------------------------------------------------#
    x_def = L_x
    y_def = v*Unit
    x = np.reshape(x_def, len(x_def))
    y = np.reshape(y_def, len(y_def))
#-----------------------------------------------------------------------------#
    axes.plot(x,y, "-b")
    axes.plot(x,y, "oy")
    axes.set_xlabel("Beam length [m]")
    axes.set_ylabel(Name)
    axes.set_title(Title)
    axes.grid(which='major', color='#666666', linestyle='-')
#-----------------------------------------------------------------------------#
    axes.fill_between(x, 0, y, where = y <= 0,color='red', alpha=.1)
    axes.fill_between(x, 0, y, where = y >= 0,color='green', alpha=.1)
#__________________________________<End>______________________________________#

def Force(Name, Name_Symbol_1, Name_Symbol_2, Unit, Title, Process = "No"):
    global G_Last
    axes = plt.subplot(G[0+G_Last:2+G_Last, :])
    G_Last = G_Last + 3
```

Slika 4.70 – „Plot“ modul (1)

```
    L_x = Fun.Input_Variable("L_x", Knot.Knot_Results)
    F_v1 = Fun.Input_Variable(Name_Symbol_1, Element.Beam_Element)
    F_v2 = Fun.Input_Variable(Name_Symbol_2, Element.Beam_Element)
#-------------------------------------------------------------------------------#
    x_def = L_x
    y_def1 = F_v1 * Unit
    y_def2 = F_v2 * Unit
    x = []
    y = []
#-------------------------------------------------------------------------------#
    for idx, x_var in enumerate(x_def):
        if (idx == 0 or idx == len(x_def)-1):
            x = np.append(x, x_var)
        else:
            x = np.append(x, x_var)
            x = np.append(x, x_var)

    for i in range(len(y_def1)):
        y = np.append(y, y_def1[i])
        if Process == "Yes":
           y = np.append(y, -y_def2[i])
        else:
           y = np.append(y, y_def2[i])
#-------------------------------------------------------------------------------#
    axes.plot(x,y, "-b")
    axes.plot(x,y, "oy")
    axes.set_xlabel("Beam length [m]")
    axes.set_ylabel(Name)
    axes.set_title(Title)
    axes.grid(which='major', color='#666666', linestyle='-')
#-------------------------------------------------------------------------------#
    axes.fill_between(x, 0, y, where = y <= 0,color='red', alpha=.1)
    axes.fill_between(x, 0, y, where = y >= 0,color='green', alpha=.1)
#_____________________________________<End>_____________________________________#

#===============================================================================#
#                                  < Diagrams >                                 #
#===============================================================================#
axes = plt.subplot(G[0:2, :])
G_Last = G_Last + 3
L_x = Fun.Input_Variable("L_x", Knot.Knot_Results)
#-------------------------------------------------------------------------------#
"""Defining Beam"""
x_def = L_x
y_def = L_x*0 + 0.06
x = np.reshape(x_def, len(x_def))
y_1 = np.reshape(y_def, len(y_def))
axes.plot(x,y_1, "black")
y_2 = -y_1
axes.plot(x,y_2, "black")
axes.fill_between(x, y_1, y_2, color="darkgray")
"""_____End_____"""

#===============================================================================#
#                              <Support Sort Functions>                         #
#===============================================================================#
def Support_Sort(v, h, r):
    Support_fix = []
    Support_f = []
    Support_v = []
    Support_h = []
```

Slika 4.71 – „Plot“ modul (2)

```
    Support_r = []
    for knot in range(len(v)):
        if (v[knot] == 1 and h[knot] == 1 and r[knot] == 1):
            Support_fix = np.append(Support_fix, knot)
        elif (v[knot] == 1 and h[knot] == 1):
            Support_f = np.append(Support_f, L_x[knot])
        elif v[knot] == 1:
            Support_v = np.append(Support_v, L_x[knot])
        elif h[knot] == 1:
            Support_h = np.append(Support_h, L_x[knot])
        elif r[knot] == 1:
            Support_r = np.append(Support_r, L_x[knot])
    return Support_fix, Support_f, Support_v, Support_h, Support_r
#_____________________________________<End>_____________________________________#
Support_fix,Support_f,Support_v,Support_h,Support_r = Support_Sort(Input.v_b,
                                                                  Input.h_b,
                                                                  Input.r_b)
#-------------------------------------------------------------------------------#
"""Define factors"""
plt.ylim([-1,1])
x_v = 0.02 * L_x[-1,0]
y_v = 0.2
x_h = 0.03 * L_x[-1,0]
y_h = 0.12
"""_____End_____"""

#===============================================================================#
#                                <Support Functions>                            #
#===============================================================================#
def Vertical_Support(support, Slider = "Yes"):
    for point in support:
        k_x = 1.5
        axes.plot(point, 0, "or")
        x = [point - x_v , point, point + x_v]
        y = [    - y_v   ,   0  ,    - y_v   ]
        axes.plot(x,y, "-r")
        x = [point - (x_v * k_x), point + (x_v * k_x)]
        y = [      - y_v        ,       - y_v        ]
        axes.plot(x,y, "-r")
        if Slider == "Yes":
            k_y = 1.2
            y = [   - y_v * k_y   ,   - y_v * k_y    ]
            axes.plot(x,y, "-r")
        else:
            k_y = 1
        n = 5
        x_n = (x[1] - x[0])/(n+1)
        k = 1.4
        for i in range(n):
            x_start = point - (x_v * k_x)
            y_start = -y_v * k_y * k
            x = [x_start + (i*x_n),x_start + ((i+1)*x_n)]
            y = [     y_start     ,     -y_v * k_y      ]
            axes.plot(x,y, "-r")
#-------------------------------------------------------------------------------#
def Horizontal_Support(support, Slider = "Yes"):
    for point in support:
        if point == 0:
            x_H = -x_h
        else:
            x_H = x_h
```

Slika 4.72 – „Plot“ modul (3)

```
            k_y = 1.5
            axes.plot(point, 0, "or")
            x = [point + x_H, point, point + x_H]
            y = [    -y_h   ,   0  ,     y_h    ]
            axes.plot(x,y, "-r")
            x = [point + x_H, point + x_H]
            y = [ -y_h * k_y,  y_h * k_y ]
            axes.plot(x,y, "-r")
            if Slider == "Yes":
                k_x = 1.25
                x = [ point + x_H * k_x, point + x_H * k_x ]
                axes.plot(x,y, "-r")
            else:
                k_y = 1
            n = 5
            y_n = (y[1] - y[0])/(n+1)
            k = 1.5
            for i in range(n):
                x_start = point + (x_H * k_x * k)
                y_start = -y_h * k_y
                x = [       x_start    ,   point + x_H * k_x  ]
                y = [ y_start + (i*y_n), y_start + ((i+1)*y_n)]
                axes.plot(x,y, "-r")
    #-------------------------------------------------------------------------------#
    def Fix_Support(support):
        for point in support:
            if point == 0:
                x_H = -x_h
            else:
                x_H = x_h

            k_y = 1.5
            x = [   point   ,   point    ]
            y = [ -y_h * k_y,  y_h * k_y ]
            axes.plot(x,y, "-r")
            n = 5
            y_n = (y[1] - y[0])/(n+1)
            k = 1
            for i in range(n):
                x_start = point + (x_H * k)
                y_start = -y_h * k_y
                x = [     x_start      ,         point        ]
                y = [ y_start + (i*y_n), y_start + ((i+1)*y_n)]
                axes.plot(x,y, "-r")
    #_____________________________________<End>_____________________________________#

    """Defining Loads"""
    F_v = Input.F_v
    F_h = Input.F_h
    M = Input.M
    q = Input.q
    """_____End_____"""

    #===============================================================================#
    #                                  <Load Functions>                             #
    #===============================================================================#
    def Vertical_Force(Force):
        for point in range(len(Force)):
            k_x= 0.4
            if Force[point] < 0:
```

Slika 4.73 – „Plot“ modul (4)

```
            k = 1
            k_y = 0.8
        elif Force[point] > 0:
            k = -1
            k_y = -0.8
        if Force[point] != 0:
            axes.plot(L_x[point], 0, "og")
            x = [L_x[point]- x_v*k_x, L_x[point], L_x[point]+ x_v*k_x]
            y = [      y_v * k_y    ,      0    ,       y_v * k_y    ]
            axes.plot(x,y, "-g")
            x = [L_x[point], L_x[point]]
            y = [    0     ,    0.8 * k    ]
            axes.plot(x,y, "-g")
            text = str(round(Force[point,0]/1000,2)) + " kN"
            plt.text(L_x[point]+x_v*0.3, 0.7*k, text , fontsize = 10)
#-------------------------------------------------------------------------------#
def Horizontal_Force(Force):
    for point in range(len(Force)):
        k_y= 0.3
        if Force[point] < 0:
            k = -4
            k_x = -1.2
            k_t = -1
        elif Force[point] > 0:
            k = 4
            k_x = 1.2
            k_t = 4
        if Force[point] != 0:
            axes.plot(L_x[point], 0, "og")
            x = [L_x[point]- x_v*k_x, L_x[point], L_x[point]- x_v*k_x]
            y = [     -y_v * k_y    ,      0    ,       y_v * k_y    ]
            axes.plot(x,y, "-g")
            x = [L_x[point]- x_v*k, L_x[point]]
            y = [          0      ,     0     ]
            axes.plot(x,y, "-g")
            text = str(round(Force[point,0]/1000,2)) + " kN"
            plt.text(L_x[point]-x_v*k_t, 0.1, text , fontsize = 10)
#-------------------------------------------------------------------------------#
def Bending_Moment(Moment):
    for point in range(len(Moment)):
        if Moment[point] != 0:
            if Moment[point] < 0:
                start = -np.pi
                end = 1
                factor = -1
                x_arrow1 = -0.5 * x_v
                y_arrow1 =  1.1 * y_h
                x_arrow2 = -1.2 * x_v
                y_arrow2 =  0.3 * y_h

            if Moment[point] > 0:
                start = -1
                end = np.pi
                factor = 0
                x_arrow1 =  0.5 * x_v
                y_arrow1 =  1.1 * y_h
                x_arrow2 =  1.2 * x_v
                y_arrow2 =  0.3 * y_h

            r = 0.2
            plt.plot(L_x[point],0,"ob")
```

Slika 4.74 – „Plot“ modul (5)

```
            t = np.arange(start, end, 0.1)

            x_end = r * np.sin(t) * 0.17 * (L_x[-1]) + L_x[point]
            y_end = r * np.cos(t)
            plt.plot(x_end,y_end,"b",linewidth = 2)

            x = [x_end[factor],x_end[factor] + x_arrow1]
            y = [y_end[factor],y_end[factor] + y_arrow1]
            plt.plot(x,y,"b",linewidth = 2)

            x = [x_end[factor],x_end[factor] + x_arrow2]
            y = [y_end[factor],y_end[factor] + y_arrow2]
            plt.plot(x,y,"b",linewidth = 2)
            text = str(round(Moment[point,0]/1000,2)) + " kN"
            plt.text(L_x[point]-x_v*1.3, 0.1+r, text , fontsize = 10)

#-------------------------------------------------------------------------------#
def Distributed_load(q):
    Count = 0
    Min = 0.2
    Max = 0.6
    Max_q = np.max(abs(q))

    for point in range(len(q)):
        k_def = (abs(q[point,0])/Max_q)*(Max-Min)+Min
        if q[point] < 0:
            k = k_def
            k_t = k + 0.05
            show = 0.06
        elif q[point] > 0:
            k = -k_def
            k_t = k - 0.12
            show = -0.06
        if q[point] != 0:
            x = [L_x[point,0], L_x[point+1,0]]
            y = [     k     ,       k      ]
            axes.plot(x,y, "black","-")
            axes.fill_between(x, show, y, color='pink')
            Count += 1
            if (point == len(q)-1 or q[point] != q[point+1]):
                text = str(round(q[point,0]/1000,2)) + " kN"
                x = L_x[point+1] - ((L_x[point+1]-L_x[point])*Count)/2 - x_v
                plt.text(x, k_t, text , fontsize = 10)
                Count = 0
#_____________________________________<End>_____________________________________#

#===============================================================================#
#                                     <Plot>                                    #
#===============================================================================#
Vertical_Support(Support_v)
Vertical_Support(Support_f, Slider="No")
Horizontal_Support(Support_h)
Fix_Support(Support_fix)
#-------------------------------------------------------------------------------#
Vertical_Force(F_v)
Horizontal_Force(F_h)
Bending_Moment(M)
Distributed_load(q)
#-------------------------------------------------------------------------------#
axes.get_yaxis().set_ticks([])
axes.set_xlabel("Beam length [m]")
```

Slika 4.75 – „Plot“ modul (6)

```
axes.set_title("Beam Calculation")
#_____________________________________<End>_____________________________________#

Force("Shear_Force [kN]", "F_v1", "F_v2", 0.001, "Shear_Force", Process="Yes")
Force("Axial_Force [kN]", "F_h1", "F_h2", 0.001, "Axial_Force")
Force("Bending moment [kNm]", "M_1", "M_2", 0.001, "Bending moment")
Deflection("Deflection [mm]", "v", 1000, "Vertical beam deflection")
Deflection("Displacement [mm]", "h", 1000, "Horizontal beam displacement")
Deflection("Deflection [deg]", "r", (180/np.pi), "Rotation beam deflection")
#_____________________________________< End >_____________________________________#
File_Location = Data.Input_Directory("Diagrams.png")
plt.savefig(File_Location, dpi=300, bbox_inches = 'tight')
#.....................................< End >.....................................#
```

Slika 4.76 – „Plot“ modul (7)

4.5.9. „Output“ modul

Modul „Output“ (*Slika 4.80*) služi za ispis rezultata u Excel datoteku. Za to koristi module kao što su „Input_Directory“ za određivanje lokacije Excel datoteke, „Knot_Results“ za preuzimanje podataka o čvorovima, te „Element_Results“ iz kojeg uzima podatke o elementima. Ti se podaci upisuju na zadane listove unutar Excel datoteke. Na sljedećim slikama možete vidjeti primjere ispisa rezultata iz Excel datoteke (*Slika 4.77*, *Slika 4.78*, *Slika 4.79*).

Knots	Length	Vertical displacement	Horizontal displacement	Rotation displacement	Vertical force	Horizontal force	Moment	Vertical support reaction	Horizontal support reaction	Moment support reaction
1.4	0,75 [m]	0,217 [mm]	0,007 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.5	1,00 [m]	0,332 [mm]	0,009 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.6	1,25 [m]	0,434 [mm]	0,011 [mm]	0,020 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.7	1,50 [m]	0,501 [mm]	0,013 [mm]	0,010 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.8	1,75 [m]	0,513 [mm]	0,015 [mm]	-0,005 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.9	2,00 [m]	0,448 [mm]	0,018 [mm]	-0,025 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.10	2,25 [m]	0,284 [mm]	0,020 [mm]	-0,050 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.1	2,50 [m]	0,000 [mm]	0,022 [mm]	-0,081 [deg]	34,94 [kN]	0,00 [kN]	0,00 [kNm]	34,94 [kN]	N/A	N/A
2.2	2,75 [m]	-0,421 [mm]	0,024 [mm]	-0,112 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.3	3,00 [m]	-0,976 [mm]	0,027 [mm]	-0,142 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.4	3,25 [m]	-1,656 [mm]	0,029 [mm]	-0,169 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A

Slika 4.77 – Izgled rezultata čvorova unutar Excel datoteke

Points	Fv1	Fh1	M1	Fv2	Fh2	M2
1.1	-22,44 [kN]	10,00 [kN]	19,22 [kNm]	22,69 [kN]	10,00 [kN]	13,58 [kNm]
1.2	-22,69 [kN]	10,00 [kN]	13,58 [kNm]	22,94 [kN]	10,00 [kN]	7,87 [kNm]
1.3	-22,94 [kN]	10,00 [kN]	7,87 [kNm]	23,19 [kN]	10,00 [kN]	2,11 [kNm]
1.4	-23,19 [kN]	10,00 [kN]	2,11 [kNm]	23,44 [kN]	10,00 [kN]	-3,72 [kNm]
1.5	-23,44 [kN]	10,00 [kN]	-3,72 [kNm]	23,69 [kN]	10,00 [kN]	-9,61 [kNm]
1.6	-23,69 [kN]	10,00 [kN]	-9,61 [kNm]	23,94 [kN]	10,00 [kN]	-15,56 [kNm]
1.7	-23,94 [kN]	10,00 [kN]	-15,56 [kNm]	24,19 [kN]	10,00 [kN]	-21,58 [kNm]
1.8	-24,19 [kN]	10,00 [kN]	-21,58 [kNm]	24,44 [kN]	10,00 [kN]	-27,66 [kNm]
1.9	-24,44 [kN]	10,00 [kN]	-27,66 [kNm]	24,69 [kN]	10,00 [kN]	-33,80 [kNm]
1.10	-24,69 [kN]	10,00 [kN]	-33,80 [kNm]	24,94 [kN]	10,00 [kN]	-40,00 [kNm]
2.1	10,00 [kN]	10,00 [kN]	-40,00 [kNm]	-10,00 [kN]	10,00 [kN]	-37,50 [kNm]

Slika 4.78 – Izgled rezultata elemenata unutar Excel datoteke

Name:	Value:
Vertical Displacement	-43,195 [mm]
Horizontal Displacement	0,088 [mm]
Rotation Displacement	-0,407 [deg]
Shear Force	-24,94 [kN]
Tension Force	10,00 [kN]
Compression Force	N/A
Bending Moment	-40,00 [kNm]

Slika 4.79 – Izgled maksimalnih vrijednosti unutar Excel datoteke

```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Output >                                   #
#...........................................................................#

"""Python libraries. """
import numpy as np

#===========================================================================#

"""User Function. """
import _1_Excel_Data as Excel
import _2_Function as Fun

#===========================================================================#

"""Code sections. """
import Input_Directory as Input
import Knot_Results as Knot
import Element_Results as Ele

#================================ Calculation ==============================#

Excel.Output(Input.Output_Excel, "Element Results", Ele.Beam_Element, 0.001)
#---------------------------------------------------------------------------#
Excel.Output(Input.Output_Excel, "Knot Results", Knot.Knot_Results[:,0:2], 1)
Excel.Output(Input.Output_Excel, "Knot Results", Knot.Knot_Results[:,2:4],
             1000, Column_Start = 3, Process = "No")
Excel.Output(Input.Output_Excel, "Knot Results", Knot.Knot_Results[:,4:5],
              (180/np.pi), Column_Start = 5, Process = "No")
Excel.Output(Input.Output_Excel, "Knot Results", Knot.Knot_Results[:,5:],
              0.001, Column_Start = 6, Process = "No")
#---------------------------------------------------------------------------#
Summary_Matrix = np.vstack((Fun.Title("Name", "Value"),
                            Knot.Max_Table,
                            Ele.Max_Table))
Excel.Output(Input.Output_Excel, "Summary", Summary_Matrix, 1,
             Row_Start=4, Column_Start = 3 )

#..................................< End >..................................#
```

Slika 4.80 – „Output“ modul

4.5.10. „Calculation“ modul

Modul „Calculation“ (*Slika 4.82*) služi za praćenje slijeda operacija i mjerenje vremena trajanja pojedinih dijelova programa. To omogućava korisnicima da prate koliko dugo svaka funkcija ili sekcija koda traje. Modul koristi funkciju „Import“ koja bilježi početak izvođenja svake sekcije i prikazuje trajanje tih operacija na terminalu. Osim toga, „Calculation“ modul prikazuje glavne rezultate programa kako bi korisnici mogli provjeriti ispravnost podataka. Na sljedećoj slici možete vidjeti ispis u Spyder konzoli koji prikazuje vrijeme izvršavanja pojedinih sekcija (vrijeme je prikazano u sekundama).

```
Input - Start
Input - Done ( 0.05 )
Beam_Characteristics - Start
Beam_Characteristics - Done ( 0.0 )
Beam_Load - Start
Beam_Load - Done ( 0.01 )
Displacement - Start
Displacement - Done ( 0.0 )
Knot_Results - Start
Knot_Results - Done ( 0.01 )
Element_Results - Start
Element_Results - Done ( 0.0 )
Output - Start
Output - Done ( 10.9 )
```

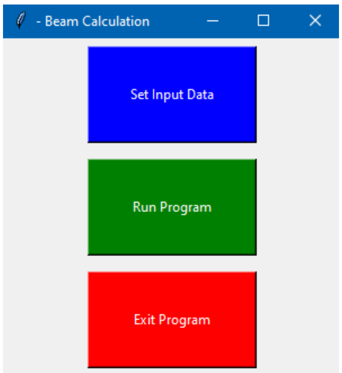
Slika 4.81 – Izgled konzole nakon izvršenja „Calculation“ modula

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                < Caculation >                              #
#............................................................................#

"""Python libraries. """
import time
import importlib

#============================================================================#
#                              <Import Function>                             #
#============================================================================#
def Import(Name, New_Name = "Default"):
    if New_Name == "Default":
        New_Name = Name
    Time_Start = time.time()
    print(New_Name, "- Start")
    #------------------------------------------------------------------------#
    importlib.import_module(Name)
    #------------------------------------------------------------------------#
    Time_End = time.time()
    Time = Time_End - Time_Start
    print(New_Name, "- Done (", round(Time, 2), ")")
#_____________________________________<End>__________________________________#

#================================= Calculation ==============================#

Import("Input")
Import("Beam_Characteristics")
Import("Beam_Load")
Import("Displacement")
Import("Knot_Results")
Import("Element_Results")
Import("Output")
import Plot
#----------------------------------------------------------------------------#
import Knot_Results
Knot = Knot_Results.Knot_Results
import Element_Results
Element = Element_Results.Beam_Element

#.................................< End >....................................#
```

Slika 4.82 – „Calculation“ modul

4.5.11. „Start Box“ modul

Modul „Start Box“ (*Slika 4.84*, *Slika 4.85*, *Slika 4.86*) služi za kreiranje grafičkog sučelja koristeći Tkinter. Sučelje (*Slika 4.83*) sadrži tri gumba: „Set Input Dana“, „Run Program“ i „Exit Program“. Svaki gumb omogućuje korisniku da pokrene specifične radnje, kao što su postavljanje ulaznih podataka, pokretanje programa ili izlazak iz aplikacije, uz prikaz odgovarajućih potvrda putem poruka.



Slika 4.83 – Izgled grafičkog sučelja

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                                 < Start Box >                              #
#............................................................................#

"""Python libraries. """
import tkinter as tk

#================================= Calculation ==============================#

root = tk.Tk(className="- Beam Calculation")
#----------------------------------------------------------------------------#
Screen_Width = root.winfo_screenwidth()
Screen_Height = root.winfo_screenmmheight()
#----------------------------------------------------------------------------#
Box_Width = 300
Box_Height = 300
#----------------------------------------------------------------------------#
Width_Position = (Screen_Width/2)-(Box_Width/2)
Height_Position = (Screen_Height/2)-(Box_Height/2)+200
#----------------------------------------------------------------------------#
root.geometry("%dx%d+%d+%d" % (Box_Width, Box_Height,
                                Width_Position, Height_Position))
#----------------------------------------------------------------------------#
canvas = tk.Canvas(root, width=Box_Width, height=Box_Height, )
canvas.pack()
#============================================================================#
def set_input():
    msg_box = tk.messagebox.askquestion("Set inputs",
                                    "Are you sure you want to set input data?",
                                    icon="warning",)

    if msg_box == "yes":
        global Process
        Process = "Input"
        root.destroy()
    else:
        tk.messagebox.showinfo("Return",
                               "You will now return to the program screen")
    return Process
#============================================================================#
def run_application():
```

Slika 4.84 – „Start Box“ modul (1)

```
    msg_box = tk.messagebox.askquestion("Run Program",
                                        "Are you sure you want to Run the program, Excel file will be closed?",
                                        icon="warning",)
    if msg_box == "yes":
        global Process
        Process = "Yes"
        root.destroy()
    else:
        tk.messagebox.showinfo("Return",
                               "You will now return to the application screen")
    return Process
#=============================================================================#
def exit_application():
    msg_box = tk.messagebox.askquestion("Exit Program",
                                        "Are you sure you want to exit the program?",
                                        icon="warning",)
    if msg_box == "yes":
        global Process
        Process = "No"
        root.destroy()
    else:
        tk.messagebox.showinfo("Return", "You will now return to the program screen")

#=============================================================================#
button = tk.Button(root, text="Set Input Data",
                   command = set_input,
                   bg="blue", fg="white",
                   height=5, width=20)
canvas.create_window(150, 50, window=button)
#-----------------------------------------------------------------------------#
button = tk.Button(root, text="Run Program",
                   command = run_application,
                   bg="green", fg="white",
                   height=5, width=20)
canvas.create_window(150, 150, window=button)
#-----------------------------------------------------------------------------#
button = tk.Button(root, text="Exit Program",
                   command = exit_application,
                   bg="red", fg="white",
                   height=5, width=20)
```

Slika 4.85 – „Start Box“ modul (2)

```
canvas.create_window(150, 250, window=button)
#-----------------------------------------------------------------------------#
root.mainloop()

#.................................< End >.................................#
```

Slika 4.86 – „Start Box“ modul (3)

4.5.12. „Global System“ modul

Modul „Global System“ (*Slika 4.87, Slika 4.88*) upravlja svim operacijama vezanim uz vanjske datoteke i osigurava da se svi potrebni procesi, kao što su otvaranje, spremanje i zatvaranje datoteka, pravilno izvode. Također, ovaj modul prati radnje programa i ispisuje ih na terminalu, čime omogućava korisnicima praćenje aktivnosti i potencijalnih problema.

Ako program naiđe na poteškoće pri izvršavanju kalkulacija, „Global System“ će ispisati upozorenje na konzoli. Ova obavijest može pomoći u identifikaciji problema, bilo da se radi o grešci u kodu ili o performansama računala na kojem se program izvodi. Ovisno o vrsti upozorenja, korisnici mogu lakše locirati uzrok problema i poduzeti odgovarajuće korake za njegovo rješavanje.

```
#''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                              < Global System >                                      #
#....................................................................................#

"""Python libraries. """
import time
import os
import xlwings as xw
from IPython import get_ipython
import importlib

#================================ Calculation ===============================#

import Start_Box as Start
Process = Start.Process

#============================================================================#
#                    <Import Input and Output file locations>                #
#============================================================================#
import Input_Directory
Input_Excel_Directory = Input_Directory.Input_Excel
Output_Excel_Directory = Input_Directory.Output_Excel
Output_Excel = Input_Directory.Output
#___________________________________<End>___________________________________#

if Process == "Input":
    os.startfile(Input_Excel_Directory)
    importlib.reload(Start)
    Process = Start.Process
#============================================================================#
if Process == "Yes":
    try:
        book = xw.Book(Output_Excel)
        book.close()
        print("Book ", Output_Excel, " is closed")
    except Exception as e:
        print(e)
    #------------------------------------------------------------------------#
    for i in range(3):
        print('\r'," ...", 3-i, end='')
        time.sleep(1)
    print('\r', " " * i, end='')
    #------------------------------------------------------------------------#
    wb = xw.Book(Input_Excel_Directory)
    wb.save()
    print('\r',end='')
    print("\rBook ", Input_Excel_Directory," is saved")
    #------------------------------------------------------------------------#
    Time_Start = time.time()
    print("Calculation - Start")
    try:
        import Calculation
        Global_System = "Done"
    except:
        print("The program has been terminated; the input data is not valid.")
        print("Please check them!")
        Global_System = "Terminated"
    Time_End = time.time()
    Time = Time_End - Time_Start
    print("Calculation - Done (", round(Time, 2), ")")
    #------------------------------------------------------------------------#
```

Slika 4.87 – „Global System“ modul (1)

```
    os.startfile(Output_Excel_Directory)
    for i in range(1000):
        try:
            with open(Output_Excel_Directory, 'r+') as f:
                Check = "Closed"
                print(".", end='')
        except:
            Check = "Open"
        time.sleep(0.5)
        if Check == "Open":
            print('\r'," " * i, end='')
            break
    #--------------------------------------------------------------------------#
    Clear_Variable = "Yes"
    if Clear_Variable == "Yes":
        get_ipython().run_line_magic('reset', '-sf')
    print('\r',end='')
    print("Program is finish")

#============================================================================#
else:
    Global_System = "Closed"
    print("Program is closed")

#.................................< End >.....................................#
```

Slika 4.88 – „Global System“ modul (2)

4.5.13. „Beam Calculation“ modul

Modul „Beam Calculation“ (*Slika 4.89*) služi za nadzor i kontrolu programa kako bi se osigurala osnovna funkcionalnost i prikaz osnovnih varijabli. Modul funkcionira na sljedeći način:

1. Pokretanje Modula „Global System“: Modul „Beam Calculation“ pokreće „Global System“, koji upravlja svim operacijama vezanim uz datoteke i prati radnje programa.
2. Prikaz Rezultata: Ako se „Global System“ uspješno izvrši, „Beam Calculation“ prikazuje rezultate iz modula „Calculation“ na prozoru. Ovi rezultati uključuju osnovne varijable i informacije o izvedbi.
3. U slučaju Greške: Ako tijekom izvođenja nastane greška, „Beam Calculation“ će prikazati samo varijablu „Error“ u prozoru za varijable. Ova informacija pomaže korisnicima da prepoznaju da je došlo do problema u „Global System“ modulu.

Ovaj pristup omogućava brz pregled osnovnih varijabli i statusa programa, dok istovremeno pruža jednostavan način za prepoznavanje i dijagnosticiranje grešaka.

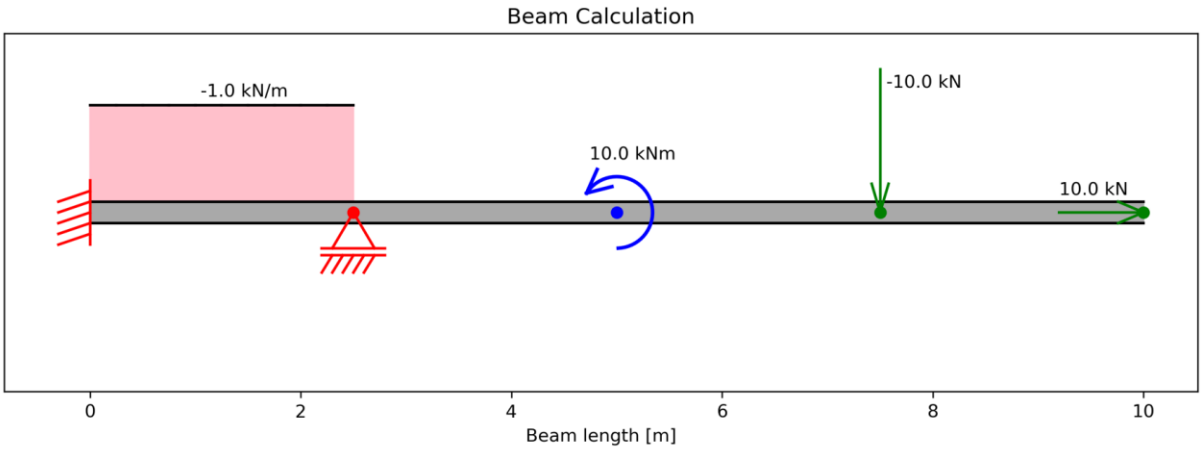
```
#'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''#
#                        < Beam Calculation >                      #
#.................................................................#

#=============================== Calculation ===============================#

import Global_System
if Global_System.Global_System == "Done":
    import Calculation
    Knot = Calculation.Knot
    Element = Calculation.Element
if Global_System.Global_System == "Terminated":
    Error = "Data is not valid"

#................................< End >................................#
```

Slika 4.89 – „Beam Calculation“ modul

5. Rješavanje zadanog problema korištenjem Python programa

5.1. Određivanje ulaznih parametara

Zadani problem obuhvaća analizu grede IPE 300 na duljini od 10 metara sa zadanim opterećenjima koji se mogu vidjeti na sljedećoj slici (*Slika 5.1*).



Slika 5.1 – Zadani problem

Specifikacije IPE 300 profila se mogu vidjeti na sljedećoj slici (*Slika 5.2*).

		Profile dimensions					Area properties					Inertia properties about major axis y-y				Inertia proper	
Profile	Drawing	Depth h [mm]	Width b [mm]	Web thickness t_w [mm]	Flange thickness t_f [mm]	Root radius r [mm]	Weight m [kg/m]	Perimeter P [m]	Area A [mm^2]	Shear area z-z $A_{v,z}$ [mm^2] (for η=1.2)	Shear area y-y $A_{v,y}$ [mm^2]	Second moment of area I_y [$\times 10^6$ mm^4]	Radius of gyration i_y [mm]	Elastic section modulus $W_{el,y}$ [$\times 10^3$ mm^3]	Plastic section modulus $W_{pl,y}$ [$\times 10^3$ mm^3]	Second moment of area I_z [$\times 10^6$ mm^4]	Radi of gyrat i_z [mn
IPE200	dxf	200	100	5.6	8.5	12	22.4	0.768	2848	1400	1700	19.43	82.6	194.3	220.6	1.424	22.
IPE220	dxf	220	110	5.9	9.2	12	26.2	0.848	3337	1588	2024	27.72	91.1	252.0	285.4	2.049	24.
IPE240	dxf	240	120	6.2	9.8	15	30.7	0.922	3912	1914	2352	38.92	99.7	324.3	366.6	2.836	26.
IPE270	dxf	270	135	6.6	10.2	15	36.1	1.041	4595	2214	2754	57.90	112.3	428.9	484.0	4.199	30.
IPE300	dxf	300	150	7.1	10.7	15	42.2	1.160	5381	2568	3210	83.56	124.6	557.1	628.4	6.038	33.
IPE330	dxf	330	160	7.5	11.5	18	49.1	1.254	6261	3081	3680	117.7	137.1	713.1	804.3	7.881	35.
IPE360	dxf	360	170	8.0	12.7	18	57.1	1.353	7273	3514	4318	162.7	149.5	903.6	1019	10.43	37.
IPE400	dxf	400	180	8.6	13.5	21	66.3	1.467	8446	4269	4860	231.3	165.5	1156	1307	13.18	39.
IPE450	dxf	450	190	9.4	14.6	21	77.6	1.605	9882	5085	5548	337.4	184.8	1500	1702	16.76	41.
IPE500	dxf	500	200	10.2	16.0	21	90.7	1.744	11552	5987	6400	482.0	204.3	1928	2194	21.42	43.

Slika 5.2 – Specifikacije IPE 300 grede

5.2. Upisivanje ulaznih podataka unutar Excel datoteke

Podaci se unose u program tako da se na mjestima gdje se nalaze koncentrirana opterećenja, oslonci ili početak/kraj kontinuiranog opterećenja postavljaju čvorovi. U ovom primjeru, greda ima pet točaka promjene opterećenja, pa se na „Knot“ listu u tablici za čvorove (*Slika 5.3*) dodaju pet redaka (čvorova).

Nakon toga, unose se oslonci na odgovarajuće lokacije. Također je potrebno unijeti zadane sile na specifične čvorove, uključujući kut pod kojim sile djeluju. Veličina opterećenja u vertikalnom i horizontalnom smjeru upisuje se u dva različita stupca. Na kraju, potrebno je unijeti momente u čvorovima. Svi stupci označeni svijetlo plavom bojom moraju biti ispunjeni kako bi program mogao pravilno funkcionirati.

Point	Support	Vertical boundary	Horizontal boundary	Rotation boundary	Force	Angle	Vertical force	Horizontal force	Moment
1	Fixed	Fix	Fix	Fix	0,00 [kN]	0 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]
2	Vertical support	Fix	Free	Free	0,00 [kN]	0 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]
3	None	Free	Free	Free	0,00 [kN]	0 [deg]	0,00 [kN]	0,00 [kN]	10,00 [kNm]
4	None	Free	Free	Free	10,00 [kN]	270 [deg]	-10,00 [kN]	0,00 [kN]	0,00 [kNm]
5	None	Free	Free	Free	10,00 [kN]	0 [deg]	0,00 [kN]	10,00 [kN]	0,00 [kNm]

Slika 5.3 – ulazni podaci o čvorovima

Nakon što su svi čvorovi uneseni, prelazi se na „Element“ list te ispunjava tablicu (*Slika 5.4*) kako bi se definirala svojstva svakog elementa.

1. Modul elastičnosti i duljina: prvo, za svaki element upisuje se modul elastičnosti i duljina elementa.
2. Kontinuirano opterećenje: ako element ima kontinuirano opterećenje, to se također unosi na ovom listu.
3. Podelementi: nakon što su osnovne karakteristike unijete, može se definirati broj podelememata za svaki element. Veći broj podelemenata može poboljšati točnost rezultata, dok premali broj može dovesti do grešaka u rezultatima. Međutim, preveliki broj može nepotrebno produžiti vrijeme kalkulacije.
4. Poprečni presjek: dalje, potrebno je unijeti dimenzije poprečnog presjeka elementa, uključujući visinu, širinu i debljinu za kvadratni (pravokutni) poprečni presjek.
5. Moment inercije i poprečni presjek: ako se koristi drugi oblik presjeka, umjesto dimenzija, unose se podaci o momentu inercije i poprečnom presjeku. U tom slučaju, program će koristiti ove nove podatke umjesto prethodno unesenih dimenzija.

Element No.	E	Lenght	q	Number of elements	Inertia	Cross section	Height	Width	Thickness
1	210 [GPa]	2,50 [m]	-1,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**	400,00 [mm]	180,00 [mm]	8,60 [mm]
2	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**	400,00 [mm]	180,00 [mm]	8,60 [mm]
3	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**	400,00 [mm]	180,00 [mm]	8,60 [mm]
4	210 [GPa]	2,50 [m]	0,00 [kN/m]	10,00 [ul]	**8356,00 [cm4]**	**53,81 [cm2]**	400,00 [mm]	180,00 [mm]	8,60 [mm]

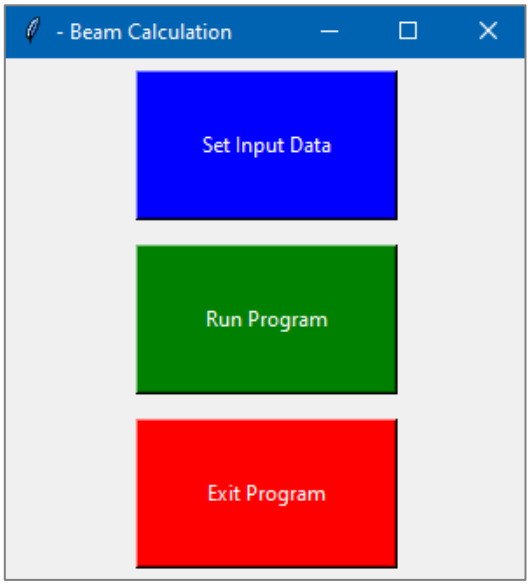
Slika 5.4 – Ulazni podaci o elementima

5.3. Pokretanje programa

Kada se program pokreće unutar Spyder editora, prikazuje se prozorčić (*Slika 5.5*) s tri osnovne funkcije:

1. „Set Input Data“: ova opcija omogućuje korisniku da unese ili ažurira ulazne podatke za program. Ako su ulazni podaci već zadani i ažurirani, ovaj korak može biti preskočen.
2. „Run Program“: kada se klikne na ovaj gumb, program se pokreće i počinje s obradom podataka. Nakon završetka obrade, program automatski otvara Excel datoteku koja sadrži sve rezultate analize.
3. „Exit Program“: ova opcija omogućuje korisniku da izađe iz programa bez pokretanja analize ili promjene ulaznih podataka.

Na taj način, korisnici mogu jednostavno upravljati ulaznim podacima i pokretati analizu unutar Spyder editora, s automatskim generiranjem i otvaranjem Excel datoteke s rezultatima nakon završetka obrade.



Slika 5.5 - Izbornik

5.4. Prikaz rezultata

Rezultati analize mogu se pregledati u Excel datoteci koja sadrži tri odvojena lista:

1. List „Knot Results“: ovaj list prikazuje tablicu (*Slika 5.6*) sa rezultatima za svaki čvor, uključujući pomake, naprezanja, i druge relevantne podatke. Podaci su organizirani u tablici koja omogućuje jednostavno pregledavanje i sortiranje prema maksimalnim, minimalnim vrijednostima itd.
2. List „Element Results“: ovaj list prikazuje tablicu (*Slika 5.7*) sa rezultatima za svaki element grede, uključujući opterećenja na početku i kraju svakog elementa te maksimalne vrijednosti elemenata. Kao i kod liste za čvorove, podaci su prikazani u tablici koja omogućuje pregled i sortiranje.
3. List „Summary“: ovaj list prikazuje tablicu (*Slika 5.8*) koja sadrži samo sažete podatke, fokusirajući se na maksimalne vrijednosti tijekom analize. Korisnici mogu lako pregledati ključne maksimalne rezultate bez potrebe za detaljnim pregledom svih podataka.

Ove tri liste omogućuju sveobuhvatan uvid u rezultate analize, omogućavajući korisnicima da brzo pronađu i analiziraju ključne informacije.

Knots	Length	Vertical displacement	Horizontal displacement	Rotation displacement	Vertical force	Horizontal force	Moment	Vertical support reaction	Horizontal support reaction	Moment support reaction
1.1	0,00 [m]	0,000 [mm]	0,000 [mm]	0,000 [deg]	-22,44 [kN]	-10,00 [kN]	-19,22 [kNm]	-22,44 [kN]	-10,00 [kN]	-19,22 [kNm]
1.2	0,25 [m]	0,031 [mm]	0,002 [mm]	0,013 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.3	0,50 [m]	0,110 [mm]	0,004 [mm]	0,022 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.4	0,75 [m]	0,217 [mm]	0,007 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.5	1,00 [m]	0,332 [mm]	0,009 [mm]	0,026 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.6	1,25 [m]	0,434 [mm]	0,011 [mm]	0,020 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.7	1,50 [m]	0,501 [mm]	0,013 [mm]	0,010 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.8	1,75 [m]	0,513 [mm]	0,015 [mm]	-0,005 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.9	2,00 [m]	0,448 [mm]	0,018 [mm]	-0,025 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
1.10	2,25 [m]	0,284 [mm]	0,020 [mm]	-0,050 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.1	2,50 [m]	0,000 [mm]	0,022 [mm]	-0,081 [deg]	34,94 [kN]	0,00 [kN]	0,00 [kNm]	34,94 [kN]	N/A	N/A
2.2	2,75 [m]	-0,421 [mm]	0,024 [mm]	-0,112 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.3	3,00 [m]	-0,976 [mm]	0,027 [mm]	-0,142 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.4	3,25 [m]	-1,656 [mm]	0,029 [mm]	-0,169 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.5	3,50 [m]	-2,451 [mm]	0,031 [mm]	-0,195 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.6	3,75 [m]	-3,353 [mm]	0,033 [mm]	-0,218 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.7	4,00 [m]	-4,353 [mm]	0,035 [mm]	-0,240 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.8	4,25 [m]	-5,442 [mm]	0,038 [mm]	-0,259 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.9	4,50 [m]	-6,611 [mm]	0,040 [mm]	-0,276 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
2.10	4,75 [m]	-7,852 [mm]	0,042 [mm]	-0,292 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.1	5,00 [m]	-9,155 [mm]	0,044 [mm]	-0,305 [deg]	0,00 [kN]	0,00 [kN]	10,00 [kNm]	N/A	N/A	N/A
3.2	5,25 [m]	-10,529 [mm]	0,046 [mm]	-0,324 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.3	5,50 [m]	-11,983 [mm]	0,049 [mm]	-0,342 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.4	5,75 [m]	-13,508 [mm]	0,051 [mm]	-0,357 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.5	6,00 [m]	-15,096 [mm]	0,053 [mm]	-0,370 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.6	6,25 [m]	-16,737 [mm]	0,055 [mm]	-0,382 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.7	6,50 [m]	-18,423 [mm]	0,058 [mm]	-0,391 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.8	6,75 [m]	-20,144 [mm]	0,060 [mm]	-0,398 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.9	7,00 [m]	-21,892 [mm]	0,062 [mm]	-0,403 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
3.10	7,25 [m]	-23,658 [mm]	0,064 [mm]	-0,406 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.1	7,50 [m]	-25,433 [mm]	0,066 [mm]	-0,407 [deg]	-10,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.2	7,75 [m]	-27,209 [mm]	0,069 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.3	8,00 [m]	-28,986 [mm]	0,071 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.4	8,25 [m]	-30,762 [mm]	0,073 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.5	8,50 [m]	-32,538 [mm]	0,075 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.6	8,75 [m]	-34,314 [mm]	0,077 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.7	9,00 [m]	-36,090 [mm]	0,080 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.8	9,25 [m]	-37,867 [mm]	0,082 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.9	9,50 [m]	-39,643 [mm]	0,084 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
4.10	9,75 [m]	-41,419 [mm]	0,086 [mm]	-0,407 [deg]	0,00 [kN]	0,00 [kN]	0,00 [kNm]	N/A	N/A	N/A
5.1	10,00 [m]	-43,195 [mm]	0,088 [mm]	-0,407 [deg]	0,00 [kN]	10,00 [kN]	0,00 [kNm]	N/A	N/A	N/A

Slika 5.6 – Rezultati čvorova

Points	Fv1	Fh1	M1	Fv2	Fh2	M2
1.1	-22,44 [kN]	10,00 [kN]	19,22 [kNm]	22,69 [kN]	10,00 [kN]	13,58 [kNm]
1.2	-22,69 [kN]	10,00 [kN]	13,58 [kNm]	22,94 [kN]	10,00 [kN]	7,87 [kNm]
1.3	-22,94 [kN]	10,00 [kN]	7,87 [kNm]	23,19 [kN]	10,00 [kN]	2,11 [kNm]
1.4	-23,19 [kN]	10,00 [kN]	2,11 [kNm]	23,44 [kN]	10,00 [kN]	-3,72 [kNm]
1.5	-23,44 [kN]	10,00 [kN]	-3,72 [kNm]	23,69 [kN]	10,00 [kN]	-9,61 [kNm]
1.6	-23,69 [kN]	10,00 [kN]	-9,61 [kNm]	23,94 [kN]	10,00 [kN]	-15,56 [kNm]
1.7	-23,94 [kN]	10,00 [kN]	-15,56 [kNm]	24,19 [kN]	10,00 [kN]	-21,58 [kNm]
1.8	-24,19 [kN]	10,00 [kN]	-21,58 [kNm]	24,44 [kN]	10,00 [kN]	-27,66 [kNm]
1.9	-24,44 [kN]	10,00 [kN]	-27,66 [kNm]	24,69 [kN]	10,00 [kN]	-33,80 [kNm]
1.10	-24,69 [kN]	10,00 [kN]	-33,80 [kNm]	24,94 [kN]	10,00 [kN]	-40,00 [kNm]
2.1	10,00 [kN]	10,00 [kN]	-40,00 [kNm]	-10,00 [kN]	10,00 [kN]	-37,50 [kNm]
2.2	10,00 [kN]	10,00 [kN]	-37,50 [kNm]	-10,00 [kN]	10,00 [kN]	-35,00 [kNm]
2.3	10,00 [kN]	10,00 [kN]	-35,00 [kNm]	-10,00 [kN]	10,00 [kN]	-32,50 [kNm]
2.4	10,00 [kN]	10,00 [kN]	-32,50 [kNm]	-10,00 [kN]	10,00 [kN]	-30,00 [kNm]
2.5	10,00 [kN]	10,00 [kN]	-30,00 [kNm]	-10,00 [kN]	10,00 [kN]	-27,50 [kNm]
2.6	10,00 [kN]	10,00 [kN]	-27,50 [kNm]	-10,00 [kN]	10,00 [kN]	-25,00 [kNm]
2.7	10,00 [kN]	10,00 [kN]	-25,00 [kNm]	-10,00 [kN]	10,00 [kN]	-22,50 [kNm]
2.8	10,00 [kN]	10,00 [kN]	-22,50 [kNm]	-10,00 [kN]	10,00 [kN]	-20,00 [kNm]
2.9	10,00 [kN]	10,00 [kN]	-20,00 [kNm]	-10,00 [kN]	10,00 [kN]	-17,50 [kNm]
2.10	10,00 [kN]	10,00 [kN]	-17,50 [kNm]	-10,00 [kN]	10,00 [kN]	-15,00 [kNm]
3.1	10,00 [kN]	10,00 [kN]	-25,00 [kNm]	-10,00 [kN]	10,00 [kN]	-22,50 [kNm]
3.2	10,00 [kN]	10,00 [kN]	-22,50 [kNm]	-10,00 [kN]	10,00 [kN]	-20,00 [kNm]
3.3	10,00 [kN]	10,00 [kN]	-20,00 [kNm]	-10,00 [kN]	10,00 [kN]	-17,50 [kNm]
3.4	10,00 [kN]	10,00 [kN]	-17,50 [kNm]	-10,00 [kN]	10,00 [kN]	-15,00 [kNm]
3.5	10,00 [kN]	10,00 [kN]	-15,00 [kNm]	-10,00 [kN]	10,00 [kN]	-12,50 [kNm]
3.6	10,00 [kN]	10,00 [kN]	-12,50 [kNm]	-10,00 [kN]	10,00 [kN]	-10,00 [kNm]
3.7	10,00 [kN]	10,00 [kN]	-10,00 [kNm]	-10,00 [kN]	10,00 [kN]	-7,50 [kNm]
3.8	10,00 [kN]	10,00 [kN]	-7,50 [kNm]	-10,00 [kN]	10,00 [kN]	-5,00 [kNm]
3.9	10,00 [kN]	10,00 [kN]	-5,00 [kNm]	-10,00 [kN]	10,00 [kN]	-2,50 [kNm]
3.10	10,00 [kN]	10,00 [kN]	-2,50 [kNm]	-10,00 [kN]	10,00 [kN]	0,00 [kNm]
4.1	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.2	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.3	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.4	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.5	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.6	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.7	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.8	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.9	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]
4.10	0,00 [kN]	10,00 [kN]	0,00 [kNm]	0,00 [kN]	10,00 [kN]	0,00 [kNm]

Slika 5.7 – Rezultati elemenata

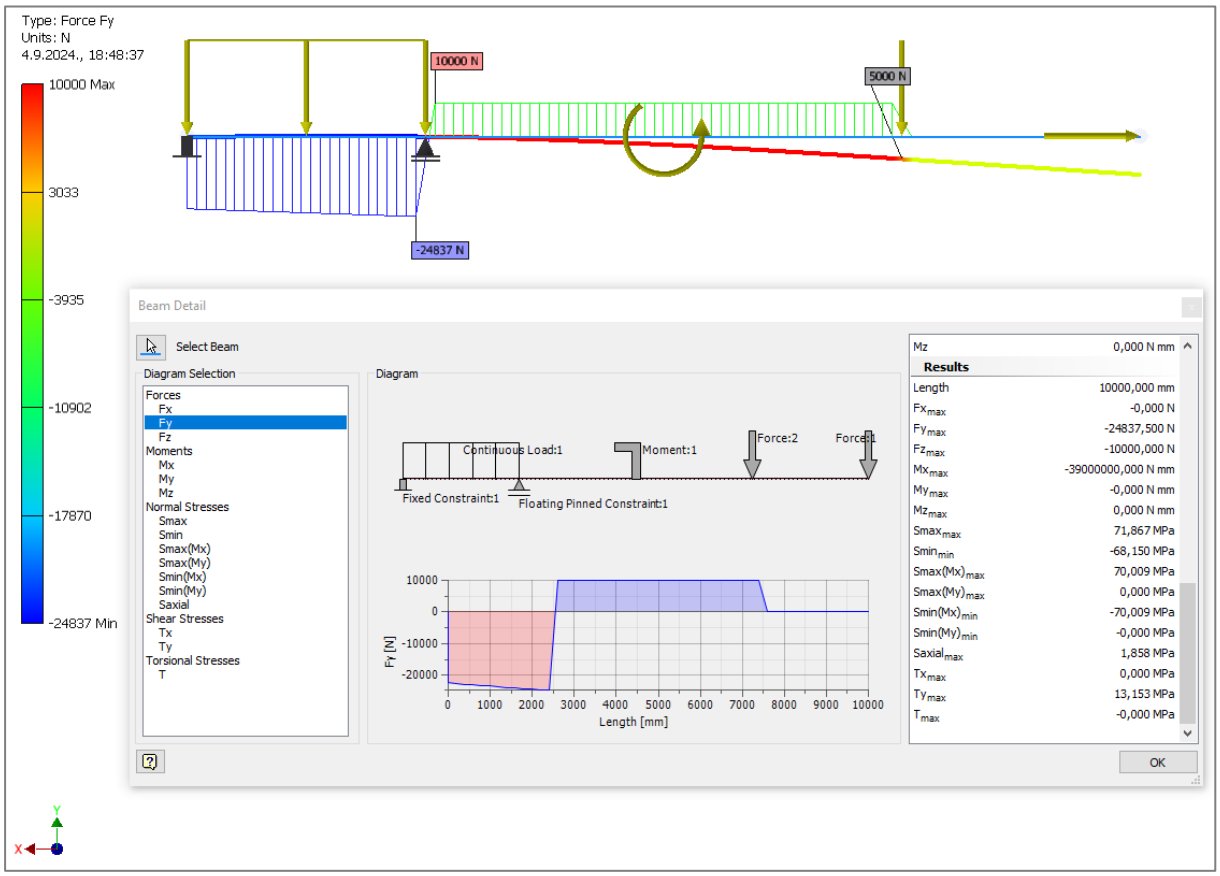
Name:	Value:
Vertical Displacement	-43,195 [mm]
Horizontal Displacement	0,088 [mm]
Rotation Displacement	-0,407 [deg]
Shear Force	-24,94 [kN]
Tension Force	10,00 [kN]
Compression Force	N/A
Bending Moment	-40,00 [kNm]

Slika 5.8 – Maksimalne vrijednosti opterećenja grede

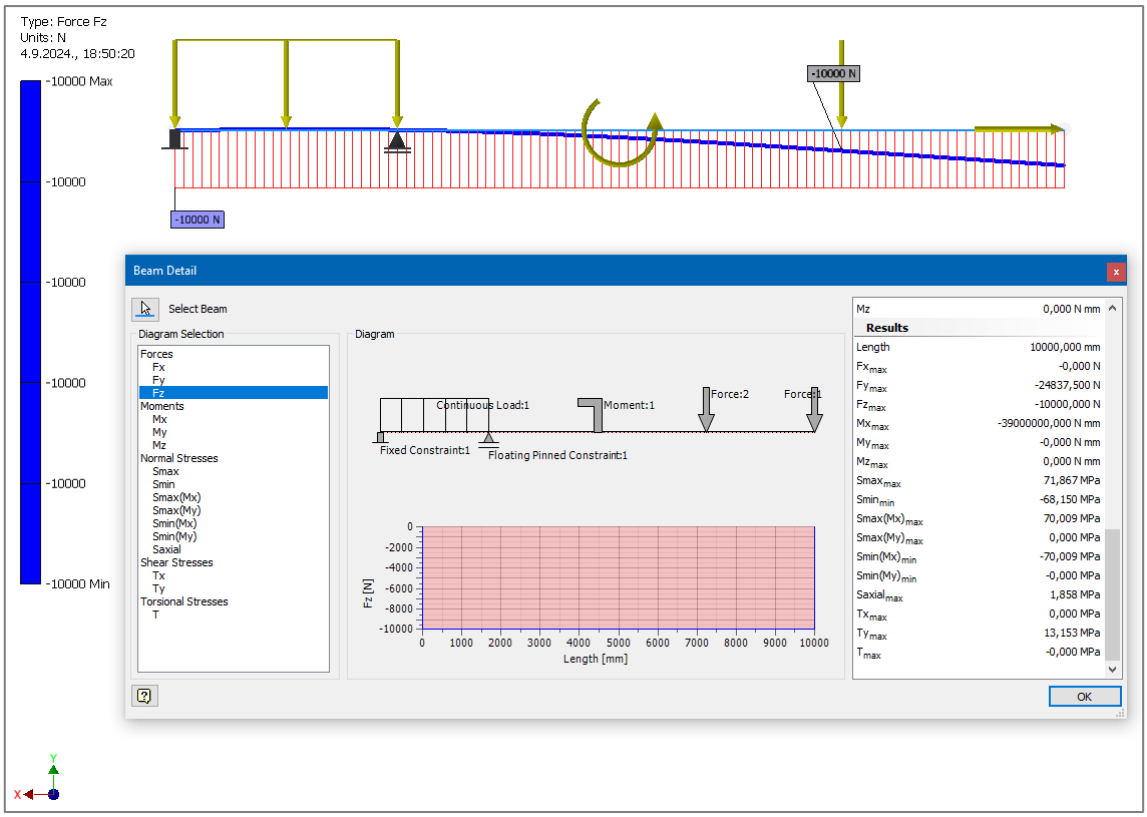
6. Rješavanje zadanog problema korištenjem Inventor programa

U ovom radu korišten je Autodesk Inventor [4] za analizu strukturnog okvira (Frame Analysis), pri čemu su rezultati uspoređeni s onima dobivenim Python programom.

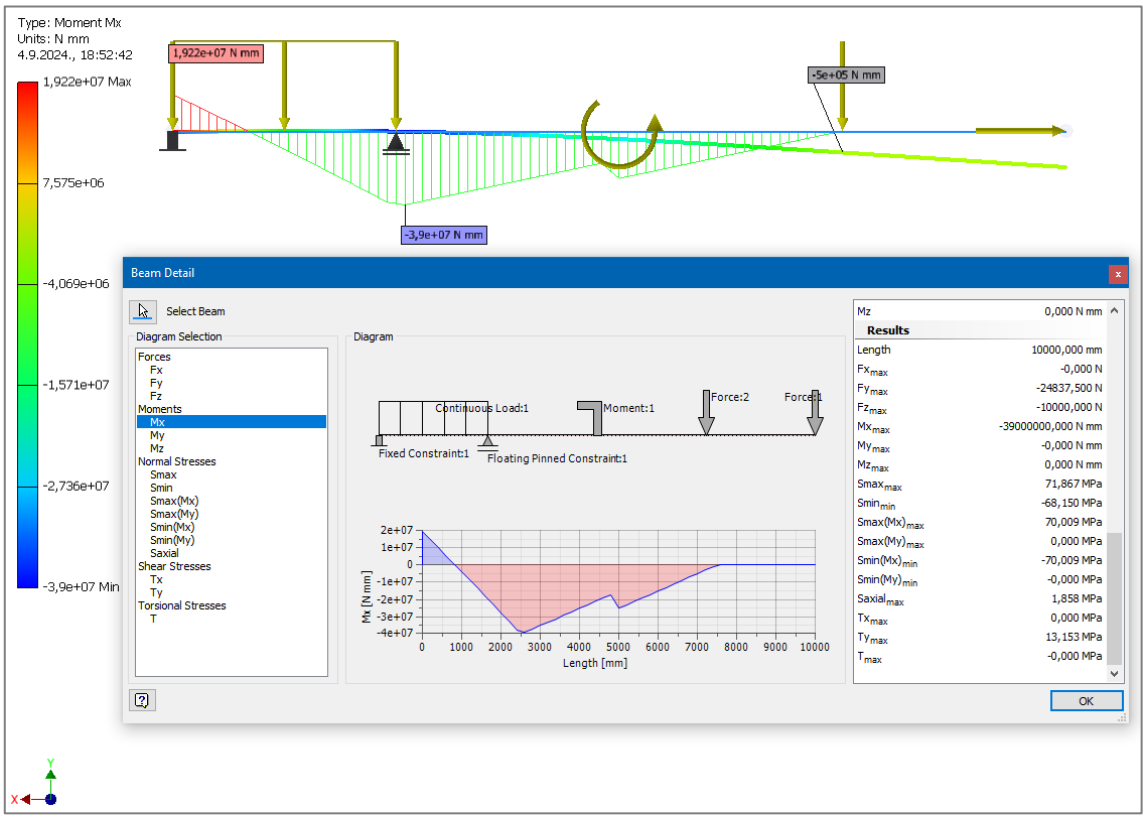
Za potrebe analize, u Autodesk Inventoru primijenjen je Frame Analysis modul. Definirana je geometrija okvira, materijalni parametri, opterećenja i uvjeti oslonaca. Analiza je provedena metodom konačnih elemenata (FEA), što je omogućilo dobivanje detaljnih podataka o silama djelujućim na grede, kao i o naprezanjima unutar samih greda. Osim toga, Inventor pruža rezultate pomaka u ravnini grede, što omogućuje sveobuhvatan pregled ponašanja okvira pod opterećenjem. Grafovi za poprečne sile, uzdužne momente i momente savijanja prikazani su na sljedećim slikama, zajedno s prikazom vertikalnog pomaka grede (*Slika 6.1*, *Slika 6.2*, *Slika 6.3*, *Slika 6.4*).



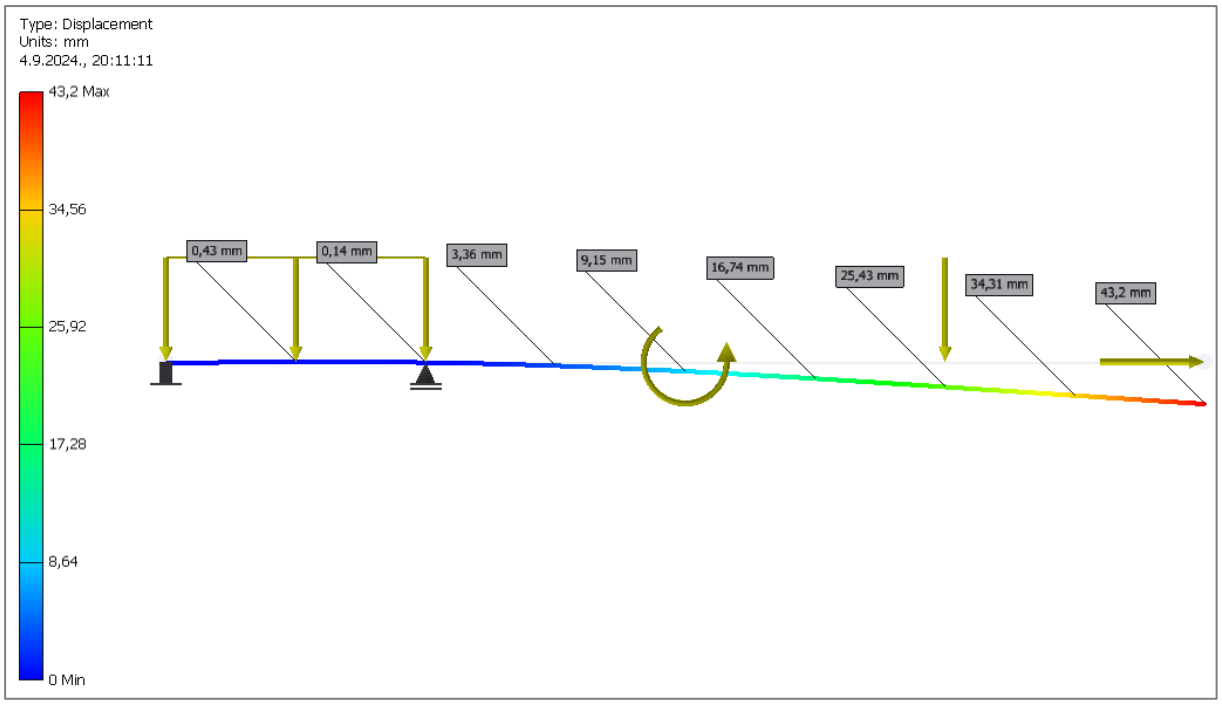
Slika 6.1 – Frame Analysis poprečne sile



Slika 6.2 – Frame Analysis uzdužne sile



Slika 6.3 – Frame Analysis moment savijanja



Slika 6.4 – Frame Analysis vertikalni pomak

Rezultati dobiveni u Inventoru pokazali su se sličnima onima izračunatima u Pythonu, s odstupanjima koja nisu velika i prihvatljiva su s obzirom da je riječ o 3D softveru koji koristi vlastite momente inercije za zadani profil, pri čemu se razlike mogu pojaviti zbog broja zadanih decimala.

Inventor Frame Analysis modul omogućio je preciznu vizualizaciju i detaljan pregled naprezanja i pomaka duž cijelog okvira, te se pokazao kao pouzdan alat za analizu složenih struktura.

Rezultati dobiveni pomoću Python programa uspoređeni su s rezultatima Inventorovog Frame Analysis te programa DSNWinbeam [5]. Razlike između rezultata su prikazane u tablici na slici 6.5 , koja ilustrira usporedbu i potvrđuje preciznost Python analize u odnosu na referentne rezultate Inventora i DSNWinbeam.

Usporedba	Python	Inventor	Razlika:	DSNWinbeam	Razlika:
Vertical Displacement	-43,195 [mm]	-43,195 [mm]	0,00%	-43,195 [mm]	0,00%
Horizontal Displacement	0,088 [mm]	N/A	N/A	N/A	N/A
Rotation Displacement	-0,407 [deg]	N/A	N/A	-0,407 [deg]	0,00%
Shear Force	-24,94 [kN]	-24,84 [kN]	0,40%	-24,94 [kN]	0,00%
Tension Force	10,00 [kN]	10,00 [kN]	0,00%	N/A	N/A
Compression Force	0,00 [kN]	0,00 [kN]	N/A	N/A	N/A
Bending Moment	-40,00 [kNm]	-39,00 [kNm]	2,56%	-40,00 [kNm]	0,00%

Slika 6.5 – Usporedba rezultata Python programa i rezultata dobivenih pomoću Inventor Frame Analysis i DSNWinbeam programa

7. Zaključak

Analizom 2D problema na gredi pomoću Python programa ostvarene su značajne prednosti u brzini i točnosti u usporedbi s tradicionalnim metodama, poput ručnog izračuna na papiru. Python omogućava fleksibilnu manipulaciju podacima, uključujući čvorove i sile, te pruža mogućnost neograničenog nadograđivanja programa.

Za rješavanje zadanog problema Python program pokazao se kao vrlo efikasan alat. Konkretno, za rješavanje zadatka bilo je potrebno 15 sekundi, pri čemu je sama kalkulacija trajala manje od 2 sekunde. Manipulacija s Excel datotekama, iako nešto sporija, dodatno je istaknula da specifikacija računala ima utjecaj na ukupnu učinkovitost programa.

Rezultati dobiveni pomoću Python programa su uspoređeni s rezultatima dobivenima korištenjem Inventor Frame Analysis i programa DSNWinbeam. Inventorov alat služio je kao sredstvo za verifikaciju točnosti rezultata koje je generirao Python program. Usporedba je pokazala da su rezultati iz Python programa u velikoj mjeri usklađeni s onima dobivenim putem Inventora. Odstupanje od 0,4% se javlja kod proračuna maksimalne smične sile, dok je odstupanje maksimalnog momenta savijanja 2,56%. U slučaju proračuna korištenjem programa DSNWinbeam nije bilo odstupanja u rezultatima u odnosu na izrađeni Python program.

Literatura

1. Python 3.12.4 [Online]. Dostupno na: https://www.python.org/downloads/. dana 06. 09. 2024
2. Python prezentacije [Online]. Dostupno na: https://moodle.oss.unist.hr/course/view.php?id=613. dana 06. 09. 2024
3. Spyder 5.5.1 IDE [Online]. Dostupno na: https://www.spyder-ide.org/. dana 06. 09. 2024
4. Autodesk Inventor [Online]. Dostupno na: https://www.autodesk.com/education/edu-software/overview. dana 06.09.2024
5. DSNWinbeam [Online]. Dostupno na: https://apps.microsoft.com/detail/9pc56zjsrdcf?hl=en-US&gl=US. dana 06. 09. 2024