Automatizirana priprema modela za analizu odziva grotlenog poklopca broda za prijevoz rasutog tereta metodom konačnih elemenata

Kos, Gordan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:940868

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Gordan Kos

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

Izv. prof. dr. sc. Pero Prebeg, dipl. ing. Gordan Kos

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Ponajprije se zahvaljujem mentoru, prof. Peri Prebegu na nebrojenim satima provedenim na konzultacijama, svom prenesenom znanju i stručnom vodstvu tijekom izrade ovog rada.

Prof. Jerolimu Andriću se zahvaljujem na svim savjetima, ustupljenim primjerima izvedenih konstrukcija i odgovorima na sva pitanja vezana za konstrukcijske detalje.

Zahvaljujem se ravnatelju Hrvatskog registra brodova, Damiru Roji na ukazanom povjerenju i dodijeljenoj stipendiji HRB-a, bez koje se ne bih mogao u potpunosti posvetiti diplomskom studiju.

Hvala kolegi Tomislavu Pavloviću na pomoći sa optimizacijom programskog koda korištenjem biblioteke NumPy i uloženom trudu u razvoj vizualizatora d3v-sgd koji je korišten za prikaz modela konstrukcije u okviru ovog rada.

Gordan Kos

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija brodogradnje

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

DIPLOMSKI ZADATAK

Student: **Gordan Kos** JMBAG: 0035201460

Naslov rada na hrvatskom jeziku: **Automatizirana priprema modela za analizu odziva grotlenog poklopca broda za prijevoz rasutog tereta metodom konačnih elemenata**

Naslov rada na engleskom jeziku: **Automatic preparation of finite element analysis model for analysis of a bulk carrier hatch cover**

Opis zadatka:

Pri projektiranju brodske konstrukcije neophodno je provesti analizu odziva metodom primjerene točnosti. U tu svrhu, sve više se koristi metoda konačnih elemenata. Program otvorenog koda d3v-sgd (Design visualizer for Ship grillage design) omogućuje pojednostavljeno, fleksibilno modeliranje konstrukcije grotlenog poklopca te njegovu trodimenzijsku vizualizaciju. U radu je potrebno, proširenjem funkcionalnosti programa otvorenog koda d3v-sgd u programskom jeziku Python, izraditi modul za automatiziranu pripremu modela za analizu odziva poklopca broda za rasuti teret metodom konačnih elemenata (MKE). Primjenom modula za automatiziranu pripremu MKE modela potrebno je automatizirati izradu modela za program OOFEM, koji omogućuje analizu odziva metodom konačnih elemenata.

Zadatak obuhvaća sljedeće:

- upoznavanje s trenutnom verzijom programa otvorenog koda d3v-sgd, koja omogućuje pojednostavljeno, fleksibilno modeliranje konstrukcije poklopca te njegovu trodimenzijsku vizualizaciju
- upoznavanje s pravilima za projektiranje konstrukcije grotlenog poklopca prema *IACS Common Structural Rules for Bulk Carriers,* July 2012 (IACS, 2012)
- izradu algoritma za diskretizaciju konstrukcije grotlenog poklopca konačnim elementima, u skladu s IACS, 2012
- implementaciju Python modula za automatiziranu pripremu MKE modela, prema prethodno izrađenom algoritmu za diskretizaciju konstrukcije grotlenog poklopca konačnim elementima
- primjenu implementiranog modula za automatsku izradu više različitih varijanti konstrukcije grotlenih poklopaca, s tim da se izrađene varijante moraju razlikovati po broju jakih uzdužnih nosača, broju jakih poprečnih nosača, dimenzijama jakih uzdužnih i poprečnih nosača, broju ukrepa te orijentaciji ukrepa
- izradu grafičkog sučelja u okviru programa otvorenog koda d3v-sgd koje omogućuje promjenu karakteristika diskretiziranog modela te odabir različitih vrsta konačnih elemenata.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
29. rujna 2022.	1. prosinca 2022.	12. – 16. prosinca 2022.

Zadatak zadao: Predsjednik Povjerenstva:

Izv. prof. dr. sc. Pero Prebeg Izv. prof. dr. sc. Ivan Ćatipović

SADRŽAJ

POPIS SLIKA

Prilog 2: Slike modela i mreža konačnih elemenata ispitnih varijanti konstrukcija

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
L_p	mm	Duljina promatrane zone oplate u smjeru globalne osi *x*
B_p	mm	Širina promatrane zone oplate u smjeru globalne osi *y*
α^{pe}	-	Aspektni odnos konačnih elemenata oplate (eng. *plate*)
α^{fe}	-	Aspektni odnos konačnih elemenata prirubnice (eng. *flange*)
α_p^{pe}	-	Poželjni aspektni odnos konačnih elemenata oplate
α_{max}^{pe}	-	Maksimalni aspektni odnos konačnih elemenata oplate
α_{max}^{fe}	-	Maksimalni aspektni odnos konačnih elemenata prirubnice
h	mm	Visina Hat profila
t	mm	Debljina lima Hat profila
φ	°	Kut nagiba struka Hat profila
S_1	mm	Razmak između strukova Hat profila
b_f	mm	Širina prirubnice T, L ili Hat profila
t_f	mm	Debljina prirubnice T, L ili Hat profila
b_p	mm	Širina sunosivog opločenja
t_p	mm	Debljina sunosivog opločenja
t_c	mm	Korozijski dodatak
h_w	mm	Visina struka T, L ili FB profila
t_w	mm	Debljina struka T, L ili FB profila
h_w'	mm	Visina HP profila
t_w'	mm	Debljina struka HP profila
dim_x	mm	Osnovna dimenzija mreže konačnih elemenata u smjeru globalne osi *x*
dim_y	mm	Osnovna dimenzija mreže konačnih elemenata u smjeru globalne osi *y*
dim_{xf}	mm	Dimenzija elementa prirubnice segmenta u smjeru globalne osi *x*
dim_{yf}	mm	Dimenzija elementa prirubnice segmenta u smjeru globalne osi *y*
$b_{f,max1}$	mm	Najveća širina prirubnice segmenta u okomitom smjeru na kraju 1
$b_{f,max2}$	mm	Najveća širina prirubnice segmenta u okomitom smjeru na kraju 2
tr_x	mm	Dimenzija prijelaznog elementa u smjeru globalne osi *x*
tr_y	mm	Dimenzija prijelaznog elementa u smjeru globalne osi *y*
tr_{xp}	mm	Dimenzija prijelaznog elementa oplate u smjeru globalne osi *x*
tr_{yp}	mm	Dimenzija prijelaznog elementa oplate u smjeru globalne osi *y*
tr_{xf}	mm	Dimenzija prijelaznog elementa prirubnice u smjeru globalne osi *x*
tr_{yf}	mm	Dimenzija prijelaznog elementa prirubnice u smjeru globalne osi *y*

SAŽETAK

Cilj ovog diplomskog rada je izrada modula za automatiziranu izradu mreže konačnih elemenata na roštiljnoj konstrukciji grotlenog poklopca u programskom jeziku Python. Modul je izveden u okviru programa otvorenog koda d3v-sgd čija je namjena modeliranje, analiza i vizualizacija jednostavnih brodskih roštiljnih konstrukcija.

Modul za automatiziranu izradu mreže konačnih elemenata izrađuje mrežu na temelju učitanog modela konstrukcije, koji sadrži sve informacije o položajima i svojstvima konstrukcijskih elemenata. Učitani model se provjerava prema kriterijima izvedivosti mreže i prolazi automatsko prepoznavanje simetričnosti konstrukcije. Na temelju prepoznatih ili ručno odabranih osi simetrija određuju se granice za izradu mreže konačnih elemenata. Unutar tih granica modela se računaju dimenzije konačnih elemenata, prema parametrima za kontrolu mreže koji se podešavaju u izrađenom korisničkom sučelju. Inicijalne vrijednosti ovih parametara su postavljene prema preporukama *Lloyd's Register* i zajedničkim konstrukcijskim pravilima *International Association of Classification Societies, Common Structural Rules for Bulk Carriers.*

Za diskretizaciju konstrukcije prema izračunatim dimenzijama konačnih elemenata su izrađene dvije varijante mreža. Prva varijanta generira mrežu isključivo sa pravokutnim *quad* elementima, a druga koristi kombinaciju četverokutnih i trokutastih konačnih elemenata. Na generiranu mrežu konačnih elemenata se izrađenim algoritmima postavljaju rubni uvjeti, opterećenje vanjskim tlakom i vlastitom težinom.

Ispitivanje automatske izrade mreže konačnih elemenata je provedeno na ukupno 8 različitih ispitnih varijanti konstrukcije. Za svaku ispitnu varijantu su priloženi nacrti, navedene specifičnosti i karakteristike, prikazani modeli konstrukcije te generirane mreže konačnih elemenata u programu d3v-sgd. Za odabranu ispitnu varijantu prikazani su rezultati analize metodom konačnih elemenata sa programom OOFEM koji je integriran u d3v-sgd. Odziv konstrukcije uspoređen je sa odzivom iste konstrukcije koja je ručno modelirana i analizirana u programu Maestro.

Ključne riječi: metoda konačnih elemenata, roštiljna konstrukcija, automatizirana izrada mreže konačnih elemenata, Python, d3v-sgd, OOFEM

SUMMARY

The objective of this master's thesis is creation of a Python module for automated finite element mesh generation of a hatch cover grillage structure. This module is developed as a part of d3v-sgd, an open-source program for modelling, analysis and visualization of simple ship grillage structures.

The finite element mesh is generated based on an input grillage model which contains all data relating to positions and properties of all structural elements. This input model is checked for mesh feasibility and goes through automatic symmetry detection. Finite element mesh extents are calculated based on the automatically discovered or overridden axis of symmetry. Finite element mesh size within the mesh extents is calculated using the input mesh control parameters, which can be set in the user interface. Initial values of mesh control parameters are set according to recommendations of *Lloyd's Register* and *International Association of Classification Societies, Common Structural Rules for Bulk Carriers*.

Discretization of the grillage structure according to the calculated mesh size can be performed using two different meshing variants. The first variant generates the finite element mesh using exclusively rectangular quad elements, while the second variant uses a combination of quad and triangle elements. Boundary conditions are automatically applied to the generated finite element mesh, along with a load case which consists of external pressure and self-weight.

Testing of automatic mesh generation has been conducted on 8 different test grillage structure variants. Blueprints, particularities, characteristics, input models and generated finite element mesh variants in d3v-sgd are shown for all test grillage variants. Results of finite element method analysis using OOFEM integrated into d3v-sgd are shown for a selected test variant. Structural response is compared with the response of the same grillage structure variant that has been manually meshed and analyzed in the program Maestro.

Key words: finite element method, grillage structure, automated finite element mesh generation, Python, d3v-sgd, OOFEM

1. UVOD

Razvojem računala i računalnih alata za analizu odziva se metoda konačnih elemenata sve više koristi za direktne proračune konstrukcija u brodogradnji i široj inženjerskoj praksi. Korištenje ove metode omogućuje ispitivanje različitih konstrukcijskih rješenja u kratkom vremenu, a time i optimizaciju konstrukcije s obzirom na odabrani cilj. U ovakvoj primjeni je automatizirana izrada mreže konačnih elemenata nezaobilazan korak u diskretizaciji konstrukcije ukoliko model sadržava geometrijske ili topološke varijable.

Metoda konačnih elemenata je numerička metoda koja se temelji na fizičkoj diskretizaciji kontinuuma. Razmatrani kontinuum s beskonačnim brojem stupnjeva slobode se zamjenjuje s diskretnim modelom međusobno povezanih elemenata s ograničenim brojem stupnjeva slobode. Drugim riječima, kontinuum se dijeli na konačan broj područja koja se nazivaju konačni elementi, čime razmatrani kontinuum postaje mreža konačnih elemenata. Konačni elementi su međusobno povezani u točkama na konturi koje se nazivaju čvorovi. Stanje u svakom elementu, kao što je polje pomaka, deformacije, naprezanja i ostalih veličina, se opisuje pomoću interpolacijskih funkcija. Te funkcije moraju zadovoljavati odgovarajuće uvjete da bi se diskretizirani model što više približio ponašanju kontinuiranog sustava. Treba imati na umu da su rješenja dobivena metodom konačnih elemenata približna, a ne egzaktna. Stvarnim vrijednostima se moguće približiti samo uz ispravan odabir proračunskog modela i konačnih elemenata koji su u stanju opisati realni proces deformacije. Uz pravilnu formulaciju konačnih elemenata, povećanjem broja elemenata raste točnost dobivenog rješenja [1].

U ovom radu je korištenjem programskog jezika Python izrađen programski kod za automatsku diskretizaciju konstrukcije grotlenog poklopca konačnim elementima. Python je interpretirani, interaktivni, objektno orijentirani programski jezik opće namjene, otvorenog koda, čiji je tvorac nizozemski programer Guido van Rossum. Odlikuju ga iznimna čitljivost, fleksibilnost proširivanja i kompatibilnost [2]. Zahvaljujući dostupnosti raznih biblioteka kao što su TensorFlow, Matplotlib, NumPy, Scipy, Pandas i druge, Python je postao *de facto* standardni programski jezik znanstvene zajednice u više znanstvenih polja.

2. MODELIRANJE KONSTRUKCIJE ROŠTILJA

Modul za definiciju modela `grillage_model.py` omogućuje pojednostavljeno modeliranje roštiljne konstrukcije grotlenog poklopca uz minimalni broj nezavisnih parametara i jedan je od osnovnih modula programa d3v-sgd (Design Visualizer for Ship Grillage Design). Modul je izrađen u okviru timskog projekta izbornog kolegija Podobnost konstrukcije plovnih objekata, a za potrebe ovog rada dodatno je proširen. Ovaj modul osim opisivanja konstrukcije ima ulogu povezivanja ostalih modula programa d3v-sgd koji služe za analizu odziva analitičkom metodom izjednačenja progiba, analizu odziva metodom konačnih elemenata i analizu podobnosti prema preskriptivnim pravilima. U ovom poglavlju biti će objašnjeni korišteni pojmovi i načini definicije pojedinih dijelova roštiljne konstrukcije zbog povezanosti modula za definiciju modela i modula za izradu mreže konačnih elemenata.

Program d3v-sgd je dostupan na poveznici: https://github.com/pprebeg/d3v-sgd [3], dok je verzija programa koja je korištena u ovom diplomskom radu dostupna na: https://github.com/pprebeg/d3v-sgd/tree/gordan_kos_final_thesis [4]

2.1 Model konstrukcije poklopca broda za prijevoz rasutog tereta

Model opisuje tipičnu izvedbu roštiljne konstrukcije grotlenog poklopca pravokutnog oblika sa sustavom jakih nosača koji teku u dva smjera i križaju se pod pravim kutom. Slikom 2.1 je prikazan primjer MacGregor podiznog poklopca broda za opći teret.

Slika 2.1 Podizni grotleni poklopac [5]

Svaka zona oplate između jakih nosača ima određeni broj sekundarnih ukrepa, a na elementarnim panelima oplate između jakih nosača i ukrepa mogu biti postavljene interkostalne ukrepe. Modelom je opisana varijanta grotlenog poklopca sljedećih karakteristika i ograničenja:

1) Tip poklopca je nepropustan za vremenske uvjete (eng. *weathertight*)
2) Opločenje poklopca je jednostruko
3) Opločenje nema preluka
4) Oplata je u ravnini *x,y* na visini struka jakih nosača
5) Na jednoj zoni oplate je moguće postaviti samo jedan raspored ukrepa
6) Raspored ukrepa može imati samo jedan tip ukrepe
7) Jedna zona oplate može imati samo jedno svojstvo opločenja
8) Jaki nosači su T, L ili FB profili
9) Segment jakog nosača može imati samo jedno svojstvo grede
10) Ukrepe mogu biti T, L, FB, Bulb ili Hat profili
11) Dopušten je mješoviti sustav orebrenja

2.2 Struktura klasa modula za definiciju modela roštiljne konstrukcije

Struktura programskog koda za modeliranje konstrukcije je izrađena primjenom oblikovnog obrasca *Composite* kojim se opisuje hijerarhija pojedinih dijelova i njihove cjeline. Ovaj obrazac omogućuje izradu više različitih primitivnih objekata, koji se kasnije spajaju u složeniji kompozitni objekt [6]. U ovom slučaju je taj kompozitni objekt sama roštiljna konstrukcija `Grillage`, koja se sastoji od sljedećih dijelova tj. objekata:

- `PrimarySuppMem` – jaki nosač
- `Plate` – zona oplate sa pridruženim sekundarnim ukrepama

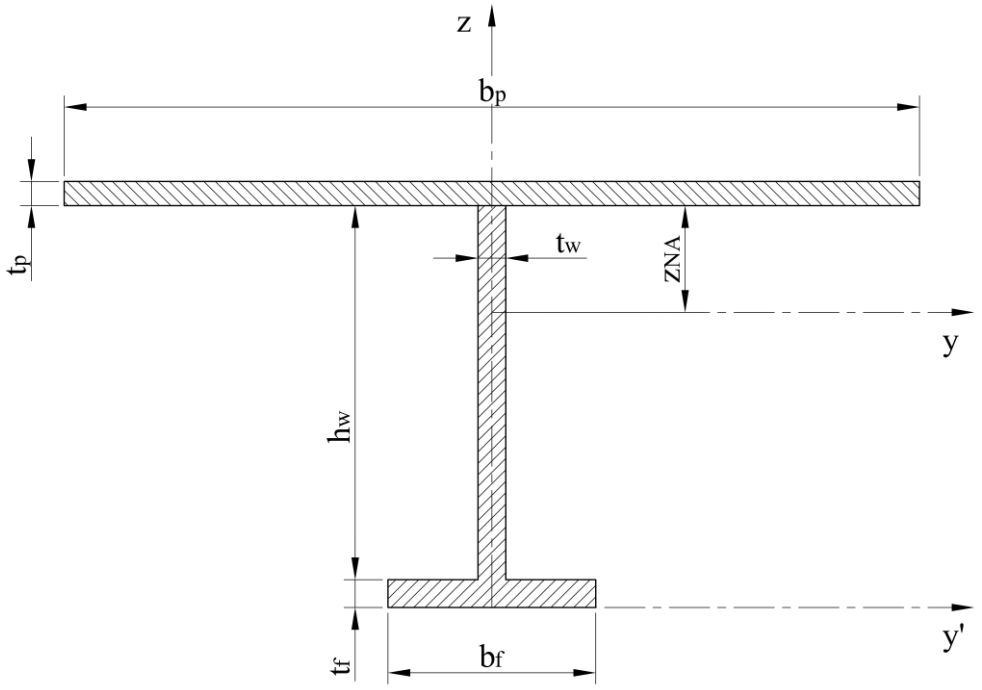
Svaki jaki nosač se može podijeliti na više dijelova, a svaki dio je opisan objektom `Segment`, kojemu se pridodaju svojstva grede i položaja u odnosu na jake nosače. Svaka zona oplate ima pridružen raspored ukrepa `StiffenerLayout` koji opisuje sve ukrepe koje se nalaze na toj zoni oplate. Ukrepama i segmentima jakih nosača se pridodaju svojstva grede u obliku objekta `BeamProperty`, koji može opisivati različite tipove poprečnih presjeka. Svakom svojstvu grede i svojstvu opločenja `PlateProperty` se pridodaje objekt koji opisuje karakteristike materijala `MaterialProperty`. U prilogu I, slika 1-1 prikazuje UML dijagram klasa programskog koda `grillage_model.py`

2.3 Karakteristike poprečnog presjeka jakih nosača i ukrepa

Svojstva greda definirana su klasom `BeamProperty` te izvedenim klasama koje sadrže posebnosti pojedine vrte profila. Ove klase sadrže metode za proračun karakteristika krutosti svih jakih nosača i ukrepa. Te metode primarno su namijenjene za analitički proračun odziva metodom izjednačenja progiba, a za potrebe ovog rada su prenesene i u modul `geofementity.py` i služe za proračun krutosti grednih konačnih elemenata običnih ukrepa. Dostupni tipovi profila unutar ovog modula su T, L, FB, Bulb i Hat. Prilikom izrade svojstava grednih konačnih elemenata koje će biti dodijeljeno konačnim elementima ukrepa se iz modela preuzimaju neto dimenzije, prema *IACS CSR Chapter 3, Section 2, Net scantling approach* [7].

2.3.1 T profil

Izvedeni izrazi za T profil prikazanog slikom 2.2 se koriste za proračun neto momenta inercije, uvrštavanjem neto dimenzija prema jednadžbama 2.1 – 2.5. Unutar programskog koda su obuhvaćeni slučajevi kada T profil nema sunosivo opločenje i/ili prirubnicu, tako da je proračun osim L profila upotrebljiv i za proračun FB profila.



Slika 2.2 Poprečni presjek T profila

Neto visina struka za slučaj kada profil ima sunosivo opločenje:

$$h_{\mathrm{w,net}} = h_{\mathrm{w}} + t_{\mathrm{c}} \tag{2.1}$$

Neto visina struka za slučaj kada profil nema sunosivo opločenje:

$$h_{\mathrm{w,net}} = h_{\mathrm{w}} \tag{2.2}$$

Neto debljina struka:

$$t_{\mathrm{w,net}} = t_{\mathrm{w}} - t_{\mathrm{c}} \tag{2.3}$$

Neto širina prirubnice:

$$b_{\mathrm{f,net}} = b_{\mathrm{f}} - t_{\mathrm{c}} \tag{2.4}$$

Neto debljina prirubnice:

$$t_{\mathrm{f,net}} = t_{\mathrm{f}} - t_{\mathrm{c}} \tag{2.5}$$

Površina poprečnog presjeka sa sunosivom širinom:

$$A = b_{\mathrm{f}} \cdot t_{\mathrm{f}} + h_{\mathrm{w}} \cdot t_{\mathrm{w}} + b_{\mathrm{p}} \cdot t_{\mathrm{p}} \tag{2.6}$$

Udaljenost neutralne linije od točke spoja sa opločenjem:

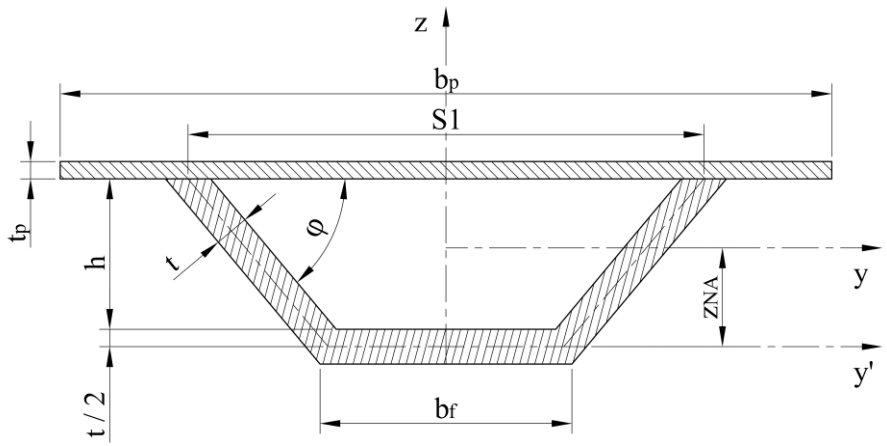
$$z_{\mathrm{NA}} = \frac{b_{\mathrm{f}} \cdot t_{\mathrm{f}} \left(t_{\mathrm{p}} + h_{\mathrm{w}} + \frac{t_{\mathrm{f}}}{2} \right) + h_{\mathrm{w}} \cdot t_{\mathrm{w}} \left(t_{\mathrm{p}} + \frac{h_{\mathrm{w}}}{2} \right) + b_{\mathrm{p}} \cdot \frac{{t_{\mathrm{p}}}^2}{2}}{A} \tag{2.7}$$

Moment inercije profila oko y osi:

$$I_{\mathrm{y}} = b_{\mathrm{f}} \cdot t_{\mathrm{f}} \cdot \left[\frac{{t_{\mathrm{f}}}^2}{12} + \left(\frac{t_{\mathrm{f}}}{2} + h_{\mathrm{w}} + t_{\mathrm{p}} - z_{\mathrm{NA}} \right)^2 \right] + h_{\mathrm{w}} \cdot t_{\mathrm{w}} \cdot \left[\frac{{h_{\mathrm{w}}}^2}{12} + \left(\frac{h_{\mathrm{w}}}{2} + t_{\mathrm{p}} - z_{\mathrm{NA}} \right)^2 \right] + \\ + b_{\mathrm{p}} \cdot t_{\mathrm{p}} \cdot \left[\frac{{t_{\mathrm{p}}}^2}{12} + \left(z_{\mathrm{NA}} - \frac{t_{\mathrm{p}}}{2} \right)^2 \right] \tag{2.8}$$

2.3.2 Hat profil

Hat profil prikazan slikom 2.3 je izveden razmatranjem ekvivalentnog profila primjenom pravila o paralelnom pomaku. Prema ovom pravilu se moment tromosti s obzirom na neku os neće promijeniti ako se pojedini dijelovi presjeka pomaknu paralelno s tom osi [8].



Slika 2.3 Poprečni presjek Hat profila

Razmak strukova u razini spoja sa opločenjem:

$$S_1 = b_\mathrm{f} + \frac{2 \cdot h + t}{\mathrm{tg}(\varphi)} - t \cdot \mathrm{tg}\left(\frac{\varphi}{2}\right) \tag{2.9}$$

Smanjenje širine prirubnice na jednoj strani zbog korozijskog dodatka:

$$b_{\mathrm{f,red}} = \frac{t_\mathrm{c}}{2} \cdot \mathrm{tg}\left(\frac{\varphi}{2}\right) \tag{2.10}$$

Neto širina prirubnice:

$$b_{\mathrm{f,net}} = b_\mathrm{f} - 2 \cdot b_{\mathrm{f,red}} \tag{2.11}$$

Neto visina struka:

$$h_\mathrm{net} = h + t_\mathrm{c} \tag{2.12}$$

Neto debljina struka i prirubnice:

$$t_\mathrm{net} = t - t_\mathrm{c} \tag{2.13}$$

Površina poprečnog presjeka sa sunosivom širinom:

$$A = \frac{2 \cdot h \cdot t}{\sin(\varphi)} + \frac{t^2}{\mathrm{tg}(\varphi)} + b_\mathrm{f} \cdot t + b_\mathrm{p} \cdot t_\mathrm{p} \tag{2.14}$$

Udaljenost neutralne linije od simetrale prirubnice:

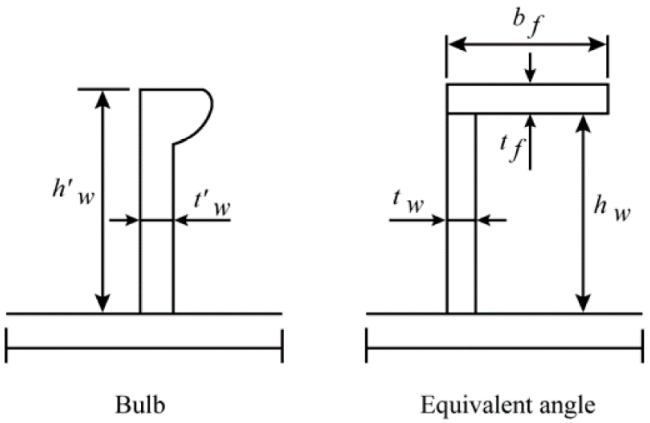
$$z_\mathrm{NA} = \frac{\frac{h \cdot t}{\sin(\varphi)} \cdot (h + t) + \frac{t^3}{6 \cdot \mathrm{tg}(\varphi)} + \left(h + \frac{t_\mathrm{p} + t}{2}\right) \cdot b_\mathrm{p} \cdot t_\mathrm{p}}{A} \tag{2.15}$$

Moment inercije profila oko y osi:

$$I_\mathrm{y} = 2 \cdot \left[\frac{t \cdot h^3}{12 \cdot \sin(\varphi)} + \left(\frac{t + h}{2} - z_\mathrm{NA}\right)^2 \cdot \frac{h \cdot t}{\sin(\varphi)}\right] + 2 \cdot \left[\frac{t^4}{36 \cdot \mathrm{tg}(\varphi)} + \left(z_\mathrm{NA} - \frac{t}{6}\right)^2 \cdot \frac{t^2}{2 \cdot \mathrm{tg}(\varphi)}\right] + \\ + \left[\frac{b_\mathrm{f} \cdot t^3}{12} + z_\mathrm{NA}^2 \cdot b_\mathrm{f} \cdot t\right] + \left[\frac{b_\mathrm{p} \cdot t_\mathrm{p}^3}{12} + \left(\frac{t_\mathrm{p} + t}{2} - z_\mathrm{NA}\right)^2 \cdot b_\mathrm{p} \cdot t_\mathrm{p}\right] \tag{2.16}$$

2.3.3 HP profil

Moment inercije HP profila se određuje prema već izvedenim izrazima za T profil, uvrštavanjem neto dimenzija ekvivalentnog L profila. Bruto dimenzije ekvivalentnog L profila prikazanog slikom 2.4 su određene prema *IACS CSR, Chapter 3, Section 6, 4.1.1 Stiffener profile with a bulb section* [7].



Slika 2.4 Dimenzije ekvivalentnog L profila [7]

Visina struka ekvivalentnog profila:

$$h_{\mathrm{w}} = h'_{\mathrm{w}} - \frac{h'_{\mathrm{w}}}{9{,}2} + 2 \tag{2.17}$$

Širina prirubnice ekvivalentnog profila:

$$b_{\mathrm{f}} = \alpha\left(t'_{\mathrm{w}} + \frac{h'_{\mathrm{w}}}{6{,}7} - 2\right) \tag{2.18}$$

Debljina prirubnice ekvivalentnog profila:

$$t_{\mathrm{f}} = \frac{h'_{\mathrm{w}}}{9{,}2} - 2 \tag{2.19}$$

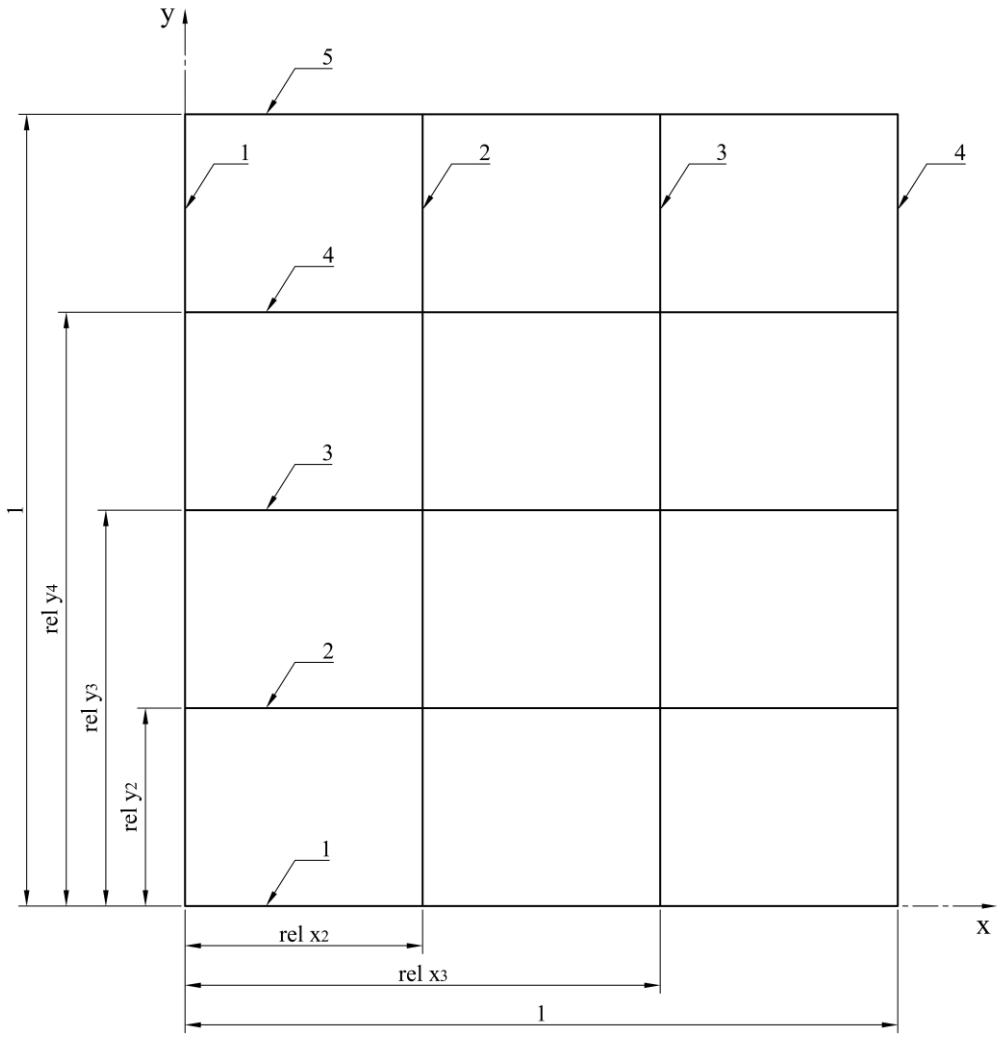
gdje su h'_{w} i t'_{w} visina i debljina struka HP profila

$$\alpha = 1{,}1 + \frac{(120 - h'_{\mathrm{w}})^2}{3000} \quad \text{za } h'_{\mathrm{w}} \leq 120\ mm \tag{2.20}$$

$$\alpha = 1 \quad \text{za } h'_{\mathrm{w}} > 120\ mm \tag{2.21}$$

2.4 Jaki nosač

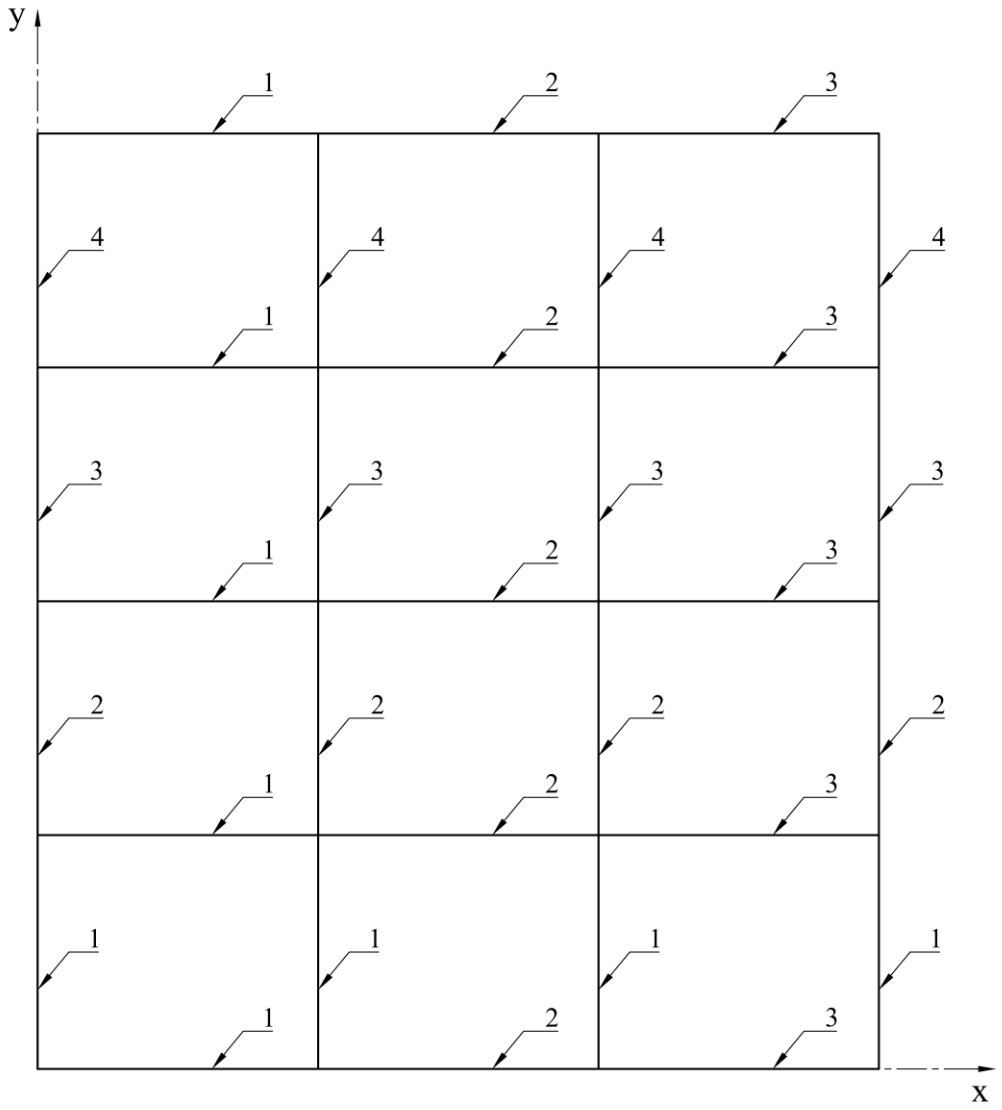
Jaki nosač definiran je klasom `PrimarySuppMem` (eng. *Primary Supporting Member*), skraćeno PSM, i opisuje gredu koja se proteže cijelom duljinom ili širinom roštiljne konstrukcije, a može biti usmjeren uzdužno (`BeamDirection.LONGITUDINAL`) ili poprečno (`BeamDirection.TRANSVERSE`). Jaki uzdužni nosač (eng. *longitudinal*) je jaki nosač koji se proteže uzdužno u smjeru globalne osi *x*. Jaki poprečni nosač (eng. *transversal*) je nosač koji se proteže poprečno u smjeru globalne osi *y*. Svaki jaki nosač osim podatka vlastitog smjera sadrži i jedinstveni identifikacijski broj `idbeam` na razini pojedinog smjera i relativnu udaljenost od globalnih koordinatnih osi `rel_dist`. Numeracija jakih nosača zasebno kreće od 1 za uzdužne i poprečne nosače, kako je prikazano slikom 2.5. Jaki rubni nosači se nalaze na relativnim koordinatama 0 i 1, a centralni jaki nosač, za neparan broj nosača, na sredini konstrukcije se nalazi na relativnoj koordinati 0.5.



Slika 2.5 Identifikacijski brojevi i relativne koordinate jakih nosača

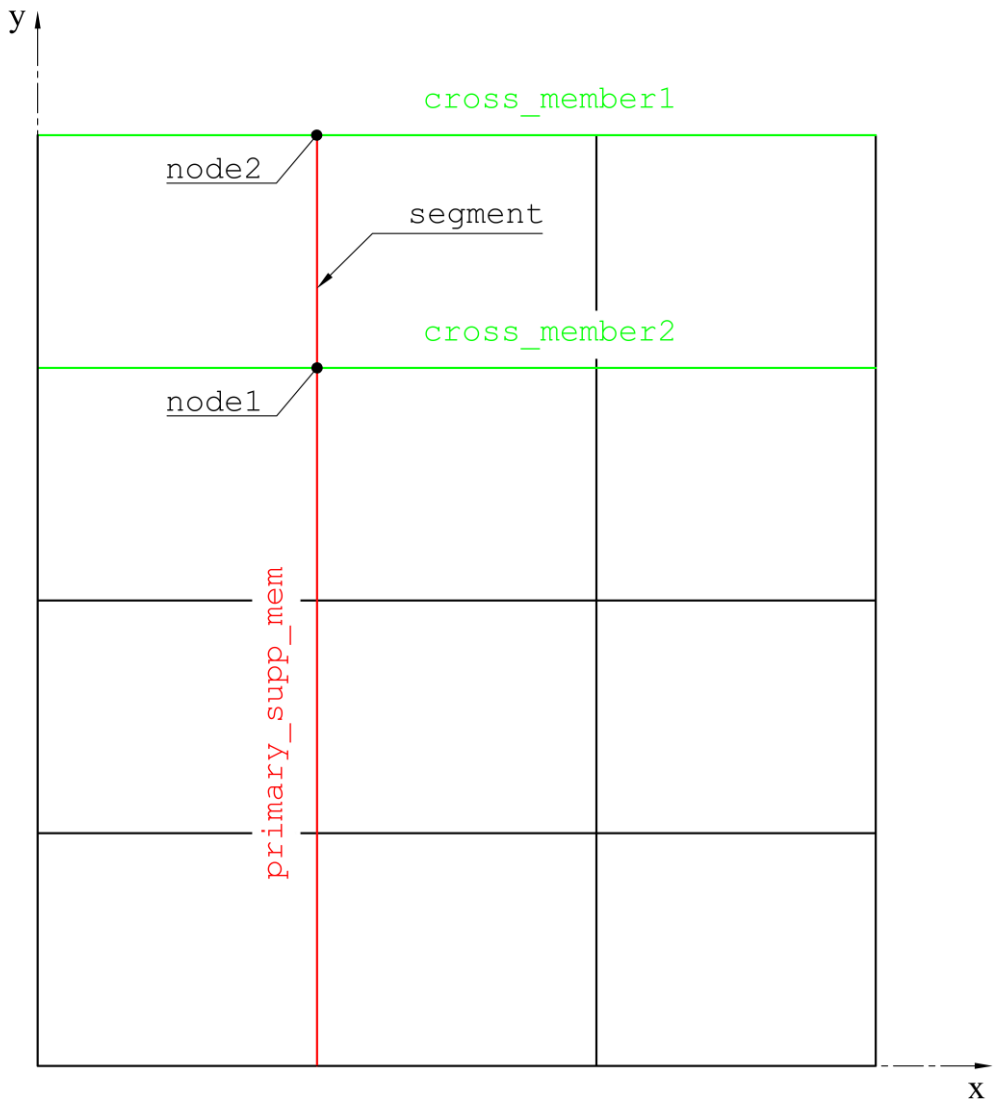
2.5 Segment jakog nosača

Segment je dio jakog nosača omeđen sa dva jaka nosača u suprotnom smjeru te može pripadati jakom uzdužnom ili jakom poprečnom nosaču, a definiran je klasom `Segment`. Uzdužni segment (eng. *longitudinal segment*) je dio jakog uzdužnog nosača omeđen sa dva jaka poprečna nosača. Poprečni segment (eng. *transverse segment*) je dio jakog poprečnog nosača omeđen sa dva jaka uzdužna nosača. Svaki segment sadrži jedinstveni identifikacijski broj `idsegment` na razini pojedinog jakog nosača, svojstvo grede `beam_prop` (eng. *beam property*), jaki nosač `primary_supp_mem` kojemu segment pripada i dva jaka nosača `cross_member1`, `cross_member2` koji definiraju segment. Numeracija segmenata zasebno kreće od 1 za svaki jaki nosač, kako je prikazano slikom 2.6.



Slika 2.6 Identifikacijski brojevi segmenata jakih nosača

Slika 2.7 prikazuje primjer definicije poprečnog segmenta jakog poprečnog nosača (crveno) koji je definiran sa dva jaka uzdužna nosača (zeleno). Na mjestima sjecišta ovih jakih nosača se nalaze referentne točke `node1`, `node2` svojstvene svakom segmentu, čije se koordinate unutar programskog koda određuju algoritmom za presjecište jakih nosača metodom `get_intersection`.



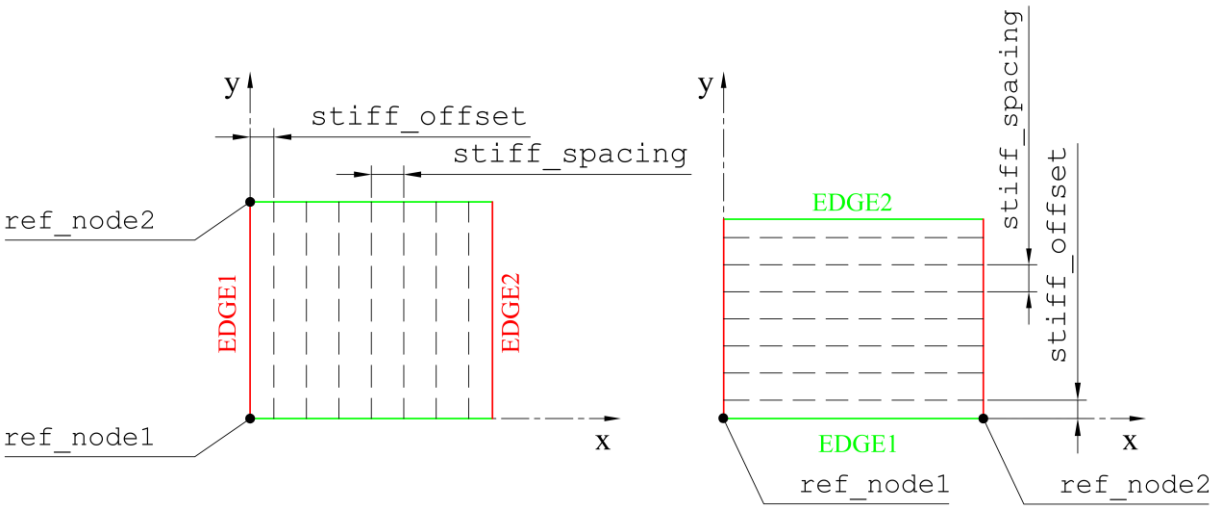
Slika 2.7 Definicija segmenta i referentnih točaka

Metodama `get_segment_node1` i `get_segment_node2` se određuju pojedine koordinate tih referentnih točaka, pri čemu je točka `node1` uvijek bliža globalnoj koordinatnoj osi. Vertikalna koordinata ovih referentnih točaka je na visini $z = h_w$, na mjestu spoja struka nosača sa opločenjem.

2.6 Raspored ukrepa

Za definiciju rasporeda ukrepa koristi se klasa `StiffenerLayout`. Na pojedinoj zoni oplate implementirane su dvije mogućnosti. Vrsta definicije ukrepa je određena sa vrijednosti `definition_type`, a može biti prema broju (`NUMBER`) ili razmaku (`SPACING`) ukrepa. Osim tipa i vrijednosti definicije, svaki raspored ukrepa sadrži jedinstveni identifikacijski broj `id_layout` i informaciju svojstva grede `beam_prop` (eng. *beam property*) koja pripada svim ukrepama pojedinog rasporeda ukrepa.

Za buduću implementaciju drugih mogućnosti definicije rasporeda ukrepa prema nekom odabranom referentnom rubu (`EDGE1` ili `EDGE2`), svaka zona oplate definira vlastiti referentni rub `ref_edge`. Ovaj referentni rub ovisi o orijentaciji ukrepa, kako je prikazano slikom 2.8.



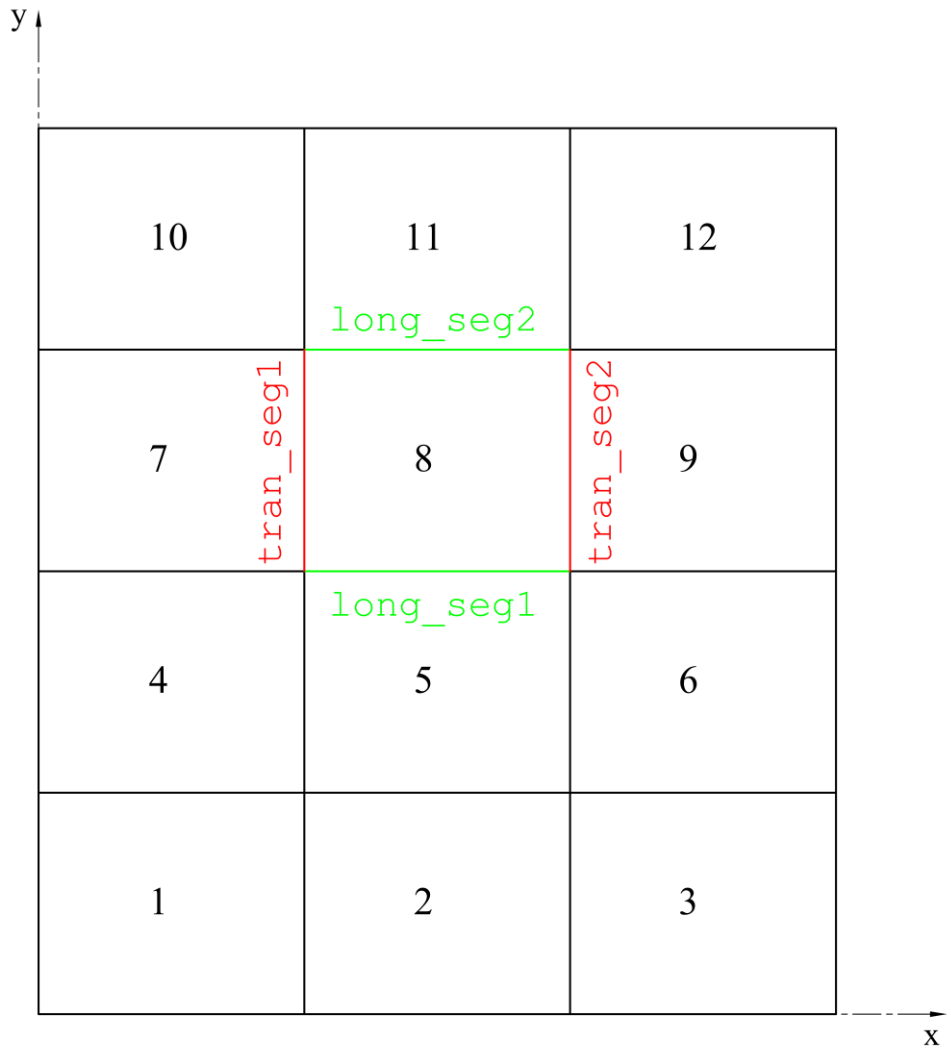
Slika 2.8 Referentni rubovi i čvorovi za postavljanje ukrepa

Svaki raspored ukrepa može imati dodijeljeno samo jedno svojstvo grede, a svaka zona oplate može imati dodijeljen samo jedan raspored ukrepa. Ovo znači da je na pojedinoj zoni oplate moguće postaviti samo jednu vrstu ukrepa, što u nekim slučajevima može biti ograničavajuće, ako je npr. samo na sredini zone oplate potrebno postaviti drugačije ukrepe.

Ovaj nedostatak bi se mogao riješiti izradom novog objekta `Stiffener` koji bi sadržavao vlastito svojstvo grede i bio dodijeljen pojedinom rasporedu ukrepa na određenoj poziciji. To bi također podrazumijevalo proširenje objekta `StiffenerLayout` koji bi trebao sadržavati dodatne informacije i metode za određivanje koja ukrepa se nalazi na kojoj poziciji.

2.7 Zona oplate

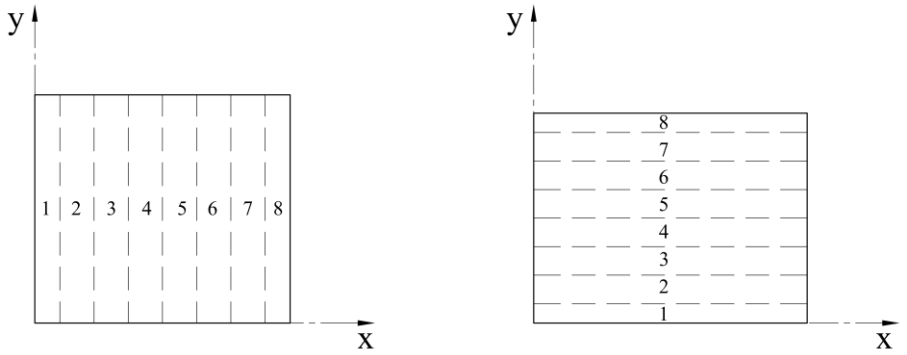
Zona oplate definirana je klasom `Plate` (eng. Plating zone) i predstavlja dio oplate poklopca omeđen sa četiri segmenta: `long_seg1`, `long_seg2`, `trans_seg1` i `trans_seg2`. Svaka zona oplate sadrži jedinstveni identifikacijski broj, svojstvo opločenja, raspored ukrepa, orijentaciju ukrepa i referentni rub. Primjer numeracije zona oplate identifikacijskim brojem `idplate` je prikazan slikom 2.9, gdje su također prikazani rubni uzdužni (zeleno) i poprečni (crveno) segmenti koji definiraju granice zone oplate 8. Svojstvo opločenja `plate_prop` (eng. *plate property*) sadrži informaciju o debljini i vrsti materijala. Raspored ukrepa `stiff_layout` (eng. *stiffener layout*) i orijentacija ukrepa `stiff_dir` (eng. *stiffener direction*) u potpunosti definiraju ukrepe na pojedinoj zoni oplate. Referentni rub `ref_edge` (eng. *reference edge*) je podatak koji može poslužiti za nesimetričnu definiciju rasporeda ukrepa u odnosu na neki segment (rub) zone oplate.



Slika 2.9 Numeracija zona oplate i rubni segmenti

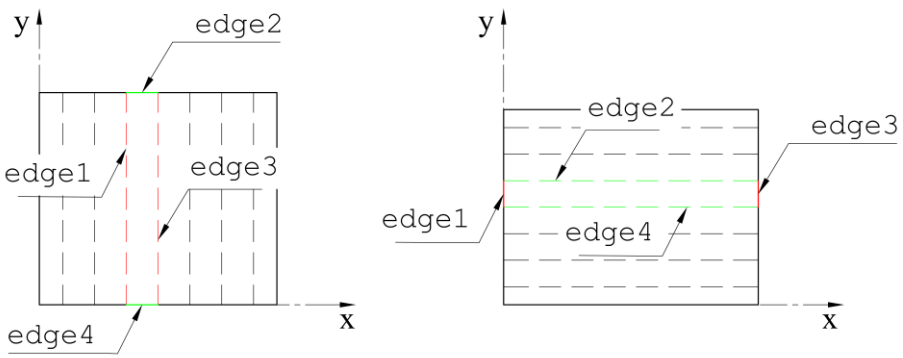
2.8 Elementarni panel oplate

Elementarni panel `ElementaryPlatePanel` je dio oplate omeđen sa jakim nosačima i rasporedom ukrepa duž svojih rubova. Svaki elementarni panel ima jedinstven identifikacijski broj `id` na razini pojedine zone oplate, broj interkostalnih ukrepa `intercostal_stiffener_num` (eng. *intercostal stiffener number*) i svojstvo tih inerkostalnih ukrepa `beam_prop` (eng. *beam property*). Metoda `generate_elementary_plate_panels` izrađuje elementarne panele na svim zonama oplate, pri čemu je inicijalna vrijednost broja interkostalnih ukrepa 0, a svojstvo grede je `None`, tj. interkostalne ukrepe inicijalno ne postoje. Numeracija elementarnih panela zasebno kreće od 1 za svaku zonu oplate, u smjeru globalnih koordinatnih osi ovisno o orijentaciji ukrepa. Slika 2.10 prikazuje numeraciju elementarnih panela oplate za poprečne ukrepe (lijevo) i za uzdužne ukrepe (desno).



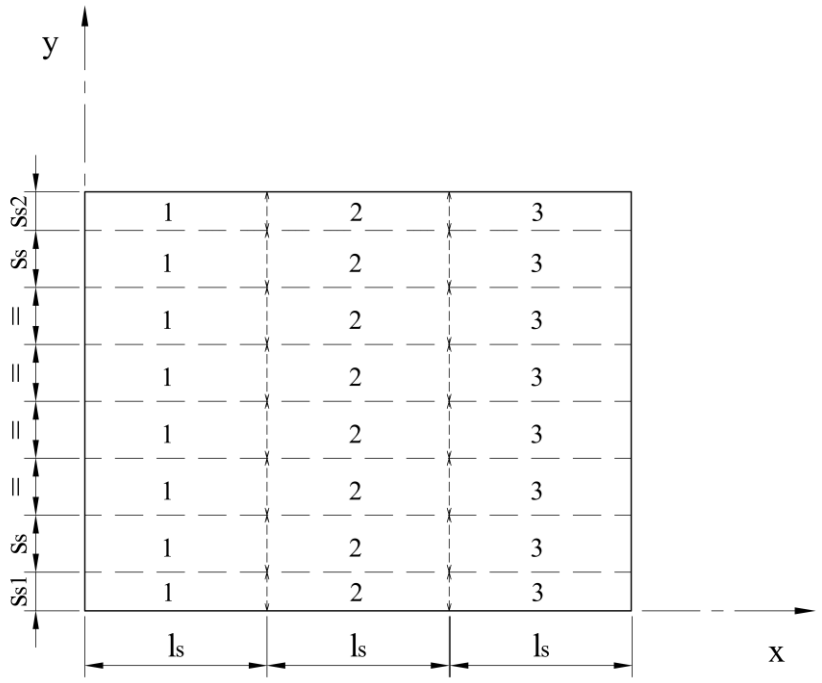
Slika 2.10 Numeracija elementarnih panela oplate

Za provjeru opločenja prema kriterijima izvijanja potrebno je imati informaciju o tipovima nosača na svakom rubu neukrepljenog polja oplate. Lista tipova ovih rubnih nosača se određuje metodom `get_edge_beam_types`. U slučaju kada nema interkostalnih ukrepa, rubni nosači mogu biti samo jaki nosači ili obične ukrepe. Na slici 2.11 je prikazan sustav označavanja rubnih nosača za slučaj poprečnih ukrepa (lijevo) i uzdužnih ukrepa (desno).



Slika 2.11 Sustav označavanja rubnih nosača elementarnog panela

Dodavanje interkostalnih ukrepa može biti pojedinačno na odabrani elementarni panel oplate ili grupno na sve elementarne panele metodom `set_intercostal_stiffeners`, pri čemu se isti broj i svojstva interkostalnih ukrepa postavljaju na sve elementarne panele odabrane zone oplate. U slučaju kada na elementarnom panelu postoje interkostalne ukrepe, dolazi do podjele elementarnog panela na više neukrepljenih polja oplate koja se nazivaju pod paneli (eng. *sub panel*). Numeracija pod panela započinje sa brojem 1 za svaki redak ili stupac elementarnih panela između ukrepa, ovisno o orijentaciji ukrepa. Primjer numeracije pod panela jedne zone oplate sa uzdužnim ukrepama i dvije interkostalne ukrepe na svakom elementarnom panelu je prikazan slikom 2.12.



Slika 2.12 Numeracija i dimenzije pod panela zone oplate

Metoda `get_elementary_plate_dimensions` vraća manju i veću dimenziju, s_s i l_s neukrepljenog polja oplate na odabranom elementarnom panelu. Dimenzija panela paralelno s ukrepama se određuje metodom `sub_panel_length`, pri čemu se nepoduprti raspon ukrepe dijeli na jednake dijelove u ovisnosti o broju interkostalnih ukrepa. Dimenzija panela okomito na ukrepe se određuje metodom `sub_panel_width`, pri čemu se uzima u obzir razmak ukrepe od jakog nosača, razmak između ukrepa i tip profila ukrepe. Ako je ukrepa Hat profil, dimenzija panela okomito na ukrepe se smanjuje za vrijednosti razmaka između strukova S_1.

2.9 Roštiljna konstrukcija

Cjelokupna konstrukcija definirana je klasom `Grillage` koja sadrži slijedeća svojstva:

- `L_overall` – ukupna duljina u uzdužnom smjeru globalne x osi, [m]
- `B_overall` – ukupna širina u poprečnom smjeru globalne y osi, [m]
- `N_longitudinal` – broj jakih uzdužnih nosača
- `N_transverse` – broj jakih poprečnih nosača

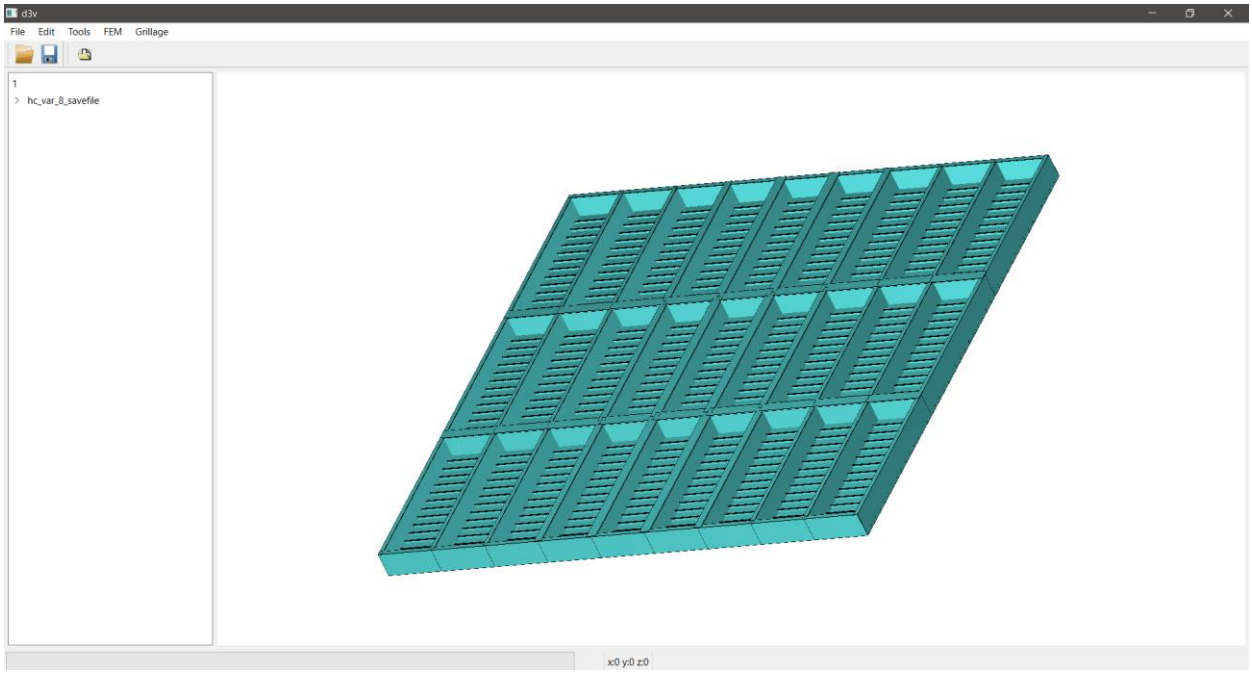
Svojstva materijala, opločenja, greda i ostalih karakteristika konstrukcije se zapisuju unutar ovog objekta u sljedećim rječnicima:

- `longitudinal_memb` – jaki uzdužni nosači
- `transverse_memb` – jaki poprečni nosači
- `material_props` – svojstva materijala
- `beam_props` – svojstva greda
- `plate_props` – svojstva opločenja
- `stiffener_layouts` – rasporedi ukrepa
- `plating` – zone oplate
- `corrosion_add` – korozijski dodatak
- `flange_direction` – usmjerenje prirubnica jakih nosača

Prilikom inicijalne generacije objekta roštiljne konstrukcije koristi se po jedno svojstvo grede za ukrepe, jake uzdužne, poprečne i rubne nosače te samo jedno svojstvo debljine oplate koje se dodjeljuje svim zonama oplate. Na ovaj način je za definiciju cijele konstrukcije potrebno definirati najviše četiri svojstva greda i jedno svojstvo opločenja, čime se pojednostavljuje i ubrzava proces izrade nove varijante konstrukcije. Za naknadnu izmjenu karakteristika segmenata i zona oplate postoje različite metode koje omogućuju pojedinačnu ili grupnu izmjenu konstrukcije. Grupne izmjene mogu biti s obzirom na simetrične elemente konstrukcije ili s obzirom na njihov položaj u odnosu na ostale elemente. Na globalnoj razini modela konstrukcije je inicijalno zadan parametar `flange_direction` koji usmjerava sve prirubnice jakih nosača prema središtu konstrukcije (`FlangeDirection.INWARD`). Naknadnom izmjenom je za pojedini jakih nosač moguće promijeniti ovaj parametar.

3. MODUL ZA IZRADU MREŽE KONAČNIH ELEMENATA

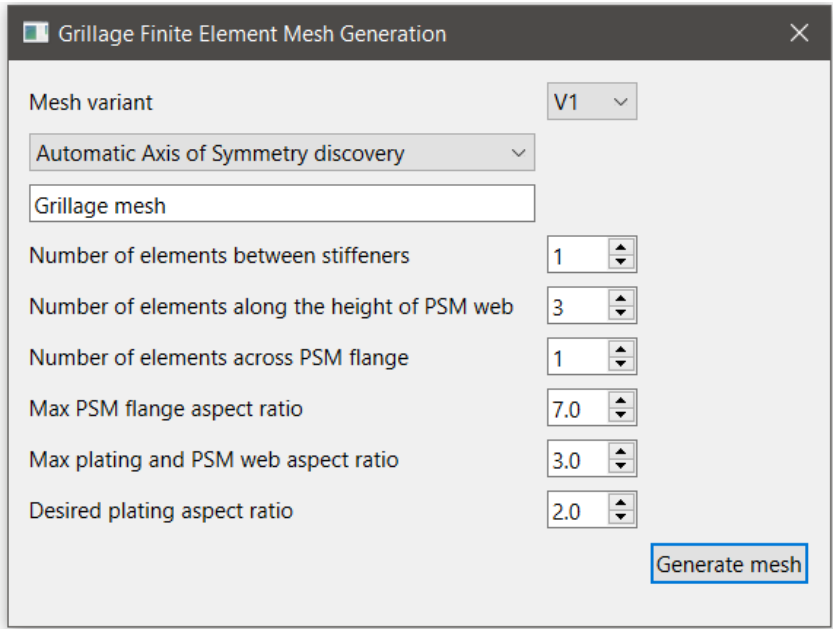
Modul za automatiziranu izradu mreže konačnih elemenata `grillage_mesher.py` je zamišljen kao jedan od dva sastavna Python modula za analizu odziva modela konstrukcije učitanog u program otvorenog koda d3v-sgd (Design Visualizer for Ship Grillage Design). Glavni prozor programa d3v-sgd sa učitanim modelom ispitne varijante konstrukcije `hc_var_8` je prikazan slikom 3.1.



Slika 3.1 Glavni prozor programa d3v-sgd

Osim izrade mreže konačnih elemenata na modelu konstrukcije, modul u potpunosti priprema ulaznu datoteku za analizu metodom konačnih elemenata pomoću programa OOFEM. U tu svrhu se na izrađenu mrežu postavljaju odgovarajući rubni uvjeti, prema preporukama sadržanim u *Lloyd's Register, Assesment of Steel Hatch Covers Using Finite Element Analysis* [9]. Opterećenje vanjskim tlakom i vlastitom težinom je zadano prema zajedničkim konstrukcijskim pravilima *IACS, Common Structural Rules for Bulk Carriers* [7].

Modul je povezan sa grafičkim sučeljem programa d3v-sgd, unutar kojega se pozivaju metode za izradu mreže konačnih elemenata. U izborniku programa *Grillage* /*Analysis* dodana je opcija *Generate FEM* koja otvara korisničko sučelje za izradu mreže konačnih elemenata, prikazano slikom 3.2. Ovo sučelje omogućuje odabir varijante mreže, odabir osi simetrije, unos imena i podešavanje svih dostupnih parametara za kontrolu mreže.



Slika 3.2 Korisničko sučelje za izradu mreže konačnih elemenata

Odabirom opcije *Generate mesh* unutar ovog grafičkog sučelja započinje automatska generacija mreže konačnih elemenata sljedećim koracima:

1.) Automatsko prepoznavanje simetričnosti modela i provjera izvedivosti mreže
2.) Identifikacija elemenata obuhvaćenih izradom mreže s obzirom na osi simetrije
3.) Izrada svojstava pločastih i grednih konačnih elemenata prema karakteristikama modela
4.) Izračun dimenzija konačnih elemenata lokalnim razmatranjem dijelova konstrukcije
5.) Usklađivanje dimenzija konačnih elemenata globalnim razmatranjem konstrukcije
6.) Izrada čvorova i pločastih konačnih elemenata na svim zonama oplate
7.) Izrada grednih konačnih elemenata svih ukrepa
8.) Izrada čvorova i pločastih konačnih elemenata na svim segmentima jakih nosača
9.) Spajanje mreže ispravkom preklapanja čvorova duž rubova elemenata konstrukcije
10.) Ispravak preklapanja konačnih elemenata prirubnica jakih nosača
11.) Identifikacija i postavljanje rubnih uvjeta prema osi simetrije
12.) Izrada slučaja opterećenja prema zadanom projektnom tlaku

Prvi korak provjere modela je opisan u četvrtom poglavlju, a određivanje opsega mreže (koraci 2 i 3) u petom poglavlju. Određivanje dimenzija mreže konačnih elemenata (koraci 4 i 5) je opisano u šestom poglavlju. Postupak izrade mreže konačnih elemenata (koraci 6 – 10) je opisan u sedmom poglavlju. Postavljanje rubnih uvjeta i opterećenja (koraci 11 i 12) je opisano u osmom poglavlju.

3.1 Struktura klasa modula za izradu mreže konačnih elemenata

Struktura klasa (eng. *class*) programskog koda je izrađena korištenjem principa SOLID za objektno orijentirano programiranje. Prema prvom principu jedne odgovornosti (eng. *Single responsibility*), svaka klasa treba imati samo jednu odgovornost tj. glavnu ili jedinu funkcionalnost koju mora dobro odrađivati. Korištenjem ovog principa se olakšava izmjena programskog koda i ispitivanje njegovih pojedinih dijelova, jer se može lako utvrditi koja klasa je odgovorna za koji zadatak [10]. Slijedeći ove principe, izvedena je implementacija modula za izradu mreže konačnih elemenata koji sadrži sljedeće klase:

- `ModelCheck` - provjera učitanog modela za osi simetrije i izvedivost izrade mreže
- `MeshExtent` - određivanje opsega izrade mreže konačnih elemenata na modelu
- `MeshSize` - određivanje dimenzija i broja konačnih elemenata
- `PlateMesh` - izrada mreže konačnih elemenata na jednoj zoni oplate
- `SegmentMesh` - izrada mreže konačnih elemenata na jednom segmentu
- `GrillageMesh` - izrada mreže konačnih elemenata na cijelom modelu

Primjenom oblikovnog obrasca *Template method* je definirana struktura algoritma za izradu mreže konačnih elemenata, koji opisuje fiksan proces u kojemu pojedini koraci mogu biti drugačiji. Ovo se postiže izradom podklasa(eng. *subclass*) za redefiniciju pojedinih dijelova, što ne mijenja osnovnu strukturu algoritma [6]. Kako bi se omogućilo jednostavno dodavanje različitih načina izrade mreže konačnih elemenata, prema ovom obrascu su izrađene podklase koji nasljeđuju zajedničke metode unutar sljedećih baznih klasa (eng. *base class*):

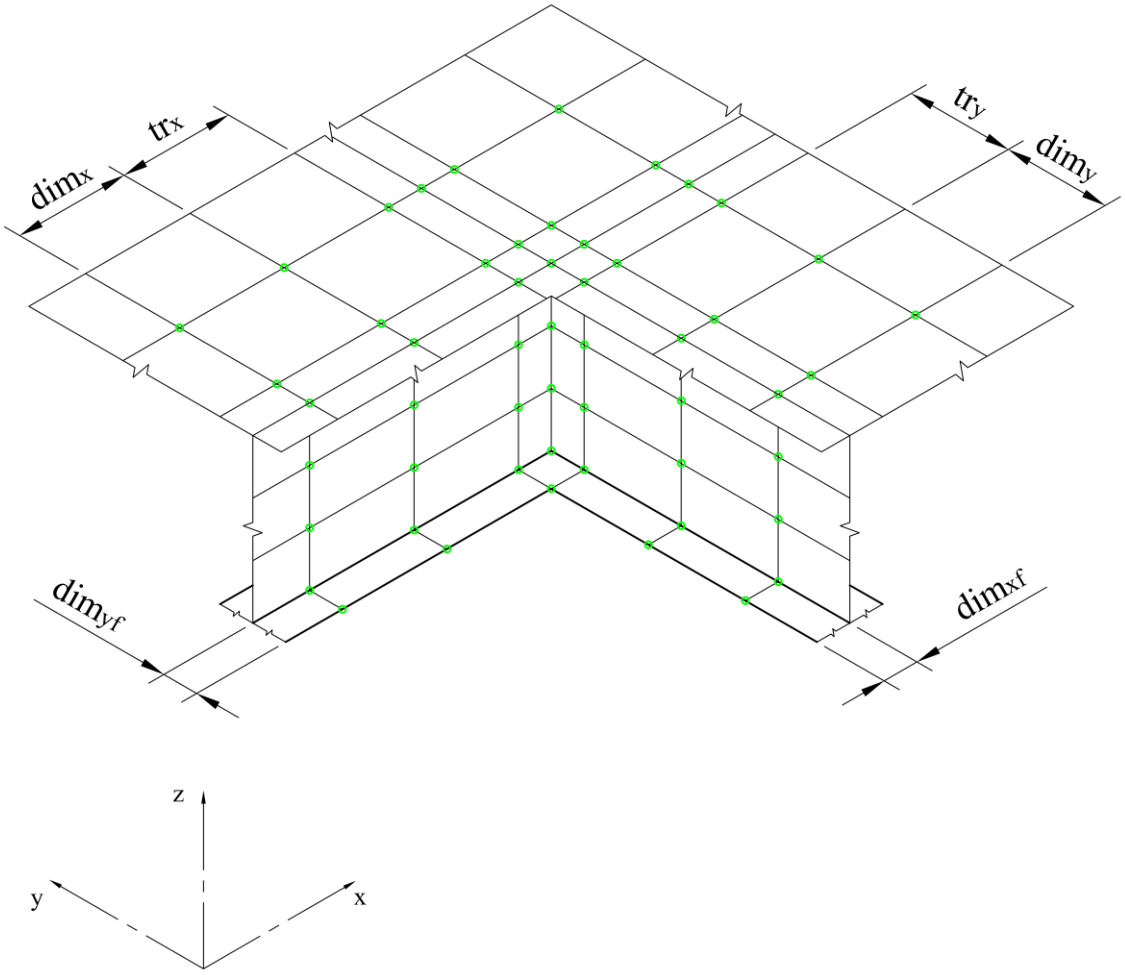
- `MeshSize`: `ElementSizeV1` i `ElementSizeV2` – određivanje dimenzija konačnih elemenata specifičnih za varijante mreže *V1* i *V2*
- `SegmentMesh`: `SegmentMeshV1` i `SegmentMeshV2` –načini izrade mreže konačnih elemenata na segmentima, specifični za varijante mreže *V1* i *V2*
- `GrillageMesh`: `MeshVariantV1` i `MeshVariantV1` -izrada mreže konačnih elemenata pojedine varijante mreže *V1* ili *V2*

Primjenom obrasca *Template method* se ujedno slijedi princip otvoren-zatvoren (eng. *open – closed*), prema kojemu je programski kod otvoren za proširenje, ali zatvoren za izmjene. Ovo znači da je proširenje koda npr. dodavanjem nove varijante mreže moguće bez potrebe za izmjenom ili preinakom postojećih metoda. U prilogu I, slika 1-2 prikazuje UML dijagram klasa programskog koda `grillage_mesher.py`.

3.2 Karakteristike i ograničenja varijanti mreža konačnih elemenata

3.2.1 Varijanta mreže V1

Prva varijanta mreže *V1* koristi tehniku preslikavanja elementa prirubnica jakih nosača na oplatu. Time su mreža oplate i prirubnica usko povezani, a određivanjem ovakve mreže oplate je također istovremeno definirana cijela mreža konačnih elemenata prirubnica. Slikom 3.3 je prikazan koncept varijante mreže *V1* na spoju dva jaka T nosača, gdje je vidljiv način preslikavanja elemenata prirubnice na mrežu opločenja.

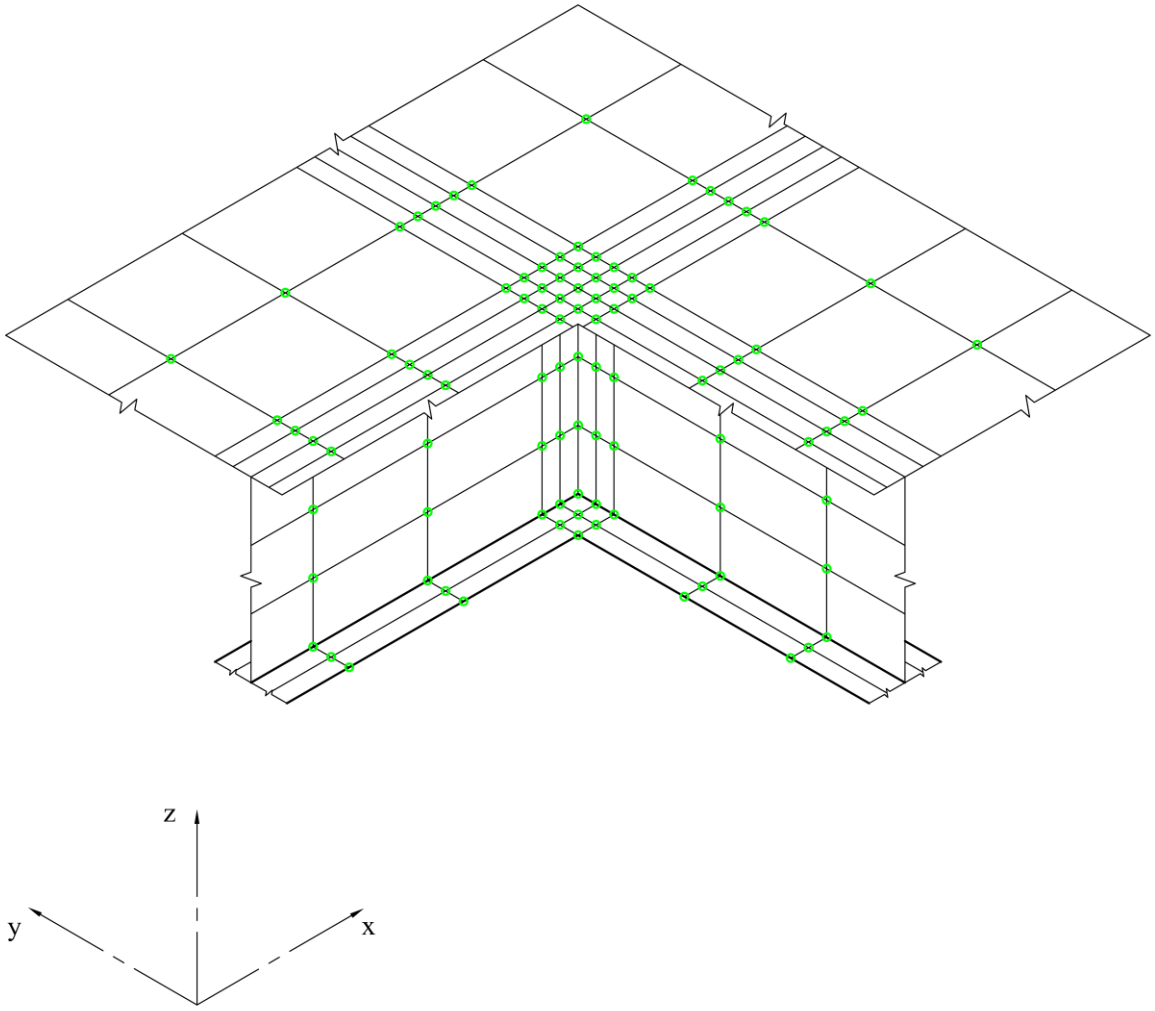


Slika 3.3 Koncept varijante mreže *V1*, jedan element po širini prirubnice

Ograničenja varijante *V1* su sljedeća:

1) Širine prirubnica moraju biti jednake duž cijelog jakog nosača
2) Visine svih jakih nosača moraju biti jednake, a prirubnice im biti u istoj ravnini

Glavni nedostaci varijante *V1* su povišena gustoća mreže konačnih elemenata oko spoja opločenja sa strukovima jakih nosača i nezadovoljavajući aspektni odnos preslikanih elemenata prirubnice na opločenje. Ovo je posebno izraženo kod grubljih mreža konačnih elemenata uz odabir većeg broja elemenata po širini prirubnice, što je vidljivo na Slici 3.4.

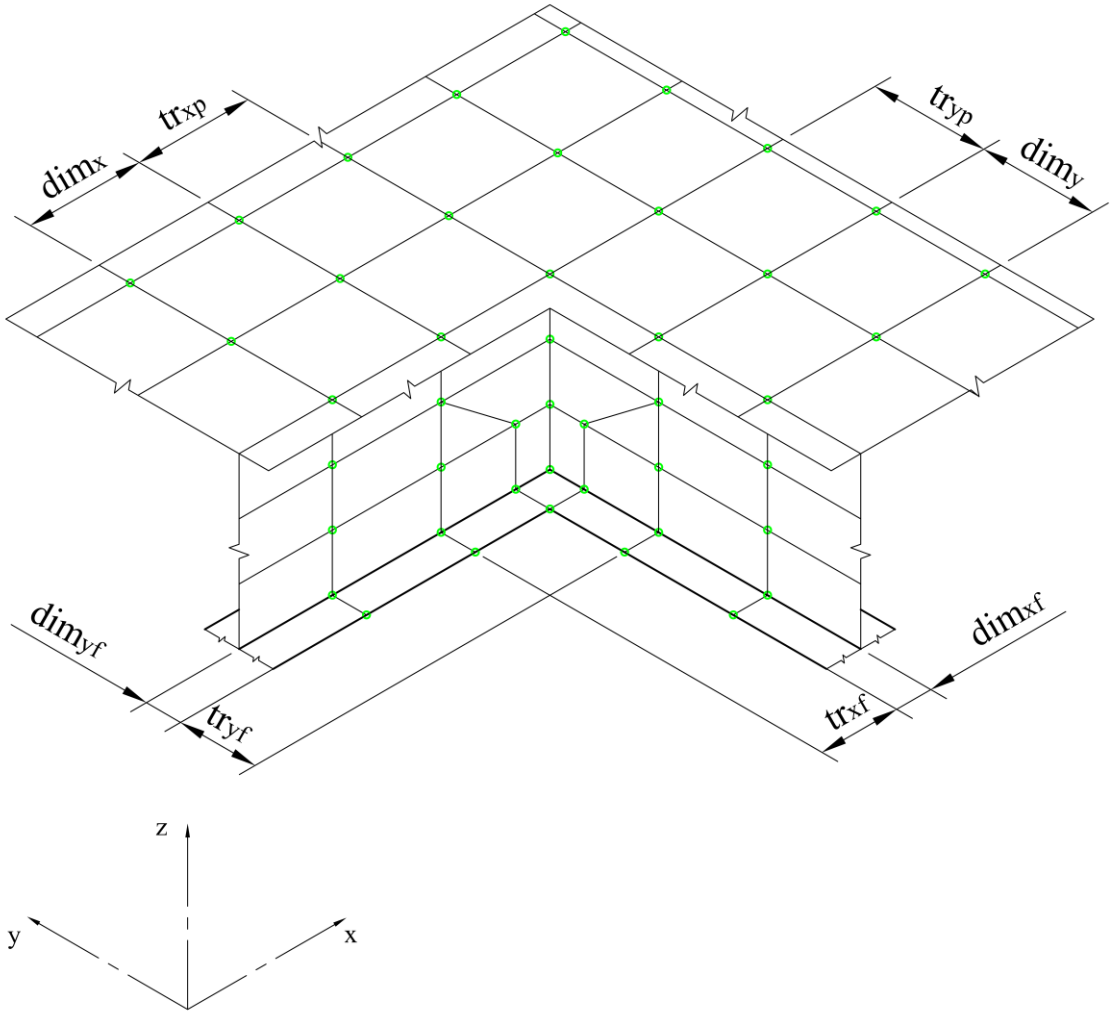


Slika 3.4 Koncept varijante mreže *V1*, dva elementa po širini prirubnice

Prednost varijante *V1* je iznimno robustan algoritam koji uspješno generira vrlo fine mreže konačnih elemenata i nema ograničenje minimalne dimenzije elemenata. Prilikom testiranja na ispitnim varijantama konstrukcije, korištenjem ovog algoritma su generirane mreže sa više od 90000 elemenata i dimenzijama elemenata reda veličine 3/2 *t*. Ograničenje koje se javlja kod izrade ovako finih mreža je vrijeme čekanja za izradu i prikaz, koje može potrajati do nekoliko minuta ovisno o računalnim kapacitetima.

3.2.2 Varijanta mreže V2

Drugom varijantom mreže *V2* se nastoji smanjiti utjecaj preklapanja prirubnica jakih nosača na mrežu oplate tako što se eliminira preslikavanje elemenata prirubnica na oplatu. Na ovaj način varijanta *V2* omogućuje izradu mreže na jakim nosačima koji nemaju konstantnu širinu prirubnice. Postupak određivanja dimenzija elemenata oplate je time značajno jednostavniji i rezultira pravilnijom mrežom opločenja. S druge strane, izrada mreže konačnih elemenata na prirubnicama i strukovima jakih nosača postaje složenija zbog potrebe za prijelazom mreže. Ovaj prijelaz mreže se izvodi kombinacijom deformiranih četvrtastih i trokutastih konačnih elemenata u prijelaznom redu elemenata. Slikom 3.5 je prikazan koncept varijante mreže *V2* na spoju dva jaka T nosača, gdje je vidljiv prijelaz mreže.



Slika 3.5 Koncept varijante mreže *V2*

Ograničenja varijante *V2* su sljedeća:

1) Visine svih jakih nosača moraju biti jednake, a prirubnice im biti u istoj ravnini
2) Broj elemenata po širini prirubnice je ograničen na 1
3) Dimenzije konačnih elemenata imaju donju granicu

Zbog složenosti prijelaza mreže u slučaju kada je broj elemenata po širini prirubnice veći od 1, za varijantu mreže *V2* se usvaja ograničenje od jednog elementa po širini prirubnice. Donja granica dimenzije konačnih elemenata proizlazi iz dimenzija prijelaznih konačnih elemenata prirubnice i širina prirubnica okomitih na promatrani segment. Ovo ograničenje je opisano u poglavlju 6.3.2.

Varijanta mreže *V2* zbog navedenih ograničenja nije prikladna za generaciju fine mreže konačnih elemenata. Prilikom korištenja ove varijante se preporuča zadržavanje inicijalnih kontrolnih parametara, postavljenih prema preporukama *Lloyd's Register* i zajedničkim konstrukcijskim pravilima *IACS Common Structural Rules for Bulk Carriers*. U slučaju izrade finijih mreža konačnih elemenata je potrebna pažljiva provjera izvedenog prijelaznog reda konačnih elemenata. Programski kod sadrži automatske provjere izvedivosti varijante mreže *V2*, koje će zaustaviti program i javiti grešku u predviđenim situacijama kada je poznato da varijanta neće generirati zadovoljavajuću mrežu konačnih elemenata. Međutim u okviru ovog rada nisu provjerene sve moguće permutacije nestandardnih roštiljnih konstrukcija i kontrolnih parametara za izradu mreže, te postoji mogućnost za neočekivane rezultate.

Prednost varijante mreže *V2* je značajno pravilnija mreža i zadovoljavajući aspektni odnos elemenata opločenja kod grubljih mreža konačnih elemenata. Zbog prijelaznog reda elemenata varijanta *V2* također omogućuje izradu mreže konačnih elemenata na jakim nosačima sa različitim širinama prirubnica pojedinih segmenata.

4. IZRADA MKE MODELA SIMETRIČNE KONSTUKCIJE

Na razini globalnog razmatranja roštiljne konstrukcije postoji mogućnost jednostruke ili dvostruke simetrije, što se uzima u obzir prilikom izrade mreže konačnih elemenata. Programski kod unutar klase `ModelCheck` sadrži niz algoritama za automatsko prepoznavanje osi simetrija učitanog modela. S obzirom na globalni parametar osi simetrije AOS (eng. *Axis of Symmetry*) moguće je generirati sljedeće vrste mreža konačnih elemenata:

- Puna - nesimetričan model, nema osi simetrije
- Polovična - postoji uzdužna ili poprečna os simetrije
- Četvrtinska - postoji obostrana simetrija

Automatski prepoznatu os simetrije je moguće poništiti odabirom parametra osi simetrije `axis_of_symm_override` i na taj način izraditi mrežu za željenu vrstu modela. Ovaj parametar može poprimiti sljedeće vrijednosti, za različite osi simetrije:

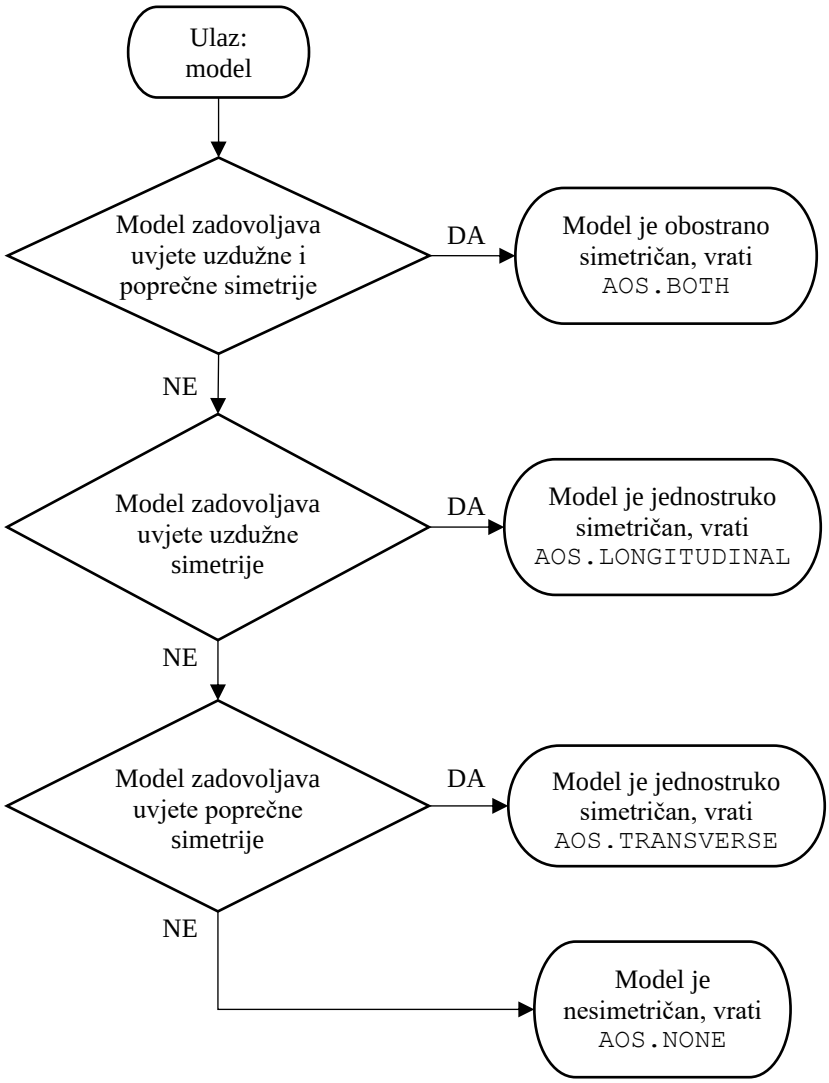
- Uzdužna os simetrije: `AOS.LONGITUDINAL`
- Poprečna os simetrije: `AOS.TRANSVERSE`
- Obostrana simetrija: `AOS.BOTH`
- Nema osi simetrije: `AOS.NONE`

4.1 Algoritam za prepoznavanje osi simetrija modela

Prepoznavanje osi simetrija se temelji na nizu testova kojima se zasebno određuje postoji li uzdužna ili poprečna os simetrije. Ukoliko svi testovi simetrije vraćaju istinitu vrijednost, prepoznata je obostrana simetrija. Alternativno ako samo testovi uzdužne ili poprečne simetrije vraćaju istinitu vrijednost, prepoznata je odgovarajuća jednostruka simetrija. Konačno ako ne prolaze testovi niti uzdužne niti poprečne simetrije, radi se o nesimetričnom modelu. Testovi simetrije uključuju sljedeće provjere:

1.) Relativne koordinate simetričnih jakih nosača
2.) Pozicija centralnog jakog nosača za neparan broj nosača
3.) Debljina lima, karakteristike ukrepa i smjer ukrepa na simetričnim zonama oplate
4.) Svojstva simetričnih segmenata jakih nosača

Metode `long_symmetry_tests` i `tran_symmetry_tests` omogućuju jednostavno dodavanje dodatnih testova simetrije u listu. Kada svi testovi simetrije odgovarajućeg smjera unutar ove liste vraćaju istinitu vrijednost, model je simetričan oko te osi. Dijagram toka konačne metode `assign_symmetry` za identifikaciju simetrije modela prikazan je slikom 4.1.

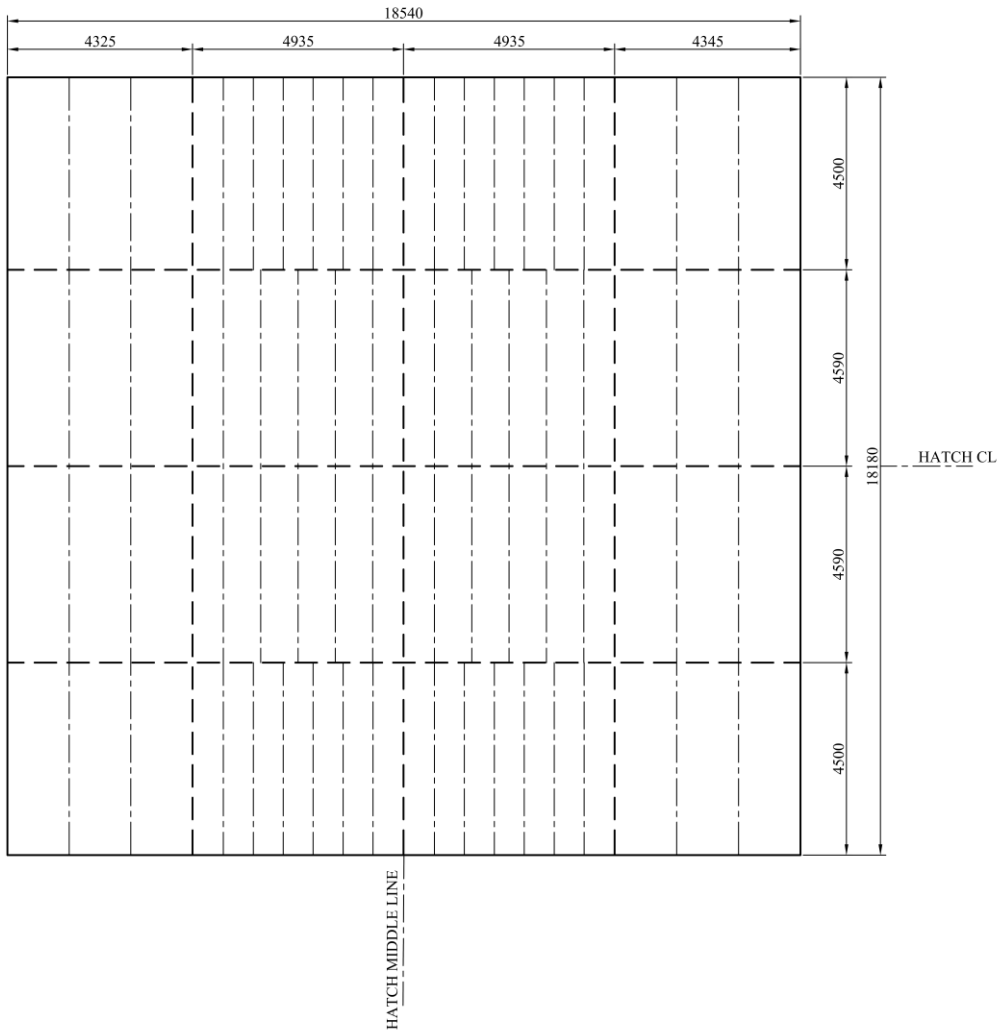


Slika 4.1 Dijagram toka metode `assign_symmetry`

Važno je napomenuti da ručnim odabirom parametra osi simetrije nije moguće generirati mrežu konačnih elemenata uz os simetrije koja na modelu zapravo ne postoji. Ako algoritam za identifikaciju osi simetrije npr. prepozna uzdužnu simetriju, dozvoljen je samo ručni odabir izrade mreže na punom modelu. Kada bi se u ovom slučaju ručno odabrala izrada mreže sa poprečnom osi simetrije, program će vratiti odgovarajuću grešku s porukom da izrada mreže nije moguća za tu ručno odabranu os simetrije.

4.2 Algoritam za provjeru izvedivosti mreže

Metoda `mesh_feasibility` osigurava da na konstrukciji ne postoji nelogično zadan mješoviti sustav orebrenja za koji implementiranim algoritmima nije moguće izraditi mrežu konačnih elemenata. Prema ovoj provjeri, zone oplate između dva susjedna jaka nosača ne smiju imati istu orijentaciju ukrepa paralelnu sa promatranim jakim nosačima i različiti razmak ukrepa. Ukoliko ovaj uvjet nije zadovoljen, metoda prekida program prije određivanja granica izrade mreže konačnih elemenata i vraća odgovarajuću grešku. Primjer varijante konstrukcije sa ovakvim nedopuštenim mješovitim sustavom orebrenja za koju nije moguća generacija mreže je prikazan slikom 4.2.

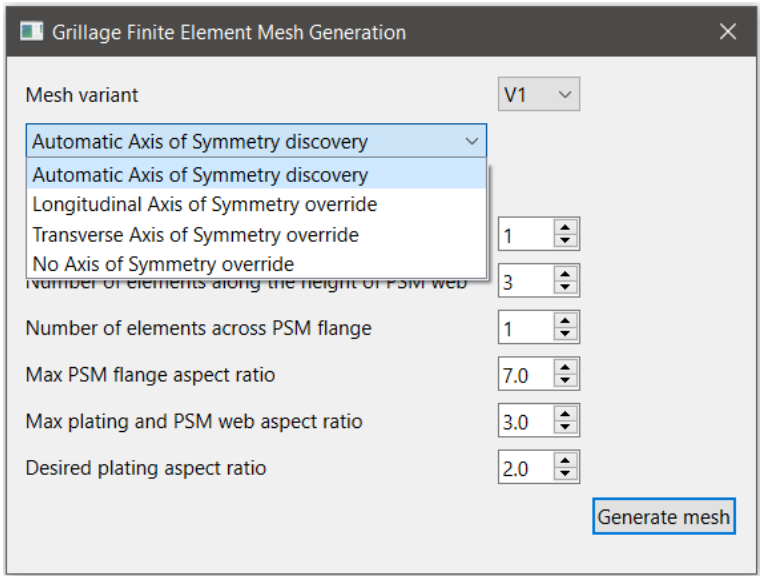


Slika 4.2 Primjer nedopuštenog mješovitog sustava orebrenja

Program će za ovaj primjer nedopuštenog mješovitog sustava orebrenja prepoznati isti smjer i različiti razmak ukrepa, te će vratiti grešku da se oni ne poklapaju na zonama oplate 2 i 6.

5. INICIJALNO ODREĐIVANJE KARAKTERISTIKA MREŽE KONAČNIH ELEMENATA

Kako bi se započeo proračun dimenzija elemenata mreže, potrebno je odrediti na kojim dijelovima konstrukcije će se izrađivati mreža i kakva će ona biti. Klasa `MeshExtent` sadrži niz metoda u kojima su implementirani algoritmi koji određuju granice izrade mreže konačnih elemenata, posebno za zone oplate i posebno za segmente jakih nosača. Ovaj korak određivanja granica mreže ovisi o već spomenutom izbornom parametru `axis_of_symm_override`, koji je moguće odabrati kroz korisničko sučelje za izradu mreže konačnih elemenata prikazano slikom 5.1. Ako ovaj izborni parametar nije zadan, tj. ako je u korisničkom sučelju ostavljena inicijalno odabrana opcija, koristi se automatski prepoznata os simetrije pomoću metode `assign_symmetry`.



Slika 5.1 Odabir osi simetrije u korisničkom sučelju

Klasa `MeshExtent` također sadrži metode koje prepoznaju korištena svojstva ploča i greda na modelu konstrukcije. Na temelju njih se izrađuju nova svojstva materijala, ploča i greda koja će biti korištena prilikom izrade pločastih i grednih konačnih elemenata.

Zbog mogućnosti da se ukrepa nađe na osi simetrije, postoje zasebne metode za izradu polovičnih svojstava grednih elemenata svakog dostupnog tipa profila. Prilikom izrade svojstva pločastih konačnih elemenata za T i L nosače se zasebno razmatraju i izrađuju svojstva za struk i prirubnicu. Za svako jedinstveno svojstvo ploče ili grede koje je korišteno na modelu konstrukcije se izrađuje novo ekvivalentno svojstvo tipa `GeoFEM` za dodjelu pločastim i grednim konačnim elementima.

5.1 Algoritmi za identifikaciju zona oplate s obzirom na osi simetrije

Na razini pojedine zone oplate su moguće sljedeće mogućnosti izrade mreže konačnih elemenata:

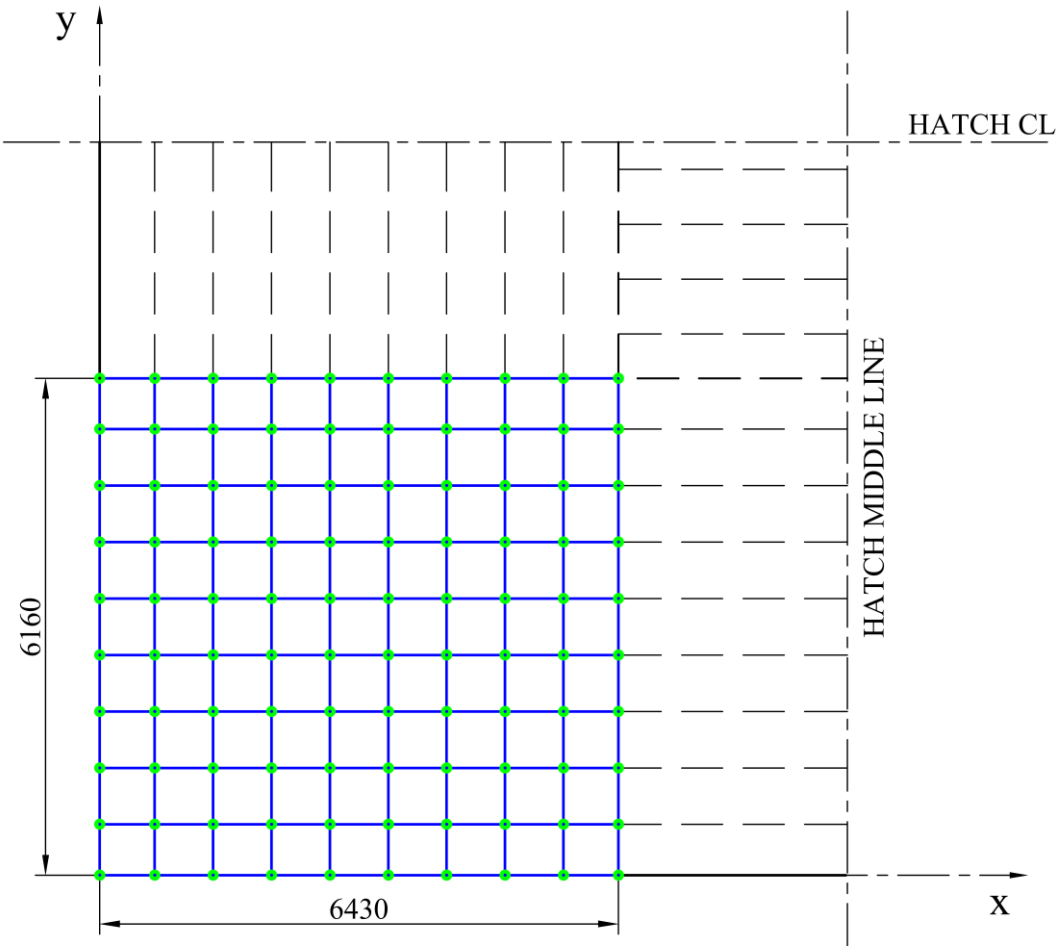
1.) Puna mreža – izrada mreže na cijeloj zoni oplate
2.) Uzdužna polovična – izrada mreže na uzdužnoj polovici zone oplate
3.) Poprečna polovična – izrada mreže na poprečnoj polovici zone oplate
4.) Četvrtinska - izrada mreže na jednoj četvrtini zone oplate

S obzirom na os simetrije modela konstrukcije, moguće su sljedeće permutacije izrade mreže na pojedinoj zoni oplate:

1.) Uzdužna os simetrije (`AOS.LONGITUDINAL`)
    - Zone sa punom mrežom
    - Zone sa uzdužnom polovičnom mrežom
2.) Poprečna os simetrije (`AOS.TRANSVERSE`)
    - Zone sa punom mrežom
    - Zone sa poprečnom polovičnom mrežom
3.) Obostrana simetrija (`AOS.BOTH`)
    - Zone sa punom mrežom
    - Zone sa uzdužnom polovičnom mrežom
    - Zone sa poprečnom polovičnom mrežom
    - Zona sa četvrtinskom mrežom
4.) Nema osi simetrije (`AOS.NONE`)
    - Zone sa punom mrežom

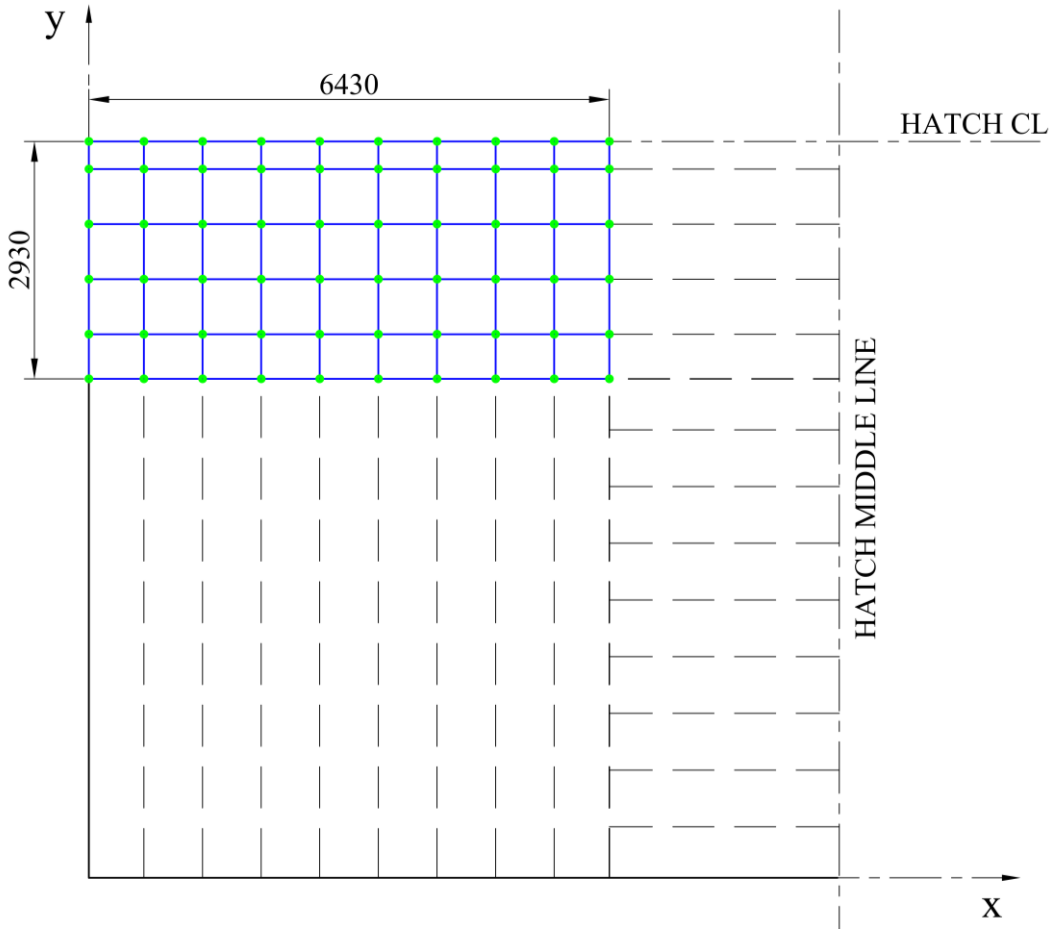
Kako bi se obuhvatile sve moguće kombinacije, izrađeno je sedam zasebnih algoritama za identifikaciju različitih mogućnosti izrade mreže na pojedinoj zoni oplate. Za ispitivanje ovih algoritama je odabrana obostrano simetrična ispitna varijanta konstrukcije `hc_var_4`, koja ima 4 jaka uzdužna i poprečna nosača, a time na polovičnom modelu ova konstrukcija sadrži sve četiri mogućnosti različitih mreža zona oplate. U nastavku će upravo na primjeru četvrtinskog modela ove ispitne varijante biti prikazani različiti slučajevi za varijantu mreže konačnih elemenata *V2* i navedene metode za identifikaciju.

Slika 5.2 prikazuje punu mrežu zone oplate broj 1, na četvrtinskom modelu ispitne varijante `hc_var_4`. U slučaju ovakve obostrano simetrične konstrukcije, zone oplate za izradu pune mreže se identificiraju metodom `identify_both_full_plate_zones`. U slučaju uzdužne simetrije, koristi se metoda `identify_long_full_plate_zones`, a za poprečno simetrične konstrukcije metoda `identify_tran_full_plate_zones`. Svaka od ovih metoda identificiranu zonu oplate sprema u rječnik `full_plate_zones`.



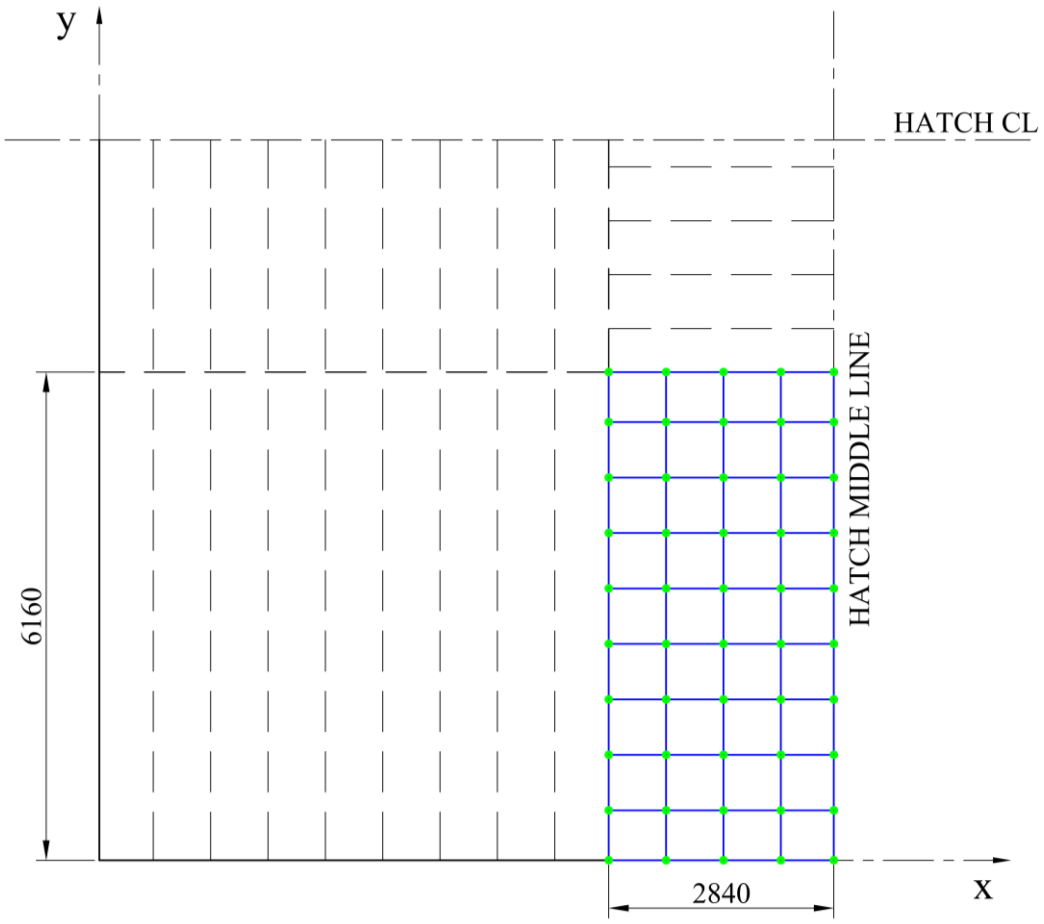
Slika 5.2 Puna mreža zone oplate, četvrtinski model `hc_var_4`

Slika 5.3 prikazuje uzdužnu polovičnu mrežu zone oplate broj 4, na modelu ispitne varijante `hc_var_4`. U slučaju obostrano simetrične konstrukcije, zone oplate za izradu uzdužne polovične mreže se identificiraju metodom `identify_both_half_plate_zones`, dok se za uzdužnu osi simetrije koristi metoda `identify_long_half_plate_zones`. Svaka od ovih metoda identificiranu zonu oplate sprema u rječnik `long_half_plate_zones`. Konstrukcije koje imaju isključivo poprečnu os simetrije ne mogu imati uzdužnu polovičnu mrežu zone oplate, stoga metoda za identifikaciju tog slučaja ne postoji.



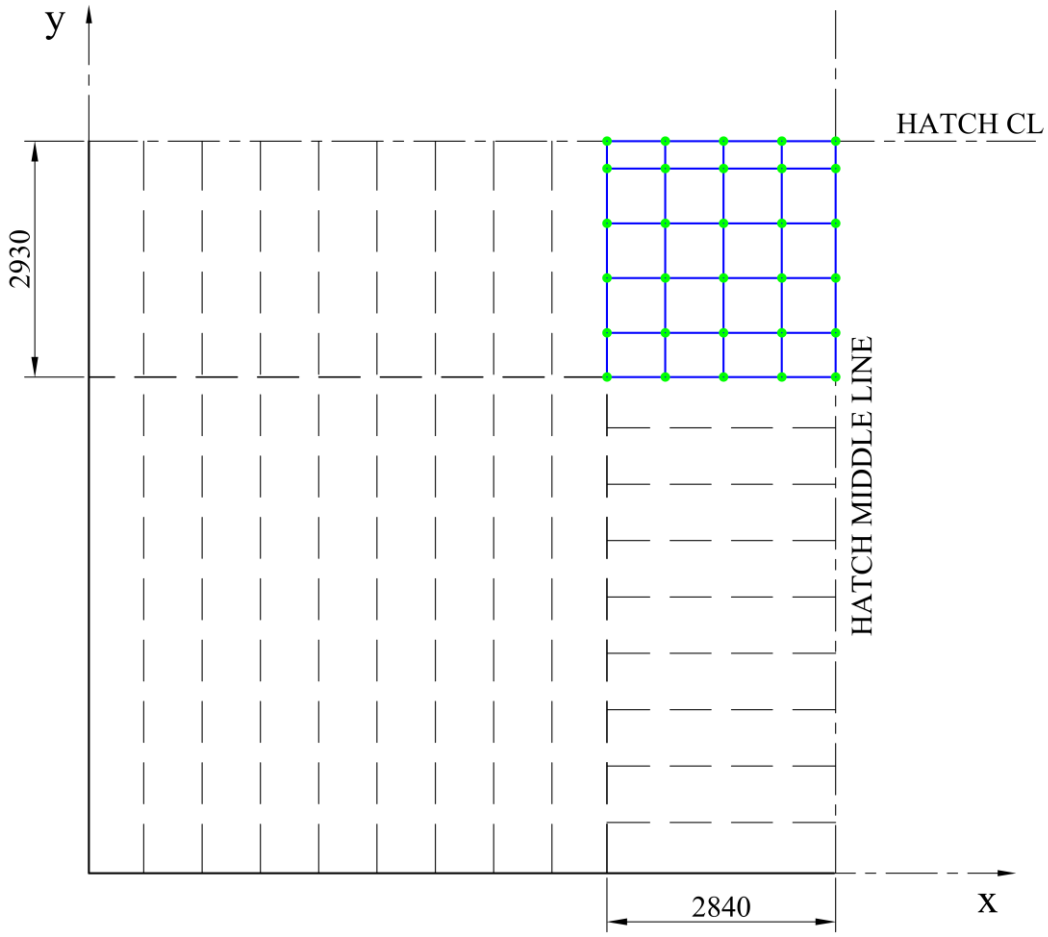
Slika 5.3 Uzdužna polovična mreža zone oplate, četvrtinski model `hc_var_4`

Slika 5.4 prikazuje poprečnu polovičnu mrežu zone oplate broj 2, na modelu ispitne varijante `hc_var_4`. U slučaju obostrano simetrične konstrukcije, zone oplate za izradu poprečne polovične mreže se identificiraju metodom `identify_both_half_plate_zones`, dok se za poprečnu osi simetrije koristi metoda `identify_tran_half_plate_zones`. Svaka od ovih metoda identificiranu zonu oplate sprema u rječnik `tran_half_plate_zones`. Konstrukcije koje imaju isključivo uzdužnu os simetrije ne mogu imati poprečnu polovičnu mrežu zone oplate, stoga metoda za identifikaciju tog slučaja ne postoji.



Slika 5.4 Poprečna polovična mreža zone oplate, četvrtinski model `hc_var_4`

Slika 5.5 prikazuje četvrtinsku mrežu zone oplate broj 5, na modelu ispitne varijante `hc_var_4`. Ovakav tip mreže zone oplate je moguć samo u slučaju obostrane simetrije sa parnim brojem jakih uzdužnih i poprečnih nosača. Metoda za identifikaciju zone oplate sa ovakvom mrežom je `identify_quarter_plate_zone`. Iako je moguće identificirati samo jednu zonu oplate u ovakvom slučaju, ta zona oplate se zbog dosljednosti sa ostalim metodama sprema u rječnik `quarter_plate_zone`.



Slika 5.5 Četvrtinska mreža zone oplate, četvrtinski model `hc_var_4`

Metodom `grillage_plate_extent` se pozivaju sve navedene metode za identifikaciju tipa mreže na pojedinoj zoni oplate, u ovisnosti o automatski prepoznatoj ili odabranoj osi simetrije. Ukoliko model konstrukcije nije simetričan oko niti jedne osi ili je odabrana izrada mreže na cijelom modelu, u rječnik `full_plate_zones` se upisuju sve zone oplate koje se nalaze na modelu unutar rječnika `plating`.

5.2 Algoritmi za identifikaciju jakih nosača s obzirom na osi simetrije

Odabir na kojim jakim nosačima će se izrađivati mreža, obzirom na osi simetrije se temelji na podjeli broja jakih nosača sa brojem dva i cjelobrojnim zaokruživanjem na prvu veću vrijednost. Na ovaj način je osigurano da će izrada mreže obuhvatiti jaki centralni nosač u slučaju neparnog broja jakih nosača. Moguće permutacije koji nosači će biti obuhvaćeni izradom mreže, s obzirom na odabranu os simetrije su sljedeće:

1.) Uzdužna ili obostrana simetrija (`AOS.LONGITUDINAL` ili `AOS.BOTH`)
   - Pola jakih uzdužnih nosača
   - Svi jaki poprečni nosači
2.) Poprečna ili obostrana simetrija (`AOS.TRANSVERSE` ili `AOS.BOTH`)
   - Pola jakih poprečnih nosača
   - Svi jaki uzdužni nosači

Metoda `longitudinal_psm_extent` za slučaj uzdužne ili obostrane simetrije konstrukcije određuje ukupan broj jakih uzdužnih nosača koji su obuhvaćeni izradom mreže, te vraća rječnik sa tim uzdužnim nosačima `longitudinals`. Ako je konstrukcija poprečno simetrična ili nema osi simetrije, metoda vraća cijeli rječnik svih uzdužnih nosača `longitudinal_members`. Analogno, metoda `transverse_psm_extent` za slučaj poprečne ili obostrane simetrije vraća rječnik poprečnih nosača `transversals`, a ako je konstrukcija uzdužno simetrična ili nema osi simetrije, metoda vraća rječnik svih poprečnih nosača `transverse_members`.

5.3 Algoritmi za identifikaciju segmenata s obzirom na osi simetrije

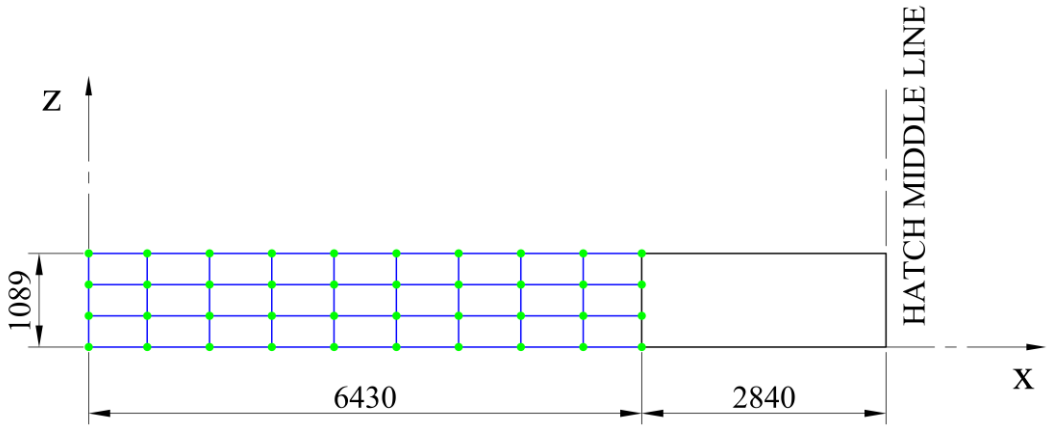
Identifikacija vrste mreže na segmentu je bitan podatak za ispravno određivanje broja konačnih elemenata na strukovima i prirubnicama jakih nosača. Na razini pojedinog segmenta jakog nosača su moguće dvije vrste mreže s obzirom na osi simetrije modela konstrukcije:

1.) Puna mreža – izrada mreže duž cijelog segmenta
2.) Polovična mreža – izrada mreže na polovici duljine segmenta

U ovisnosti o odabranoj ili automatski prepoznatoj osi simetrije, segmenti za izradu pune mreže konačnih elemenata se identificiraju sljedećim metodama:

- Uzdužna simetrija (`AOS.LONGITUDINAL`): `identify_long_full_segments`
- Poprečna simetrija (`AOS.TRANSVERSE`): `identify_tran_full_segments`
- Obostrana simetrija (`AOS.BOTH`): `identify_both_full_segments`
- Nema osi simetrije (`AOS.NONE`): `identify_none_full_segments`

Slika 5.6 prikazuje punu mrežu prvog segmenta jakog uzdužnog nosača na primjeru četvrtinske mreže konačnih elemenata ispitne varijante konstrukcije `hc_var_5`.

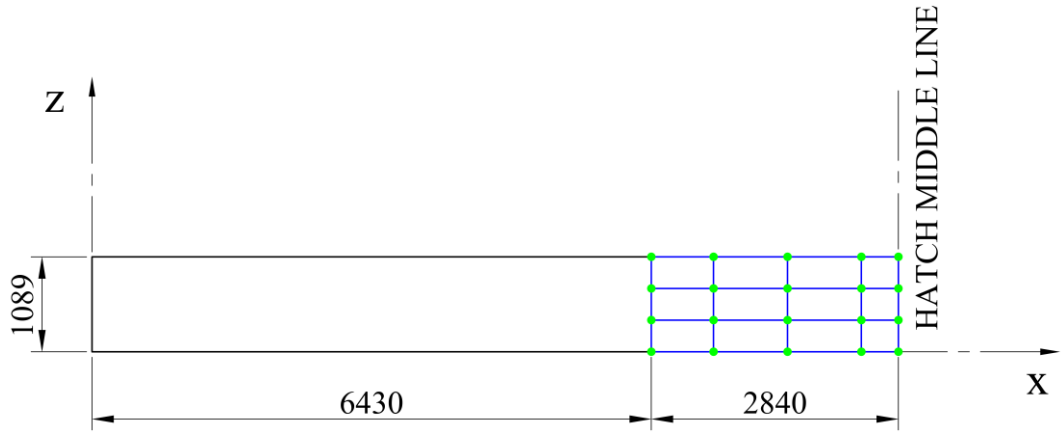


Slika 5.6 Puna mreža segmenta, četvrtinski model ispitne varijante `hc_var_5`

Polovična mreža konačnih elemenata segmenta postoji u slučaju osi simetrije koja je paralelna sa parnim brojem jakih nosača. Segmenti za izradu polovične mreže konačnih elemenata se identificiraju sljedećim metodama:

- Uzdužna simetrija (`AOS.LONGITUDINAL`): `identify_long_half_segments`
- Poprečna simetrija (`AOS.TRANSVERSE`): `identify_tran_half_segments`
- Obostrana simetrija (`AOS.BOTH`): `identify_both_half_segments`

Slikom 5.7 je prikazan primjer polovične mreže drugog segmenta jakog uzdužnog nosača na primjeru četvrtinske mreže konačnih elemenata ispitne varijante konstrukcije `hc_var_5`. Ova ispitna varijanta ima paran broj jakih uzdužnih i poprečnih nosača, pa u slučaju izrade mreže sa obostranom simetrijom postoje uzdužni i poprečni segmenti sa polovičnom mrežom.



Slika 5.7 Polovična mreža segmenta, četvrtinski model ispitne varijante `hc_var_5`

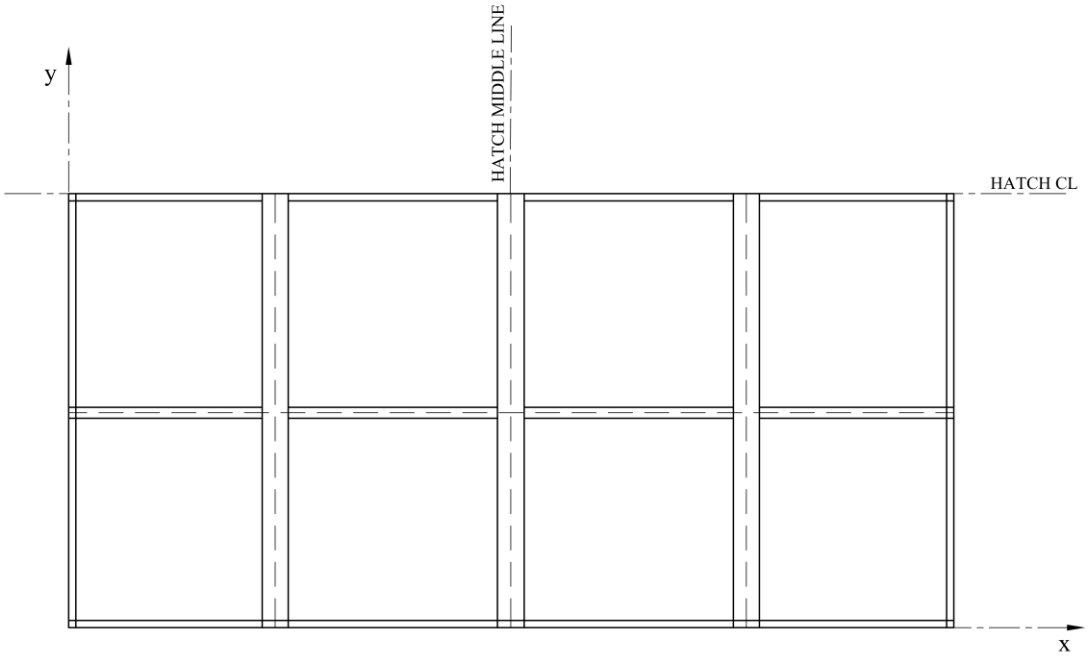
5.4 Algoritmi za prepoznavanje položaja osi simetrije

Položaj osi simetrije u odnosu na elemente konstrukcije modela je bitan podatak za ispravno određivanje broja konačnih elemenata, dodjelu svojstava konačnih elemenata i izradu mreže konačnih elemenata na ispravnoj strani prirubnice jakih nosača. Programski kod unutar klase `MeshExtent` sadrži algoritme koji obuhvaćaju provjeru za svaki od sljedećih slučajeva:

1. Jaki nosač se nalazi na osi simetrije
2. Ukrepa se nalazi na osi simetrije
3. Os simetrije prolazi između ukrepa

5.4.1 Jaki nosač na osi simetrije

U slučaju da se jaki nosač nalazi na os simetrije, konačnim elementima struka tog nosača je potrebno dodijeliti svojstvo sa upola manjom debljinom. Ukoliko je taj jaki nosač T profil, elementi prirubnice se izrađuju na samo jednoj strani. Slika 5.8 prikazuje primjer polovičnog modela ispitne varijante konstrukcije `hc_var_1` sa jakim uzdužnim nosačem na uzdužnoj osi simetrije.

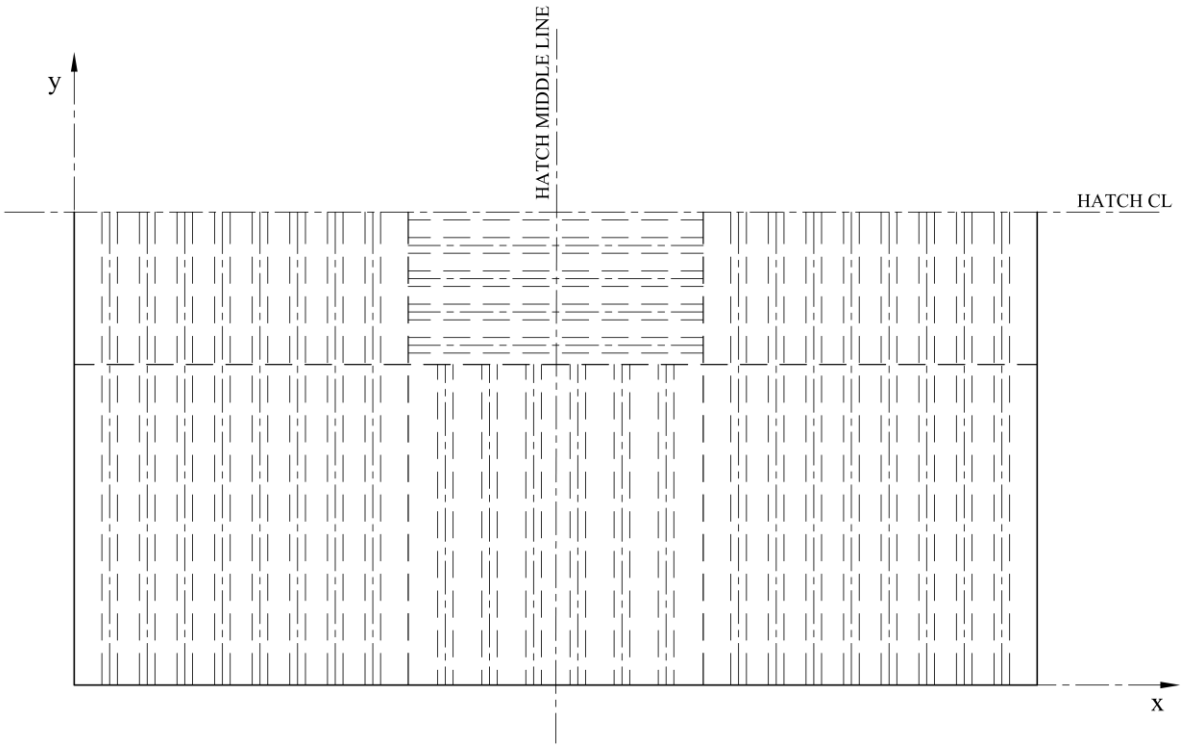


Slika 5.8 Jaki nosač na osi simetrije, polovični model ispitne varijante `hc_var_1`

Metodom `aos_on_segment` se identificiraju ovakvi slučajevi za svaki pojedini segment, usporedbom smjera jakog nosača sa osima simetrije i provjerom nalazi li se jaki nosač na relativnoj koordinati 0.5. Ako se promatrani segment nalazi na osi simetrije, metoda vraća logičku vrijednost istine *True*.

5.4.2 Ukrepa na osi simetrije

U slučaju da se ukrepa nalazi na osi simetrije, grednim konačnim elementima te ukrepe je potrebno dodijeliti pola originalnih karakteristika krutosti. Ovo podrazumijeva izradu novog svojstva grede sa polovičnim karakteristikama poprečne površine presjeka i momenta inercije. Slika 5.9 prikazuje primjer polovičnog modela ispitne varijante `hc_var_5` sa uzdužnom ukrepom, Hat tipa profila, na zoni oplate 5 koja se nalazi na uzdužnoj osi simetrije.



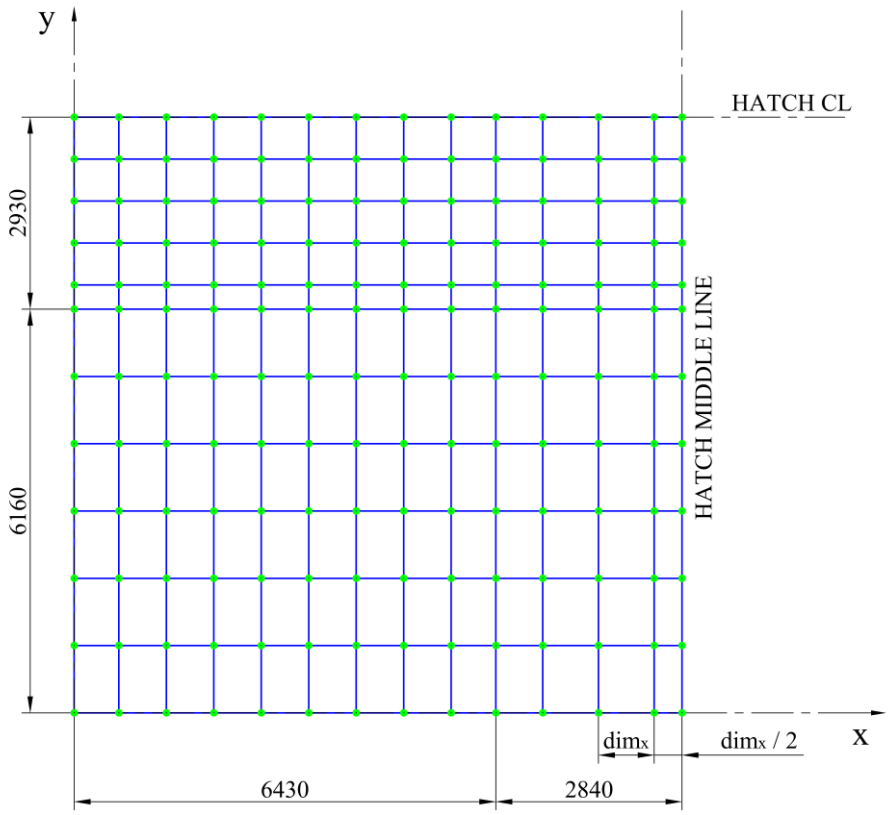
Slika 5.9 Ukrepa na osi simetrije, polovični model ispitne varijante `hc_var_5`

Metodom `aos_on_stiffener` se identificiraju ovakvi slučajevi za svaku pojedinu zonu oplate, usporedbom smjera ukrepa sa osima simetrije i provjerom je li broj ukrepa paran ili neparan. Ako se na promatranoj zoni oplate nalazi ukrepa na osi simetrije, metoda vraća logičku vrijednost istine *True*.

Algoritam ove metode je specifično izrađen s obzirom na moguće vrste definicije rasporeda ukrepa na modelu, prema kojima nije moguće nesimetrično postavljanje ukrepa na zoni oplate. Ukoliko bi postojao neki drugi način zadavanja ukrepa koji omogućuje njihovo nesimetrično postavljanje, ova metoda neće ispravno identificirati ukrepu na osi simetrije. U tom slučaju je potrebna izmjena algoritma koja će omogućiti usporedbu koordinata krajeva pojedine ukrepe sa položajem osi simetrije.

5.4.3 Os simetrije između ukrepa

U slučaju da između dvije ukrepe prolazi os simetrije, postoji mogućnost da se konačni element oplate dijeli na pola zbog te osi simetrije. Slika 5.10 prikazuje ovaj slučaj na primjeru četvrtinske mreže konačnih elemenata na ispitnoj varijanti konstrukcije `hc_var_5`. Element osnovne dimenzije dim_x na zoni oplate 2 se na ovom primjeru zbog poprečne osi simetrije dijeli na pola.



Slika 5.10 Os simetrije između ukrepa, četvrtinski model ispitne varijante `hc_var_5`

Metodom `aos_between_stiffener` se identificiraju ovakvi slučajevi za svaku pojedinu zonu oplate, analogno metodi za određivanje ukrepa na osi simetrije. Ako na promatranoj zoni oplate osi simetrije prolazi između dvije ukrepe, metoda vraća logičku vrijednost istine *True*. Ova metoda je sastavni dio drugih metoda kojima se provjerava dolazi li zaista do podjele konačnog elementa na pola. Koristi se unutar metoda `identify_long_split_zones` i `identify_tran_split_zones` kojima se identificiraju sve zone oplate na kojima postoji ovakav slučaj osi simetrije između ukrepa. Metodama `long_split_element` i `tran_split_element` se tada na temelju dimenzija osnovne mreže konačnih elemenata utvrđuje dolazi li do prepolavljanja elemenata.

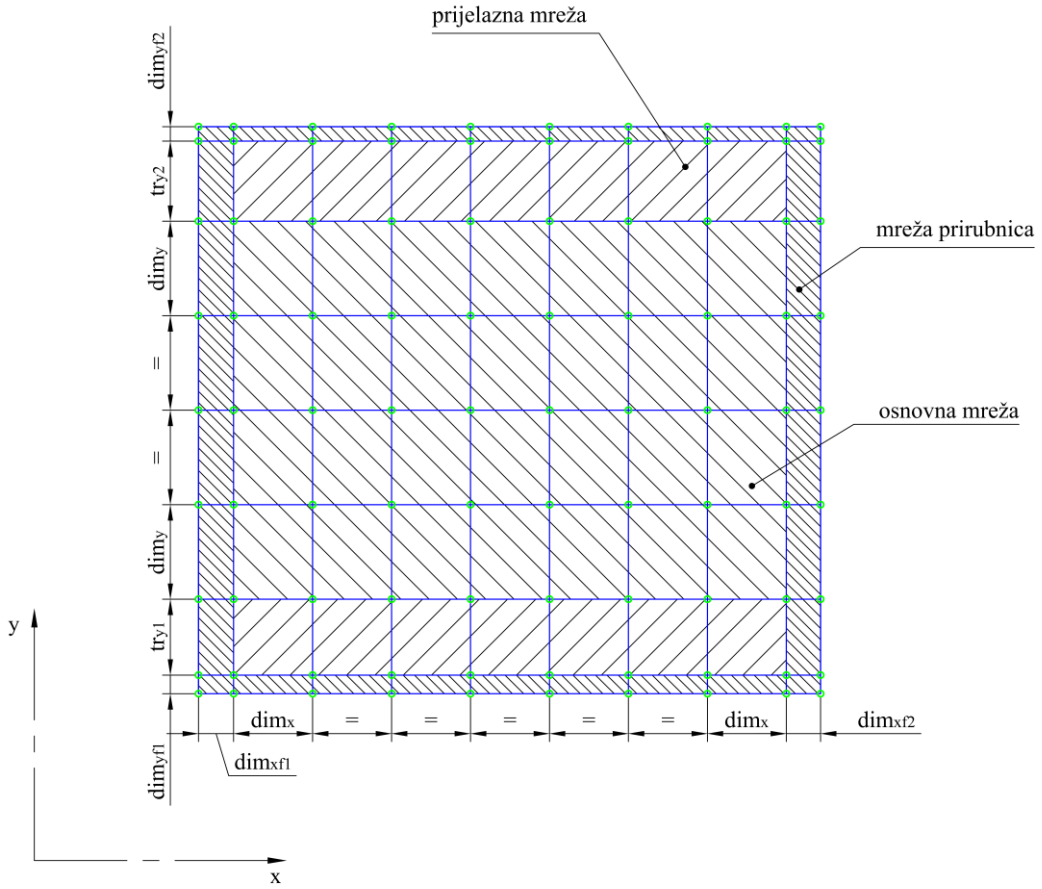
6. ODREĐIVANJE DIMENZIJA KONAČNIH ELEMENATA

Nakon definiranja granica izrade mreže konačnih elemenata je moguće krenuti sa korakom određivanja dimenzija konačnih elemenata na svakom razmatranom dijelu konstrukcije. Glavni cilj ovog koraka je određivanje dimenzija elemenata, koje će osigurati pravilnost mreže i podudaranje čvorova duž rubova svakog razmatranog dijela konstrukcije.

Na svakoj zoni oplate i segmentu se mreža konačnih elemenata može podijeliti na:

- Osnovnu mrežu (eng. *base mesh*)
- Prijelaznu mrežu (eng. *transition mesh*)
- Mrežu prirubnica (eng. *flange mesh*)

Slikom 6.1 je prikazana podjela mreže konačnih elemenata na primjeru jedne zone oplate varijante mreže *V1*.



Slika 6.1 Podjela mreže jedne zone oplate na varijanti mreže *V1*

Metodama unutar bazne klase `MeshSize` se određuju dimenzije osnovne i prijelazne mreže, a mreža prirubnica posljedično slijedi iz tih dimenzija. Za svaku izrađenu varijantu mreže postoje podklase `ElementSize` pomoću kojih se određuju dimenzije konačnih elemenata specifične za pojedino rješenje izrade tj. varijantu mreže. U okviru ovog rada su izrađene dvije varijante mreže, za koje se specifične metode nalaze unutar `ElementSizeV1` i `ElementSizeV2`.

6.1 Kontrola mreže

Dimenzije mreže konačnih elemenata se određuju na temelju niza varijabli koje je prije izrade mreže moguće odabrati, a svaka ima predodređene početne vrijednosti. Vrijednosti ovih varijabli se u toku izvođenja algoritama neće automatski promijeniti, ali ovisno o varijabli neće biti strogo usvojene. Tablicom 6.1 su prikazani dostupni parametri za kontrolu mreže, korištena imena argumenata u programskom kodu, opis i njihove inicijalne vrijednosti.

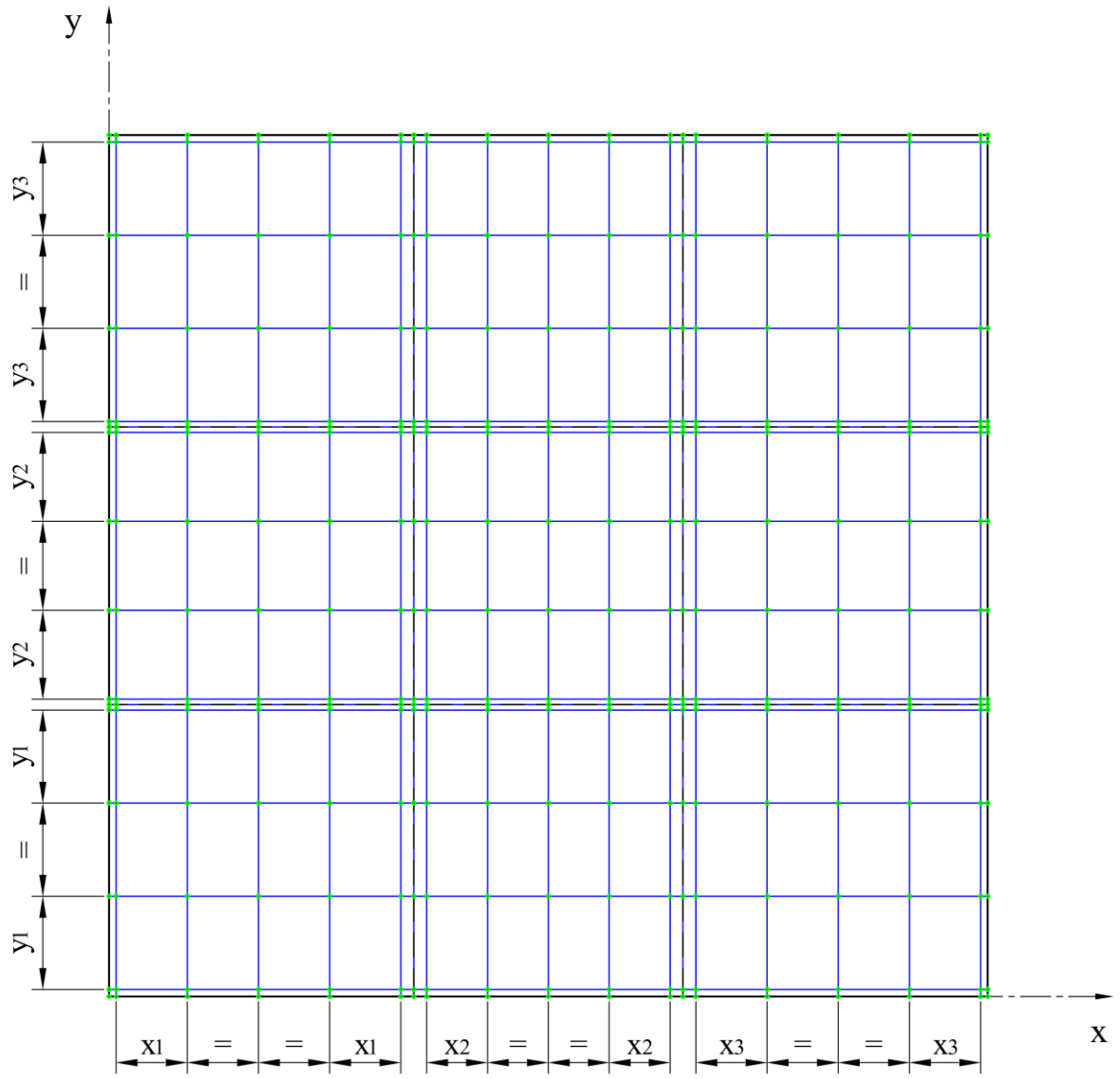
Tablica 6.1 Parametri za kontrolu mreže

Naziv parametra	Opis	Početna vrijednost
`min_num_ebs`	Minimalni broj elemenata između ukrepa (eng. *minimum number of elements between stiffeners*)	1
`min_num_eweb`	Minimalni broj elemenata po visini struka (eng. *minimum number of web elements*)	3
`num_eaf`	Broj elemenata po širini prirubnice (eng. *number of elements across flange*)	1
`flange_aspect_ratio`	Maksimalni aspektni odnos konačnih elemenata prirubnice (eng. *flange aspect ratio*)	7
`plate_aspect_ratio`	Maksimalni aspektni odnos konačnih elemenata oplate (eng. *plate aspect ratio*)	3
`des_plate_aspect_ratio`	Poželjni aspektni odnos konačnih elemenata oplate (eng. *desired plate aspect ratio*)	2

6.2 Osnovna mreža konačnih elemenata

Osnovna mreža konačnih elemenata definira pravilnu mrežu pločastih *quad* elemenata koja se nalazi na većem dijelu mreže oplate, strukova i prirubnica. Dimenzije ove osnovne mreže su određene na temelju kriterija razmaka ukrepa, širina prirubnica, poželjnog i maksimalnog aspektnog odnosa konačnih elemenata opločenja i prirubnica. Osnovna dimenzija konačnog elementa je jednaka na svim zonama oplate u smjeru osi *x* između dva susjedna jaka poprečna nosača, a u smjeru osi *y* između dva susjedna jaka uzdužna nosača.

Slikom 6.2 su na primjeru modificirane ispitne varijante poklopca `hc_var_5` prikazane različite osnovne dimenzije x_1, x_2 i x_3 za svaki stupac, te y_1, y_2 i y_3 za svaki redak zona oplate između jakih nosača.



Slika 6.2 Osnovna mreža konačnih elemenata

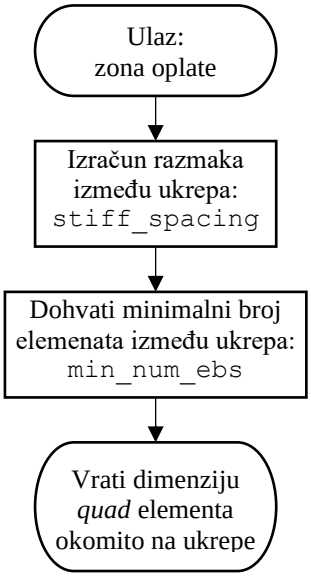
Usvajanjem samo jedne osnovne dimenzije za cijeli redak ili stupac zona oplate između susjednih uzdužnih ili poprečnih nosača se osigurava pravilnost mreže. Ove dimenzije se također prenose na elemente strukova i prirubnica, čime se eliminira mogućnost nepoklapanja rubnih čvorova mreže oplate, strukova i prirubnica. Dimenzije elemenata osnovne mreže se određuju kroz četiri koraka:

1.) Prema kriteriju razmaka između ukrepa na pojedinoj zoni oplate
2.) Prema dimenzijama elemenata prirubnica segmenata koji definiraju zonu oplate
3.) Kombinacijom prva dva kriterija, lokalnim razmatranjem pojedine zone oplate
4.) Globalnim razmatranjem redova i stupaca zona oplate između jakih nosača

Zbog važnosti određivanja dimenzija elemenata prema kojima će biti generirana cijela mreža konačnih elemenata, u nastavku ovog poglavlja su prikazani i objašnjeni korišteni algoritmi u svakom od navedenih koraka.

6.2.1 Dimenzije elemenata oplate prema kriteriju razmaka ukrepa

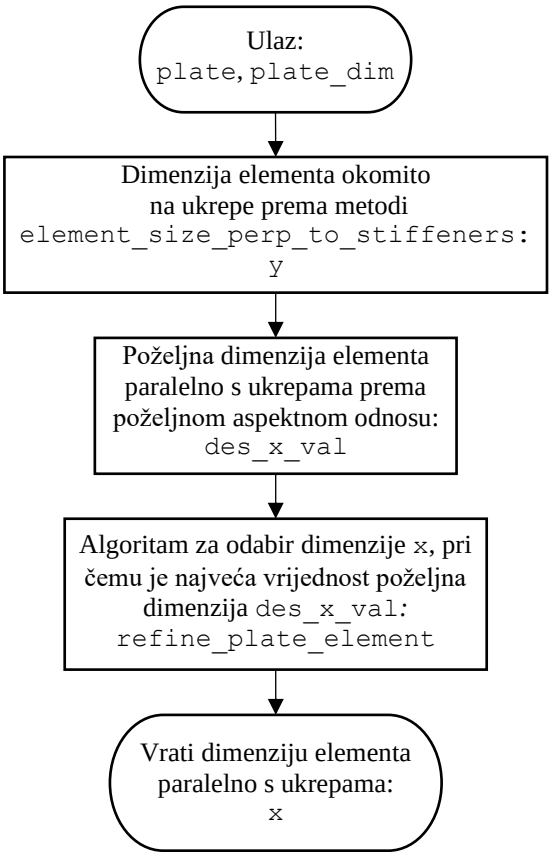
Prvi korak određivanja osnovnih dimenzija *quad* elemenata pojedine zone oplate je prema kriteriju razmaka ukrepa. Unutar programskog koda je izračun podijeljen na više metoda. Prva u nizu je metoda `element_size_perp_to_stiffener`s, za koju je prikazan dijagram toka na slici 6.3, koja definira najveću moguću dimenziju konačnih elemenata okomito na ukrepe isključivo prema minimalnom broju elemenata između ukrepa i razmaku ukrepa. Minimalni broj elemenata između ukrepa `min_num_ebs` prema pravilima *IACS CSR* za veličinu mreže konačnih elemenata iznosi 1 , što je i zadana početna vrijednost.



Slika 6.3 Dijagram toka metode element_size_perp_to_stiffeners

Odabrani minimalni broj elemenata između ukrepa predstavlja donju granicu broja elemenata, a konačni broj elemenata između ukrepa ovisi o odabranim aspektnim odnosima. Poseban utjecaj kod manjih dimenzija širine prirubnica ima odabrani maksimalni aspektni odnos prirubnica `flange_aspect_ratio`. Zbog ovog aspektnog odnosa može doći do profinjenja mreže i povećanja broja elemenata između ukrepa, što je na odabranim ispitnim varijantama konstrukcije bio redovit slučaj.

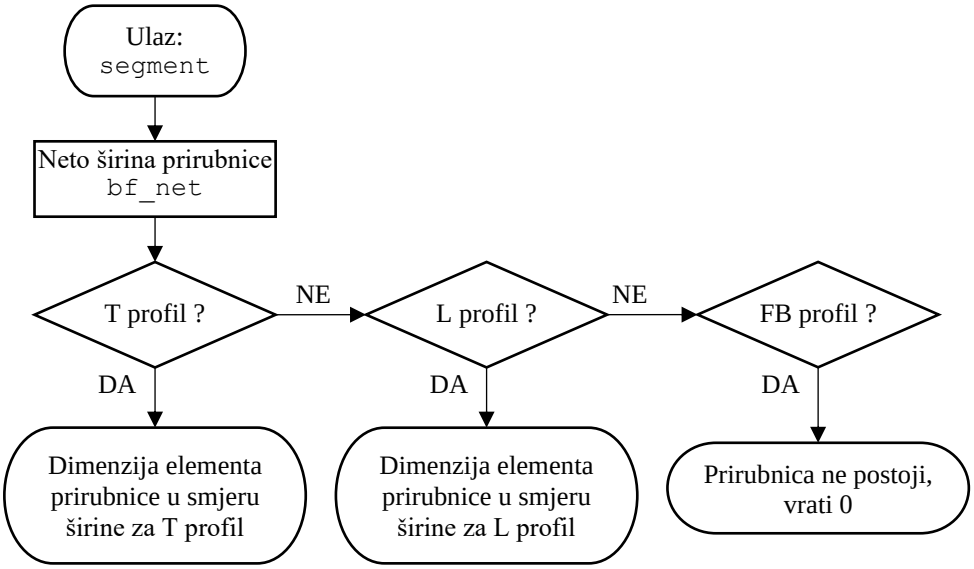
Osnovna dimenzija mreže oplate u smjeru paralelnom s ukrepama se određuje metodom `element_size_para_to_stiffeners`, čiji je dijagram toka prikazan slikom 6.4. Ova metoda koristi algoritam `refine_plate_element` s obzirom na poželjnu dimenziju konačnog elementa `des_x_val`. Ta poželjna dimenzija je definirana kao umnožak poželjnog aspektnog odnosa i dimenzije elementa okomito na ukrepe. Primarna svrha poželjnog aspektnog odnosa je kontrola grubljih mreža konačnih elemenata, kod kojih se zadržava inicijalno postavljena vrijednost od jednog elementa između ukrepa. Ovaj kontrolni parametar gubi svoj utjecaj u trenutku kada zbog ostalih kriterija dođe do povećanja broja elemenata između ukrepa. Tada se dimenzije elemenata određuju prema maksimalnom dopuštenom aspektnom odnosu.



Slika 6.4 Dijagram toka metode `element_size_para_to_stiffeners`

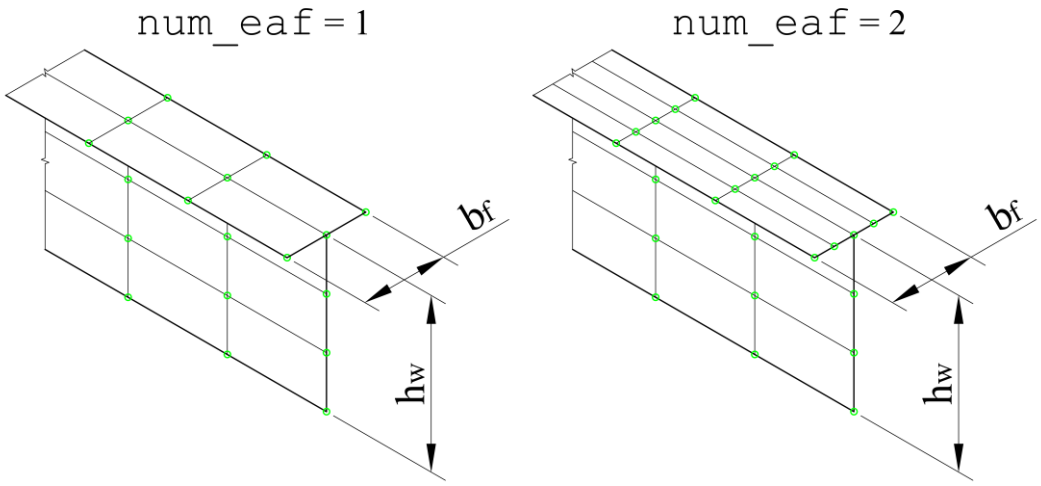
6.2.2 Maksimalne dimenzije elemenata prirubnica prema aspektnom odnosu

Drugi korak određivanja osnovnih dimenzija elemenata je određivanje maksimalnih dopuštenih dimenzija prema širini i aspektnom odnosu prirubnica, za sve segmente koji omeđuju zonu oplate. Metoda `get_flange_el_width` vraća dimenziju *quad* elementa prirubnice u smjeru njezine širine, ovisno o tipu profila nosača. Za uzdužno orijentirane segmente se prema tome određuje dimenzija prirubnice u smjeru globalne osi *y*, a za poprečno orijentirane segmente se određuje dimenzija u smjeru globalne osi *x*. Dijagram toka ove metode prikazan je slikom 6.5.



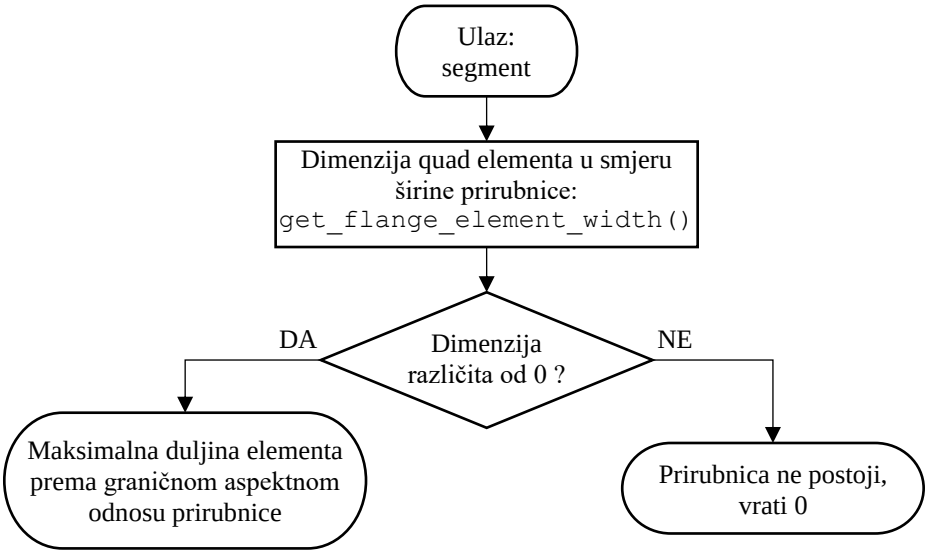
Slika 6.5 Dijagram toka metode `get_flange_el_width`

Prema inicijalno postavljenoj vrijednosti parametra `num_eaf`, prirubnica T profila se diskretizira sa dva elementa u smjeru širine, a prirubnica L profila sa jednim elementom. Izmjenom ovog kontrolnog parametra je moguće odabrati broj elemenata u smjeru širine prirubnice, npr. postavljanjem njezine vrijednosti na 2 će prirubnica T profila biti diskretizirana sa ukupno 4 elementa, kako je prikazano na slici 6.6. Analogno će u tom slučaju prirubnica L profila biti diskretizirana sa 2 elementa. Ako je segment FB profil, metoda će vratiti 0, što je bitno za kasnije provjere i određivanje dimenzija prijelaznih elemenata.



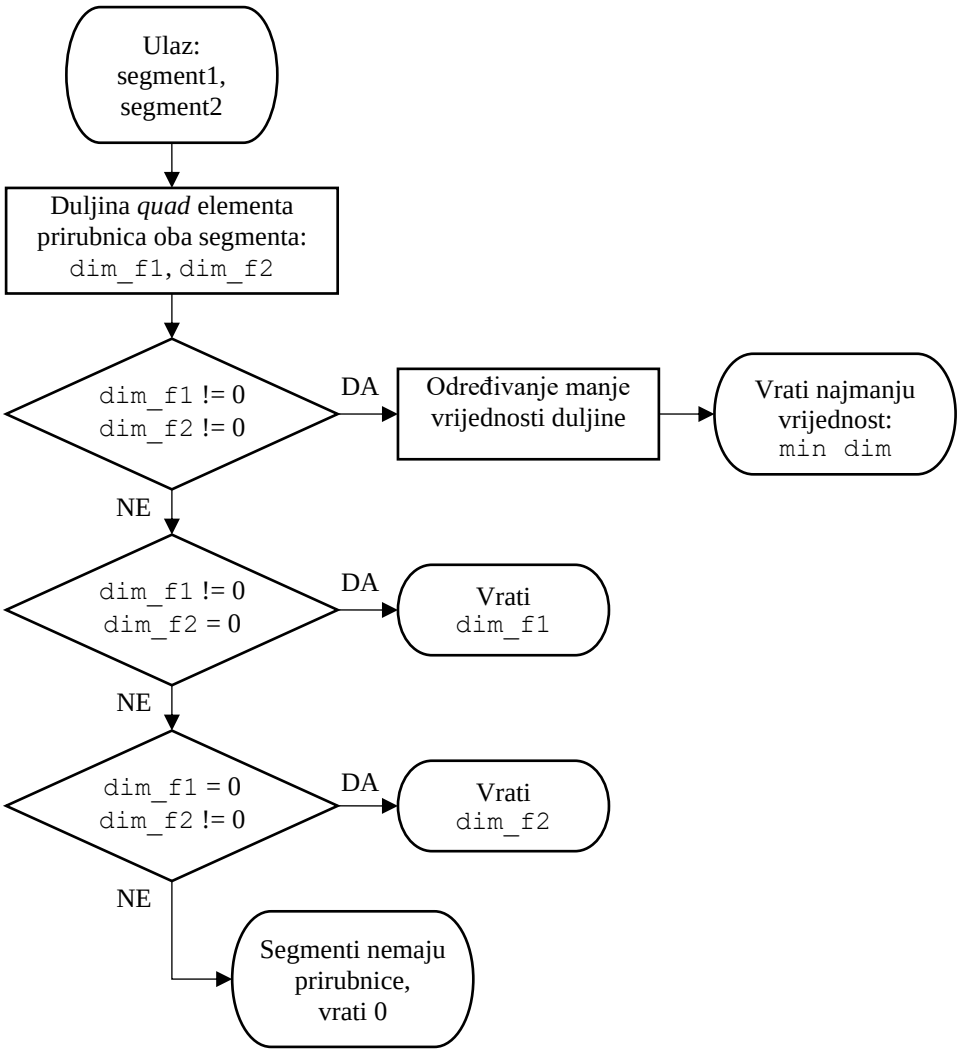
Slika 6.6 Diskretizacija prirubnice T profila

Metoda `get_flange_el_length` vraća maksimalnu dozvoljenu duljinu *quad* elementa prirubnice, prema dimenziji elementa u smjeru širine prirubnice i maksimalnom aspektnom odnosu. Za uzdužno orijentirane segmente prema tome metoda vraća dimenziju u smjeru globalne osi *x*, za poprečno orijentirane segmente dimenziju u smjeru osi *y*. Dijagram toka ove metode prikazan je slikom 6.7.



Slika 6.7 Dijagram toka metode `get_flange_el_length`

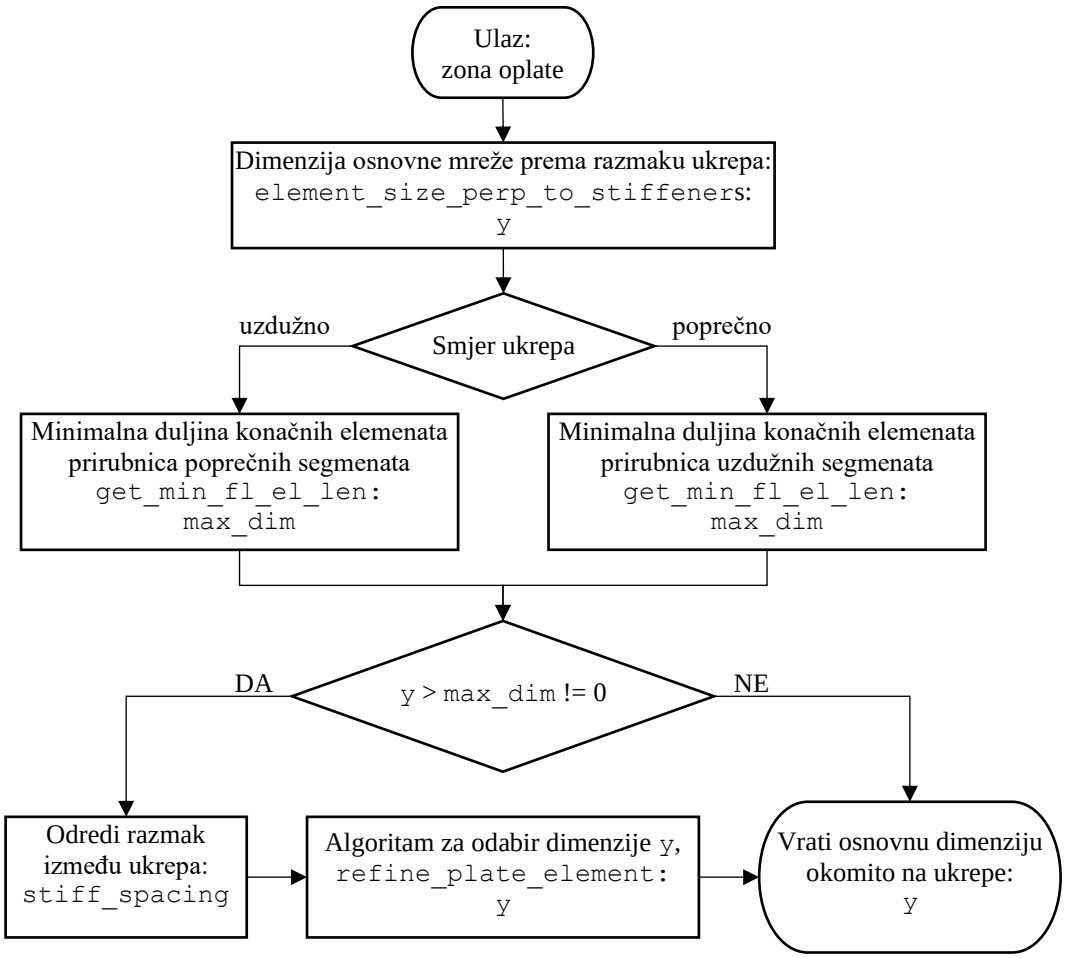
Metoda `get_min_fl_el_len` kao ulazne parametre uzima dva segmenta, za svaki određuje maksimalnu dozvoljenu duljinu *quad* elementa prirubnice i vraća manju vrijednost. Također sadrži provjeru je li segment FB profil, prepoznavanjem je li duljina elementa prirubnice jednaka nuli. Dijagram toka ove metode prikazan je slikom 6.8. Potreba za određivanjem ove minimalne vrijednosti proizlazi iz toga da je svaka zona oplate definirana sa dva uzdužna i dva poprečna segmenta. Najveća dozvoljena dimenzija elementa osnovne mreže je upravo definirana najmanjom vrijednosti duljine elementa prirubnice. Zbog istog razloga je potrebno odrediti minimalnu vrijednost duljine elemenata prirubnice svih segmenata jakih nosača koji se nalaze između dva susjedna jaka nosača u suprotnom smjeru, metodom `get_min_fl_el_len_between_psm`.



Slika 6.8 Dijagram toka metode `get_min_fl_el_len`

6.2.3 Lokalno razmatranje i profinjenje osnovnih dimenzija mreže

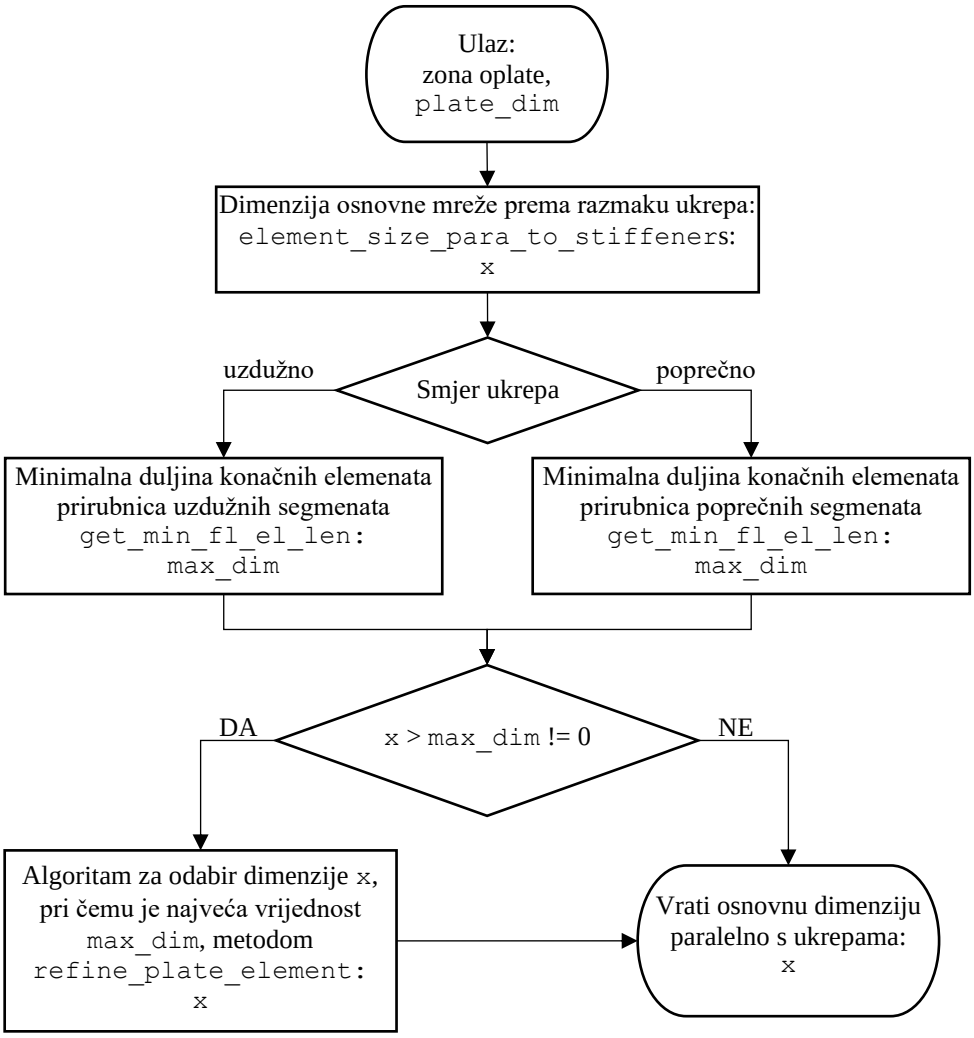
Treći korak određivanja osnovnih dimenzija elemenata je lokalno razmatranje svake zone oplate zasebno. Lokalno razmatranje osnovnih dimenzija elemenata na pojedinoj zoni oplate se provodi kombinacijom kriterija razmaka ukrepa i maksimalnih širina prirubnica, posebnim metodama za smjer okomit i paralelan s ukrepama. Metoda `element_size_plating_zone_perp` za odabranu zonu oplate određuje dimenziju elementa osnovne mreže okomito na ukrepe. Dijagram toka ove metode prikazan je slikom 6.9.



Slika 6.9 Dijagram toka metode `element_size_plating_zone_perp`

Ova metoda za uzdužno usmjerenje ukrepa traži minimalnu vrijednost duljine konačnih elemenata prirubnica prvog i drugog poprečnog segmenta koji definiraju promatranu zonu oplate, dok se za poprečne ukrepe razmatraju uzdužni segmenti. Ta vrijednost predstavlja maksimalnu dozvoljenu dimenziju osnovne mreže okomito na ukrepe. Ako dimenzija `y` prema kriteriju razmaka ukrepa prelazi ovaj maksimum, provodi se profinjavanje mreže na duljini razmaka između ukrepa `stiff_spacing`, metodom `refine_plate_element` pri čemu je najveća dozvoljena vrijednost `max_dim`.

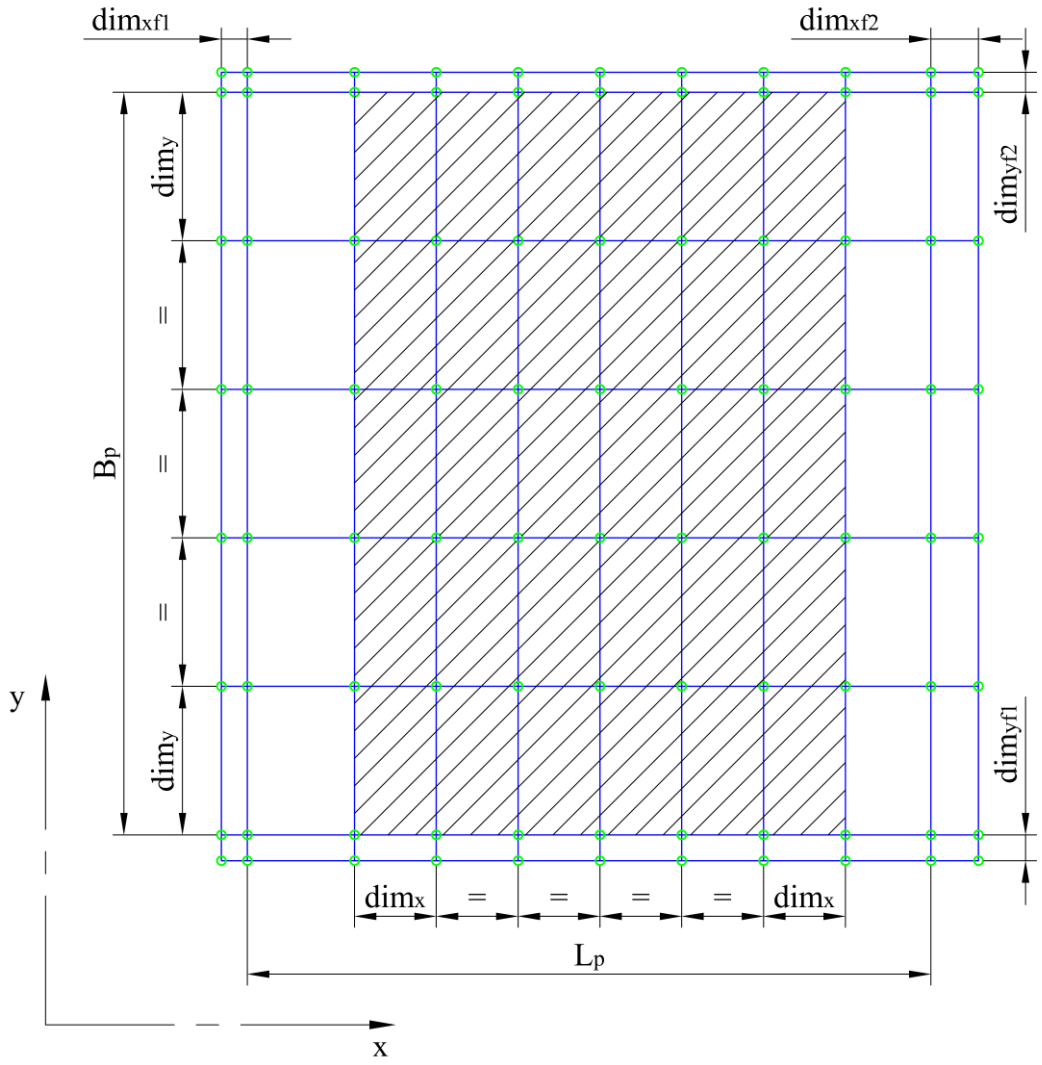
Metoda `element_size_plating_zone_para` za odabranu zonu oplate određuje dimenziju elementa osnovne mreže paralelno s ukrepama. Dijagram toka ove metode prikazan je slikom 6.10.



Slika 6.10 Dijagram toka metode `element_size_plating_zone_para`

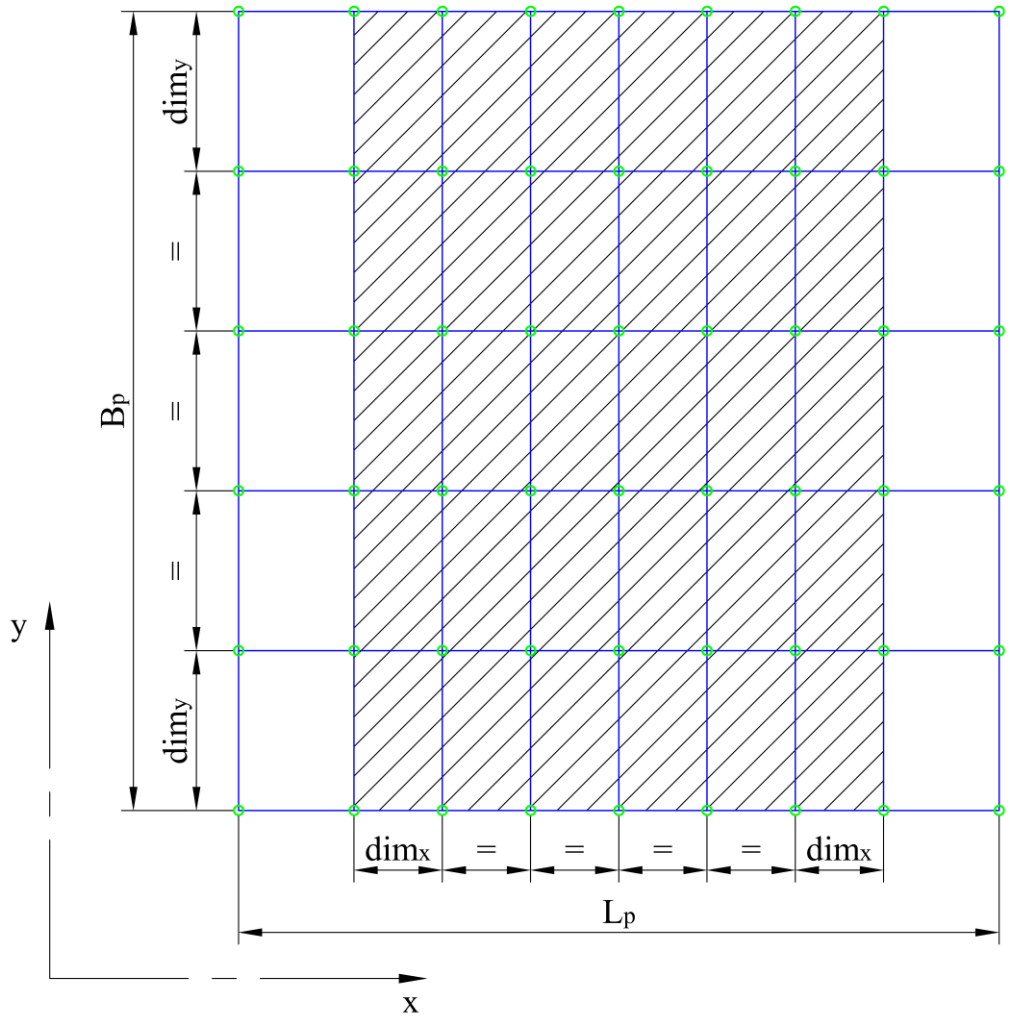
Za razliku od metode za određivanje dimenzije osnovne mreže okomito na ukrepe, ova metoda zahtjeva ulazni parametar `plate_dim`, dimenziju oplate paralelno s ukrepama koja se može razlikovati ovisno o razmatranoj varijanti mreže ili postojanju prirubnica na jakim nosačima. Na ovaj način se omogućuje korištenje metode kao zajedničke za više različitih varijanti mreža konačnih elemenata. Metoda također na ovaj način nastoji proširiti opseg osnovne mreže konačnih elemenata na što je moguće veće područje pojedine zone oplate. Time se za slučaj istog smjera ukrepa na cijelom opločenju eliminira potreba za prijelaznim elementima u smjeru paralelnom sa ukrepama.

Slika 6.11 prikazuje primjer varijante mreže *V1* na jednoj zoni oplate sa poprečnim ukrepama gdje su šrafurom istaknuti konačni elementi osnovne mreže. U ovom slučaju se metodom `element_size_plating_zone_para` određuje dimenzija dim_y, a za ulazni parametar `plate_dim` bi bila usvojena reducirana širina zone oplate dimenzije B_p. Usvajanjem ove reducirane dimenzije se opseg osnovne mreže po *y* osi proširuje sve do konačnih elemenata prirubnica dimenzija dim_{yf1} i dim_{yf2}.



Slika 6.11 Osnovna mreža konačnih elemenata na zoni oplate, varijanta mreže *V1*

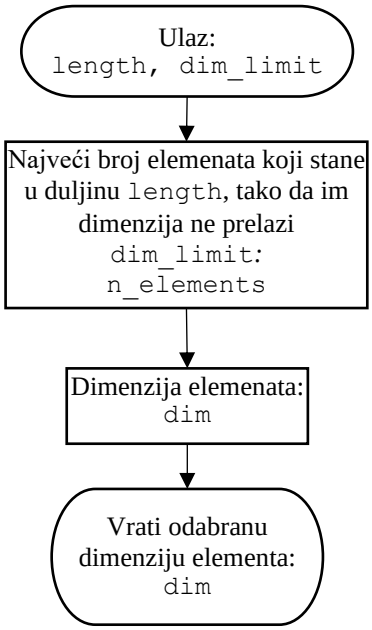
Slika 6.12 slično prikazuje primjer mreže na jednoj zoni oplate sa poprečnim ukrepama, ali za varijantu mreže *V2* gdje su L_p i B_p pune dimenzije promatrane zone oplate. Na ovoj varijanti mreže ne postoji preslikavanje elemenata prirubnice na mrežu opločenja, pa se opseg osnovne mreže konačnih elemenata u smjeru osi *y* proširuje sve do krajnjih rubova zone oplate.



Slika 6.12 Osnovna mreža konačnih elemenata na zoni oplate, varijanta mreže *V2*

Na obje slike 6.11 i 6.12 su vidljivi konačni elementi na rubovima zone oplate čija se dimenzija u smjeru osi *x* razlikuje od ostalih, a na ovim slikama nisu kotirani. Riječ je o prijelaznim elementima koji mogu biti prisutni na svim varijantama mreža, a koji će biti opisani u zasebnom poglavlju 6.3.

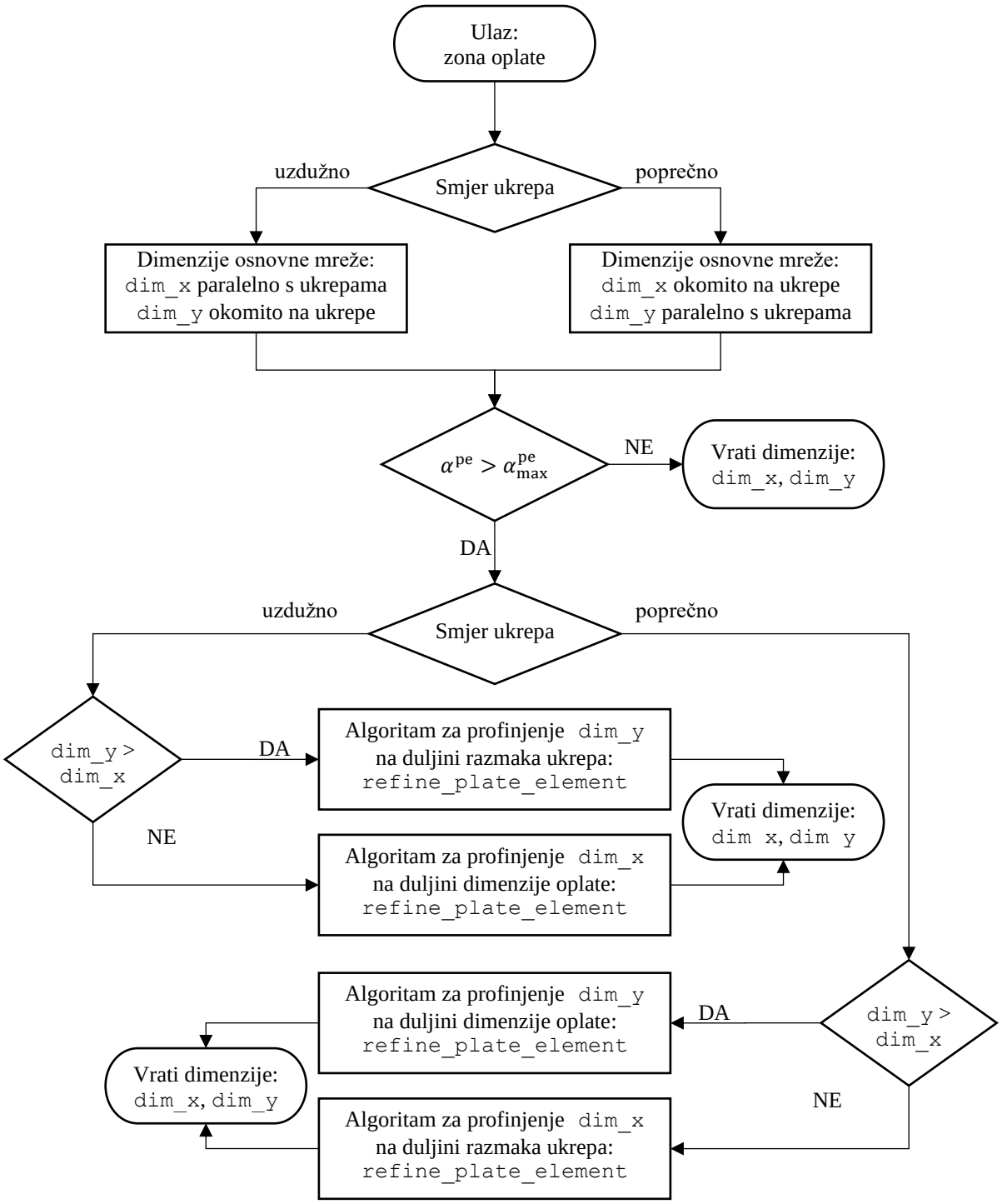
Za profinjenje dimenzija konačnih elemenata oplate se koristi metoda `refine_plate_element`. Ova metoda nastoji odrediti najveći broj konačnih elemenata `n_elements` istih dimenzija koji se mogu smjestiti duž neke duljine `length` tako da im je dimenzija manja od postavljenog maksimuma `dim_limit`. Sa tim brojem elemenata se dijeli duljina `length` i vraća kao odabrana dimenzija konačnog elementa `dim`. Dijagram toka ove metode prikazan je slikom 6.13



Slika 6.13 Dijagram toka metode `refine_plate_element`

Završna metoda za lokalno razmatranje osnovnih dimenzija elemenata na pojedinoj zoni oplate je `element_size_plating_zone`, čiji je dijagram toka prikazan slikom 6.14. Ova metoda spaja metode za izračun dimenzija osnovne mreže okomito i paralelno s ukrepama, te ove vrijednosti dodjeljuje varijablama `dim_x` i `dim_y` sukladno orijentaciji ukrepa. Nakon određivanja ovih osnovnih dimenzija mreže slijedi provjera aspektnog odnosa konačnih elemenata oplate. Ako se odabranim osnovnim dimenzijama mreže ne prelazi maksimalni aspektni odnos elemenata oplate, ove dimenzije se usvajaju za tu zonu oplate. U suprotnom se profinjuje veća dimenzija korištenjem metode `refine_plate_element` prema odgovarajućoj duljini i maksimalnoj vrijednost.

Za slučaj da se profinjuje dimenzija okomito na ukrepe, odgovarajuća duljina koja se dijeli na jednake dijelove metodom `refine_plate_element` je vrijednost razmaka ukrepa. Ako se profinjuje dimenzija paralelno s ukrepama, ova odgovarajuća duljina je jednaka punoj ili reduciranoj dimenziji zone oplate, ovisno o varijanti mreže.



Slika 6.14 Dijagram toka metode `element_size_plating_zone`

6.2.4 Globalno razmatranje i odabir osnovnih dimenzija mreže

Četvrti i zadnji korak određivanja osnovnih dimenzija elemenata je globalno razmatranje cijele konstrukcije. Susjedne zone oplate mogu imati različiti razmak ukrepa, orijentaciju ukrepa i različite širine prirubnica na segmentima jakih nosača koji definiraju te zone. Prema tome svaka zona oplate može zahtijevati različite osnovne dimenzije konačnih elemenata.

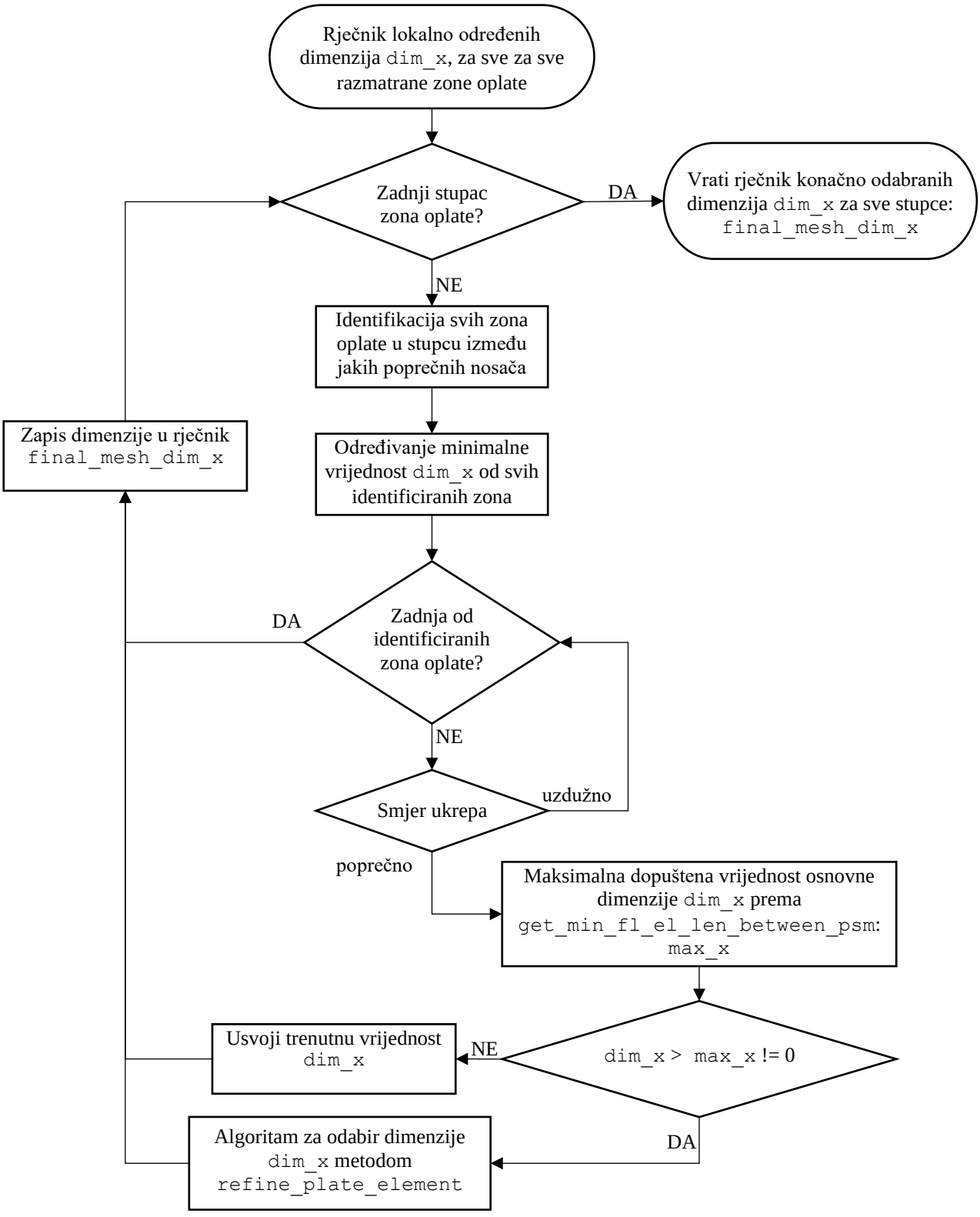
Usklađivanje i odabir osnovnih dimenzija konačnih elemenata opločenja i prirubnica jakih nosača se razmatra zasebno za cijeli redak zona oplate između para susjednih jakih uzdužnih nosača i za cijeli stupac zona oplate između para susjednih jakih poprečnih nosača. Ovakav pristup osigurava da je zadovoljen uvjet kompatibilnosti mreža svih zona oplate prema kojemu susjedne zone oplate duž zajedničkog ruba moraju imati čvorove na istim koordinatama. Dijagram toka algoritma `assign_base_dim_x` za odabir dimenzija osnovne mreže `dim_x` koje će biti usvojene za pojedini stupac zona oplate između parova susjednih jakih poprečnih nosača je prikazan je slikom 6.15. Ovaj algoritam se može podijeliti na dva zasebna slučaja:

1.) Između susjednih poprečnih nosača postoji zona oplate sa poprečnim ukrepama

Ako u cijelom stupcu zona oplate postoji barem jedna zona sa poprečnim ukrepama, odabir osnovne dimenzije `dim_x` je ograničen razmakom ukrepa i odabranim brojem konačnih elemenata između ukrepa, time što dimenzija elementa okomito na smjer ukrepa mora biti višekratnik razmaka ukrepa. Istovremeno vrijednost `dim_x` ne smije prelaziti maksimalnu dopuštenu vrijednost `max_x` koja je određena prema maksimalnom aspektnom odnosu elemenata prirubnica svih uzdužnih segmenata između susjednih jakih poprečnih nosača. Ukoliko se ne prelazi maksimalna dimenzija, usvaja se vrijednost `dim_x` identificirane zone oplate sa poprečnim ukrepama. Ukoliko se prelazi maksimalna dimenzija, metodom `refine_plate_element` se profinjuje `dim_x`, pri čemu je najveća vrijednost `max_x`. U trenutku kada se pronađe prva zona oplate sa poprečnim ukrepama, usvaja se ovako izračunata dimenzija i izlazi iz petlje naredbom *break*.

2.) Između susjednih uzdužnih nosača ne postoji zona oplate sa uzdužnim ukrepama.

Ako u stupcu zona oplate nema niti jedne zone sa poprečnim ukrepama, odabir osnovne dimenzije `dim_x` nije ograničen. U ovom slučaju se usvaja najmanja vrijednost `dim_x` od svih zona oplate u promatranom stupcu zbog mogućnosti da pojedine zone oplate zahtijevaju različitu dimenziju zbog različitih širina prirubnica uzdužnih nosača.



Slika 6.15 Dijagram toka metode `assign_base_dim_x`

Algoritam `assign_base_dim_y` za usklađivanje osnovne dimenzije `dim_y` analogno razmatra zone oplate između susjednih jakih uzdužnih nosača i koristi istu logiku.

6.3 Prijelazna mreža konačnih elemenata

Prijelazna mreža konačnih elemenata definira mrežu pločastih elemenata koji se nalaze na rubnim dijelovima oplate, strukova i prirubnica jakih nosača. Dimenzije elemenata ove prijelazne mreže ovise o orijentaciji i razmaku ukrepa od jakih nosača `stiff_offset`, maksimalnom aspektnom odnosu konačnih elemenata opločenja i prirubnica, dimenziji elemenata osnovne mreže, te o odabranoj varijanti mreže.

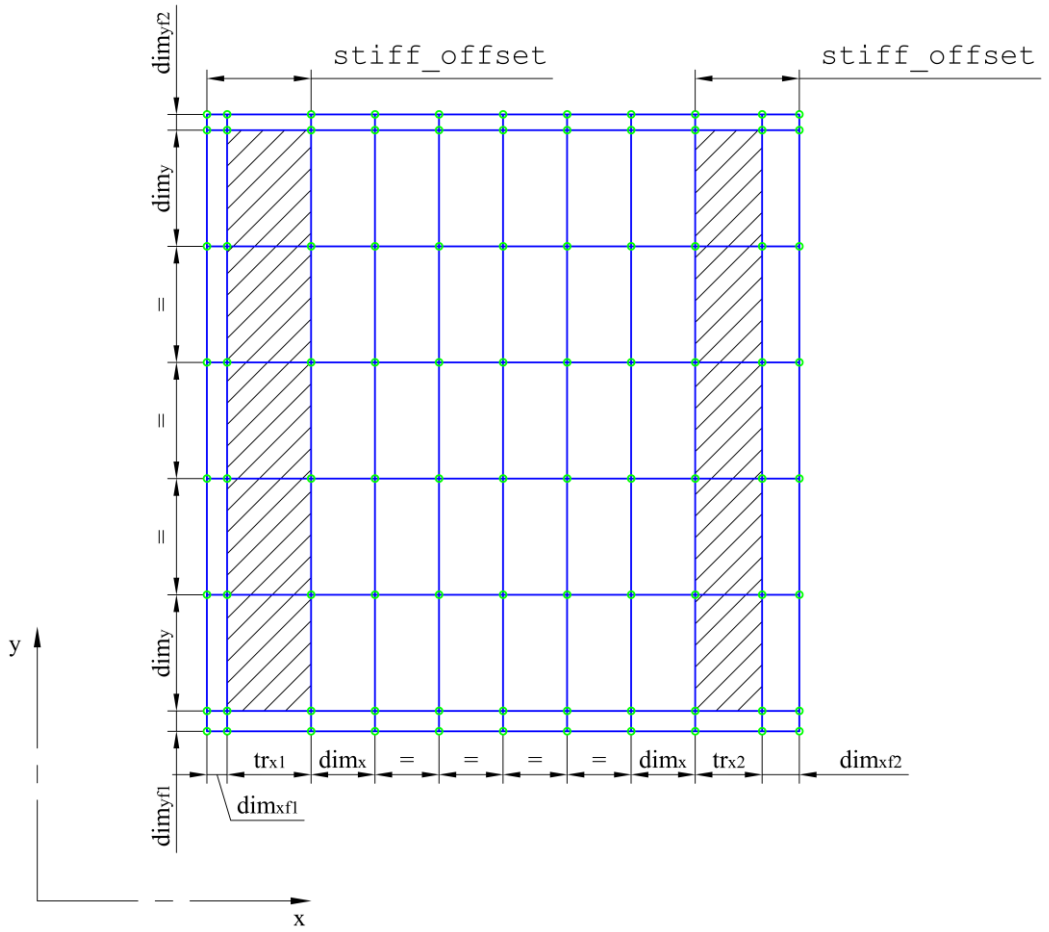
Ako je orebrenje na zoni oplate uzdužno, prijelazni elementi potencijalno postoje samo uz uzdužne segmente zone oplate, jer algoritam za određivanje osnovne dimenzije `dim_x` jednoliko dijeli punu ili reduciranu dimenziju zone oplate. Analogno vrijedi za poprečno orebrenje zone oplate, u kojem slučaju prijelazni elementi mogu postojati samo uz poprečne segmente.

Za razliku od osnovne mreže, svaka zona oplate može imati najviše 4, a najmanje 0 različitih dimenzija prijelaznih elemenata. Zbog bitnih razlika između varijanti mreža *V1* i *V2* su u nastavku za svaku zasebno prikazani primjeri prijelaznih elemenata.

6.3.1 Prijelazni elementi varijante mreže V1

Prijelazni konačni elementi oplate na varijanti mreže *V1* se nalaze između preslikanih elemenata prirubnica i elemenata osnovnih dimenzija. Slika 6.16 prikazuje dimenzije i opseg prijelazne mreže na primjeru varijante mreže *V1* na jednoj zoni oplate sa poprečnim ukrepama, gdje su šrafurom istaknuti prijelazni konačni elementi.

Budući da se na varijanti *V1* konačni elementi prirubnica preslikavanju na opločenje, jednom određene dimenzije prijelaznih elemenata opločenja istovremeno definiraju dimenzije prijelaznih elemenata prirubnica. Ove dimenzije se zasebno određuju u smjeru globalne osi *x* metodom `transition_dim_x` i u smjeru globalne osi *y* metodom `transition_dim_y` unutar podklase `ElementSizeV1`. Slika 6.17 prikazuje dijagram toka određivanja dimenzije prijelaznog elementa `dim_tr_x` unutar metode `transition_dim_x`. Lokalnim razmatranjem pojedine zone oplate varijante mreže *V1*, svaka zona oplate može imati najviše dvije različite vrijednosti prijelaznih elemenata različite od nule.

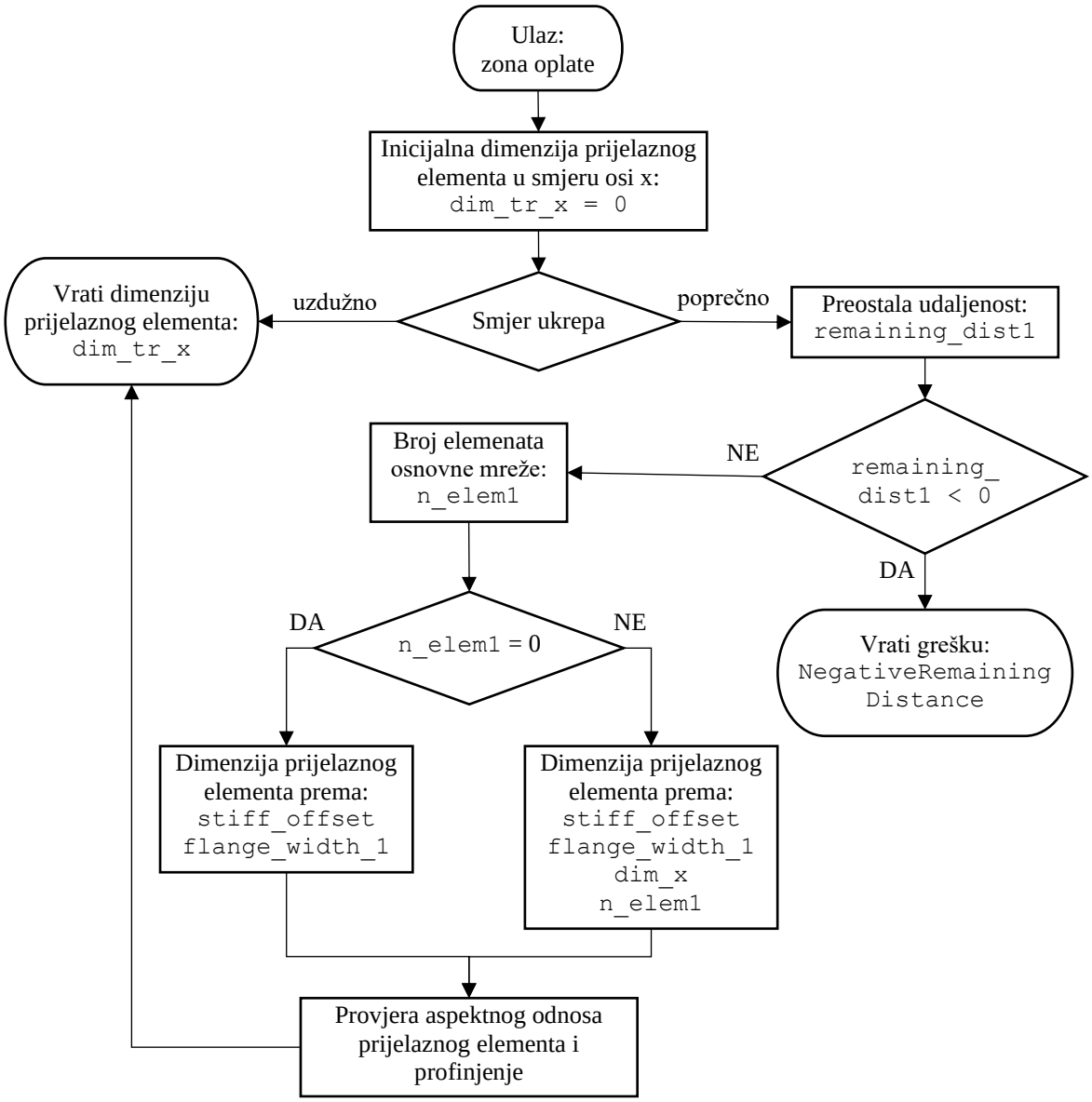


Slika 6.16 Opseg prijelazne mreže konačnih elemenata oplate na varijanti mreže *V1*

Na varijanti mreže *V1* preostala udaljenost `remaining_dist` predstavlja razliku između razmaka prve ukrepe od jakog nosača `stiff_offset` i širine elemenata prirubnice paralelne sa ukrepama. Algoritam `transition_dim_x` određuje broj elemenata osnovne mreže dimenzije `dim_x` koji potpuno stanu u ovu preostalu udaljenost. Ako ne stane niti jedan element, dimenzija prijelaznog elementa se računa kao razlika razmaka `stiff_offset` i širine elemenata prirubnice `flange_width`. Ako stane neki broj elemenata `n_elem`, u izračun dimenzije prijelaznog elementa ulazi i dimenzija osnovne mreže `dim_x` pomnožena sa tim brojem elemenata.

Pretpostavlja se da će razmak ukrepe do najbližeg jakog nosača `stiff_offset` biti veći od poluširine prirubnice T nosača, ili širine prirubnice L nosača, tj. da u tlocrtu prirubnica jakog nosača neće prekrivati ukrepu. U suprotnom algoritam vraća grešku sa porukom da je preostala udaljenost negativna jer dolazi do preklapanja.

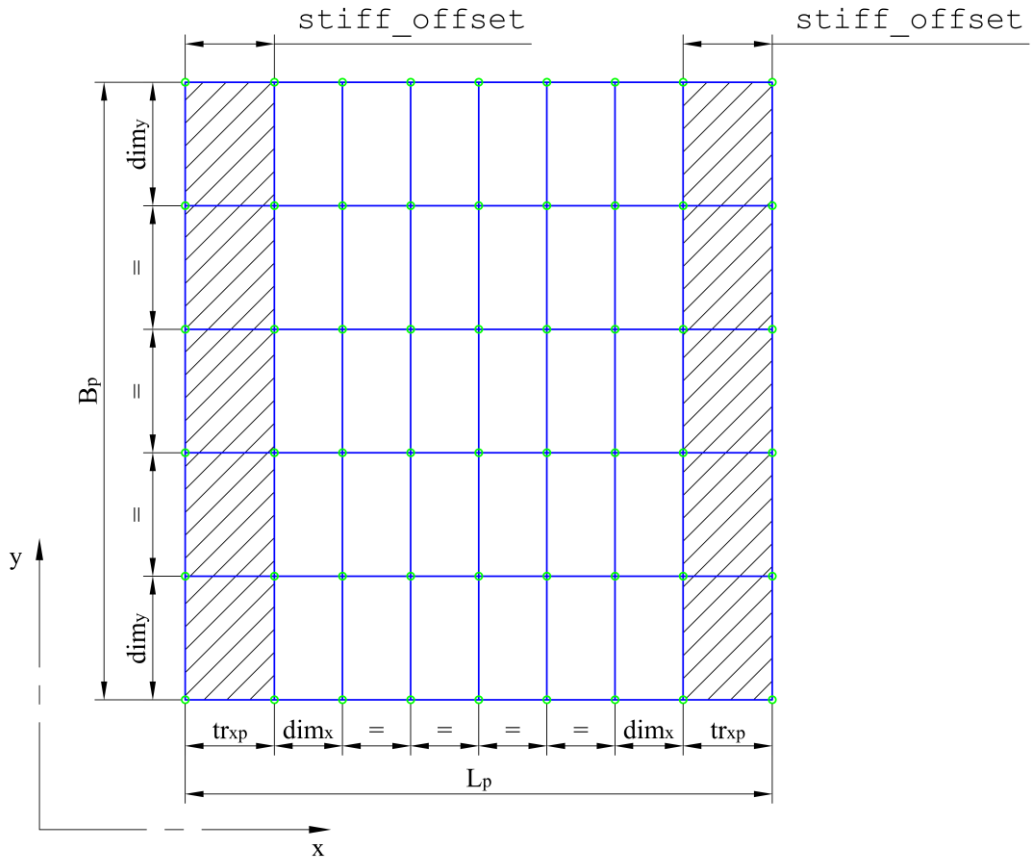
Nakon izračuna dimenzije slijedi provjera aspektnog odnosa prijelaznog elementa prema maksimalnim dopuštenim aspektnim odnosom elemenata oplate $\alpha_{\max}^{\mathrm{pe}}$. U slučaju da se prelazi ovaj aspektni odnos, susjedni konačni element osnovne mreže opločenja se spaja sa prijelaznim elementom, tj. prijelaznom elementu se pridodaje dimenzija `dim_x` osnovne mreže konačnih elemenata. Istovremeno mora biti ispunjen uvjet da je preostala udaljenost veća od osnovne dimenzije mreže, jer u suprotnom može doći do neželjenog spajanja sa susjednim elementom osnovne mreže. Ova provjera aspektnog odnosa pokriva slučaj kada kombinacija dimenzije preostale udaljenosti, broja elemenata `n_elem` i dimenzije osnovne mreže `dim_x` rezultira infinitezimalno malom vrijednosti.



Slika 6.17 Dijagram toka metode `transition_dim_x`

6.3.2 Prijelazni elementi varijante mreže V2

Prijelazni konačni elementi oplate na varijanti mreže *V2* se nalaze između elemenata osnovnih dimenzija i granica pojedine zone oplate ili segmenta. Slika 6.18 prikazuje dimenzije i opseg prijelazne mreže na primjeru varijante mreže *V2* na jednoj zoni oplate sa poprečnim ukrepama, gdje su šrafurom istaknuti prijelazni konačni elementi.



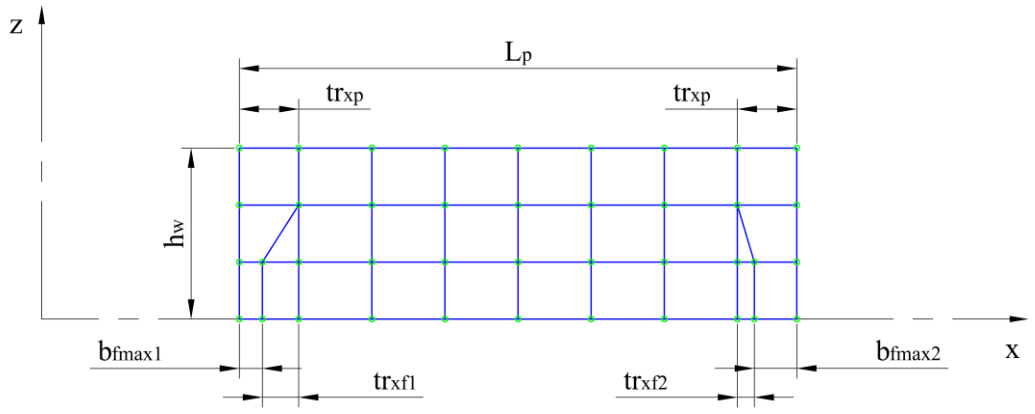
Slika 6.18 Opseg prijelazne mreže konačnih elemenata oplate na varijanti mreže *V2*

Dimenzije prijelaznih elemenata oplate na varijanti mreže *V2* se određuju metodama `transition_dim_x` i `transition_dim_y` unutar bazne klase `MeshSize`. Princip algoritma je jednak kao i kod varijante mreže *V1*, a razlika u odnosu na metode unutar podklase `ElementSizeV1` je ta što u proračun ne ulaze širine prirubnica segmenata koji omeđuju zonu oplate. Lokalnim razmatranjem pojedine zone oplate varijante mreže *V2*, svaka zona oplate može imati najviše jednu vrijednost dimenzije prijelaznog elemenata različitu od nule zbog simetričnog pozicioniranja ukrepa. Ove metode bi postale značajno složenije kada bi na modelu bilo omogućeno nesimetrično postavljanje ukrepa na zoni oplate.

Dimenzije prijelaznih elemenata prirubnice na varijanti mreže *V2* se određuju pomoćnom metodom `transition_dim` koja sadrži proračun za sve moguće slučajeve dopuštene varijantom mreže *V2*. Metoda `flange_transition_dim` zatim koristi ovu pomoćnu metodu i za odabrani segment vraća dimenziju prijelaznih elemenata na njegova oba kraja. Slika 6.19 prikazuje dimenzije prijelaznih konačnih elemenata varijante mreže *V2* na primjeru struka jednog uzdužnog segmenta, u slučaju kada postoje prijelazni elementi oplate. Ovdje je vidljiva razlika između prijelaznih elemenata oplate dimenzija tr_{xp} i prijelaznih elemenata prirubnice na donjem rubu segmenta dimenzija tr_{xf1}, tr_{xf2}. Dimenzije prijelaznih elemenata prirubnice se računaju u odnosu na prijelazne elemente oplate, kada su oni različiti od nule.

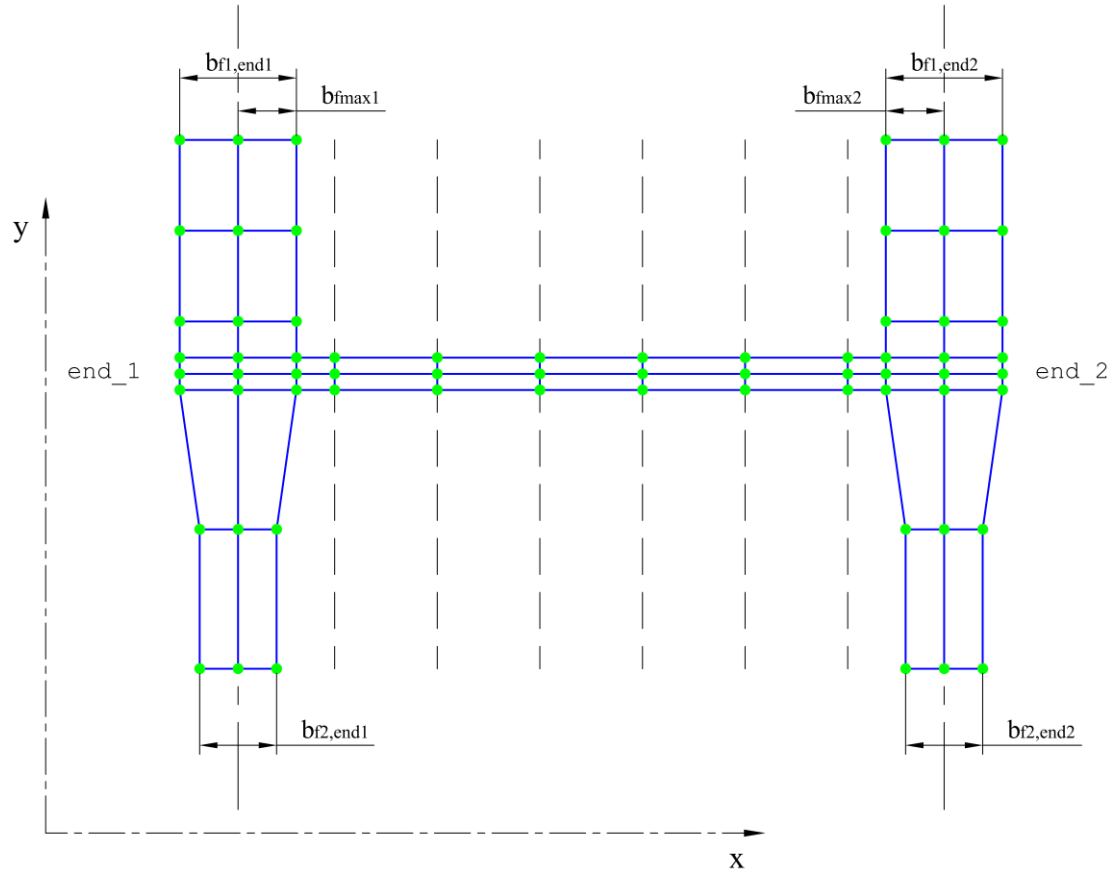
$$tr_{\mathrm{xf1}} = tr_{\mathrm{xp}} - b_{\mathrm{fmax1}} \tag{6.1}$$

$$tr_{\mathrm{xf2}} = tr_{\mathrm{xp}} - b_{\mathrm{fmax2}} \tag{6.2}$$



Slika 6.19 Prijelazni konačni elementi struka segmenta na varijanti mreže *V2*

Bitan podatak za određivanje dimenzija prijelaznih elemenata prirubnice i donjeg reda elemenata struka je širina, odnosno poluširina prirubnica segmenata jakih nosača u smjeru okomitom na promatrani segment. Ove širine prirubnica se određuju metodom `opposite_flange_width` koja vraća najveću širinu ili poluširinu prirubnica za oba kraja promatranog segmenta. Na slici 6.19 su poluširine prirubnica označene sa b_{fmax1} na prvom kraju (`end_1`) i b_{fmax2} na drugom kraju (`end_2`) segmenta. Na ovaj način se omogućuje izrada ispravnog prijelaza mreže za slučaj kada jaki nosači imaju segmente različitih širina prirubnica. Metoda također sadrži provjeru tipa nosača, te vraća poluširinu prirubnice u slučaju da je nosač T tipa, kao što je prikazano slikom 6.20. Ukoliko je nosač L tipa, postoje provjere kojima se određuje treba li dimenzija b_{fmax1} ili b_{fmax2} biti jednaka nuli. Zbog ograničenja varijante mreže *V2* se unutar ove metode ne razmatra broj elemenata po širini prirubnice različit od 1.

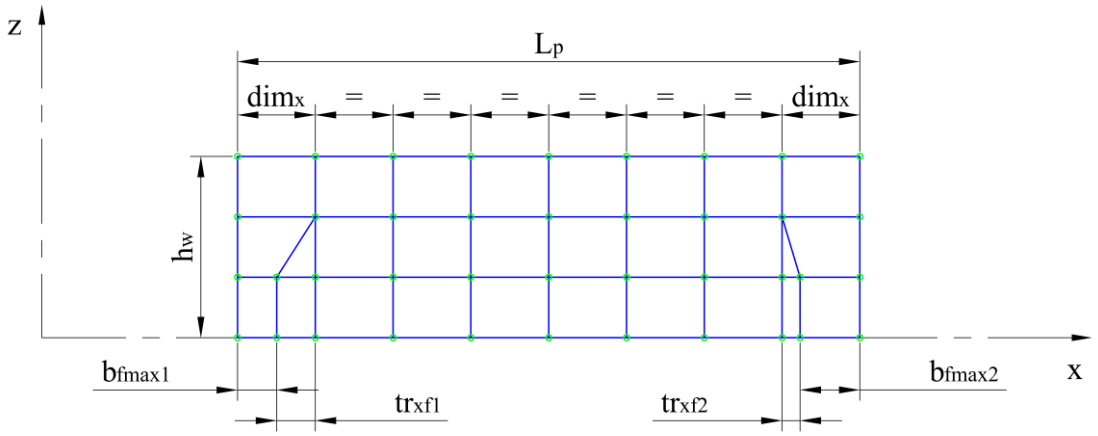


Slika 6.20 Širine prirubnica okomitih na promatrani segment

U slučaju da na pripadajućem retku ili stupcu zona oplate nema prijelaznih elemenata oplate kako je prikazano slikom 6.21, dimenzije prijelaznih elemenata na struku segmenta se računaju u odnosu na veličinu elementa osnovne mreže `dim_x` ili `dim_y`.

$$tr_{\mathrm{xf1}} = dim_{\mathrm{x}} - b_{\mathrm{fmax1}} \tag{6.3}$$

$$tr_{\mathrm{xf2}} = dim_{\mathrm{x}} - b_{\mathrm{fmax2}} \tag{6.4}$$

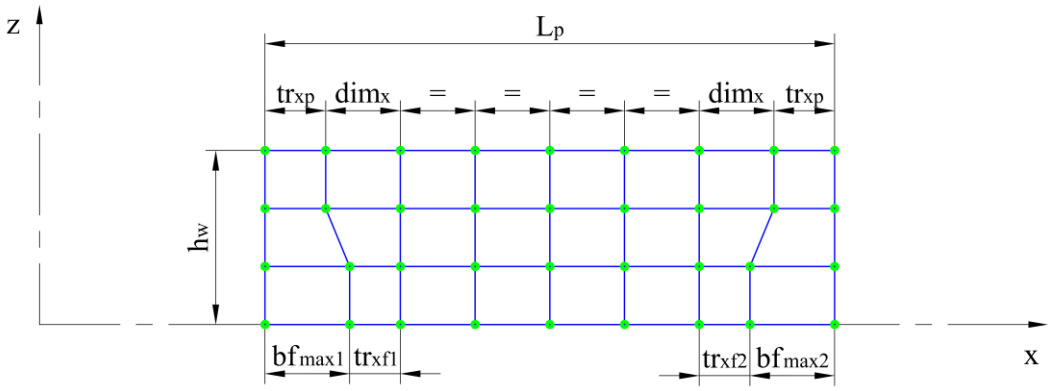


Slika 6.21 Prijelazni konačni elementi struka, bez prijelaznih elemenata oplate

Metodom `transition_dim` su obuhvaćeni i slučajevi kada na prijelaznom redu elemenata nema trokutastih konačnih elemenata zbog odnosa veličina širine prirubnice i dimenzija konačnih elemenata oplate. Slikom 6.22 je prikazan primjer mreže na struku segmenta gdje su širine prirubnica okomitih segmenata veće od prijelaznih elemenata oplate. U ovom slučaju se dimenzije prijelaznih elemenata prirubnice računaju u odnosu na veličinu elemenata osnovne mreže i prijelazne mreže oplate:

$$tr_{\mathrm{xf1}} = tr_{\mathrm{xp}} + dim_{\mathrm{x}} - b_{\mathrm{fmax1}} \tag{6.5}$$

$$tr_{\mathrm{xf2}} = tr_{\mathrm{xp}} + dim_{\mathrm{x}} - b_{\mathrm{fmax2}} \tag{6.6}$$

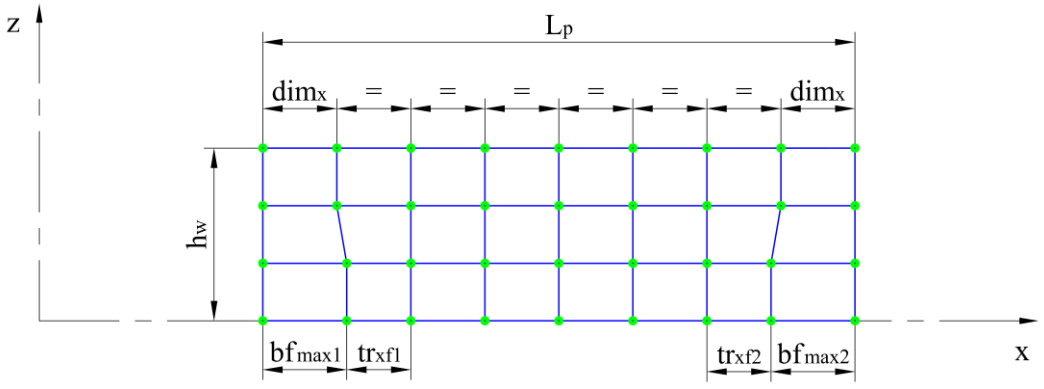


Slika 6.22 Prijelazni konačni elementi struka, bez trokuta

Slikom 6.23 je prikazan primjer mreže na struku segmenta gdje su širine prirubnica okomitih segmenata veće od prijelaznih elemenata oplate, a na oplati nema prijelaznih elemenata. U ovom slučaju se dimenzije prijelaznih elemenata prirubnice računaju u odnosu na veličinu elemenata osnovne mreže.

$$tr_{\mathrm{xf1}} = 2 \cdot dim_{\mathrm{x}} - b_{\mathrm{fmax1}} \tag{6.7}$$

$$tr_{\mathrm{xf2}} = 2 \cdot dim_{\mathrm{x}} - b_{\mathrm{fmax2}} \tag{6.8}$$



Slika 6.23 Prijelazni konačni elementi struka, bez trokuta i prijelaznih elemenata oplate

Granični slučaj koji predstavlja ograničenje izvedivosti varijante mreže *V2* je kada dimenzija prirubnice segmenta okomitog na promatrani prelazi sumu prve dvije dimenzije konačnih elemenata opločenja. Za slučaj kada postoje prijelazni elementi oplate su granice prikazane jednadžbama 6.9 i 6.10, a kada ne postoje jednadžbama 6.11 i 6.12.

$$b_{\mathrm{fmax1}} > tr_{\mathrm{xp}} + dim_{\mathrm{x}} \tag{6.9}$$

$$b_{\mathrm{fmax2}} > tr_{\mathrm{xp}} + dim_{\mathrm{x}} \tag{6.10}$$

$$b_{\mathrm{fmax1}} > 2 \cdot dim_{\mathrm{x}} \tag{6.11}$$

$$b_{\mathrm{fmax2}} > 2 \cdot dim_{\mathrm{x}} \tag{6.12}$$

Metoda `mesh_V2_tr_check` unutar klase `ModelCheck` sadrži kriterije za provjeru ovih graničnih slučajeva i poziva prilagođene greške iz modula `custom_exceptions.py`. Provjera ovih graničnih slučajeva je implementirana u okviru izrade prijelaznog reda elemenata, u koraku određivanja broja trokutastih elemenata unutar metode `identify_num_of_tris`. Kada se identificira granični slučaj, program zaustavlja izradu mreže i vraća odgovarajuću grešku.

6.3.3 Globalno razmatranje dimenzija prijelaznih elemenata oplate

Susjedne zone opločenja mogu imati različito usmjerenje ukrepa budući da se dozvoljava mješoviti sustav orebrenja konstrukcije. Prema tome svaka zona oplate može zahtijevati različite dimenzije prijelaznih elemenata, što je potrebno uskladiti usvajanjem iste dimenzije prijelaznih elemenata za cijeli redak zona oplate između para susjednih jakih uzdužnih nosača i za cijeli stupac između para susjednih jakih poprečnih nosača.

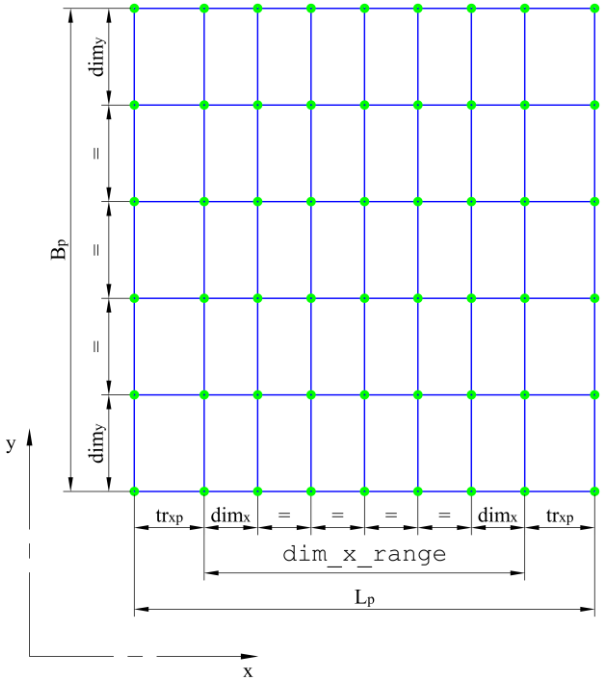
Metoda `assign_transition_dim_x` vraća polje sa svim dimenzijama prijelaznih elemenata u smjeru globalne *x* osi za sve stupce zona oplate obuhvaćene izradom mreže. Prvi redak sadrži dimenziju uz prvi poprečni segment, a drugi redak dimenziju uz drugi poprečni segment zone oplate. Analogno metoda `assign_transition_dim_x` vraća dimenzije u smjeru globalne osi *y* uz uzdužne segmente.

Budući da varijanta mreže *V1* ne dozvoljava različite širine prirubnica duž pojedinog jakog nosača, a prijelazni elementi oplate varijante *V2* ne ovise o širini prirubnica, globalno usklađivanje ovih dimenzija je isključivo na temelju orijentacije ukrepa. Ako se između dva susjedna jaka uzdužna nosača pronađe zona oplate sa uzdužnim orebrenjem, usvajaju se lokalno određene dimenzije tr_{yp} prijelaznih elemenata te zone oplate i izlazi iz petlje naredbom *break*.

U protivnom, ako su sve zone oplate između uzdužnih nosača orebrene poprečno, dimenzija tr_{yp} prijelaznih elemenata će biti 0. Analogno vrijedi i za određivanje dimenzije prijelaznih elemenata tr_{xp} u smjeru globalne osi *x*, između dva susjedna jaka poprečna nosača. U slučaju da prirubnica ne postoji, dimenzija prijelaznog elemenata je isključivo određena prema osnovnim dimenzijama mreže i razmaku ukrepa od jakih nosača `stiff_offset`.

6.4 Broj konačnih elemenata

Broj elemenata na zoni oplate sa dimenzijama osnovne mreže konačnih elemenata se određuje prema rasponu na kojemu se ovi elementi nalaze, metodom `get_base_element_number`. Prilikom određivanja ovog raspona dimenzija se uzima u obzir vrsta mreže na zoni oplate, a prema tome se preuzima puna ili polovična dimenzija zone oplate metodama `get_long_plate_dim` i `get_tran_plate_dim`. Slikom 6.24 je na primjeru varijante mreže *V2* prikazan raspon `dim_x_range` osnovnih dimenzija mreže dim_x u smjeru globalne osi x. Broj elemenata u ovom najjednostavnijem slučaju se određuje dijeljenjem raspona `dim_x_range` sa osnovnom dimenzijom mreže dim_x i zaokruživanjem na cjelobrojnu vrijednost.



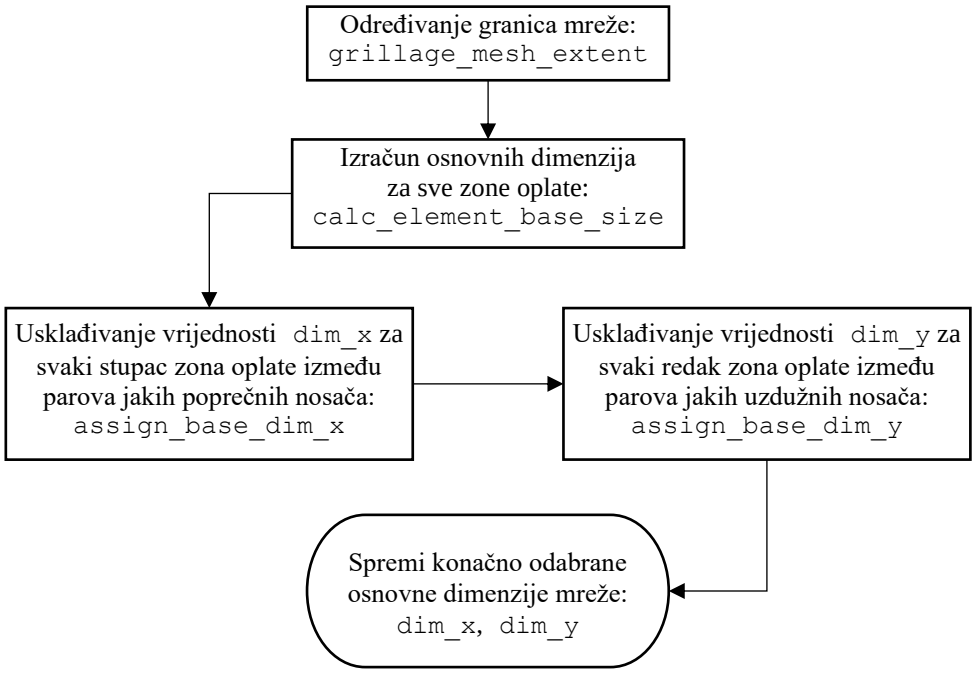
Slika 6.24 Broj konačnih elemenata osnovne mreže oplate

Ako između ukrepa prolazi os simetrije i ta os simetrije prepolavlja konačni element osnovne mreže, rezultat dijeljenja se ne smije odmah zaokružiti zbog ostatka dijeljenja od 0.5 i moguće numeričke greške. Broj prijelaznih elemenata i elemenata prirubnica se temelji na odgovarajućoj dimenziji istog. Ako je dimenzija jednaka nuli, konačni element ne postoji.

6.5 Razmaci između rubnih čvorova

Na temelju izračunatih dimenzija i broja konačnih elemenata osnovne i prijelazne mreže, za svaki element konstrukcije se generira niz razmaka između rubnih čvorova. Za svaku zonu oplate su to dimenzije u smjeru globalnih *x* i *y* osi, a za svaki segment dimenzije u pozitivnom smjeru orijentacije jakog nosača kojemu segment pripada.

Prije određivanja razmaka čvorova je potrebno provesti proračun dimenzija osnovne mreže konačnih elemenata metodom `calculate_mesh_dimensions`. Ova metoda poziva metodu za određivanje granica mreže, na temelju kojih se lokalno određuju dimenzije osnovne mreže metodom `calc_element_base_size`. Te dimenzije su usklađene globalnim razmatranjem konstrukcije metodama `assign_base_dim_x` i `assign_base_dim_x`, nakon čega se spremaju u rječnike `mesh_dim_x` i `mesh_dim_y`. Na ovaj način se u idućim koracima izbjegava ponovni izračun osnovnih dimenzija i višestruko izvršavanje brojnih petlji. Dijagram toka metode `calculate_mesh_dimensions` prikazan je slikom 6.25.



Slika 6.25 Dijagram toka metode `calculate_mesh_dimensions`

Metode `plate_edge_node_spacing_x` i `plate_edge_node_spacing_y` služe za izradu niza razmaka rubnih čvorova svake zone oplate. Za pojednostavljenje programskog koda se koristi pomoćna metoda `save_node_spacing`, koja vrijednost razmaka sprema odabrani broj puta u rječnik. Ukoliko npr. elementi prirubnice ili prijelazni elementi ne postoje, njihov broj je jednak nuli, a time je i odabrani broj unosa u rječnik jednak nuli. U ovom slučaju će početna i krajnja vrijednost raspona biti jednaka, a tada funkcija `range` rezultira praznom listom [11] i metoda `save_node_spacing` preskače unos u rječnik.

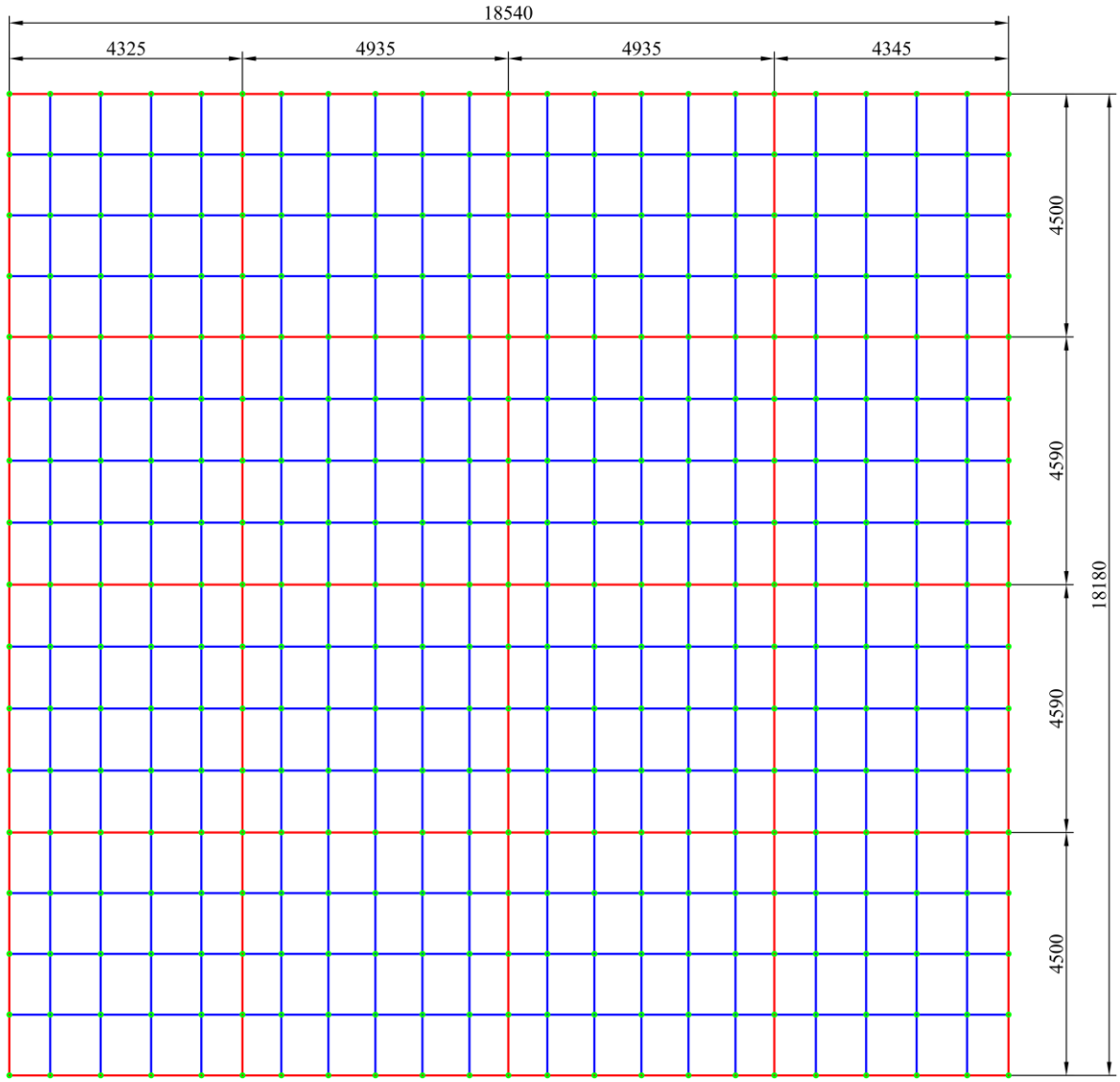
Razmaci između rubnih čvorova oplate se spremaju sljedećim redoslijedom:

1. Elementi prirubnice na prvom kraju
2. Prijelazni elementi na prvom kraju
3. Elementi osnovne mreže
4. Polovični elementi osnovne mreže
5. Prijelazni elementi na drugom kraju
6. Elementi prirubnice na drugom kraju

Za varijante mreže bez preslikavanja elemenata prirubnica na mrežu oplate kao što je *V2*, koriste se metode `plate_edge_node_spacing_x` i `plate_edge_node_spacing_y` u baznoj klasi `MeshSize`. Razlika u odnosu na metode za varijantu mreže *V1* je ta što nema upisa dimenzija elemenata prirubnica u niz dimenzija razmaka čvorova oplate. Za razmake između čvorova prirubnica varijante mreže *V2* služi posebna metoda `flange_edge_node_spacing` unutar podklase `ElementSizeV2`.

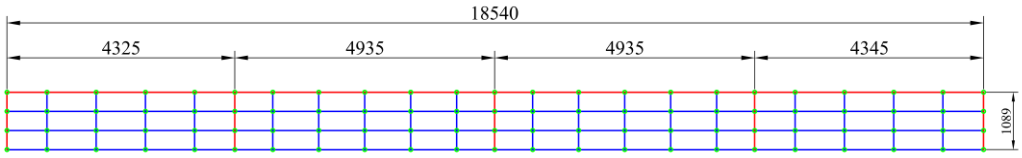
7. IZRADA MREŽE KONAČNIH ELEMENATA

Mreža konačnih elemenata se zasebno izrađuje za svaki element konstrukcije koji je obuhvaćen opsegom izrade mreže, sukladno odabranoj ili automatski prepoznatoj osi simetrije. Duž rubova svake zone oplate i krajeva segmenta dolazi do preklapanja inicijalno generiranih čvorova zbog odabranog pristupa nezavisne izrade mreže. Ovi rubovi sa preklapanjem čvorova opločenja su prikazani crvenom bojom na slici 7.1 za punu varijantu mreže *V2* ispitne varijante konstrukcije `hc_var_1`. Odabrani aspektni odnosi za izradu ove mreže su $\alpha_{\max}^{\mathrm{fe}} = 9,\ \alpha_{\max}^{\mathrm{pe}} = 4,\ \alpha_{\mathrm{p}}^{\mathrm{pe}} = 3.$



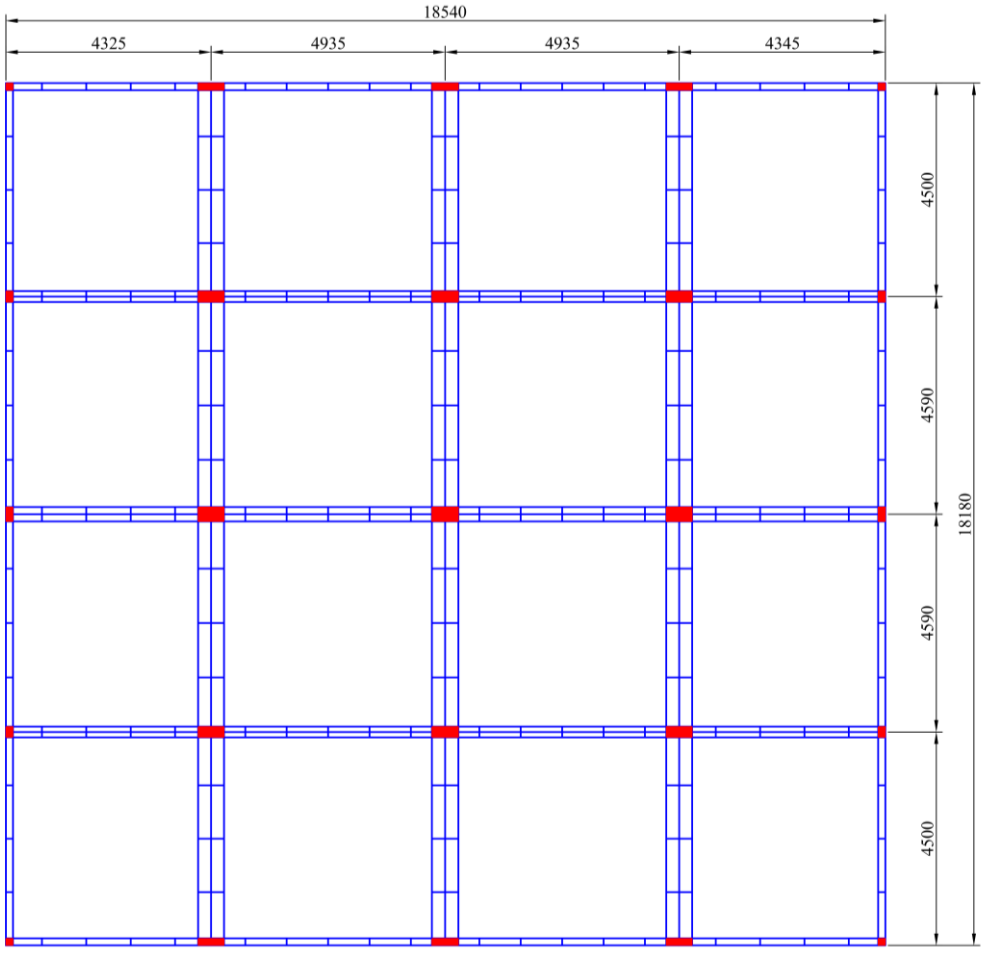
Slika 7.1 Rubovi sa preklapanjem čvorova oplate, ispitna varijanta `hc_var_1`

Na slici 7.2 je prikazana ista varijanta mreže na `hc_var_1`, gdje su crvenom bojom označeni rubovi strukova segmenata duž kojih dolazi do preklapanja čvorova. Prilikom izrade mreže se ovi čvorovi spremaju u zaseban rječnik kako bi se ubrzao rad algoritma za identifikaciju preklapanja.



Slika 7.2 Rubovi sa preklapanjem čvorova struka, ispitna varijanta `hc_var_1`

Na mreži prirubnica dolazi do preklapanja elemenata na mjestima križanja jakih nosača, te se iz istih razloga ovi elementi spremaju u zaseban rječnik zbog algoritma za identifikaciju preklopljenih elemenata. Na slici 7.3 su na primjeru iste ispitne varijante `hc_var_1` za mrežu punog modela crvenom bojom označeni konačni elementi prirubnica na mjestima preklapanja elemenata.



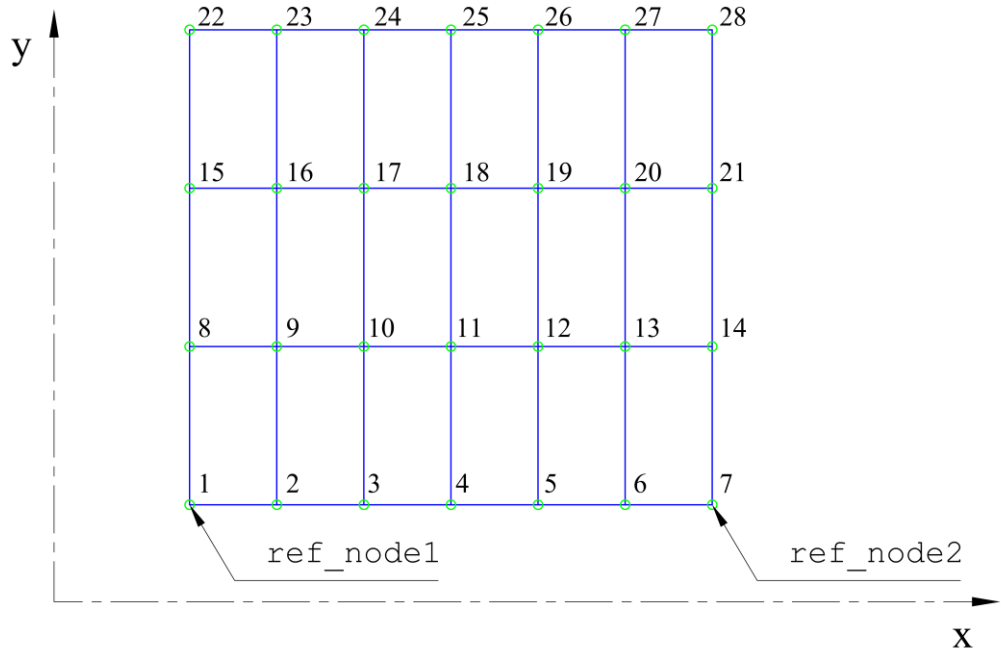
Slika 7.3 Preklapanje elemenata prirubnica, ispitna varijanta konstrukcije `hc_var_1`

7.1 Izrada mreže oplate

Mreža konačnih elemenata opločenja se izrađuje zasebno za svaku zonu oplate koja je uključena u opseg izrade mreže. Mreža opločenja nema više različitih varijanti, prema tome u programskom kodu postoji samo jedna klasa `PlateMesh` koja sadrži algoritme za izradu mreže konačnih elemenata na jednoj zoni oplate.

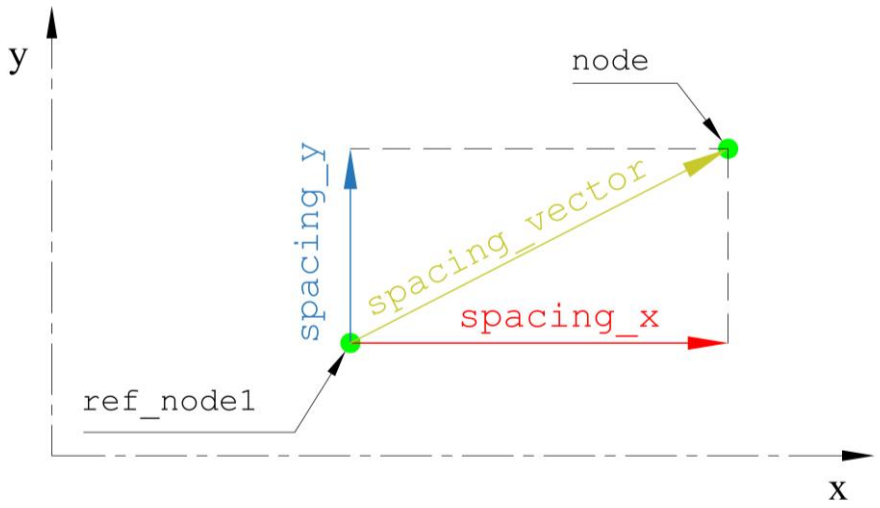
7.1.1 Čvorovi zone oplate

Čvorovi mreže pojedine zone oplate se generiraju na temelju niza dimenzija razmaka između rubnih čvorova i podataka sadržanih u objektu `Plate`. Slika 7.4 prikazuje primjer numeracije čvorova na jednoj zoni oplate sa ukupno 28 čvorova i 18 konačnih elemenata. Referentni čvorovi `ref_node1` i `ref_node2` se nalaze na krajevima prvog uzdužnog segmenta `long_seg1` koji definira tu zonu oplate, a služe za određivanje jediničnog vektora smjera `unit_ref_vector`.



Slika 7.4 Referentne točke i numeracija čvorova na zoni oplate

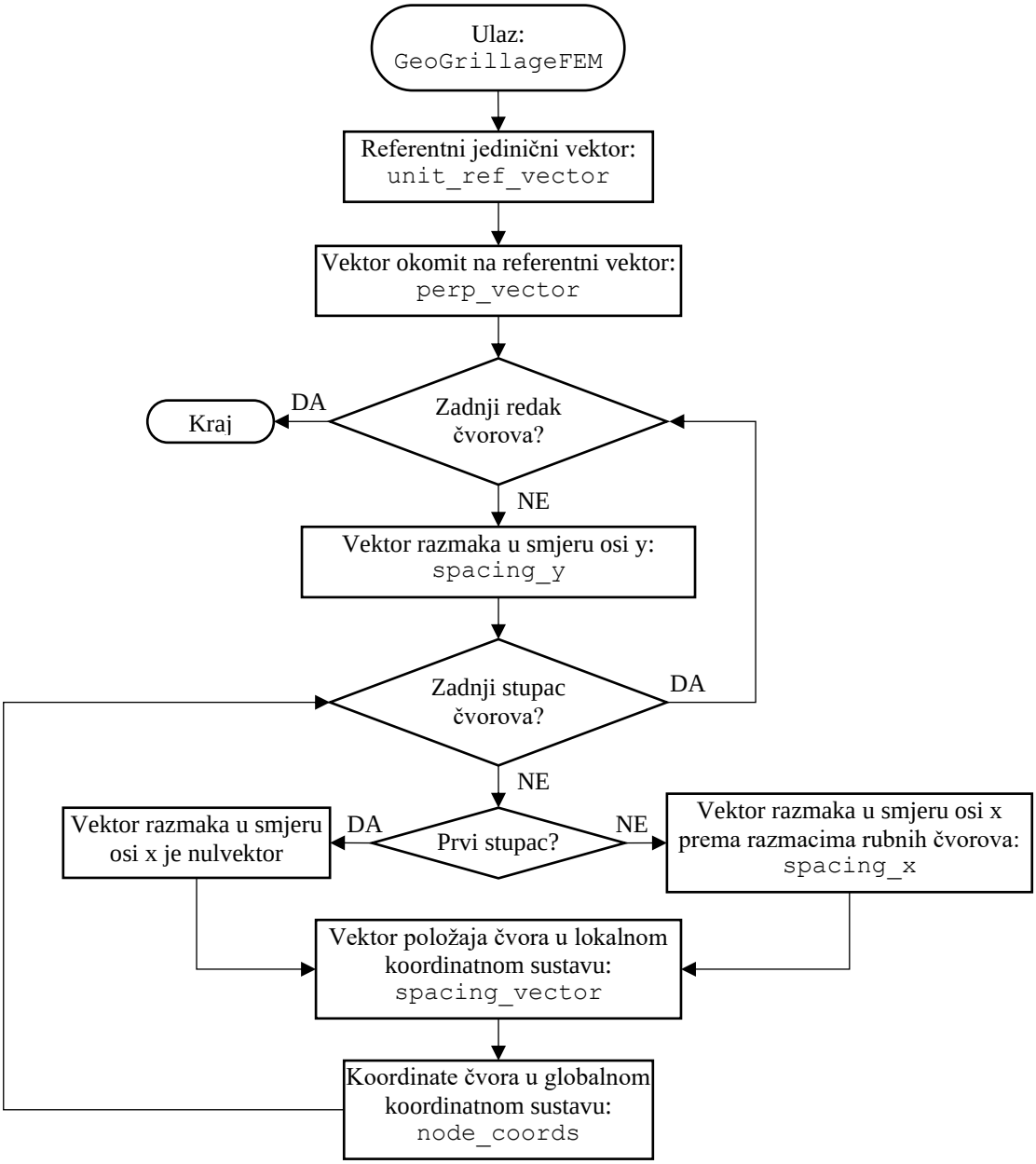
Metodom `generate_plate_nodes` unutar klase `PlateMesh` se generiraju svi čvorovi na jednoj zoni oplate. Lokalne koordinate ovih čvorova se određuju vektorski u odnosu na prvi referentni čvor, vektorskim zbrojem `spacing_x` i `spacing_y` prema slici 7.5. Koordinate čvora `node` u globalnom koordinatnom sustavu se zatim određuju zbrojem vektora položaja `spacing_vector` i koordinata prvog referentnog čvora `ref_node1`.



Slika 7.5 Vektorsko određivanje koordinata čvorova oplate

Numeracija i izrada čvorova kreće od prvog referentnog čvora `ref_node1` i nastavlja u smjeru jediničnog vektora smjera za svaki redak čvorova. Prvi čvor prve zone oplate uvijek kreće sa početnim identifikacijskim brojem 1, a numeracija čvorova na idućim zonama oplate i segmentima se nastavlja prema identifikacijskom broju zadnjeg upisanog čvora uvećanog za 1. Dijagram toka metode `generate_plate_nodes` je prikazan slikom 7.6.

Ukupan broj redaka i stupaca čvorova pojedine zone oplate se određuje metodom `get_mesh_limits`, koja broj čvorova određuje na temelju broja upisanih dimenzija razmaka rubnih čvorova. Za broj stupaca čvorova se razmatra broj dimenzija u smjeru globalne osi *x*, a za broj redaka se razmatra broj dimenzija u smjeru osi *y*.



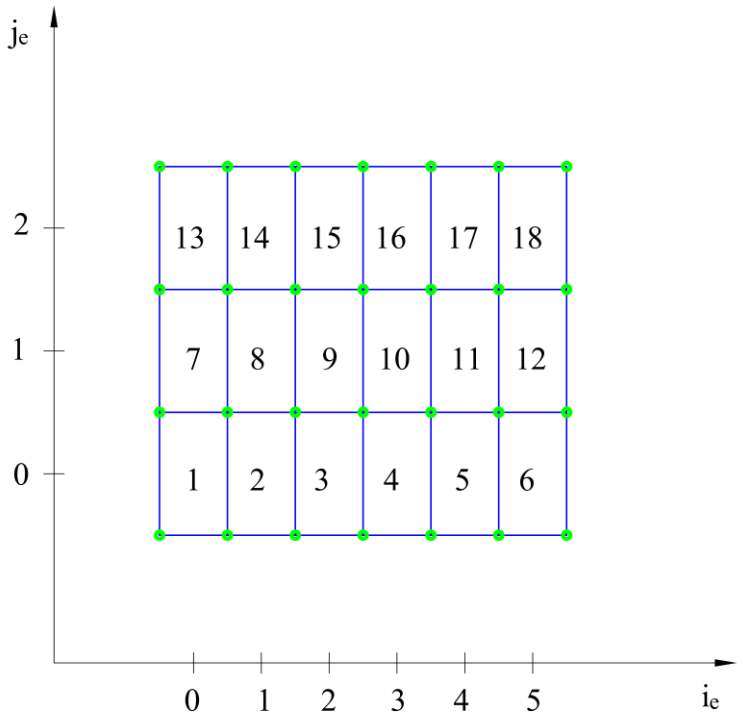
Slika 7.6 Dijagram toka metode `generate_plate_nodes`

7.1.2 Konačni elementi opločenja

Metodom `generate_plate_elements` unutar klase `PlateMesh` se generiraju svi elementi na pojedinoj zoni oplate. Za svaku zonu oplate se izrađuje referentno dvodimenzijsko polje identifikacijskih brojeva svih čvorova koji se nalaze na toj zoni oplate. Ovo referentno polje raspoređuje identifikacijske brojeve čvorova prema njihovom relativnom položaju i služi za izradu svih konačnih elemenata na odabranoj zoni oplate. Za zonu oplate prikazanu na slici 7.4 referentno polje čvorova ima sljedeći oblik:

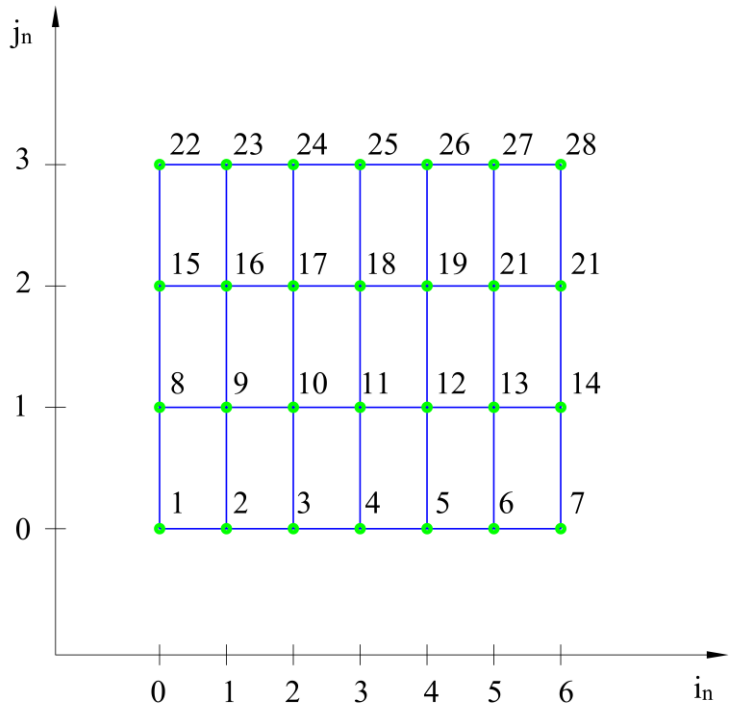
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 & 26 & 27 & 28 \end{bmatrix} \tag{7.1}$$

Numeracija i izrada konačnih elemenata, jednako kao i za čvorove, kreće od prvog referentnog čvora `ref_node1` i nastavlja u smjeru jediničnog vektora smjera za svaki redak elemenata. Slika 7.7 prikazuje numeraciju elemenata na primjeru zone oplate prikazane slikom 7.4.



Slika 7.7 Numeracija i indeksi pločastih konačnih elemenata na zoni oplate

Čvorovi koji pripadaju promatranom konačnom elementu se određuju petljom kroz sve retke j_e i stupce i_e elemenata, unutar granica određenih metodom `get_mesh_limits`. Indeksi redaka j_n i stupaca i_n čvorova istog primjera zone oplate, koji odgovaraju indeksima referentnog polja čvorova prikazani su slikom 7.8. Unutar tijela petlje se preko indeksa redaka j_e i stupaca i_e za svaki element određuju indeksi položaja pripadajućih identifikacijskih brojeva čvora u referentnom polju čvorova. Jednadžbama 7.2 – 7.5 su prikazani izrazi kojima se povezuju indeksi elemenata i indeksi identifikacijskih brojeva pripadajućih čvorova u referentnom polju.



Slika 7.8 Numeracija i indeksi čvorova na zoni oplate

$$node_1 = j_e, i_e \tag{7.2}$$

$$node_2 = j_e, i_e + 1 \tag{7.3}$$

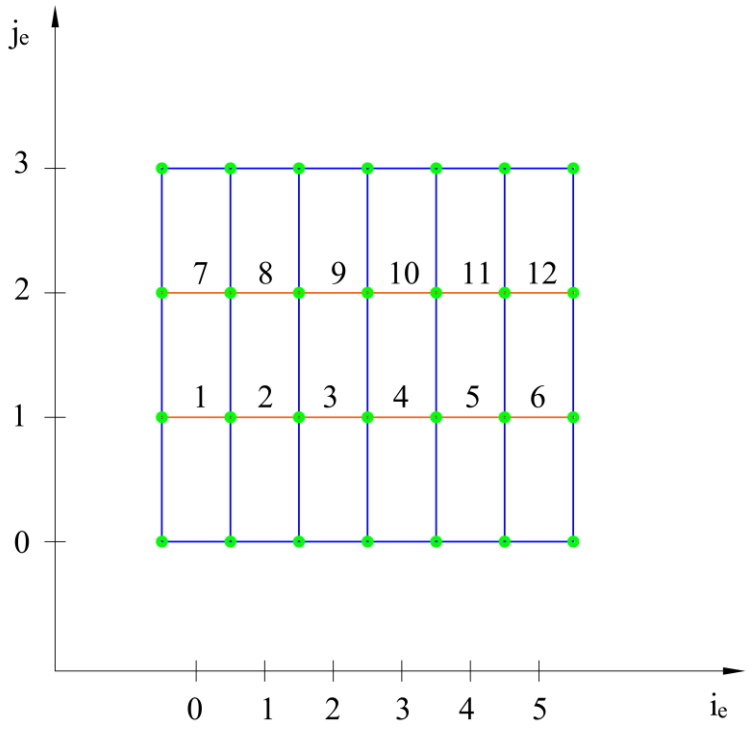
$$node_3 = j_e + 1, i_e + 1 \tag{7.4}$$

$$node_4 = j_e + 1, i_e \tag{7.5}$$

7.1.3 Konačni elementi ukrepa

Metodom `identify_beam_nodes` unutar klase `PlateMesh` se identificiraju položaji ukrepa na pojedinoj zoni oplate neovisno o dimenzijama konačnih elemenata. Identifikacija se temelji na poznatom razmaku između ukrepa i nizu dimenzija rubnih čvorova `edge_nodes` u odgovarajućem smjeru. Ova metoda vraća listu indeksa redaka ili stupaca referentnog polja čvorova na kojima se nalaze ukrepe, ovisno o orijentaciji.

Na temelju ove liste indeksa se metodom `generate_beam_elements` izrađuju gredni konačni elementi između odgovarajućih čvorova oplate. Slično metodi za izradu pločastih konačnih elemenata oplate, za svaku zonu oplate se izrađuje referentno dvodimenzijsko polje identifikacijskih brojeva čvorova `node_id_array`. Slikom 7.9 je prikazana numeracija grednih konačnih elemenata uzdužnih ukrepa na zoni oplate prikazane slikom 7.8. Na ovom primjeru će metoda `identify_beam_nodes` vratiti listu sa indeksima redaka 1 i 2 na kojima se nalaze ukrepe. Petljom kroz sve stupce elemenata na ovim redcima se preko indeksa redaka j_e i stupaca i_e za svaki gredni element određuju pripadni identifikacijski brojevi čvorova u referentnom polju čvorova. Metoda također sadrži provjeru nalazi li se ukrepa na osi simetrije i u tom slučaju dodjeljuje svojstvo grede sa pola originalnih karakteristika krutosti pomoću metode `get_half_stiffener_beam_property`.



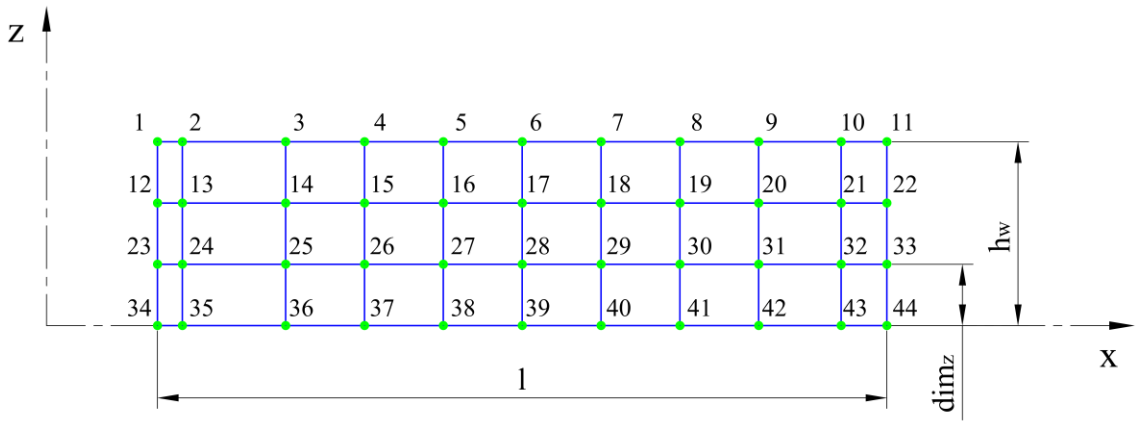
Slika 7.9 Numeracija i indeksi grednih konačnih elemenata na zoni oplate

7.2 Izrada konačnih elemenata jakih nosača

Mreža konačnih elemenata jakih nosača se izrađuje zasebno za svaki segment koji pripada pojedinom jakom nosaču, a koji je uključen u opseg izrade mreže. Mreža strukova segmenata ima više različitih varijanti, prema tome u programskom kodu za dvije varijante mreže *V1* i *V2* osim bazne klase `SegmentMesh` postoje podklase `SegmentMeshV1` i `SegmentMeshV2`.

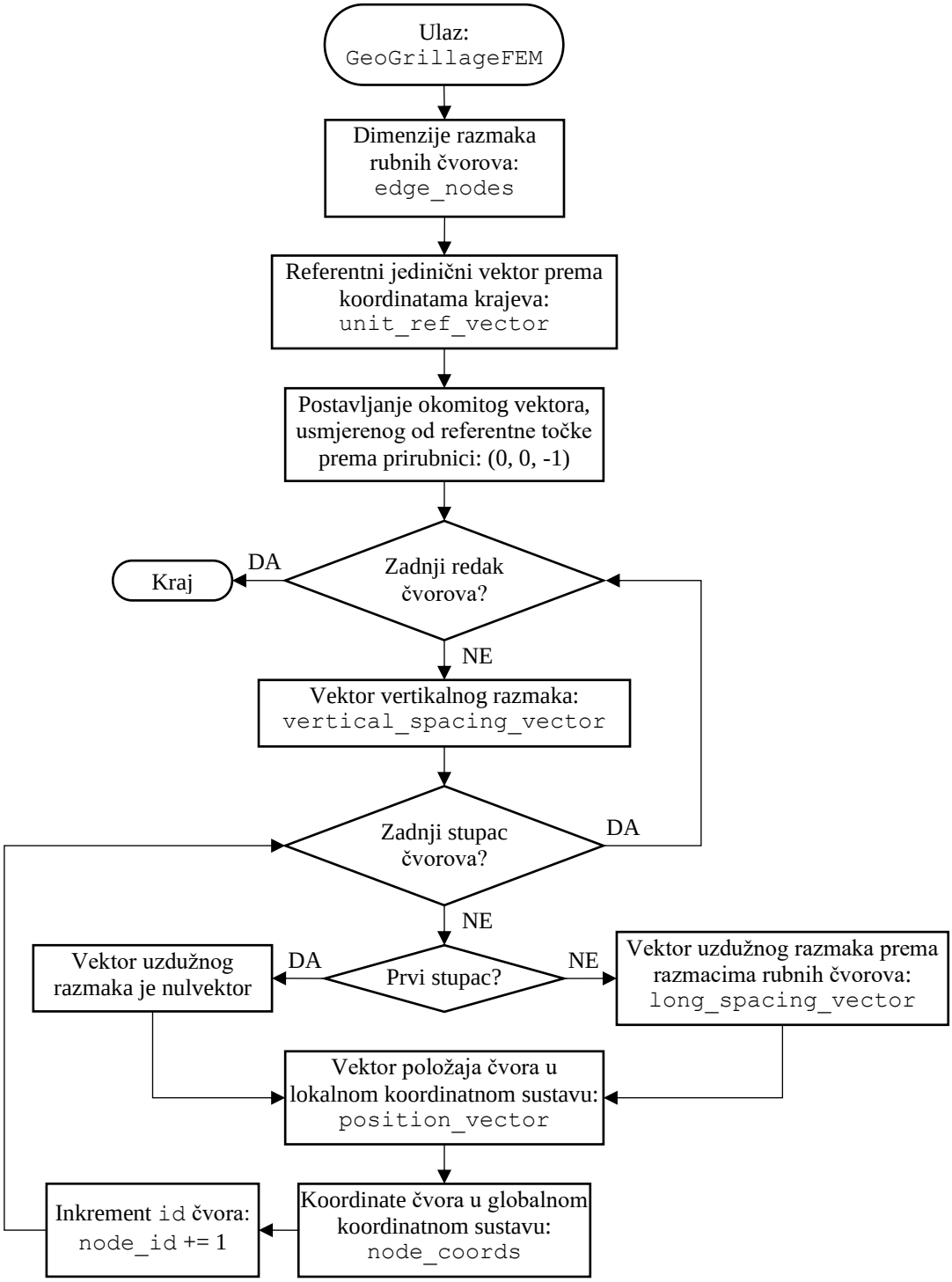
7.2.1 Čvorovi struka segmenta za varijantu mreže V1

Čvorovi mreže struka pojedinog segmenta za varijantu mreže *V1* se generiraju na temelju podataka sadržanih u objektu `Segment` i niza dimenzija razmaka između rubnih čvorova, preuzetih od pripadajuće zone oplate. Metoda `get_plate_edge_nodes` unutar bazne klase `SegmentMesh` služi za identifikaciju prve zone oplate koja je definirana promatranim segmentom. Ovisno o smjeru jakog nosača `direction` kojemu pripada promatrani segment, sa te zone oplate se preuzimaju dimenzije razmaka rubnih čvorova oplate u smjeru globalne osi *x* ili *y*. Ove dimenzije se zatim zapisuju u rječnik `edge_plate_nodes` unutar bazne klase `SegmentMesh`, kako se proces identifikacije pripadajuće zone oplate ne bi više puta ponavljao za isti segment. Slikom 7.10 je prikazan primjer numeracije čvorova na struku jednog uzdužnog segmenta sa ukupno 44 čvora i 30 konačnih elemenata za varijantu mreže *V1*..



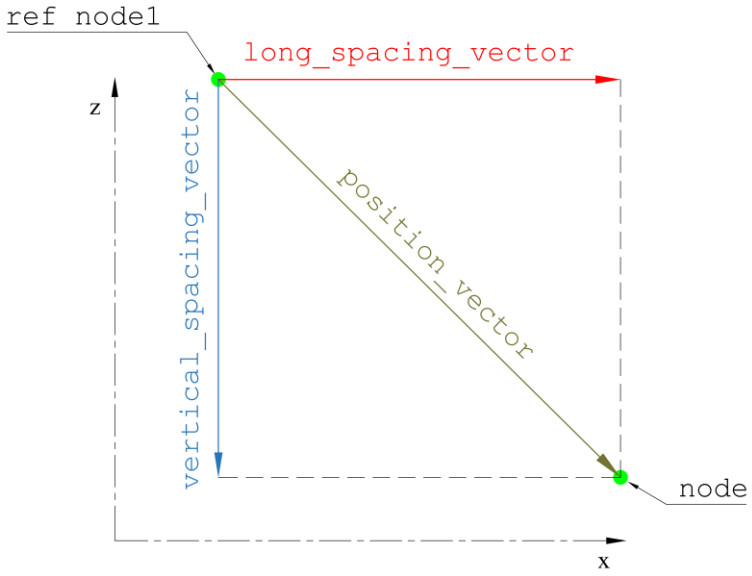
Slika 7.10 Numeracija čvorova struka segmenta, varijanta mreže *V1*

Metodom `generate_web_nodes` unutar podklase `SegmentMeshV1` se generiraju svi čvorovi na struku pojedinog segmenta za varijantu mreže *V1*. Dijagram toka ove metode je prikazan slikom 7.11. Algoritmom unutar ove metode se lokalne koordinate čvorova struka određuju vektorski u odnosu na prvi referentni čvor segmenta `ref_node1`, koji je definiran metodom `get_segment_node1`. Na slici 7.10 se ovaj čvor nalazi u razini spoja segmenta sa opločenjem, na visini $z = h_w$ i označen je identifikacijskim brojem čvora 1.



Slika 7.11 Dijagram toka metode `generate_web_nodes`

Slika 7.12 prikazuje vektor položaja pojedinog čvora `position_vector`, određen zbrojem vektora razmaka u aksijalnom smjeru segmenta `long_spacing_vector` i vektora u smjeru visine struka `vertical_spacing_vector`. Koordinate čvora `node` u globalnom koordinatnom sustavu se određuju zbrojem vektora položaja `position_vector` i koordinata referentnog čvora `ref_node1`.



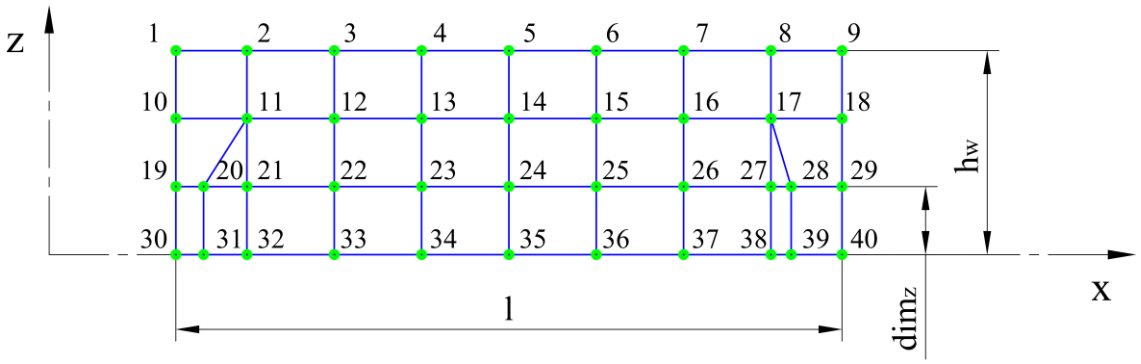
Slika 7.12 Vektorsko određivanje koordinata čvorova segmenta

Ukupan broj stupaca čvorova pojedinog segmenta se ne može odrediti jednako kao i za zonu oplate metodom `get_mesh_limits`, zbog potrebe za identifikacijom smjera jakog nosača. Broj stupaca čvorova se tada određuje na temelju broja upisanih dimenzija razmaka rubnih čvorova sadržanih u rječniku `edge_plate_nodes`. Broj redaka čvorova je određen prema ulaznom parametru broja elemenata po visini struka `min_num_eweb`.

7.2.2 Čvorovi struka segmenta za varijantu mreže V2

Čvorovi mreže struka pojedinog segmenta za varijantu mreže *V2* se generiraju na temelju podataka sadržanih u objektu `Segment`, te niza dimenzija razmaka između rubnih čvorova prirubnice i pripadajuće zone oplate. Razmaci između rubnih čvorova opločenja se preuzimaju od pripadajuće zone oplate, jednako kao i za varijantu mreže *V1*. Razmaci između rubnih čvorova prirubnica se određuju metodom `flange_edge_node_spacing` unutar podklase `ElementSizeV2`, a prilikom izrade mreže svakog segmenta se zapisuju u rječnik `edge_flange_nodes`.

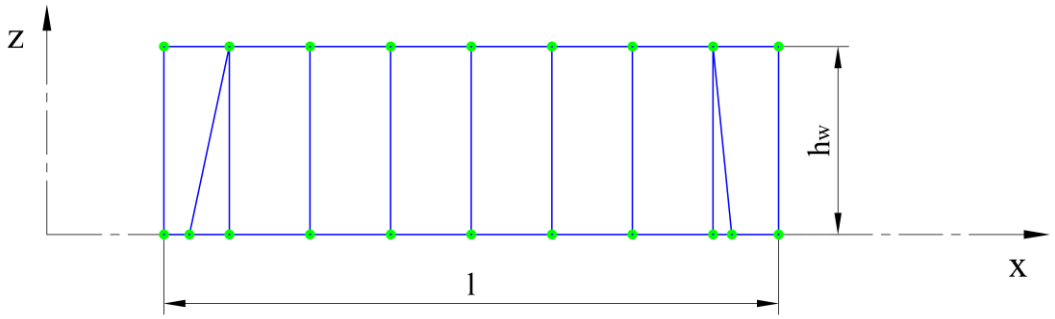
Zbog zasebnog određivanja niza dimenzija rubnih čvorova oplate i prirubnica na varijanti *V2*, broj čvorova ne mora biti jednak na gornjem i donjem rubu struka segmenta. Slikom 7.13 je prikazan primjer numeracije čvorova na struku jednog uzdužnog segmenta duljine *l* sa ukupno 40 čvorova, 26 četverokutnih i 2 trokutasta konačna elementa za varijantu mreže *V2*. Na ovom primjeru sa 3 elementa po visini struka h_w je vidljiv prijelazni red konačnih elemenata na sredini, koji se sastoji od dva trokutasta, dva deformirana četverokutna i 6 pravokutnih konačnih elemenata.



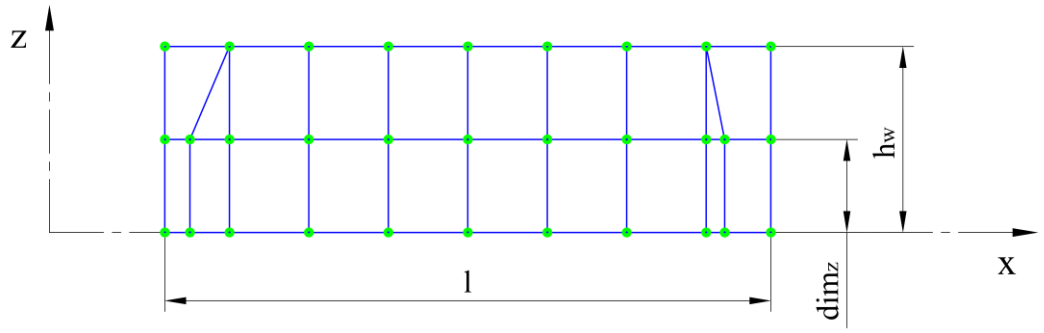
Slika 7.13 Numeracija čvorova struka segmenta, varijanta mreže *V2*

Metodom `generate_web_nodes` unutar podklase `SegmentMeshV2` se generiraju svi čvorovi na struku pojedinog segmenta za varijantu mreže *V2*. Dijagram toka ove metode je iznimno sličan metodi za varijantu mreže *V1* prikazan slikom 8.10, ali se sastoji od dvije petlje kroz retke i stupce čvorova zbog različitog broja stupaca u pojedinom retku. Različiti broj stupaca čvorova zahtijeva korištenje posebnih metoda za određivanje granica tj. broja redaka u ovisnosti o broju elemenata po visini struka `min_num_eweb`. Metodom `plate_node_row_number` se određuje broj redaka čvorova struka koji imaju dimenziju razmaka između rubnih čvorova preuzetih od pripadajuće zone oplate, a metodom `flange_node_row_number` broj redaka čvorova koji imaju dimenzije razmaka čvorova prirubnice. Ovim metodama se također u obzir uzimaju različite kombinacije broja elemenata po visini struka, tako da se prijelazni red elemenata nastoji smjestiti u drugi red elemenata od spoja sa prirubnicom.

U posebnim slučajevima kada to nije moguće, npr. ako je broj elemenata po visini struka jednak 1 ili 2, prijelazni red elemenata se smješta neposredno uz opločenje kako je prikazano slikama 7.14. i 7.15.

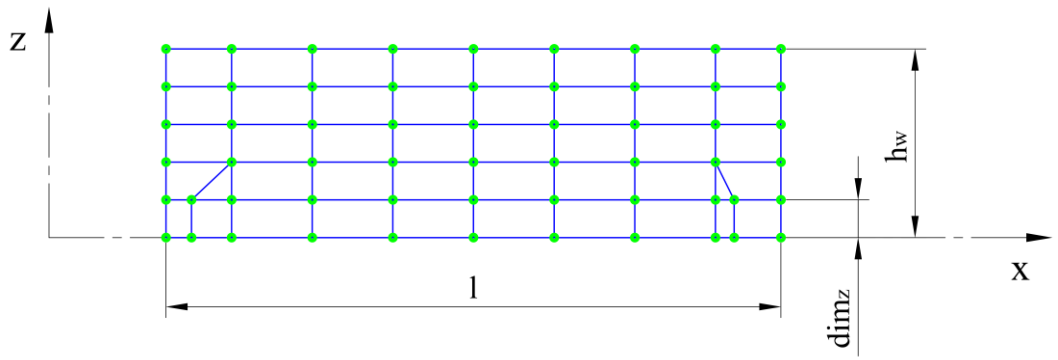


Slika 7.14 Jedan element po visini struka, varijanta mreže *V2*



Slika 7.15 Prijelazni red elemenata sa 2 elemenata po visini struka, varijanta mreže *V2*

Slikom 7.16 je prikazan primjer mreže na struku segmenta iste duljine l i visine struka h_w sa 5 elemenata po visini struka, gdje je vidljivo postavljanje prijelaznog reda u drugi red elemenata.



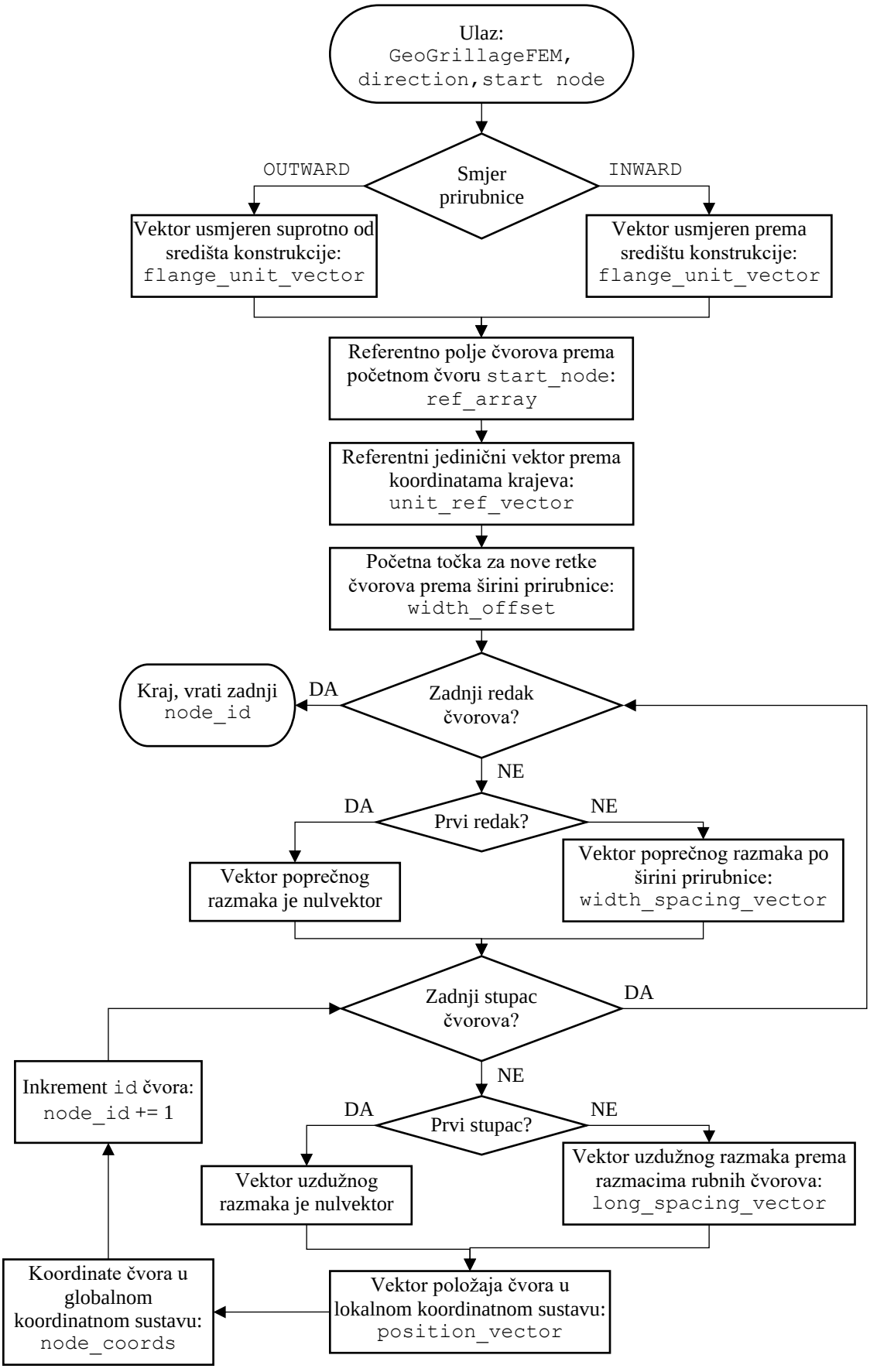
Slika 7.16 Prijelazni red elemenata sa 5 elemenata po visini struka, varijanta mreže *V2*

Broj stupaca čvorova se određuje na temelju broja upisanih dimenzija razmaka rubnih čvorova. Za retke čvorova koji preuzimaju dimenzije razmaka čvorova opločenja je to broj upisanih dimenzija u rječniku `plate_edge_nodes`, a za redak čvorova uz prirubnicu `flange_edge_nodes`.

7.2.3 Čvorovi prirubnice segmenta za varijantu mreže V1

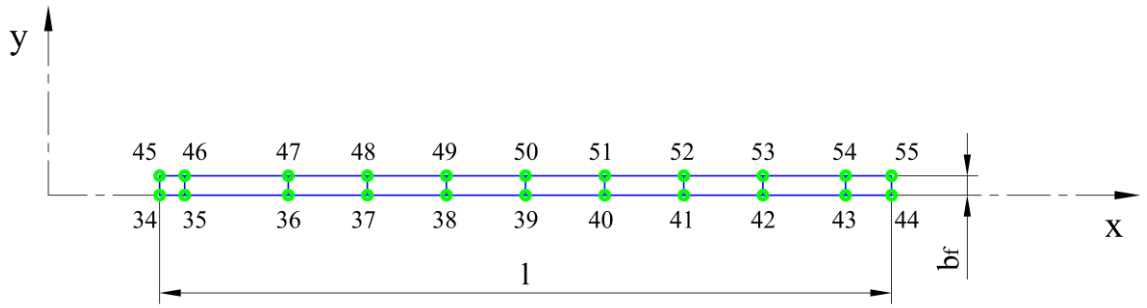
Čvorovi mreže prirubnice pojedinog segmenta se generiraju na temelju podataka sadržanih u objektu `Segment`, niza dimenzija razmaka između rubnih čvorova i identifikacijskih brojeva čvorova zadnjeg reda čvorova struka. Budući da varijanta mreže *V1* preslikava elemente prirubnice na opločenje, razmaci između rubnih čvorova se preuzimaju od pripadajuće zone oplate.

Metodom `generate_flange_nodes` unutar podklase `SegmentMeshV1` se generiraju svi novi čvorovi na prirubnici pojedinog segmenta za varijantu mreže *V1*. Dijagram toka ove metode je prikazan slikom 7.17. Ulaznim parametrom `direction` tipa `FlangeDirection` se odabire usmjerenje prirubnice L profila, a za T profile omogućuje izradu mreže prirubnice na samo jednoj strani profila, ovisno o osima simetrije za koje se izrađuje mreža konačnih elemenata.



Slika 7.17 Dijagram toka metode `generate_flange_nodes`

Kako bi se izbjeglo preklapanje čvorova, algoritam neće izraditi nove čvorove na mjestu spoja struka i prirubnice jakih nosača u obliku T i L profila, već nastavlja numeraciju čvorova od zadnjeg čvora struka. Metodom `ref_flange_node_ID_array` unutar podklase `SegmentMeshV1` se izrađuje referentno dvodimenzijsko polje identifikacijskih brojeva svih čvorova koji se nalaze na prirubnici. Ovo referentno polje raspoređuje identifikacijske brojeve čvorova prema njihovom relativnom položaju, a služi za izradu čvorova i konačnih elemenata na odabranom segmentu. Prilikom izrade referentnog polja čvorova se preuzima zadnji redak referentnog polja čvorova struka. Slika 7.18 prikazuje primjer preuzimanja zadnjeg retka čvorova struka sa mreže struka segmenta prikazanog slikom 7.10 i nastavak numeracije čvorova na prirubnici L profila.

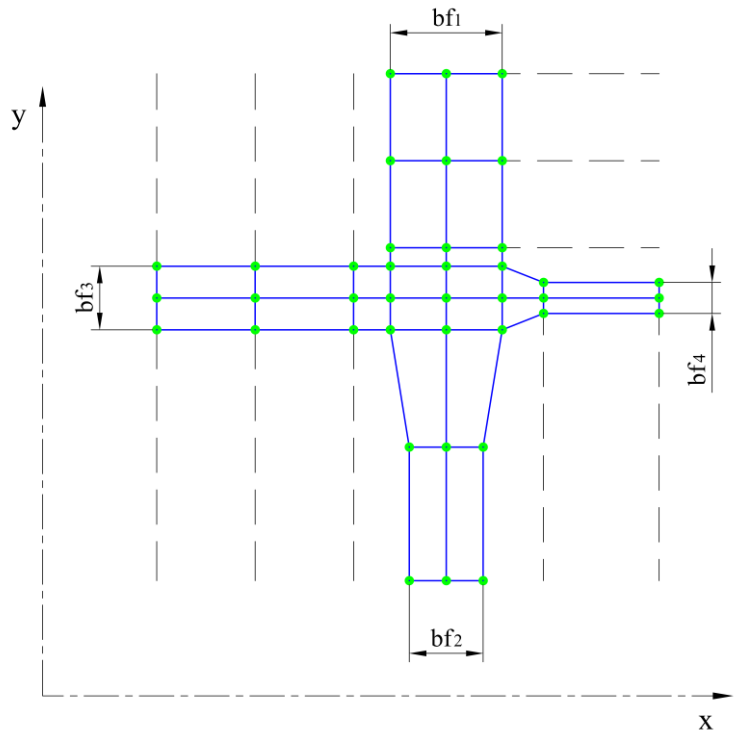


Slika 7.18 Numeracija čvorova prirubnice segmenta, varijanta mreže *V1*

Izrada novih redova čvorova prirubnice započinje u početnoj točki na udaljenosti širine prirubnice L profila ili polušširine prirubnice T profila od struka segmenta. Početna točka na primjeru čvorova prirubnice prikazanih slikom 7.18 je označena brojem čvora 45.

7.2.4 Čvorovi prirubnice segmenta za varijantu mreže V2

Čvorovi mreže prirubnice varijante mreže *V2* se izrađuju slično kao i za mrežu *V1*, uz nekoliko razlika. Za varijantu *V2* se dimenzije razmaka rubnih čvorova prirubnice preuzimaju za promatrani segment iz rječnika `edge_flange_nodes`. Zbog različitog broja stupaca čvorova u pojedinom retku, umjesto referentnog dvodimenzijskog polja čvorova se koristi referentna lista `reference_web_node_ID_list`. Specifičnost varijante mreže *V2*, zbog prijelaznog retka elemenata struka, je izrada mreže konačnih elemenata na jakim nosačima sa segmentima različitih širina prirubnica. Slikom 7.19 je prikazan primjer spoja prirubnica jakih nosača na jednom mjestu križanja, gdje dolazi do promjene širine prirubnice poprečnog nosača sa širine b_{f1} na b_{f2} i uzdužnog nosača sa širine b_{f3} na b_{f4}. Položaji čvorova na ovom primjeru su dijelom uvjetovani ukrepama koje su prikazane isprekidanim linijama.



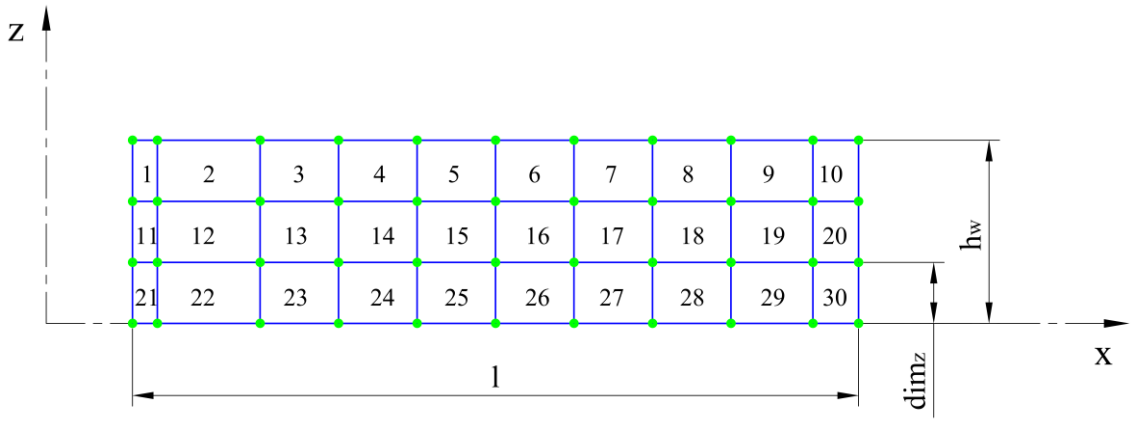
Slika 7.19 Promjena širine prirubnica na spoju jakih nosača

Koordinate čvorova na području preklopa se određuju principom zadržavanja veće širine prirubnice. Za svaki segment na kojemu se izrađuje mreža konačnih elemenata se identificira maksimalna širina prirubnice na oba kraja, na mjestima spoja sa paralelnim segmentima, metodama `get_end1_max_flange_width` i `get_end2_max_flange_width`. Za identifikaciju susjednih segmenata unutar ovih metoda, koji se nalaze na istom jakom nosaču, se koristi metoda `get_parallel_segments`. Principom zadržavanja veće širine prirubnice se na ovom primjeru kroz područje preklopa za uzdužne segmente zadržava širina b_{f3}, a za poprečne segmente širina b_{f1}.

Metodom `generate_flange_nodes` unutar podklase `SegmentMeshV2` se zadržavanje veće širine prirubnice izvodi tako da prva i zadnja dva čvora preuzimaju najveću širinu prirubnice na odgovarajućem kraju. Prijelaz na manju širinu prirubnice je zatim izveden na razmaku jednog konačnog elementa.

7.2.5 Elementi struka segmenta za varijantu mreže V1

Metodom `generate_web_elements` unutar podklase `SegmentMeshV1` se generiraju svi pločasti konačni elementi na struku segmenta za varijantu mreže *V1*. Numeracija i izrada konačnih elemenata, jednako kao i za čvorove, kreće od prvog referentnog čvora segmenta `ref_node1`. Slika 7.20 prikazuje numeraciju elemenata na primjeru struka segmenta prikazanog slikom 7.10.



Slika 7.20 Numeracija elemenata struka segmenta, varijanta mreže *V1*

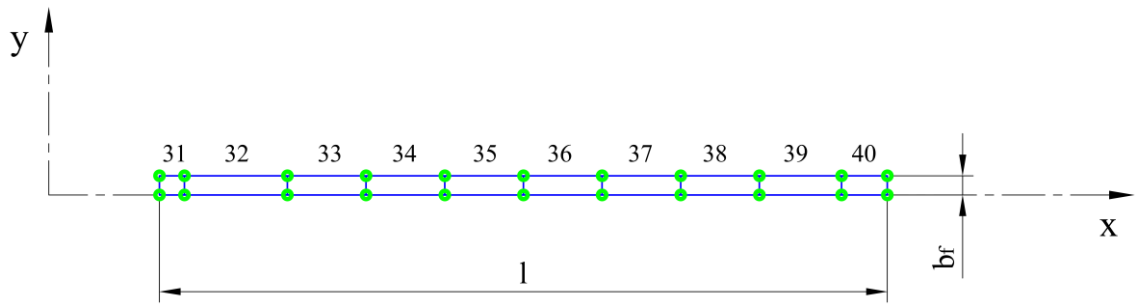
Analogno izradi konačnih elemenata opločenja, za svaki segment se izrađuje referentno dvodimenzijsko polje identifikacijskih brojeva svih čvorova koji se nalaze na struku. Petljom kroz sve retke i stupce konačnih elemenata se određuju indeksi položaja pripadajućih čvorova u referentnom polju čvorova `reference_web_node_ID_array`.

7.2.6 Elementi struka segmenta za varijantu mreže V2

Metoda `generate_web_elements` unutar podklase `SegmentMeshV2` za izradu konačnih elemenata na struku segmenta varijante mreže *V2* se značajno razlikuje od metode u podklasi `SegmentMeshV1`. Zbog postojanja prijelaznog reda elemenata, mreža struka se izrađuje po redcima, zasebno za retke iznad i ispod prijelaznog reda elemenata. Metodom `top_web_element_row` se izrađuju redovi elemenata iznad prijelaznog reda, a metodom `bot_web_element_row` red elemenata ispod prijelaznog reda elemenata. Sami prijelazni red elemenata se izrađuje metodom `tr_web_element_row` uz pomoć više pomoćnih metoda koje izrađuju pojedini element u nizu. Pomoćne metode se pozivaju odabranim redoslijedom i generiraju element po element prijelaznog reda, u ovisnosti o broju trokutastih konačnih elemenata koji se određuje metodom `identify_num_of_tris`.

7.2.7 Elementi prirubnice segmenta

Za izradu pločastih konačnih elemenata prirubnica nema razlike između varijanti mreža, te se koristi zajednička metoda `generate_flange_elements` unutar klase `SegmentMesh`. Numeracija i izrada konačnih elemenata, jednako kao i za čvorove, započinje u početnoj točki na udaljenosti širine ili poluširine prirubnice od struka segmenta, te nastavlja u smjeru pozitivne globalne koordinatne osi. Slika 7.21 prikazuje primjer nastavka numeracije elemenata prirubnice L profila sa mreže struka segmenta prikazanog slikom 7.20.



Slika 7.21 Numeracija elemenata prirubnice L profila

Analogno izradi konačnih elemenata struka, za svaki segment se izrađuje referentno dvodimenzijsko polje identifikacijskih brojeva svih čvorova koji se nalaze na prirubnici. Petljom kroz sve retke i stupce konačnih elemenata se određuju indeksi položaja pripadajućih čvorova u referentnom polju čvorova `ref_flange_node_ID_array`. Prilikom izrade ovog referentnog polja čvorova se preuzima zadnji redak čvorova struka. Za varijantu mreže *V1* se ovaj redak preuzima iz dvodimenzijskog polja `reference_web_node_ID_array`, a za varijantu mreže *V2* iz referentne liste `reference_web_node_ID_list`.

Prilikom izrade konačnih elemenata prirubnice se svaki element dodaje u poseban rječnik metodom `add_element_to_element_overlaps`. Za elemente unutar ovog rječnika nakon izrade mreže slijedi provjera preklapanja čvorova.

7.3 Spajanje mreže konačnih elemenata

Nakon nezavisne izrade mreže na svim elementima konstrukcije obuhvaćenim izradom mreže slijedi ispravak preklapanja čvorova i konačnih elemenata, metodama unutar modula `grillage_fem.py`. Postupak se sastoji od sljedećih koraka:

1. Identifikacija preklapanja čvorova
2. Sortiranje i odabir koji čvor ostaje na koordinatama, a koji se briše
3. Zamjena čvorova koji će se obrisati u definiciji konačnih elemenata
4. Brisanje preklopljenih čvorova
5. Brisanje preklopljenih elemenata prirubnica

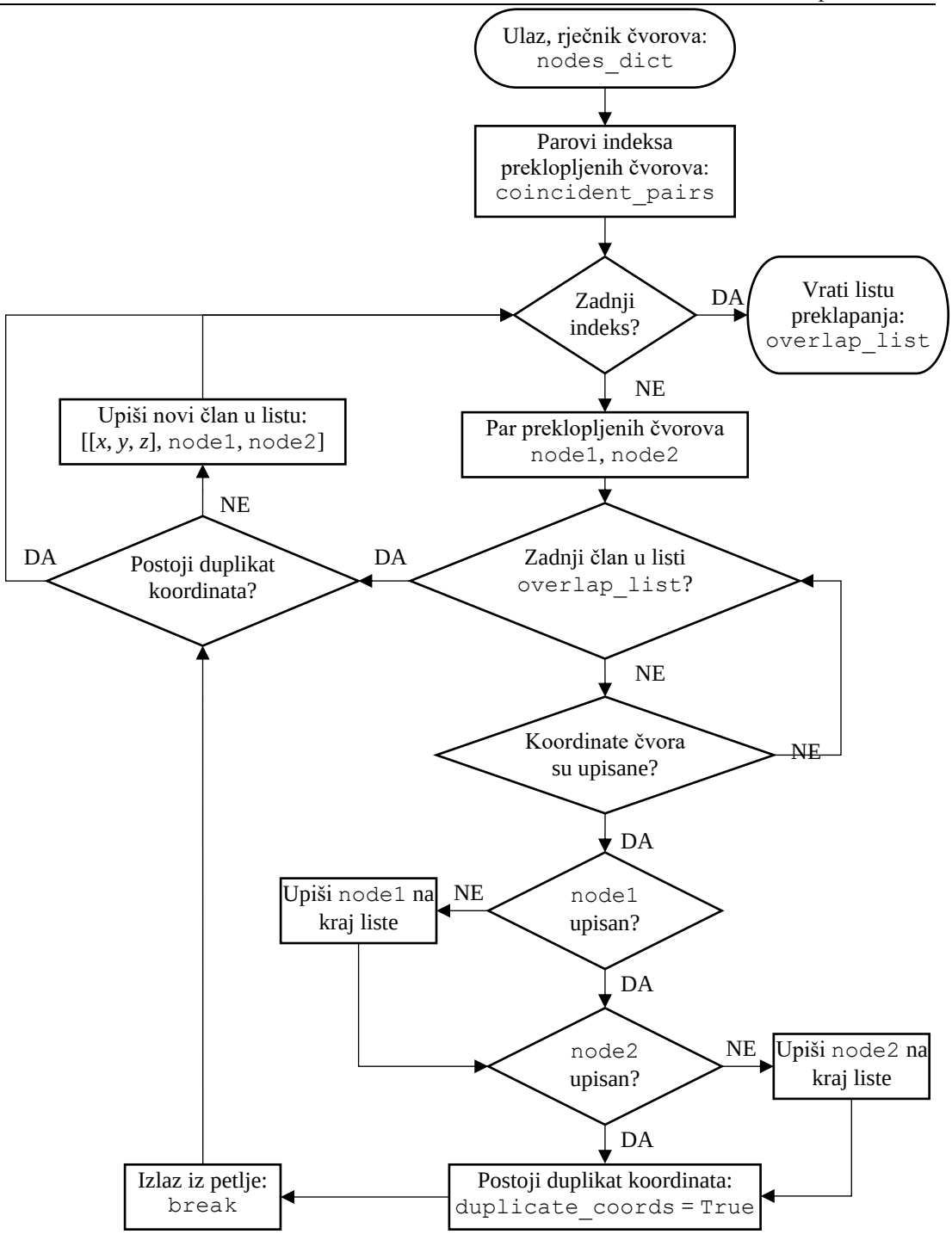
7.3.1 Algoritam za ispravak preklapanja čvorova

Metoda `check_node_overlap` služi za identifikaciju svih preklopljenih čvorova koji se nalaze na jedinstvenim koordinatama. Metoda `check_node_overlap_np` je optimizirana verzija originalne metode, korištenjem *NumPy* operacija [12] za identifikaciju parova preklopljenih čvorova. Prilikom testiranja optimizirane verzije je zabilježena 60 puta brža identifikacija preklapanja od originalne metode. Dijagram toka ove metode je prikazan slikom 7.22.

Za svaki čvor upisan u ulaznom rječniku čvorova `nodes_dict` se provjerava jesu li koordinate tih čvorova već upisane u listu preklapanja `overlap_list`. Vizualizacija zapisa ove liste preklapanja je jednostavnija ako se lista promatra kao dvodimenzijsko polje sa jednim stupcem. Prvih nekoliko redaka ovakvog polja, za primjer ispitne varijante `hc_var_5` je prikazano jednadžbom 7.5.

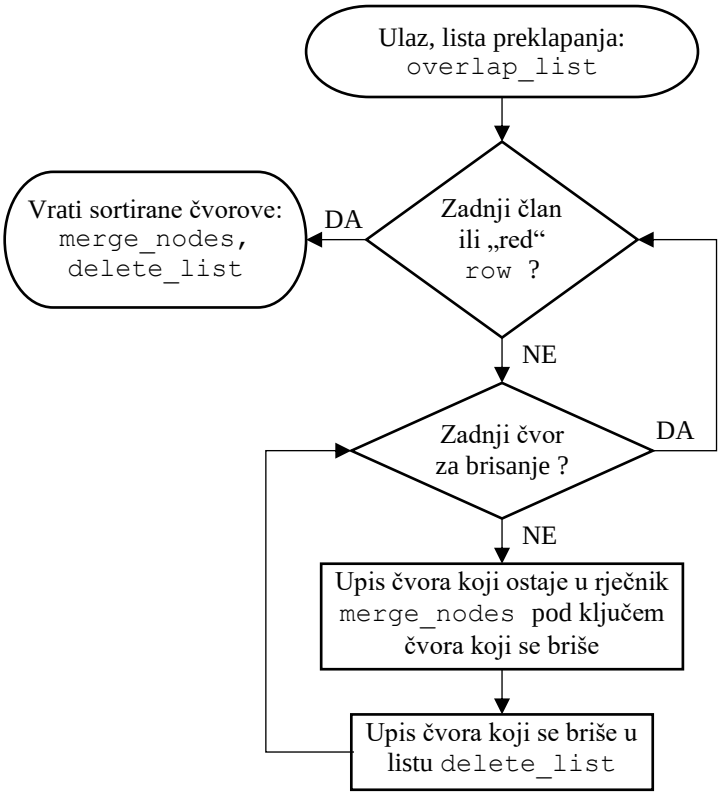
$$\begin{bmatrix} [[0,6060,1089], & node71, & node61, & node278, & node352, & node521], \\ [[681,6060,1089], & node72, & node62, & node279], & & \\ [[1405,6060,1089], & node73, & node63, & node280], & & \\ [[2129,6060,1089], & node74, & node64, & node281], & & \\ [[2853,6060,1089], & node75, & node65, & node282], & & \\ [[3577,6060,1089], & node76, & node66, & node283] & & \end{bmatrix} \quad (7.5)$$

U svakom retku ovakvog polja je lista, gdje je prvi član lista jedinstvenih koordinata [*x*, *y*, *z*] na kojima dolazi do preklapanja čvorova, a preostali članovi su objekti čvorova `node` koji se nalaze na tim koordinatama. Ako su koordinate identificiranog čvora već upisane u polje, preklopljeni čvorovi se upisuju uz pripadajuće koordinate na kraj liste. Ako koordinate nisu upisane, izrađuje se novi redak oblika [[*x*, *y*, *z*], `node1`, `node2`].



Slika 7.22 Dijagram toka metode `check_node_overlap_np`

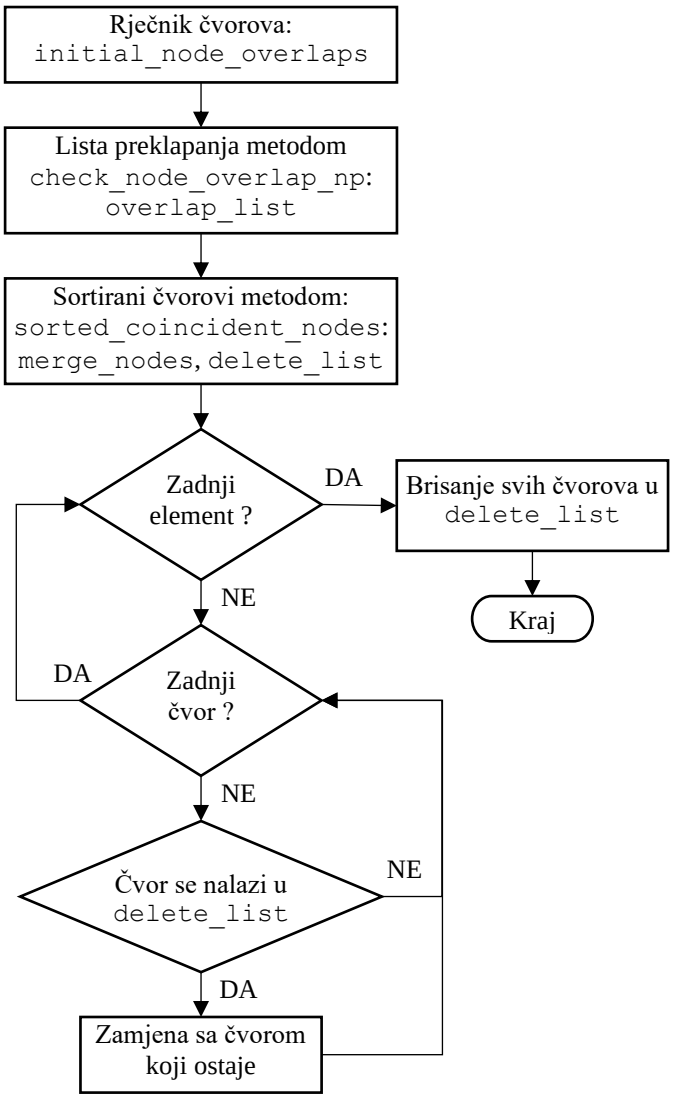
Metodom `sorted_coincident_nodes` se petljom kroz prikazanu listu preklapanja `overlap_list` određuju preklopljeni čvorovi koji će biti obrisani. Dijagram toka ove metode je prikazan slikom 7.23. Odabrano je da prvi upisani čvor na jedinstvenim koordinatama ostaje i upisuje se kao vrijednost u rječniku `merge_nodes`, gdje su ključevi preostali čvorovi koji će biti obrisani. Metoda zajedno sa rječnikom `merge_nodes` vraća sve čvorove koji će biti obrisani, zapisani unutar liste `delete_list`.



Slika 7.23 Dijagram toka metode `sorted_coincident_nodes`

Završna metoda za ispravak preklapanja čvorova `merge_coincident_nodes` preuzima rječnik `initial_node_overlaps` u kojemu su sadržani svi čvorovi na rubovima zona oplate i strukova segmenata duž kojih se očekuje preklapanja čvorova. Za ove čvorove se određuju preklapanja metodom `check_node_overlap_np`, a identificirani čvorovi se sortiraju metodom `sorted_coincident_nodes`. Dijagram toka ove metode je prikazan slikom 7.24.

Petljom kroz sve upisane elemente u rječniku `elements` se traže preklopljeni čvorovi koji će biti obrisani tj. koji se nalaze u listi `delete_list`. Kada se pronađe konačni element koji je definiran čvorom koji će biti obrisan, izvršava se zamjena sa čvorom koji preostaje tj. koji je upisan kao vrijednost u rječniku `merge_nodes`. Nakon zamjene čvorova je moguće obrisati sve čvorove koji se nalaze u listi `delete_list`.

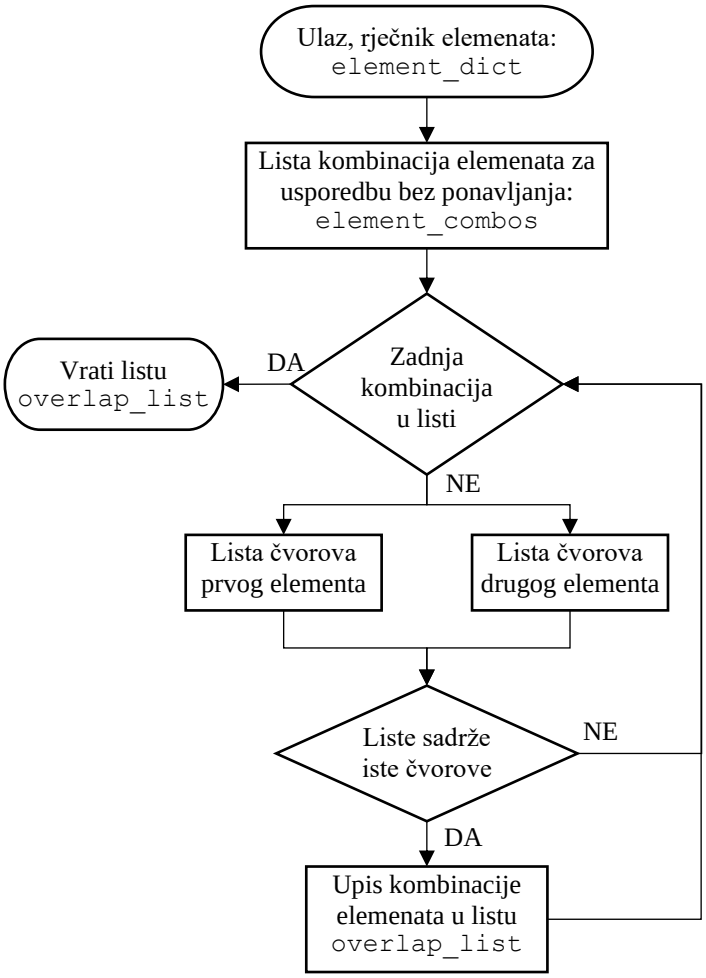


Slika 7.24 Dijagram toka metode `merge_coincident_nodes`

7.3.2 Algoritam za ispravak preklapanja elemenata

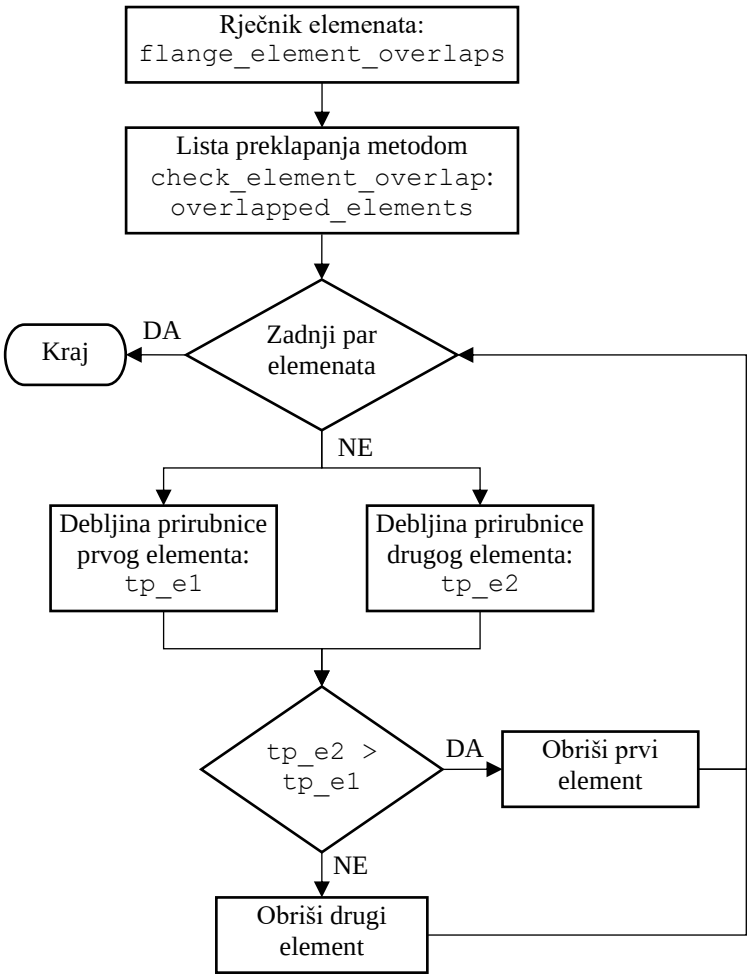
Nakon što je izvršena zamjena preklopljenih čvorova, preklopljeni konačni elementi će biti definirani sa istim čvorovima. Na temelju te činjenice slijedi relativno jednostavna pretraga rječnika elemenata, metodom `check_element_overlap` koja vraća listu parova preklopljenih elemenata `overlap_list`. Dijagram toka ove metode je prikazan slikom 7.25.

Kako bi se izbjegle višestruke usporedbe, funkcijom `itertools.combinations` se izrađuje lista kombinacija elemenata koja omogućuje usporedbu elemenata svakog sa svakim bez ponavljanja. Za svaku kombinaciju se za oba elementa izrađuje lista identifikacijskih brojeva čvorova koji ih definiraju. Usporedba sadrže li ove liste iste brojeve, neovisno o redoslijedu u kojemu su upisani, se izvodi pretvorbom liste u set.



Slika 7.25 Dijagram toka metode `check_element_overlap`

Metoda za ispravak preklapanja elemenata `merge_coincident_elements` preuzima rječnik `flange_element_overlaps` u kojemu su sadržani svi konačni elementi prirubnica jakih nosača na kojima se očekuje preklapanje elemenata. Za ove elemente se određuje lista parova preklopljenih elemenata metodom `check_element_overlap`, nakon čega za svaki par elemenata slijedi odabir koji element će ostati, a koji će biti obrisan. Princip odabira u ovom koraku se temelji na debljini lima prirubnice, slijedeći logiku tehnološke izvedbe kojom će prirubnica veće debljine biti neprekinuta, a prirubnica manje debljine se prekida i vari. Stoga se konačni element manje debljine materijala odmah briše iz rječnika svih elemenata `elements`. Dijagram toka ove metode je prikazan slikom 7.26.

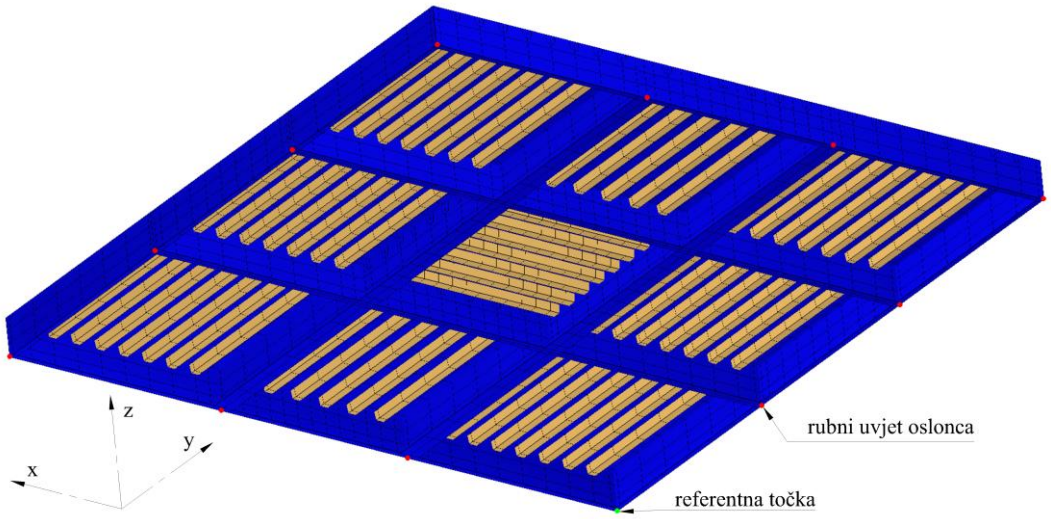


Slika 7.26 Dijagram toka metode `merge_coincident_elements`

8. RUBNI UVJETI I OPTEREĆENJA

8.1 Rubni uvjeti na krajevima jakih nosača

Rubni uvjeti na mjestima oslonaca su zadani prema preporukama *Lloyd's Register, Assesment of Steel Hatch Covers Using Finite Element Analysis, Chapter 2, Section 2: Boundary conditions, 2.2*. Na čvorovima koji se nalaze na krajevima jakih nosača se sprječava translacija u smjeru vertikalne osi [9]. Slikom 8.1 su prikazani postavljeni rubni uvjeti na primjeru pune mreže konačnih elemenata ispitne varijante konstrukcije `hc_var_5`. Crvenim točkama su označeni rubni uvjeti oslonca na kojima su spriječene translacije u smjeru vertikalne osi *z*. Zelenom točkom je označena referentna točka u ishodištu koordinatnog sustava, gdje su postavljeni rubni uvjeti koji sprječavaju pomake modela kao krutog tijela. U ovom čvoru su osim rubnog uvjeta oslonca postavljeni i rubni uvjeti koji sprječavaju translacije u smjeru globalnih *x* i *y* osi.



Slika 8.1 Rubni uvjeti, puna mreža ispitne varijante `hc_var_5`

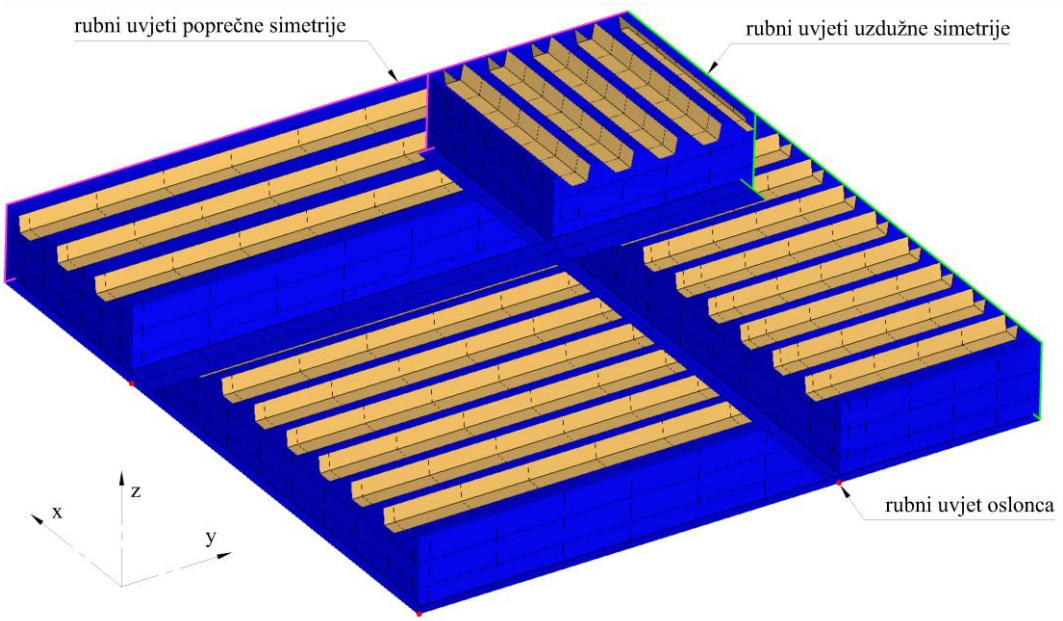
Metodom `identify_both_psm_ends` unutar klase `MeshExtent` se identificiraju koordinate krajeva svih jakih nosača koji su uključeni u opseg izrade mreže, s obzirom na os simetrije za koju je izrađena mreža. Metodom `vertical_bc_node_group` unutar klase `GrillageMesh` se zatim pretražuje koji čvorovi se nalaze na tim koordinatama. Svi pronađeni čvorovi se dodaju u grupu čvorova 1, te se za tu grupu čvorova postavljaju rubni uvjeti metodom `generate_pinned_bc`. Čvor u referentnoj točki se određuje metodom `origin_bc_node` i dodaje u grupu broj 2.

8.2 Rubni uvjeti simetrije

Rubni uvjeti simetrije su zadani prema preporukama *Lloyd's Register, Assesment of Steel Hatch Covers Using Finite Element Analysis, Chapter 2, Section 2: Boundary conditions, 2.5*. Na čvorovima koji se nalaze na osima simetrije se primjenjuju sljedeći rubni uvjeti [9]:

- Za simetriju oko uzdužne osi x, spriječene su translacije u smjeru *y* i rotacije oko *x* i *z* osi
- Za simetriju oko poprečne osi y, spriječene su translacije u smjeru *x* i rotacije oko *y* i *z* osi

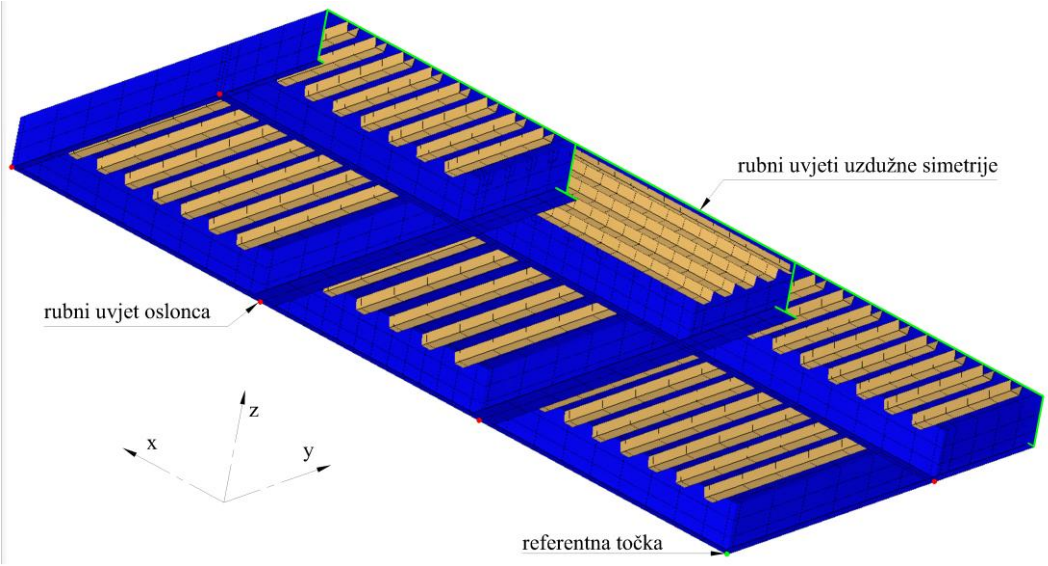
Slika 8.2 prikazuje primjer zadanih rubnih uvjeta na četvrtinskoj mreži konačnih elemenata ispitne varijante konstrukcije `hc_var_5`. Ružičastom linijom su označeni rubovi modela duž kojih su na čvorovima postavljeni rubni uvjeti poprečne simetrije, a zelenom linijom rubni uvjeti uzdužne simetrije. Crvenim točkama su označeni rubni uvjeti oslonaca, gdje su spriječene translacije u smjeru osi *z*. Na četvrtinskoj mreži konačnih elemenata ne postoji referentna točka u ishodištu tj. poseban čvor u kojemu se sprječavaju pomaci modela kao krutog tijela.



Slika 8.2 Rubni uvjeti, četvrtinska mreža ispitne varijante `hc_var_5`

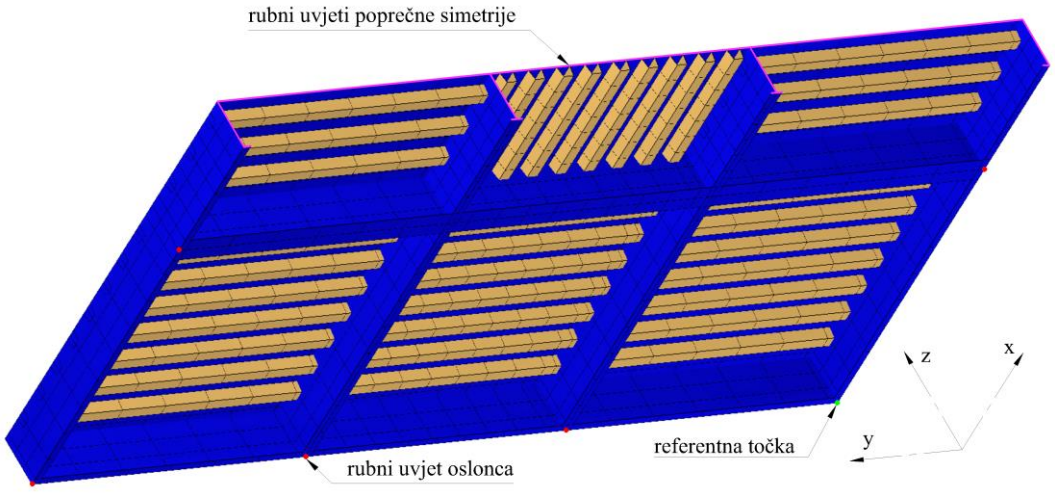
Metodom `long_symm_bc_node_group` se pretražuju svi čvorovi koji se nalaze na ravnini uzdužne simetrije i dodaju u grupu čvorova broj 4. Metodom `tran_symm_bc_node_group` se pretražuju svi čvorovi koji se nalaze na ravnini poprečne simetrije i dodaju u grupu čvorova broj 5. Rubni uvjeti simetrije se za ove grupe čvorova postavljaju metodom `generate_symm_bc` u ovisnosti o osi simetrije za koju je izrađena mreža konačnih elemenata.

Slika 8.3 prikazuje primjer zadanih rubnih uvjeta na uzdužnoj polovičnoj mreži ispitne varijante konstrukcije `hc_var_5`. Zelenom linijom su označeni rubovi modela duž kojih su na čvorovima postavljeni rubni uvjeti uzdužne simetrije. Crvenim točkama su označeni rubni uvjeti oslonaca, gdje su spriječene translacije u smjeru osi *z*. Zelenom točkom je označena referentna točka u ishodištu koordinatnog sustava, postavljen rubni uvjet koji sprječava pomak modela kao krutog tijela. Na uzdužnim polovičnim mrežama se u tom čvoru, osim translacije u smjeru z osi, dodatno sprječava translacija u smjeru globalne *x* osi.



Slika 8.3 Rubni uvjeti, uzdužna polovična mreža ispitne varijante `hc_var_5`

Slikom 8.4 je prikazana poprečna polovična mreža iste ispitne varijante `hc_var_5`. Za ovaj tip mreže je u čvoru koji se nalazi na referentnoj točki dodatno spriječena translacija u smjeru globalne *y* osi.



Slika 8.4 Rubni uvjeti, poprečna polovična mreža ispitne varijante `hc_var_5`

8.3 Opterećenje tlakom

Opterećenje grotlenog poklopca se određuje kao vanjski tlak prema *IACS CSR, Chapter 4, Section 5, 5. External pressures on hatch covers* [7]. Prilikom izrade pločastih konačnih elemenata opločenja se svaki element oplate sprema u poseban rječnik pomoćnom metodom `add_to_plate_elements`. Ovaj rječnik se nalazi u pripadnom objektu mreže konačnih elemenata `GeoGrillageFEM`, koji je definiran unutar Python modula `grillage_fem.py`.

Metodom `generate_pressure_load` u klasi `GrillageMesh` se svaki element oplate unutar ovog rječnika dodaje u grupu broj 6, na koju se zatim zadaje tlak metodom `add_pressure_load`. Nakon izrade mreže konačnih elemenata je metodom `generate_loadcase` moguće odabrati vrijednost projektnog tlaka koji će biti zadan na sve elemente opločenja konstrukcije.

8.4 Opterećenje vlastitom težinom

Prema pravilima *IACS CSR, Chapter 7, Section 1, 1.5. Applied loads* za direktni proračun jakih nosača brodova > 150m metodom konačnih elemenata, konstrukcija mora biti opterećena vlastitom težinom. Za standardnu gustoću čelika se pri tome uzima vrijednost 7.85t/m^3 [7].

Prema preporukama *Lloyd's Register, Chapter 2, Section 3: General load application,* težina strukture grotlenog poklopca mora biti uključena u analizi metodom konačnih elemenata [9]. Metodom `add_self_weight` unutar klase `GeoGrillageFEM` je omogućeno zadavanje opterećenja vlastitom težinom za odabranu vrijednost gravitacije. Opterećenja vlastitom težinom se dodaje postojećem slučaju opterećenja metodom `generate_self_weight` unutar klase `GrillageMesh`.

9. PRIMJENA MODULA ZA AUTOMATIZIRANU PRIPREMU MKE MODELA GROTLENOG POKLOPCA

9.1 Ispitne varijante konstrukcija

Ispitivanje automatizirane izrade mreže konačnih elemenata je provedeno na ukupno 8 ispitnih varijanti različitih karakteristika navedenih u tablici 9.1. Prva ispitna varijanta `hc_var_1` je bazirana na izvedenoj konstrukciji grotlenog poklopca broda za prijevoz rasutog tereta prema dobivenom predlošku. Većina ostalih ispitnih varijanti su modifikacije te izvedene konstrukcije, na kojima su testirani algoritmi za izradu mreže u različitim graničnim slučajevima. Zadnja ispitna varijanta konstrukcije `hc_var_8` je projektirana u okviru završnog rada: Projektiranje konstrukcije grotlenog poklopca broda za prijevoz rasutog tereta [13], a odabrana je zbog postojećih rezultata analize metodom konačnih elemenata u programu Maestro.

Tablica 9.1 Topološke karakteristike ispitnih varijanti konstrukcija

Ispitna varijanta	Simetrija	Broj uzdužnih nosača	Broj poprečnih nosača	Broj ukrepa između nosača	Sustav orebrenja
`hc_var_1`	Uzdužna	5	5	4, 5	Poprečni
`hc_var_2`	Uzdužna	5	5	4, 5, 6	Mješoviti
`hc_var_3`	Uzdužna	5	5	2, 5	Mješoviti
`hc_var_4`	Obostrana	4	4	8	Mješoviti
`hc_var_5`	Obostrana	4	4	6, 8, 9	Mješoviti
`hc_var_6`	Obostrana	5	5	4, 5, 6	Mješoviti
`hc_var_7`	Obostrana	5	4	5, 8	Mješoviti
`hc_var_8`	Obostrana	4	10	14	Uzdužni

Tablicom 9.2 su prikazane specifičnosti pojedine ispitne varijante zbog kojih su odabrane. Varijante sa parnim brojem jakih nosača su služile za ispitivanje algoritama za identifikaciju i izradu mreže na polovičnim i četvrtinskim zonama oplate. Na varijantama sa neparnim brojem jakih nosača su testirani algoritmi za prepoznavanje nosača na osi simetrije i dodjelu polovičnih svojstava krutosti. Ispitnim varijantama su također obuhvaćeni slučajevi položaja osi simetrije na i između ukrepa.

Tablica 9.2 Specifičnosti ispitnih varijanti konstrukcija

Ispitna varijanta	Napomena
`hc_var_1`	Izvedena konstrukcija grotlenog poklopca
`hc_var_2`	Mješoviti sustav orebrenja sa različitim brojem ukrepa
`hc_var_3`	Mješoviti sustav orebrenja sa značajnim razlikama razmaka između ukrepa
`hc_var_4`	Paran broj jakih uzdužnih i poprečnih nosača, obostrano simetrična konstrukcija, uzdužna os simetrije prolazi između uzdužnih ukrepa
`hc_var_5`	Modifikacija varijante `hc_var_4`, različite širine prirubnica jakih nosača, uzdužna ukrepa se nalazi na uzdužnoj osi simetrije
`hc_var_6`	Modifikacija varijante `hc_var_2`, obostrano simetrična konstrukcija, izmjena razmaka između jakih nosača, razmaka i tipa ukrepa
`hc_var_7`	Kombinacija neparnog broja jakih uzdužnih i parnog broja jakih poprečnih nosača sa mješovitim sustavom orebrenja
`hc_var_8`	Odabrana konstrukcija grotlenog poklopca projektirana u okviru završnog rada, topologija 4x10x14L [13]

U prilogu II se nalaze slike modela konstrukcije u programu d3v-sgd i izrađene ispitne mreže konačnih elemenata, a u prilogu III se nalaze nacrti izrađenih ispitnih varijanti. Za svaku ispitnu varijantu su prikazane dostupne varijante mreže sa inicijalno postavljenim parametrima za kontrolu mreže i automatski prepoznatom osi simetrije. Uz slike generiranih ispitnih mreža je naveden broj čvorova, ukupan broj elemenata i broj pojedinih vrsta konačnih elemenata.

Obje varijante mreža *V1* i *V2* uspješno generiraju mrežu konačnih elemenata za sve prikladne ispitne varijante konstrukcija. Budući da je model ispitne varijante `hc_var_5` prilagođen za ispitivanje izrade mreže na različitim širinama prirubnica duž jakih nosača, izrada varijante mreže *V1* nije moguća na tom modelu zbog već opisanih ograničenja. Varijanta mreže *V2* u prosjeku generira mrežu konačnih elemenata sa 18% manje čvorova i konačnih elemenata, čime je vrijeme potrebno za izradu mreže i prikaz primjetno kraće.

9.2 Proračun odziva konstrukcije primjenom generiranih MKE modela

Proračun odziva metodom konačnih elemenata proveden je varijantom programa OOFEM [14] koja je prilagođena za linearnu analizu tankostjenih brodskih konstrukcija [15], [16]. Navedena varijanta programa OOFEM prethodno je integrirana u d3v-sgd primjenom Python sučelja. Proračun odziva proveden je za ispitnu varijantu `hc_var_8`. Ulazni podaci modela su postavljeni u sljedećem sustavu mjernih jedinica:

- Duljina: [mm]
- Masa: [t]
- Sila: [N]
- Brzina: [s]

Materijal izrade ove varijante konstrukcije je čelik povišene čvrstoće AH36 sljedećih karakteristika:

Modul elastičnosti:

$$E = 210000\ \mathrm{N/mm^2}$$

Poissonov koeficijent:

$$\nu = 0{,}3$$

Gustoća:

$$\rho = 7{,}85 \cdot 10^{-9}\ \mathrm{t/mm^2}$$

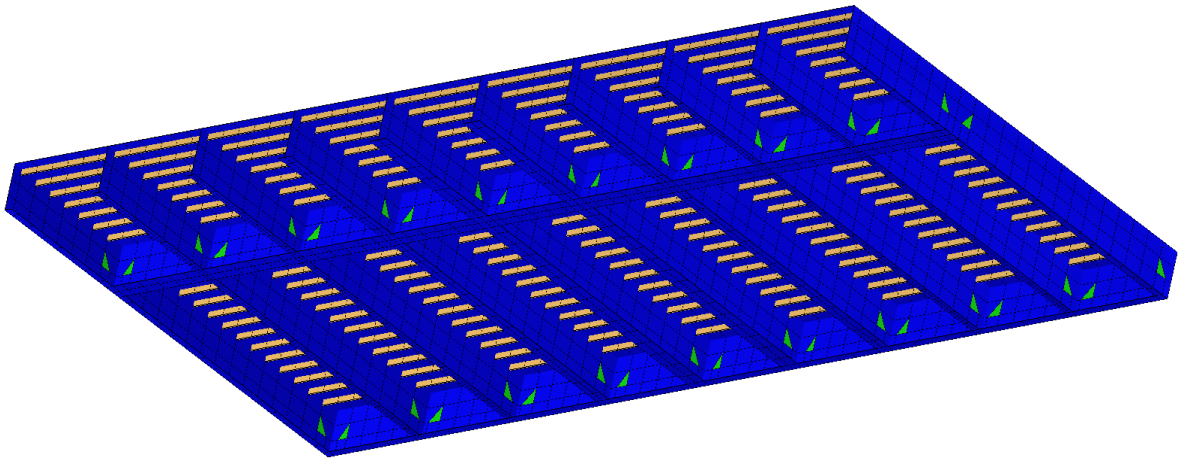
Prilikom definicije opterećenja vlastitom težinom je prema odabranom sustavu mjernih jedinica postavljen iznos ubrzanja sile teže:

$$g = 9810\ \mathrm{mm/s^2}$$

Projektni tlak kojim je opterećena konstrukcija:

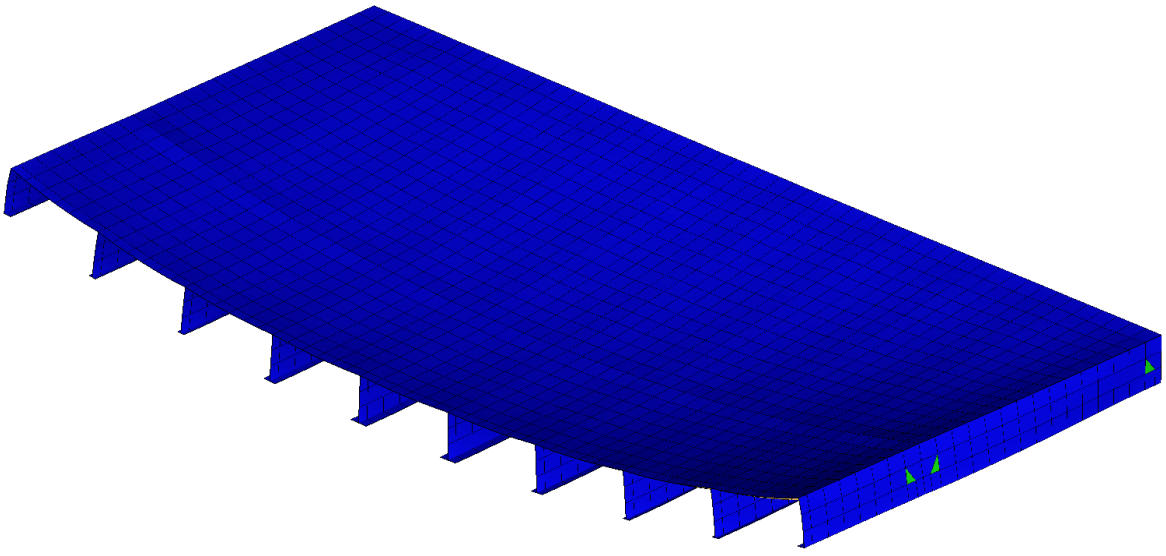
$$p = 0{,}0343\ \mathrm{N/mm^2}$$

Za izradu mreže konačnih elemenata je odabrana varijanta mreže *V2*, prikazana slikom 9.1. Prilikom izrade mreže su zadržani svi inicijalni kontrolni parametri osim poželjnog aspektnog odnosa oplate, za koji je odabrana vrijednost 1.0. Umjesto automatskog prepoznavanja osi simetrije je odabrana izrada uzdužne polovične mreže konačnih elemenata.



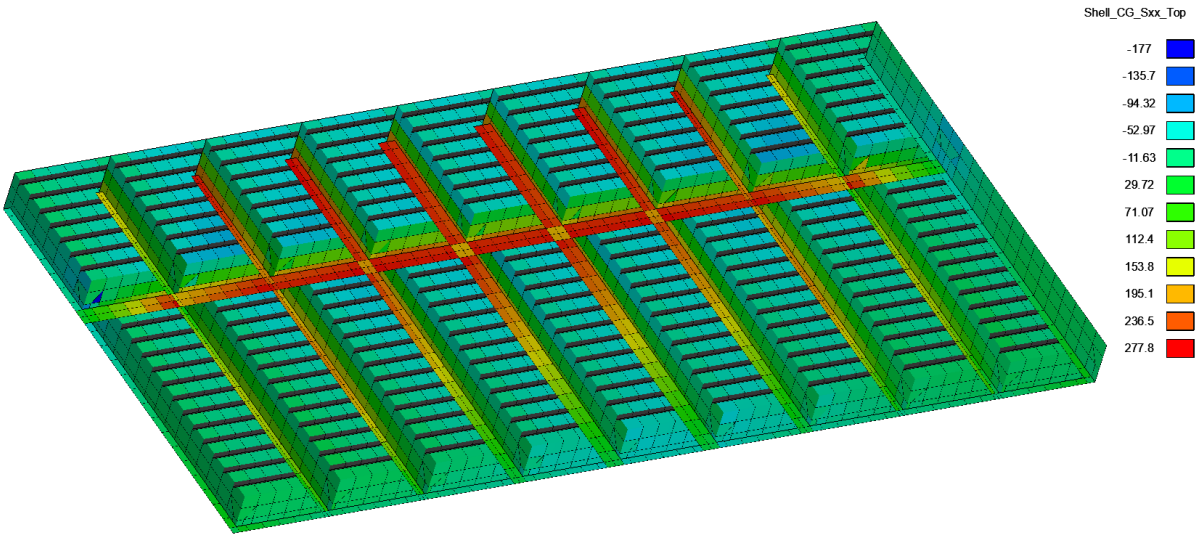
Slika 9.1 Uzdužna polovična mreža konačnih elemenata, ispitna varijanta `hc_var_8`

Slikom 9.2 je prikazan deformirani model konstrukcije uz odabrani faktor povećanja deformacije 15 (eng. *Deformation scale factor*). Maksimalni progib konstrukcije iznosi 79,37mm.



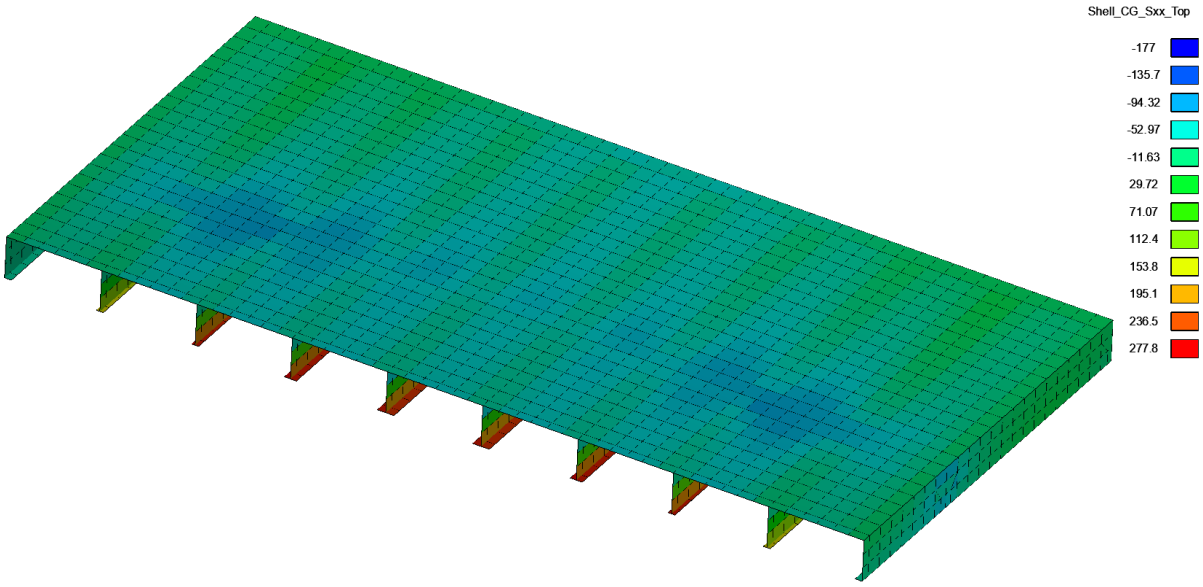
Slika 9.2 Progib ispitne varijante `hc_var_8` uz faktor povećanja deformacije 15

Slika 9.3 prikazuje naprezanja u prirubnicama jakih nosača, pri čemu su naprezanja prikazana u aksijalnom smjeru pojedinog jakog nosača. Drugim riječima, slika prikazuje naprezanja u smjeru globalne *x* osi za uzdužno usmjerene nosače i naprezanja u smjeru *y* osi za poprečno usmjerene nosače. Najveće naprezanje u prirubnicama iznosi 277,8 N/mm^2.



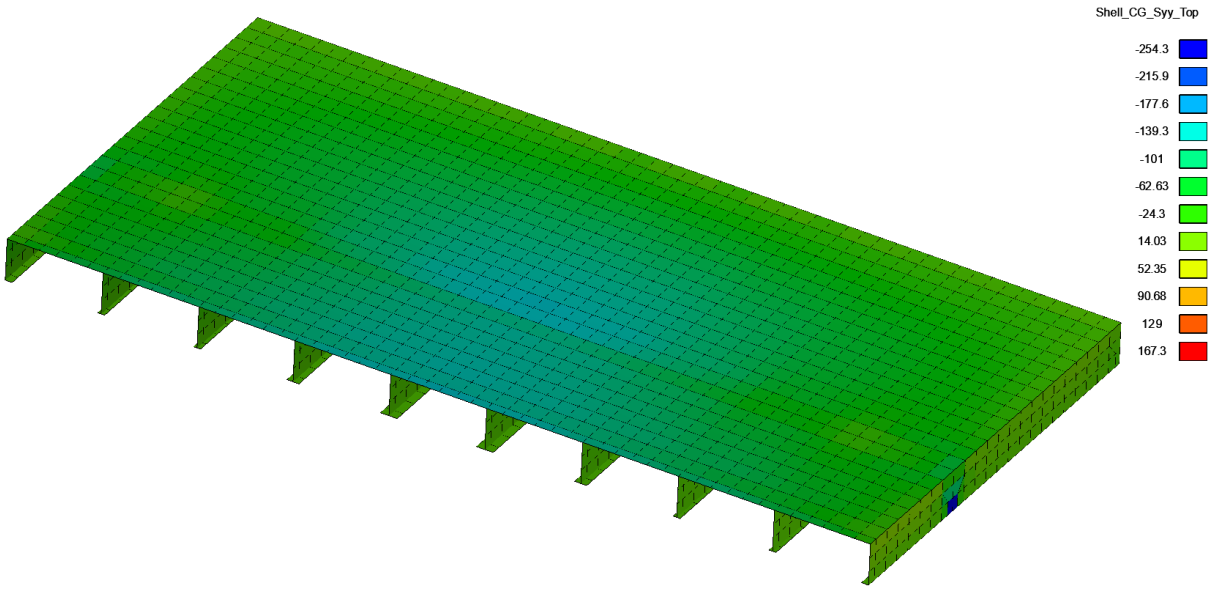
Slika 9.3 Naprezanja u prirubnicama ispitne varijante `hc_var_8`

Slika 9.4 prikazuje naprezanja u oplati u smjeru globalne osi *x*.



Slika 9.4 Naprezanja u oplati, *x* smjer, ispitna varijanta `hc_var_8`

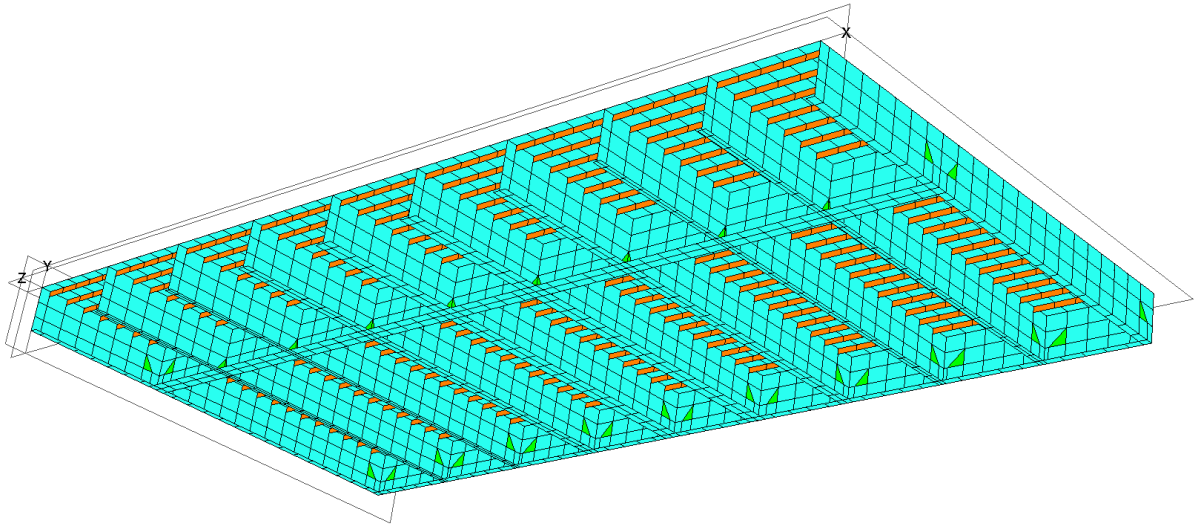
Slika 9.5 prikazuje naprezanja u oplati u smjeru globalne osi y.



Slika 9.5 Naprezanja u oplati, y smjer, ispitna varijanta `hc_var_8`

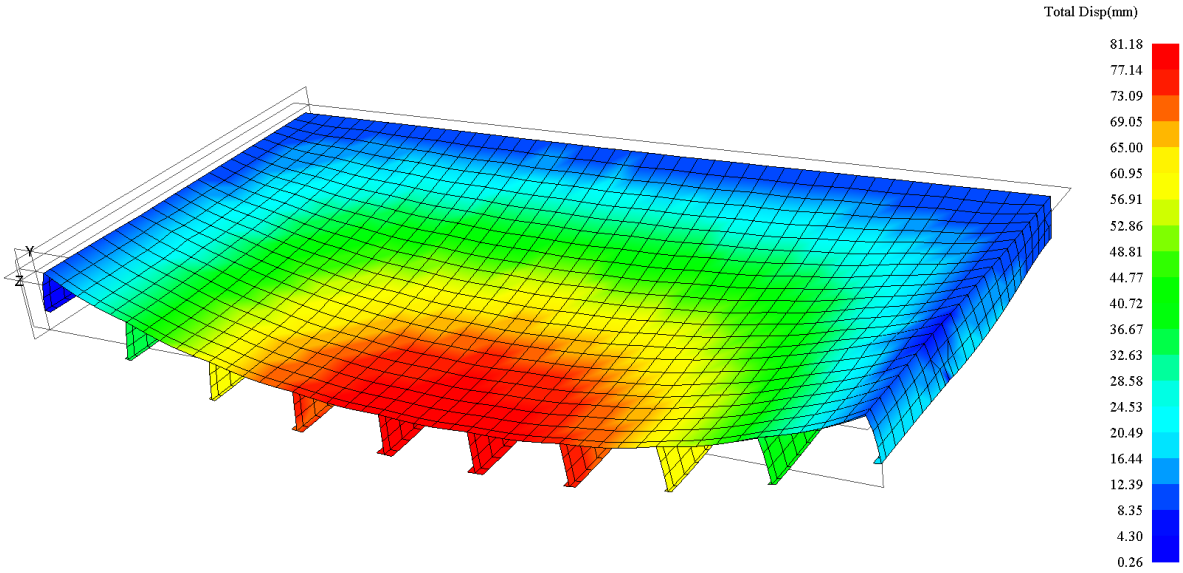
9.3 Validacija rezultata

Rezultati analize odziva metodom konačnih elemenata u programu d3v-sgd su uspoređeni sa rezultatima analize topologije 4x10x14L dobivenim u okviru završnog rada: Projektiranje konstrukcije grotlenog poklopca broda za prijevoz rasutog tereta [13]. Slika 9.6 prikazuje mrežu konačnih elemenata odabrane topologije 4x10x14L koja je izrađena u programu Maestro. Ova mreža konačnih elemenata je ekvivalentna uzdužnoj polovičnoj mreži ispitne varijante hc_var_8 koja je prikazana u poglavlju 9.2.



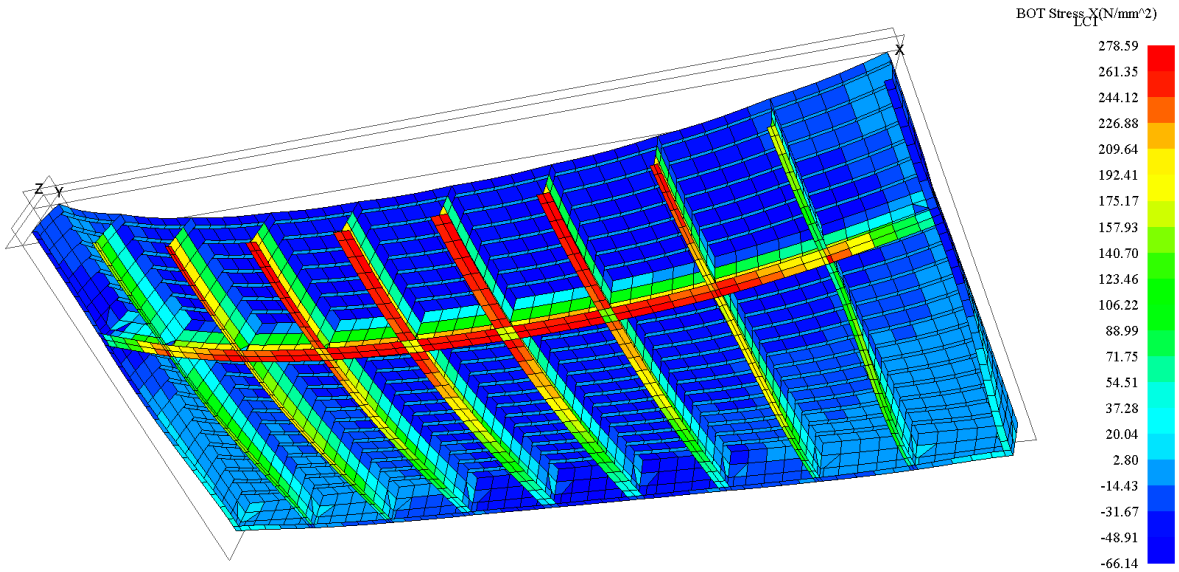
Slika 9.6 Polovični model topologije 4x10x14L [13]

Slika 9.7 prikazuje progib konstrukcije u programu Maestro. Maksimalni progib iznosi 81,18mm, prema čemu relativno odstupanje rezultata dobiveno u programu d3v-sgd iznosi -2,23%.



Slika 9.7 Progib konstrukcije, topologija 4x10x14L [13]

Slika 9.8 prikazuje naprezanja u prirubnicama u programu Maestro. Najveće vlačno naprezanje iznosi 278,59 N/mm^2, prema čemu relativno odstupanje rezultata dobiveno u programu d3v-sgd iznosi -0,39%.



Slika 9.8 Naprezanja u prirubnicama, topologija 4x10x14L [13]

10. ZAKLJUČAK

Kroz ovaj rad je izrađen modul koji omogućuje automatsku generaciju mreže konačnih elemenata učitanog modela konstrukcije u programu d3v-sgd za modeliranje i vizualizaciju roštilja brodskih konstrukcija. Algoritmi za diskretizaciju konstrukcije omogućuju izradu mreže na dva načina kroz dvije različite implementacije varijanti mreža. Varijanta V1 generira mrežu korištenjem isključivo pravokutnih quad elemenata tehnikom preslikavanja elemenata prirubnice na opločenje. Varijanta V2 generira pravilniju mrežu opločenja sa prijelazom na mrežu prirubnica i kombinacijom četverokutnih i trokutastih konačnih elemenata. Sadržane u algoritmima su metode za definiciju slučaja opterećenja i automatsko postavljanje rubnih uvjeta, koje omogućuju izradu OOFEM ulazne datoteke i analizu učitanog modela konstrukcije metodom konačnih elemenata.

Izrađeno je grafičko sučelje u okviru programa d3v-sgd za izradu mreže konačnih elemenata na učitanom modelu konstrukcije, koje korisniku omogućuje odabir varijante mreže i svih kontrolnih parametara za izradu mreže konačnih elemenata. Proširen je prethodno izrađeni programski kod za definiciju modela roštiljne konstrukcije, implementacijom raznih metoda koje omogućuju jednostavniju identifikaciju položaja strukturnih elemenata, vraćaju potrebne podatke za vizualizaciju modela, analizu podobnosti i ostalo.

U okviru rada je proširen već postojeći modul za automatiziranu pripremu ulaznih podataka za analizu metodom konačnih elemenata programom OOFEM, implementacijom izvedenih izraza za momente inercije i omogućivanjem prikaza poprečnog presjeka grednih konačnih elemenata svih korištenih tipova profila. Proširen je i postojeći kod za vizualizaciju mreže konačnih elemenata, omogućivanjem prikaza deformacija pravokutnih, trokutastih i grednih konačnih elemenata sadržanih na modelu prema rezultatima analize metodom konačnih elemenata.

Testiranje implementiranih algoritama za izradu mreže konačnih elemenata je provedeno na ukupno 8 različitih ispitnih varijanti. Ove ispitne varijante se razlikuju po simetriji, broju, razmaku i dimenzijama strukturnih elemenata, te obuhvaćaju predviđene permutacije specifičnih slučajeva koje automatska izrada mreže konačnih elemenata uspješno savladava. Rezultati analize metodom konačnih elemenata u d3v-sgd su uspoređeni sa postojećim rezultatima analize u programu Maestro za istu konstrukciju grotlenog poklopca i ekvivalentnu mrežu konačnih elemenata.

U okviru diplomskog rada utvrđeni su neki mogući daljnji koraci na razvoju programa d3v-sgd:

Modul za modeliranje konstrukcije:

1. Izrada više različitih klasa `Grillage` koje će služiti za opis različitih vrsta modela konstrukcije npr. sa dvostrukom oplatom (pontonski poklopci), sa prelukom, varijabilnim poprečnim presjecima jakih nosača i sl.
2. Opis topoloških varijanti roštiljnih konstrukcija koje nisu pravokutnog oblika, npr. zakrivljeni brid palube broda.
3. Modifikacija programskog koda koja će omogućiti izradu modela konstrukcije bez jakih rubnih nosača za npr. neke konstrukcijske izvedbe podiznih palubica RO-RO brodova.
4. Proširenje objekta `StiffenerLayout` koje će omogućiti fleksibilnije zadavanje ukrepa i dodavanje različitih ukrepa na jednoj zoni oplate.
5. Proširenje definicije segmenta koja će omogućiti podjelu segmenta na više dijelova, a svakom dijelu dodjelu vlastitog svojstva za npr. ojačavanje struka jakog nosača na krajevima.
6. Implementacija novog tipa svojstva grede `VaraibleSection` koji će omogućiti opis varijabilnog poprečnog presjeka jakog nosača.
7. Implementacija novog tipa svojstva grede u obliku kutijastog nosača (eng. *box girder*).
8. Modifikacija objekta `Plate` koja će omogućiti zadavanja zone opločenja bez ukrepa.
9. Proširenje definicije objekta `Plate` dodatnim parametrom koji bi služio za jednoznačno pozicioniranje zone oplate na modelima konstrukcije sa dvostrukom oplatom.
10. Dodjela svojstva oplate `PlateProperty` objektu elementarnog panela opločenja `ElementaryPlatePanel` umjesto objektu zone oplate `Plate`, čime bi se omogućilo zadavanje različite debljine lima na jednoj zoni oplate.

Modul za izradu mreže konačnih elemenata:

1. Izrada algoritma za automatski odabir i globalno usklađivanje broja elemenata po visini struka, prema dimenzijama elemenata na temelju aspektnog odnosa elemenata struka.
2. Proširenje koda za izradu mreže struka koji će omogućiti različite visine jakih nosača, implementacijom algoritma za usklađivanje dimenzija elemenata po visini struka.
3. Izrada algoritma za automatsko povećanje broja elemenata po širini prirubnice za fine mreže konačnih elemenata.
4. Modifikacija varijante mreže *V2* koja će omogućiti izradu vrlo finih mreža konačnih elemenata, implementacijom metode za automatsko povećanje broja elemenata po širini prirubnice.
5. Mogućnost odabira različitih tipova rubnih uvjeta i slučajeva opterećenja, za analizu konstrukcija palubica, jednostrukog ili dvostrukog dna, rampi i sl.
6. Izrada univerzalnog algoritma za izradu pravilne mreže konačnih elemenata na temelju niza objekata rubnih čvorova koji omeđuju promatrani element konstrukcije.
7. Izrada novih varijanti mreža korištenjem različitih tipova konačnih elemenata i različitih tehnika prijelaza sa grublje na finu mrežu konačnih elemenata.
8. Izrada metoda za identifikaciju koji elementi pripadaju kojem dijelu konstrukcije za automatiziranu analizu podobnosti prema preskriptivnim pravilima.

LITERATURA

[1] Sorić, J., Metoda konačnih elemenata, Tehnička knjiga, Zagreb, 2004.

[2] Python, https://www.python.org

[3] Javni repozitorij programa d3v-sgd, https://github.com/pprebeg/d3v-sgd

[4] Javni repozitorij verzije programa d3v-sgd koja je korištena u ovom diplomskom radu, https://github.com/pprebeg/d3v-sgd/tree/gordan_kos_final_thesis

[5] MacGregor, Cargo Handling Book, MacGregor, 2016.

[6] Gamma. E., Design Patterns, 1995.

[7] IACS, Common Structural Rules for Bulk Carriers, July, 2012.

[8] Matejiček, F., Semenski, D., Vnučec, Z., Uvod u statiku, Slavonski Brod, 2012.

[9] Lloyd's Register, Assesment of Steel Hatch Covers Using Finite Element Analysis, July 2007.

[10] Jović, A., Frid, N., Ivošević, D., Procesi programskog inženjerstva, E-skripta, Fakultet elektrotehnike i računarstva, Zagreb, rujan 2019.

[11] Hruška, M., Osnove programiranja (Python), Sveučilišni računski centar, 2008.

[12] NumPy API Reference, https://numpy.org/doc/stable/reference/index.html

[13] Kos, G., Projektiranje konstrukcije grotlenog poklopca broda za prijevoz rasutog tereta, Završni rad, Fakultet strojarstva i brodogradnje, Zagreb, 2021.

[14] Javni repozitorij programa OOFEM, https://github.com/oofem/oofem

[15] Javni repozitorij verzije programa OOFEM koja je modificirana za primjenu u analizi tankostjenih brodskih konstrukcija, https://github.com/unizg-fsb-nav/oofem

[16] Pero Prebeg, Marin Palaversa, Jerolim Andric & Matea Tomicic (2022) Adaptation of FEM-based open-source software for ship structural analysis, Ships and Offshore Structures, DOI: 10.1080/17445302.2022.2035568.

PRILOZI

I. UML dijagrami klasa

II. Slike modela i mreža konačnih elemenata ispitnih varijanti konstrukcija

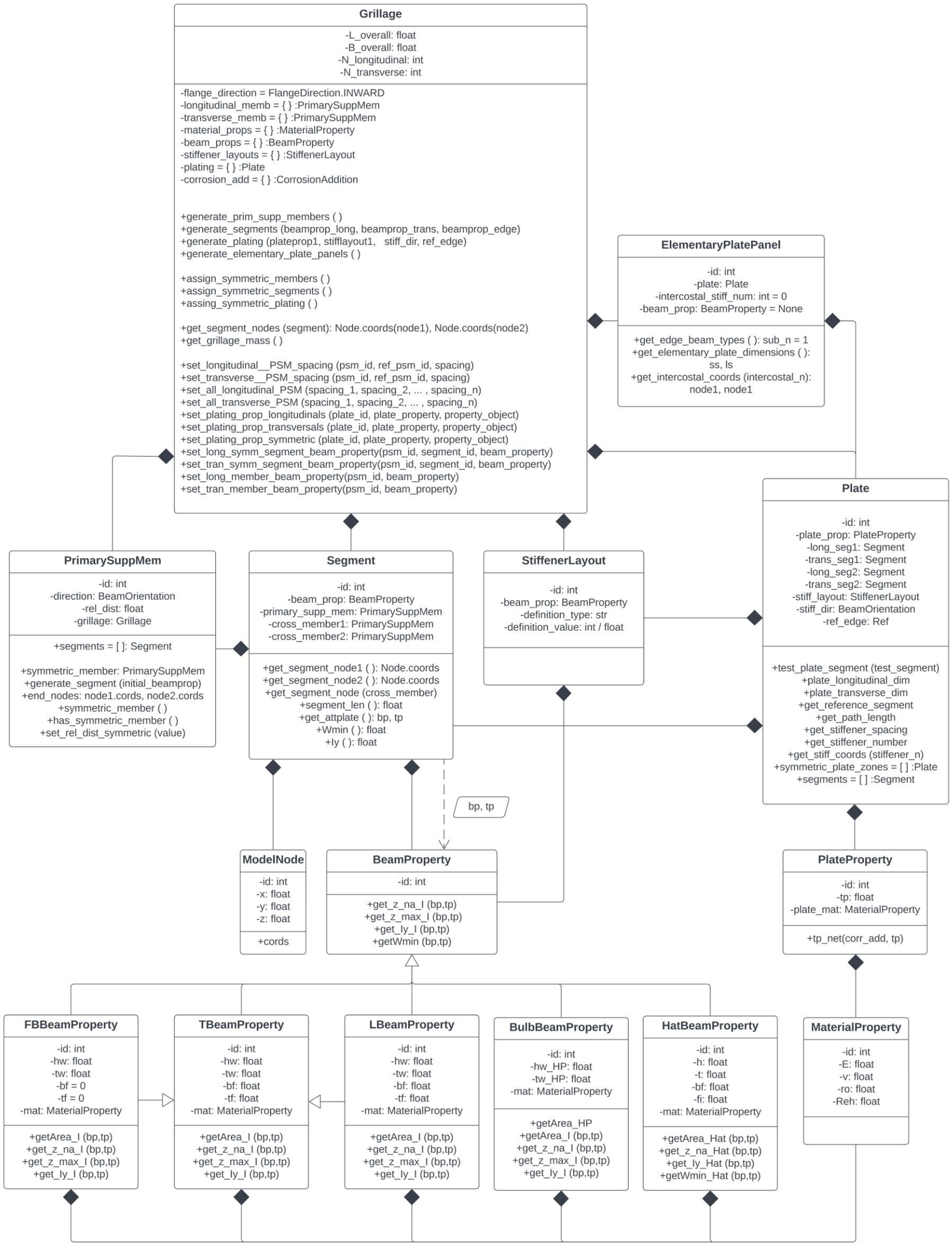
III. Nacrti ispitnih varijanti konstrukcija grotlenih poklopaca

IV. Programski kod modula `grillage_mesher.py`

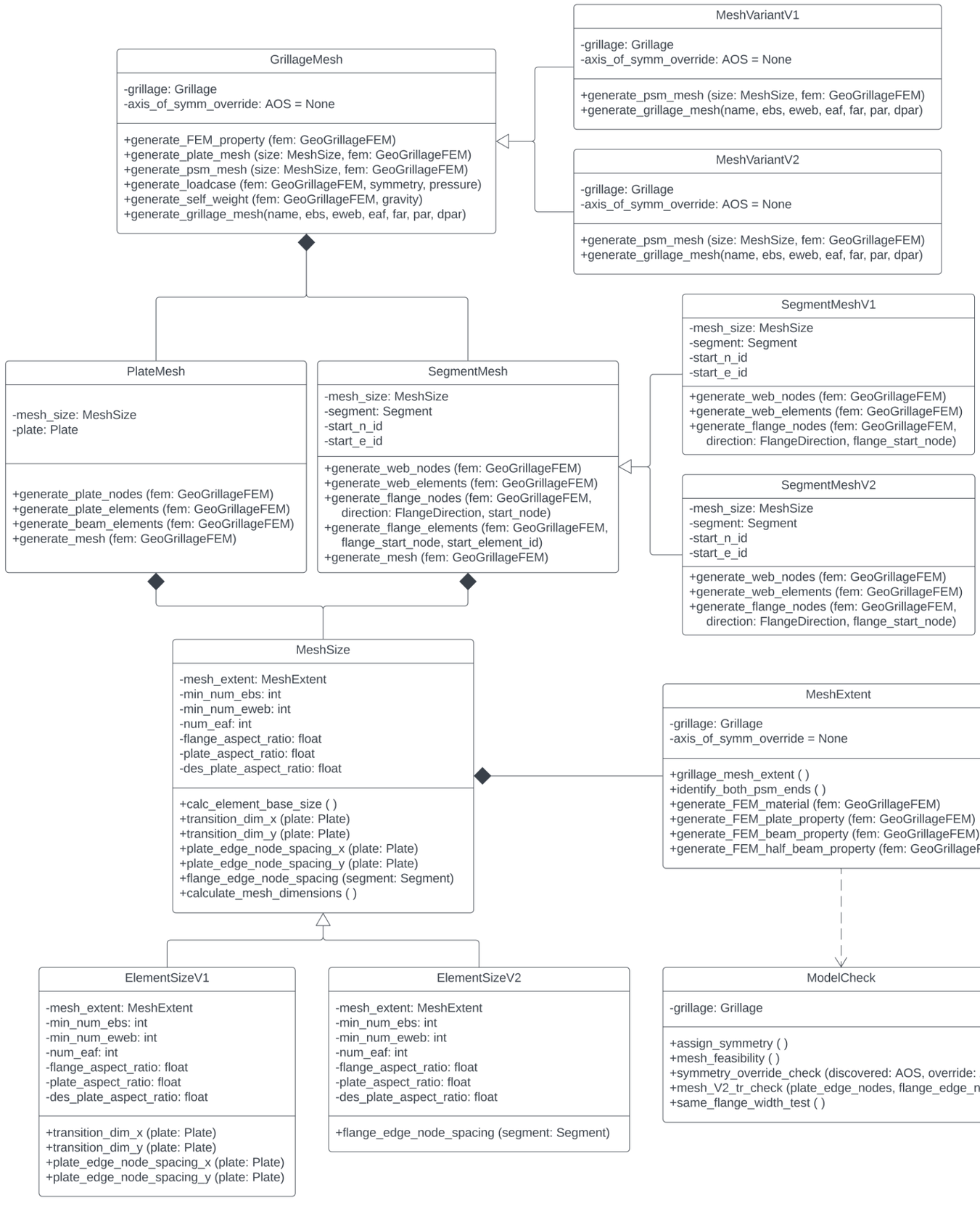
V. Programski kod modula `grillage_fem.py`

PRILOG I

UML dijagrami klasa



Slika 1-1 UML dijagram klasa programskog koda `grillage_model.py`



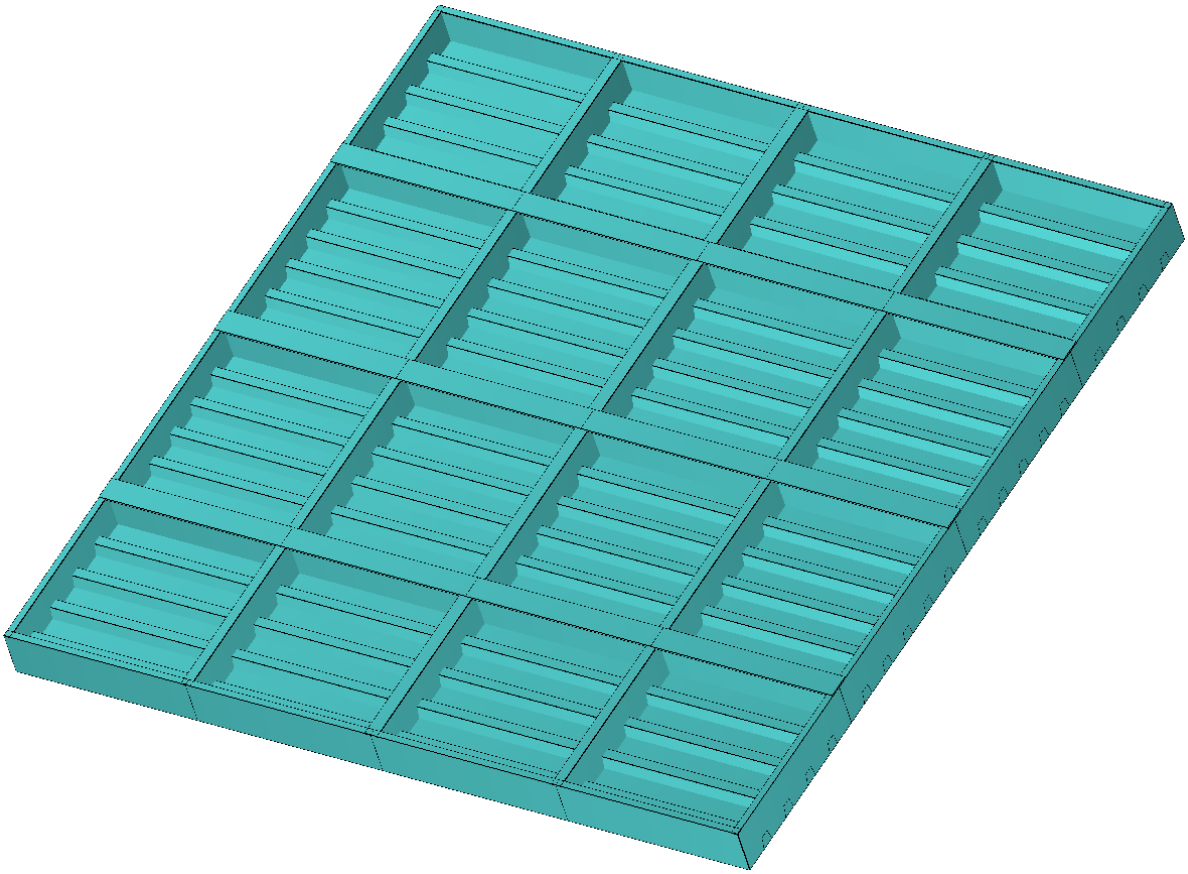
Slika 1-2 UML dijagram klasa programskog koda `grillage_mesher.py`

PRILOG II

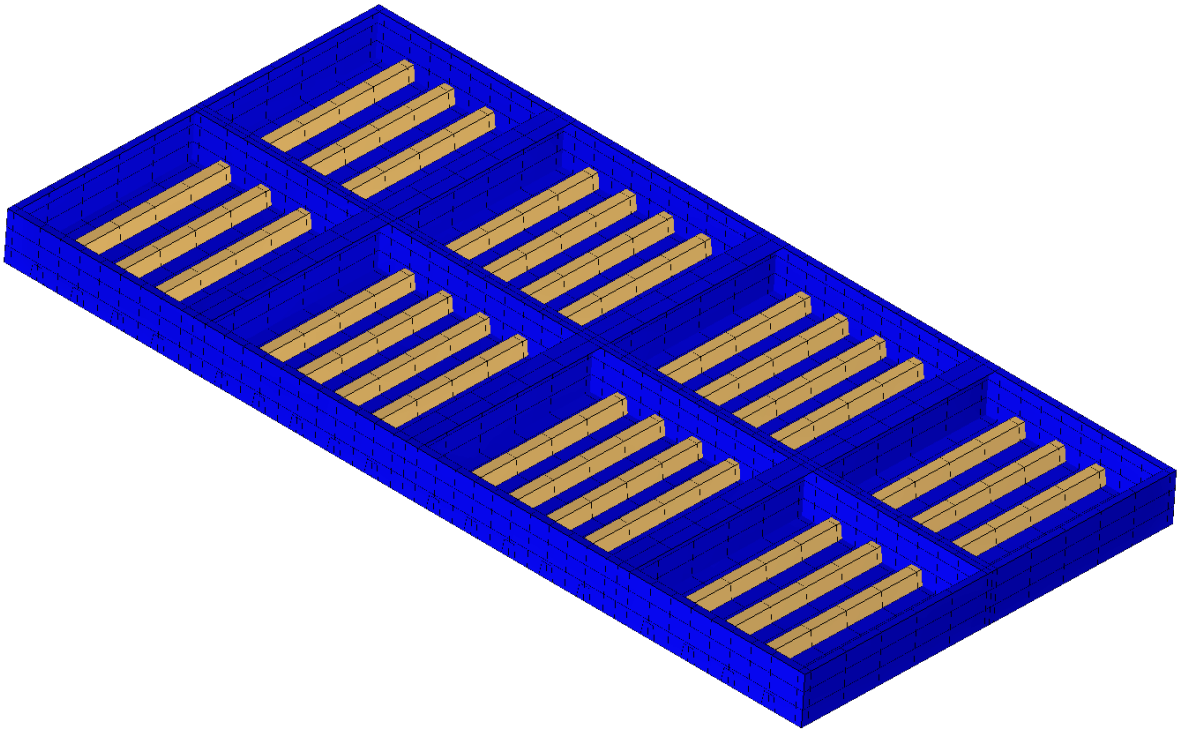
Slike modela i mreža konačnih elemenata ispitnih varijanti konstrukcija

Prilog II: Slike modela i mreža konačnih elemenata ispitnih varijanti konstrukcija

1.) Ispitna varijanta `hc_var_1`

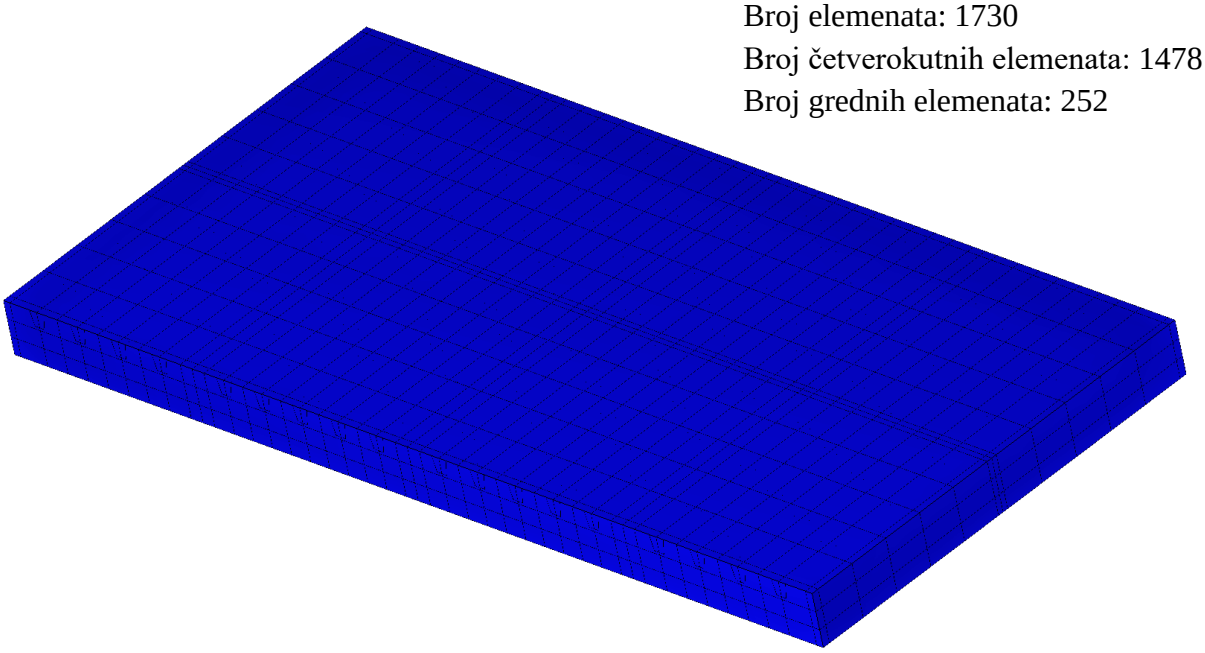


Slika 2-1 Model ispitne varijante konstrukcije `hc_var_1`

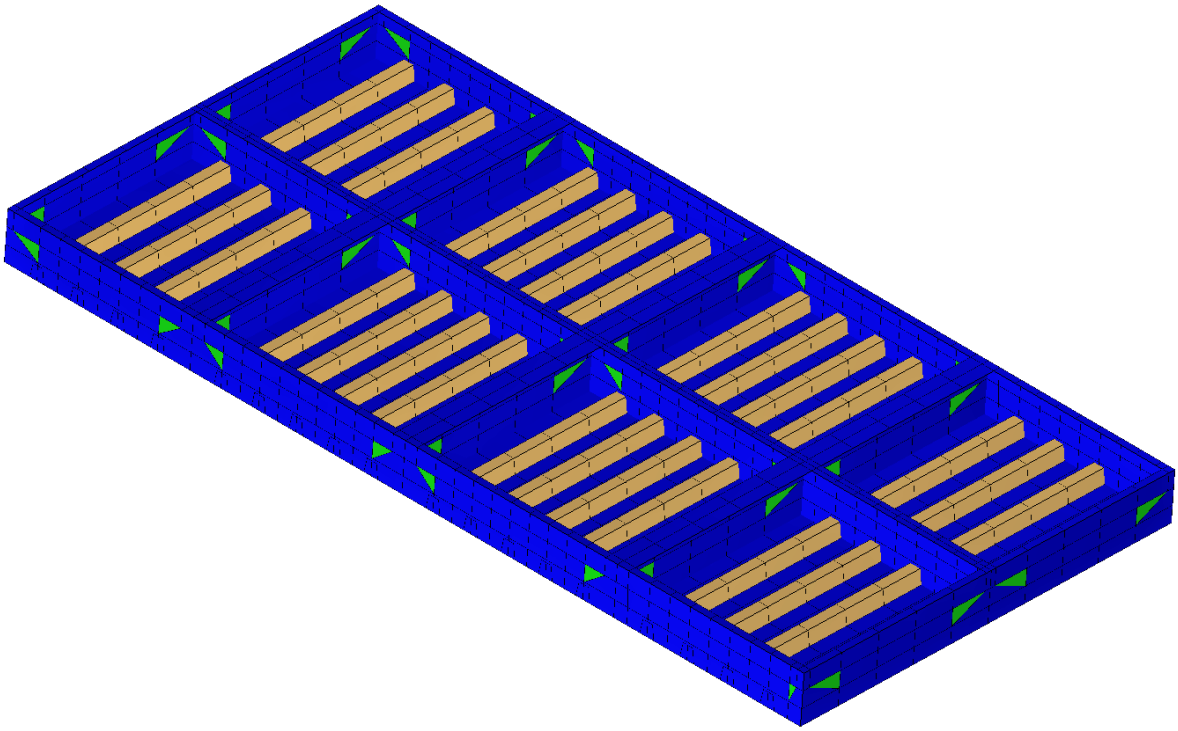


Slika 2-2 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_1`

Varijanta mreže *V1*
Broj čvorova: 1484
Broj elemenata: 1730
Broj četverokutnih elemenata: 1478
Broj grednih elemenata: 252

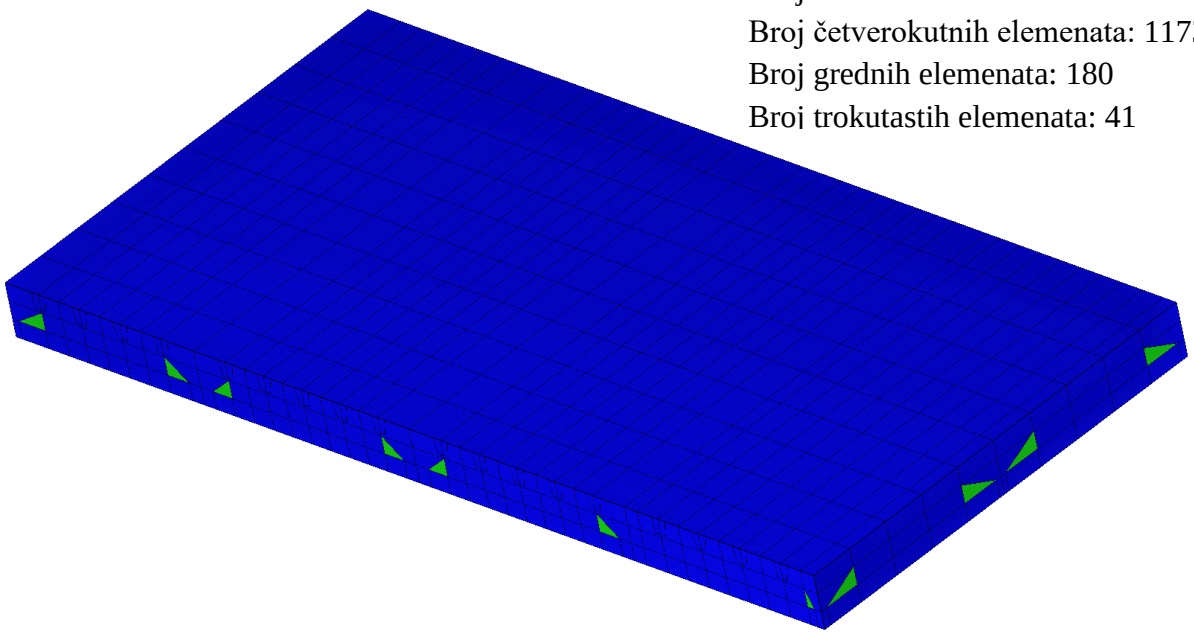


Slika 2-3 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_1`



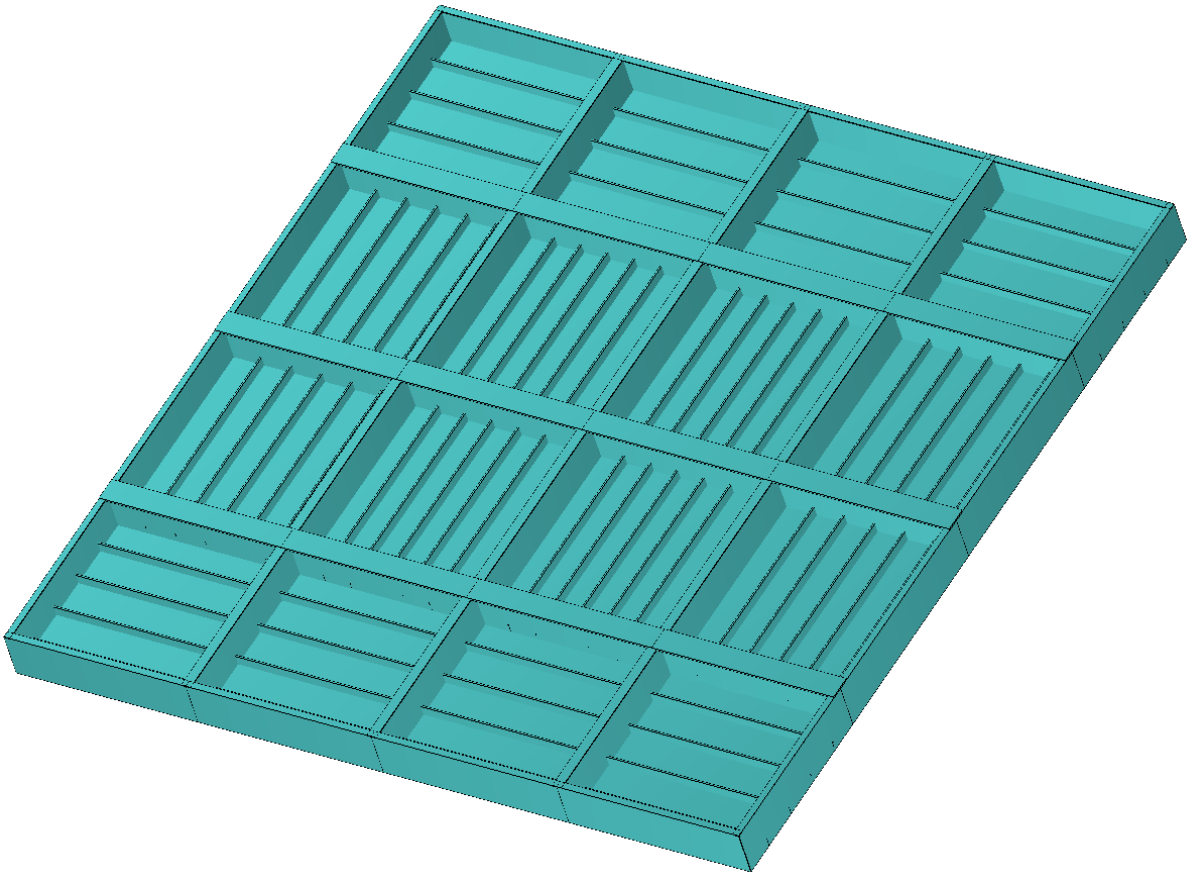
Slika 2-4 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_1`

Varijanta mreže *V2*
Broj čvorova: 1210
Broj elemenata: 1394
Broj četverokutnih elemenata: 1173
Broj grednih elemenata: 180
Broj trokutastih elemenata: 41

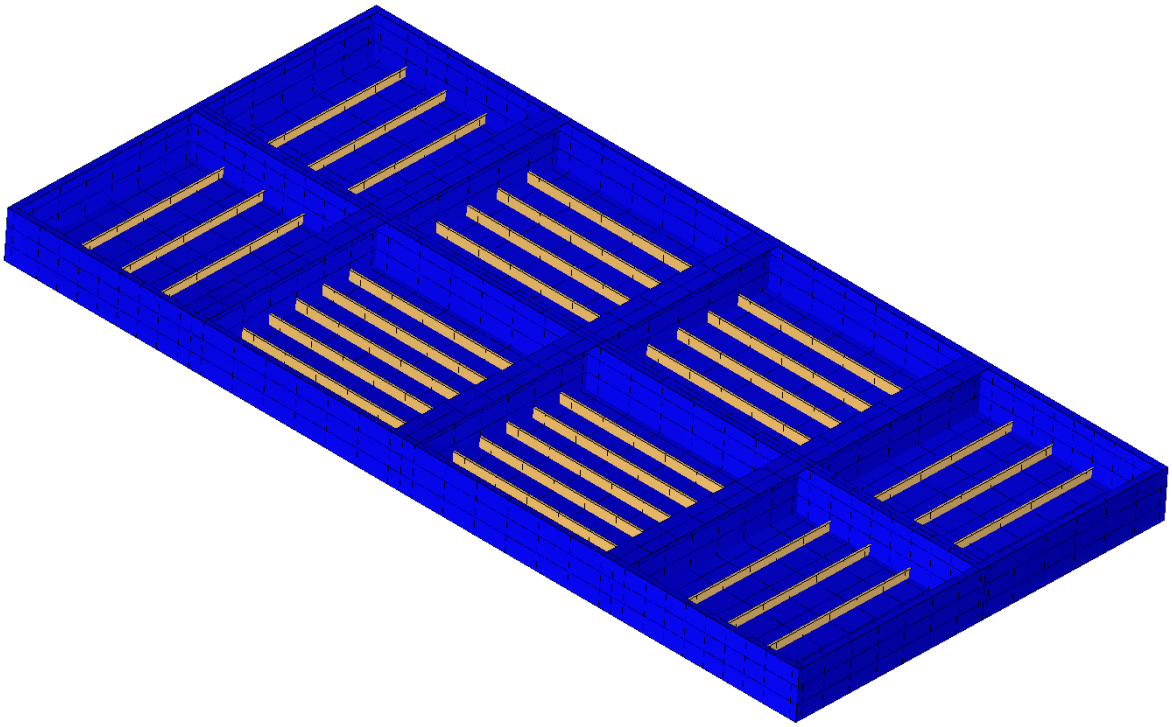


Slika 2-5 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_1`

2.) Ispitna varijanta `hc_var_2`

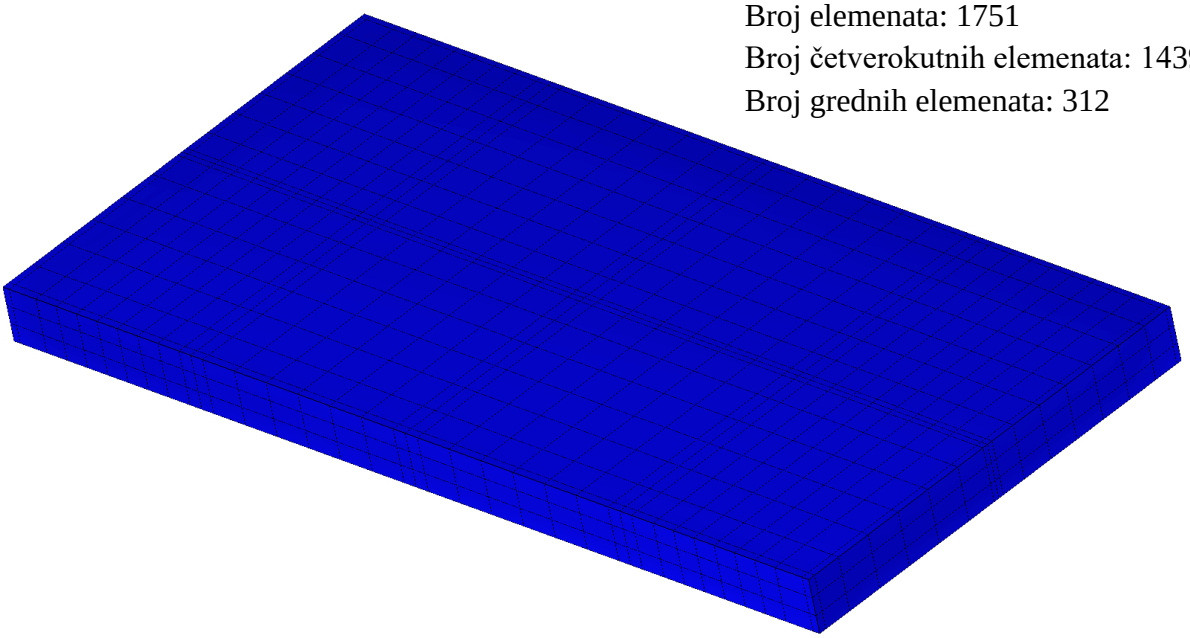


Slika 2-6 Model ispitne varijante konstrukcije `hc_var_2`

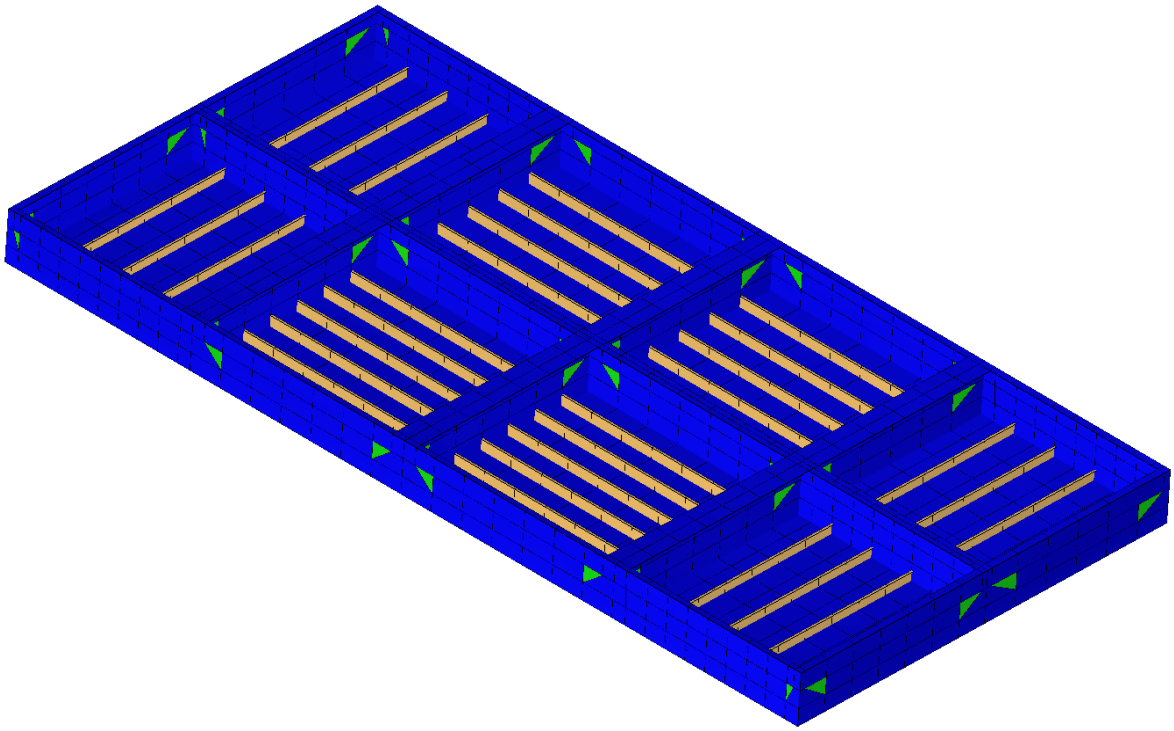


Slika 2-7 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_2`

Varijanta mreže *V1*
Broj čvorova: 1440
Broj elemenata: 1751
Broj četverokutnih elemenata: 1439
Broj grednih elemenata: 312

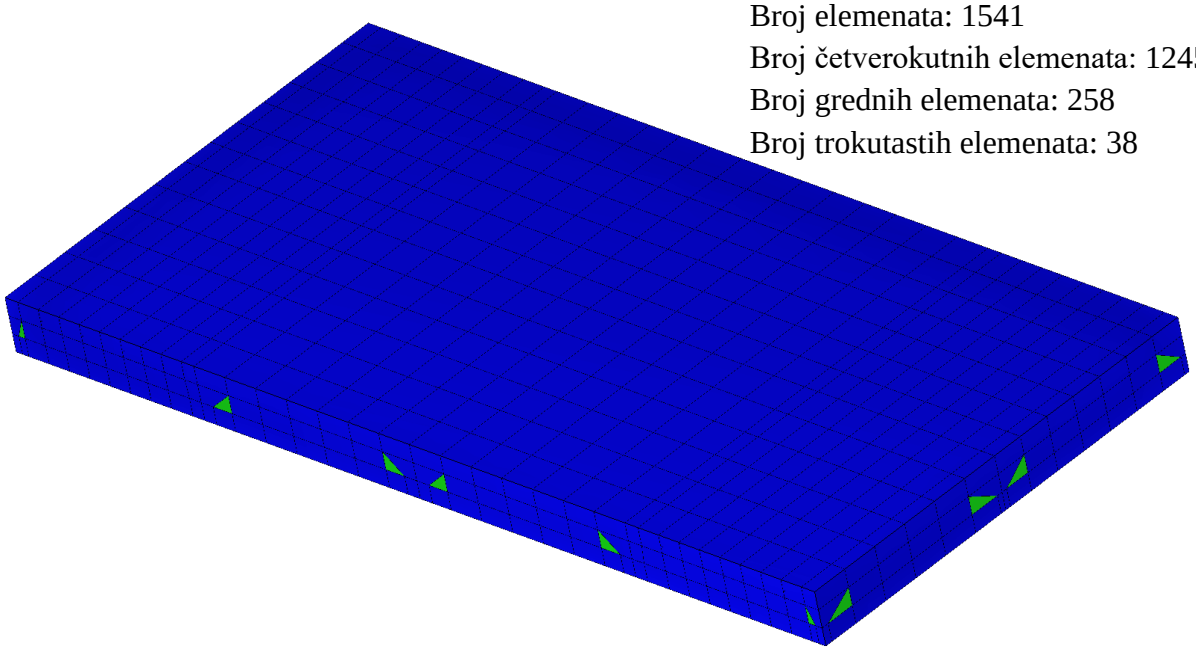


Slika 2-8 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_2`



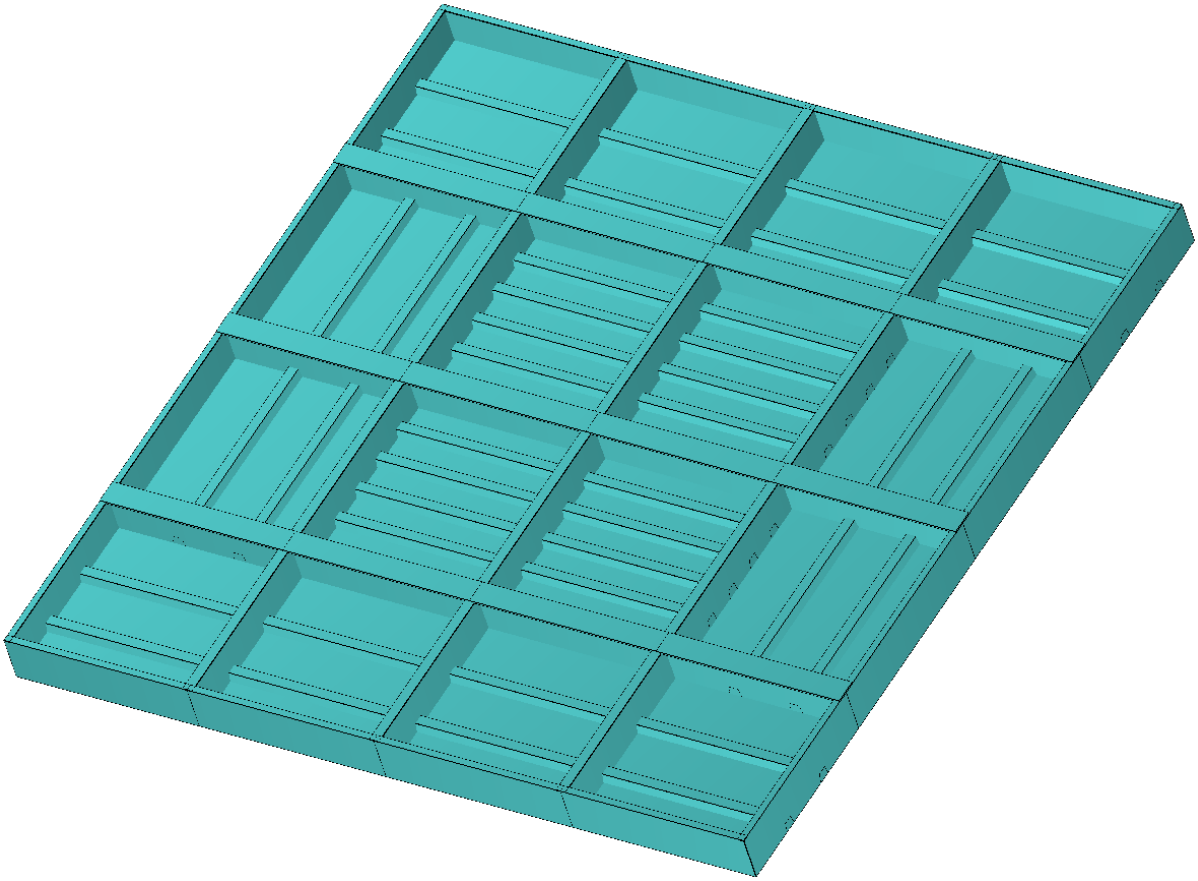
Slika 2-9 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_2`

Varijanta mreže *V2*
Broj čvorova: 1278
Broj elemenata: 1541
Broj četverokutnih elemenata: 1245
Broj grednih elemenata: 258
Broj trokutastih elemenata: 38

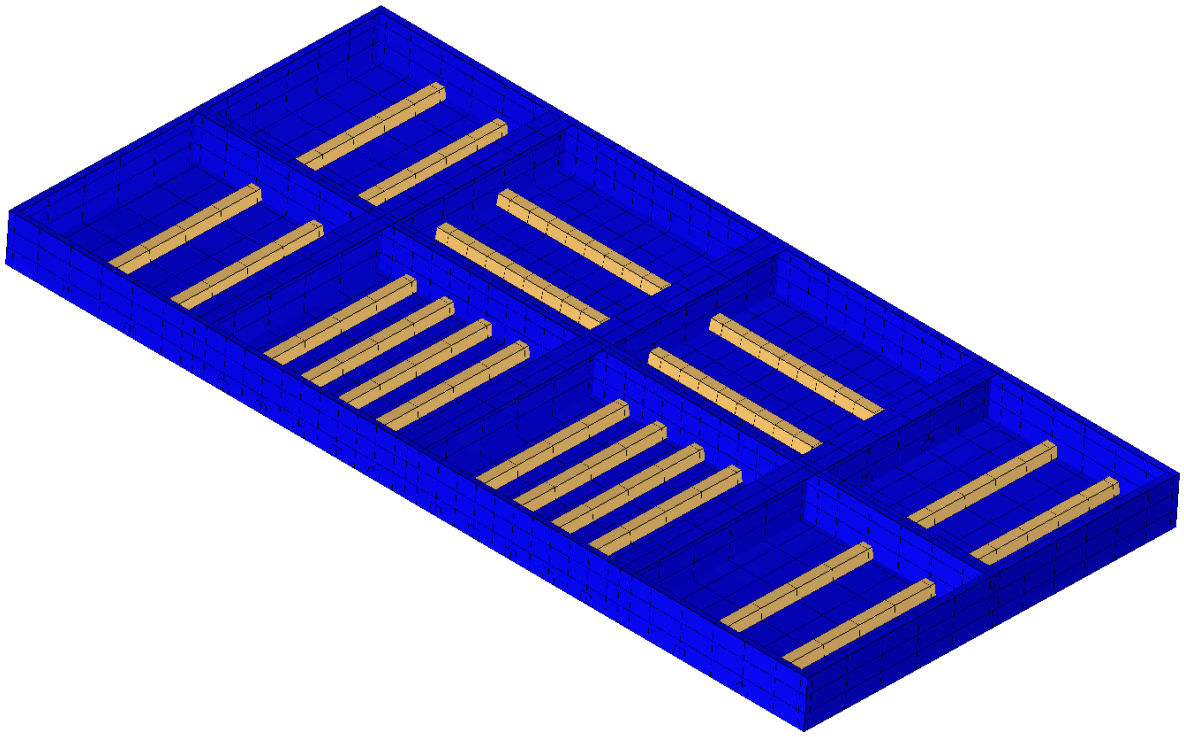


Slika 2-10 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_2`

3.) Ispitna varijanta `hc_var_3`

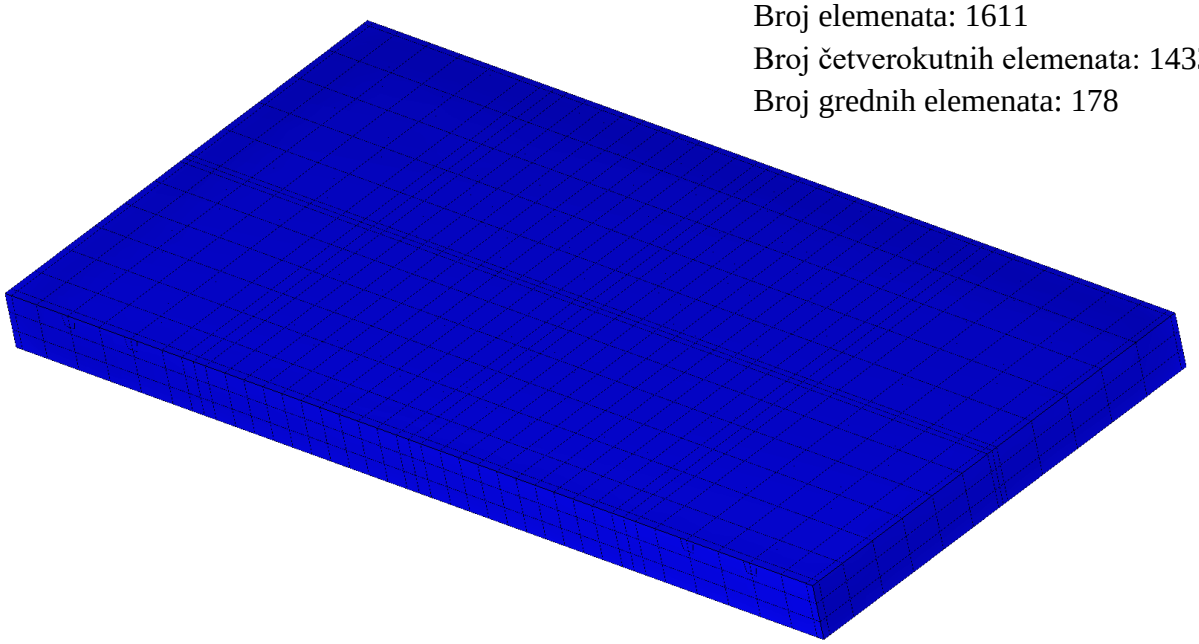


Slika 2-11 Model ispitne varijante konstrukcije `hc_var_3`

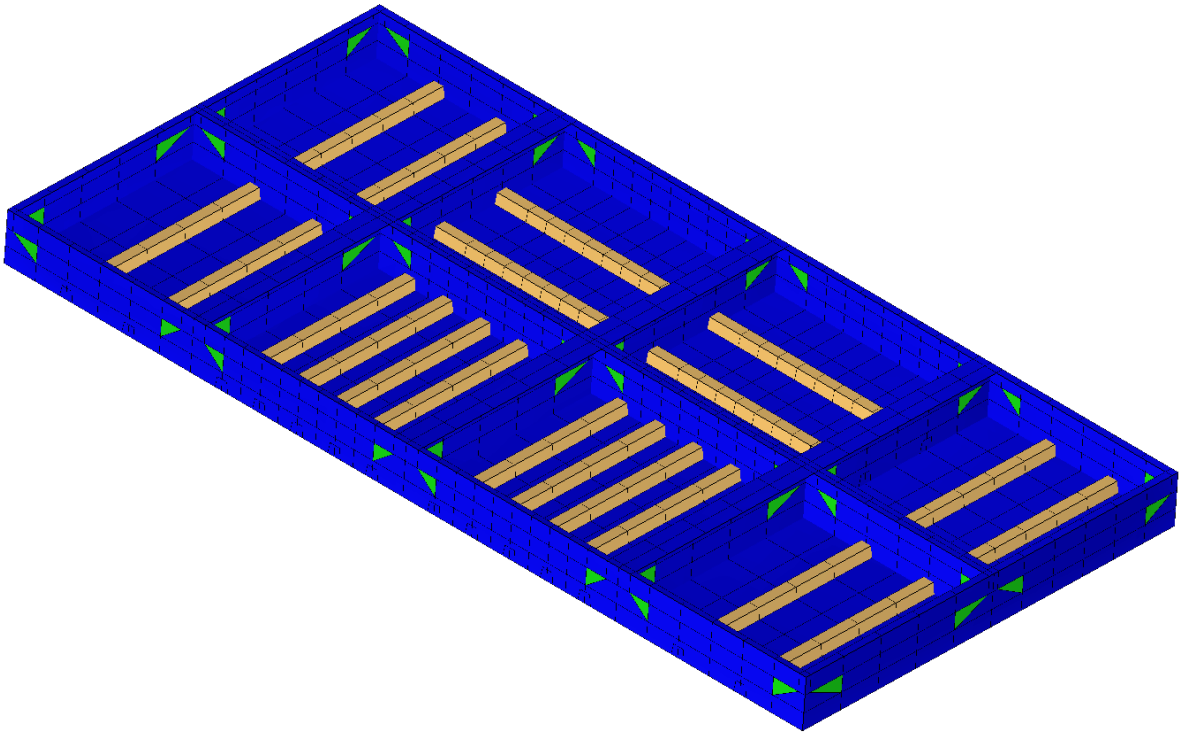


Slika 2-12 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_3`

Varijanta mreže *V1*
Broj čvorova: 1436
Broj elemenata: 1611
Broj četverokutnih elemenata: 1433
Broj grednih elemenata: 178

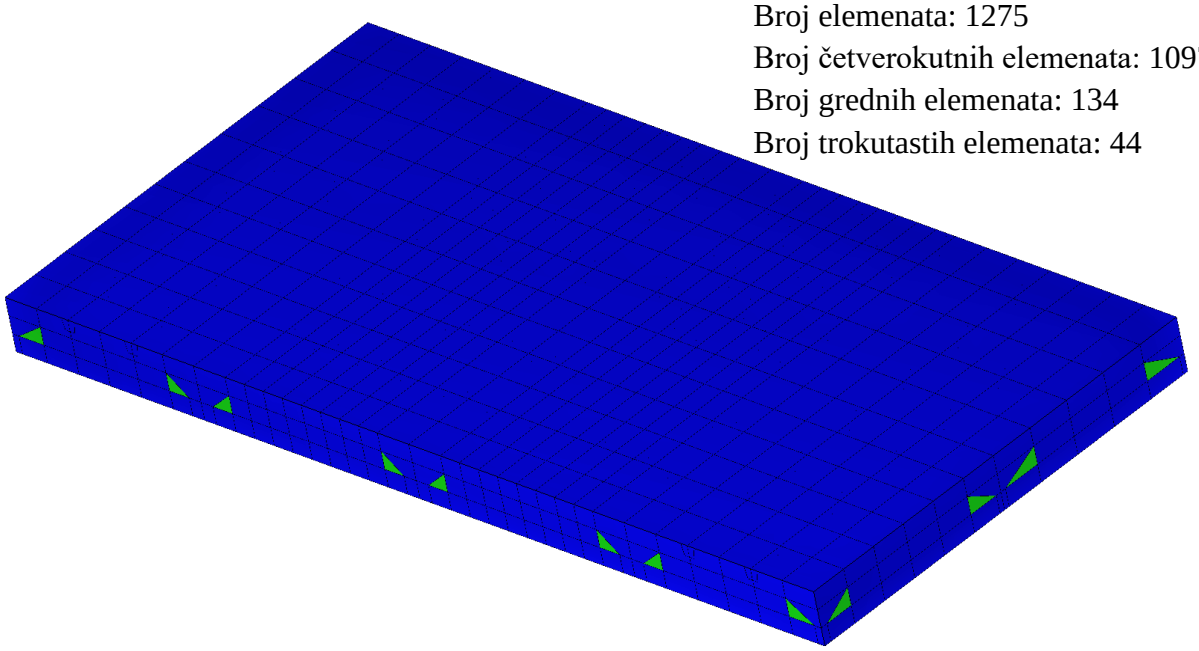


Slika 2-13 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_3`



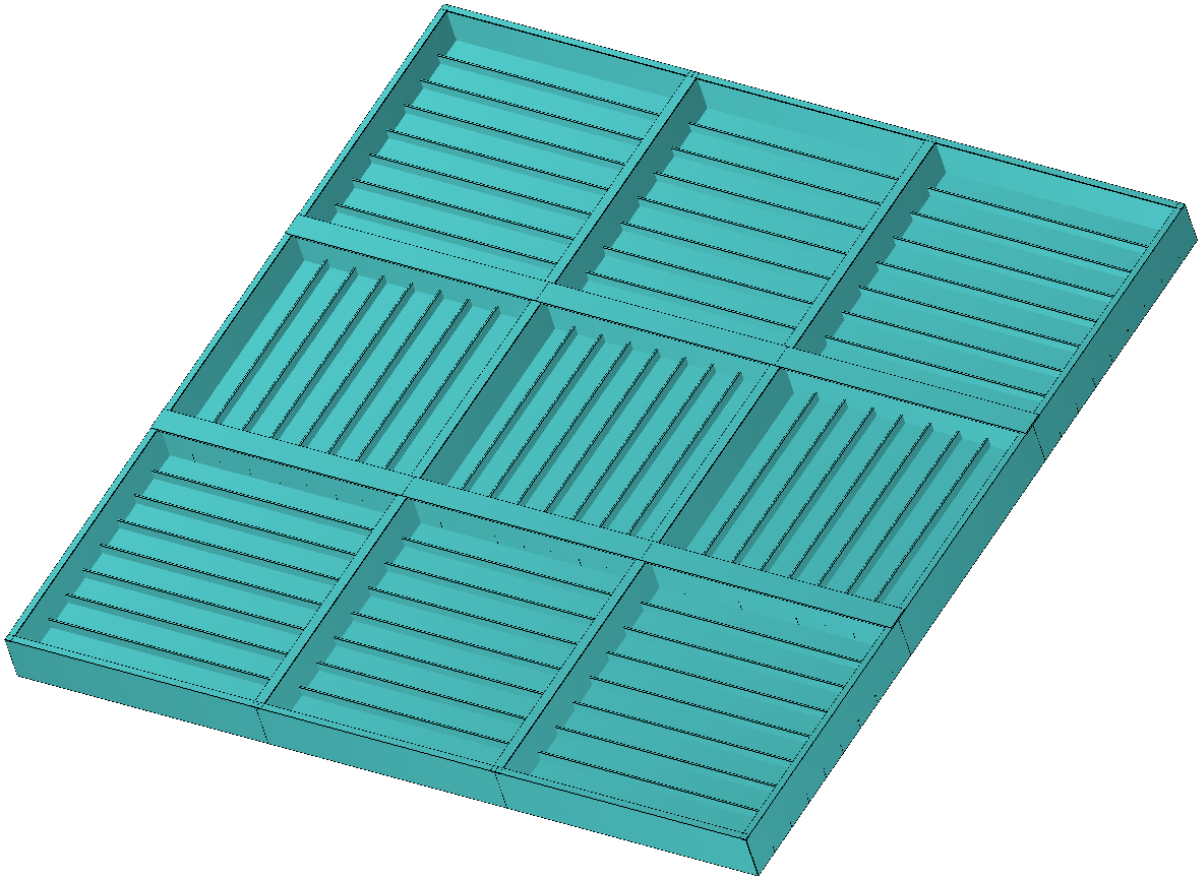
Slika 2-14 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_3`

Varijanta mreže *V2*
Broj čvorova: 1132
Broj elemenata: 1275
Broj četverokutnih elemenata: 1097
Broj grednih elemenata: 134
Broj trokutastih elemenata: 44

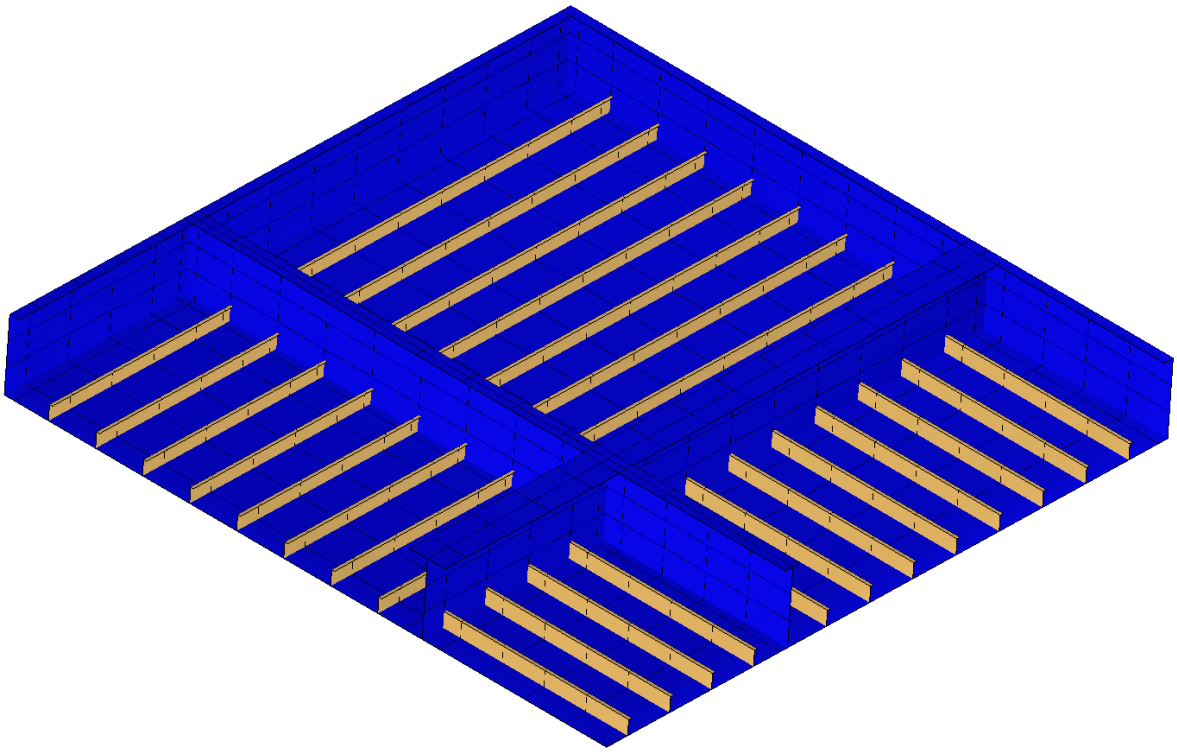


Slika 2-15 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_3`

4.) Ispitna varijanta `hc_var_4`

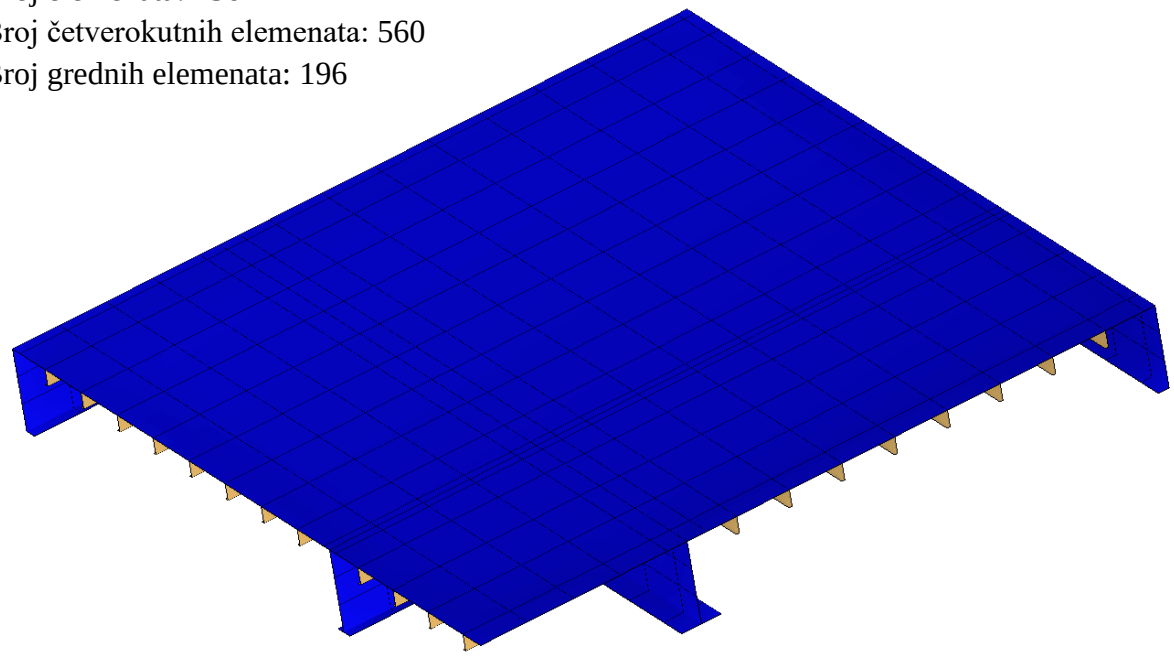


Slika 2-16 Model ispitne varijante konstrukcije `hc_var_4`

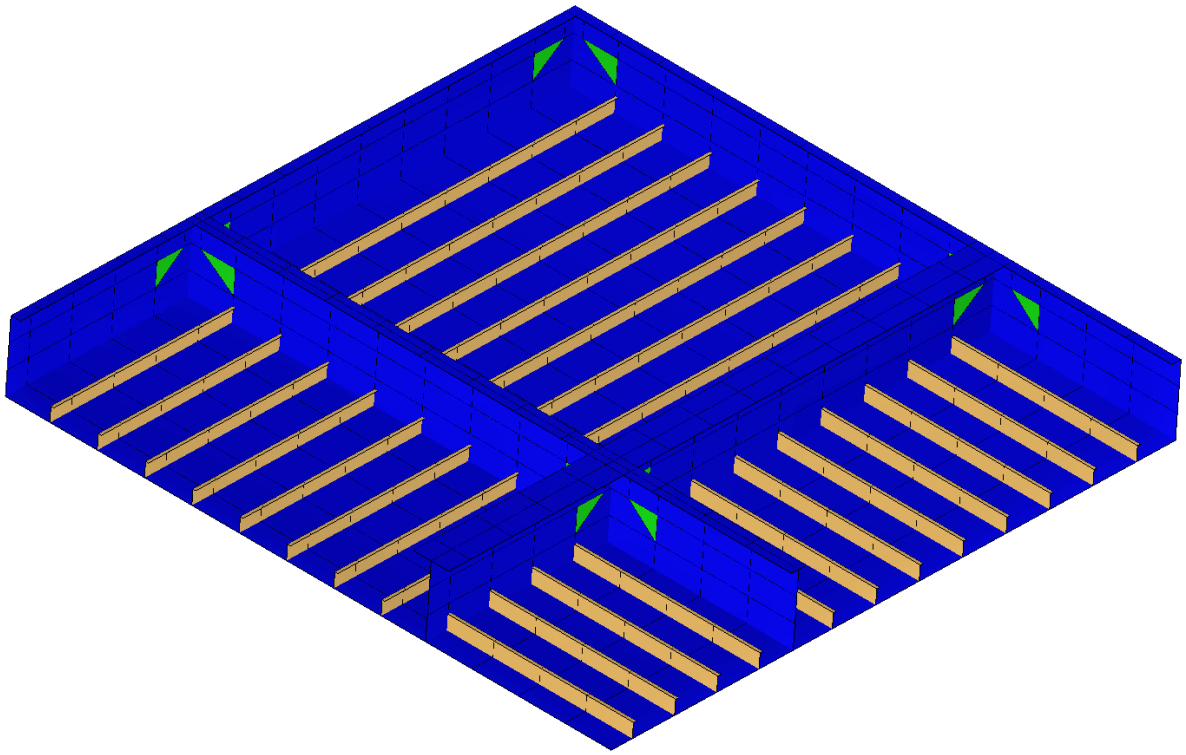


Slika 2-17 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_4`

Varijanta mreže *V1*
Broj čvorova: 588
Broj elemenata: 756
Broj četverokutnih elemenata: 560
Broj grednih elemenata: 196

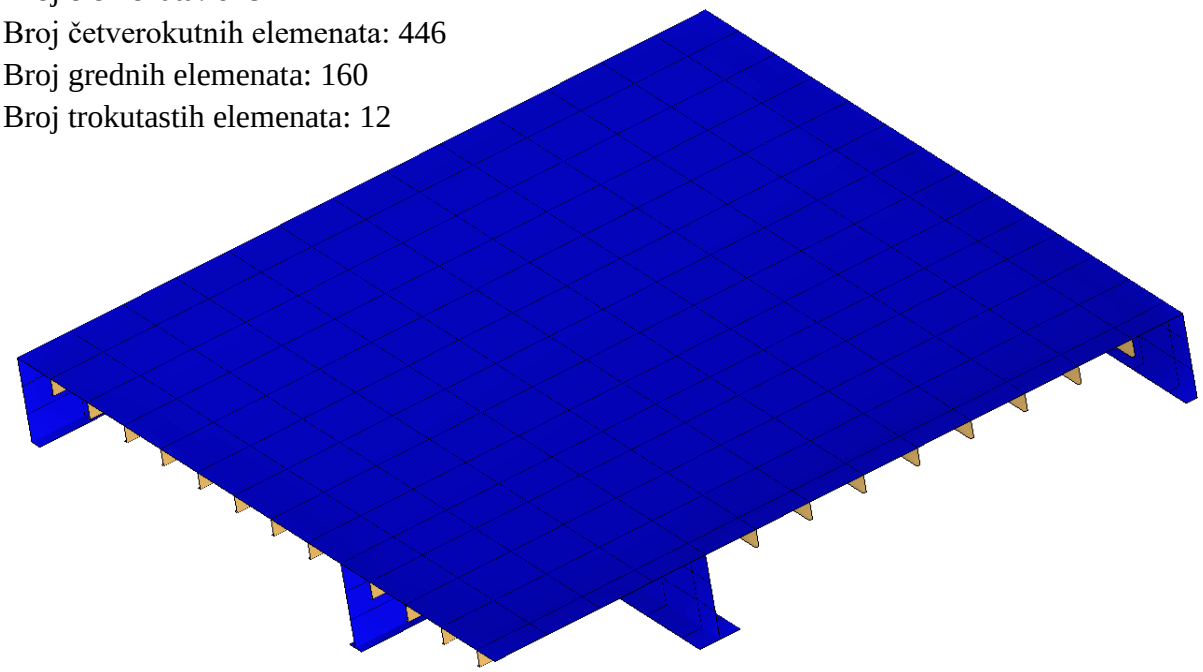


Slika 2-18 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_4`



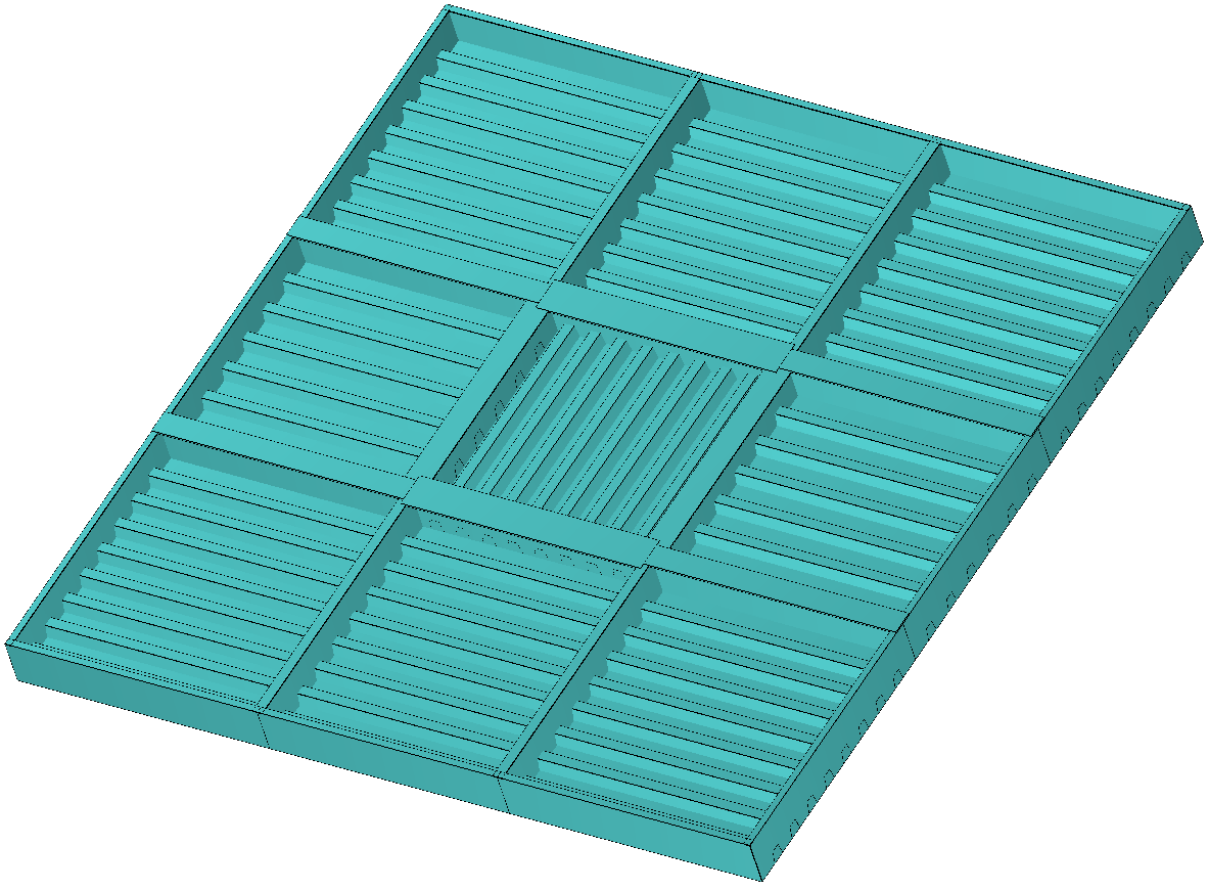
Slika 2-19 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_4`

Varijanta mreže *V2*
Broj čvorova: 480
Broj elemenata: 618
Broj četverokutnih elemenata: 446
Broj grednih elemenata: 160
Broj trokutastih elemenata: 12

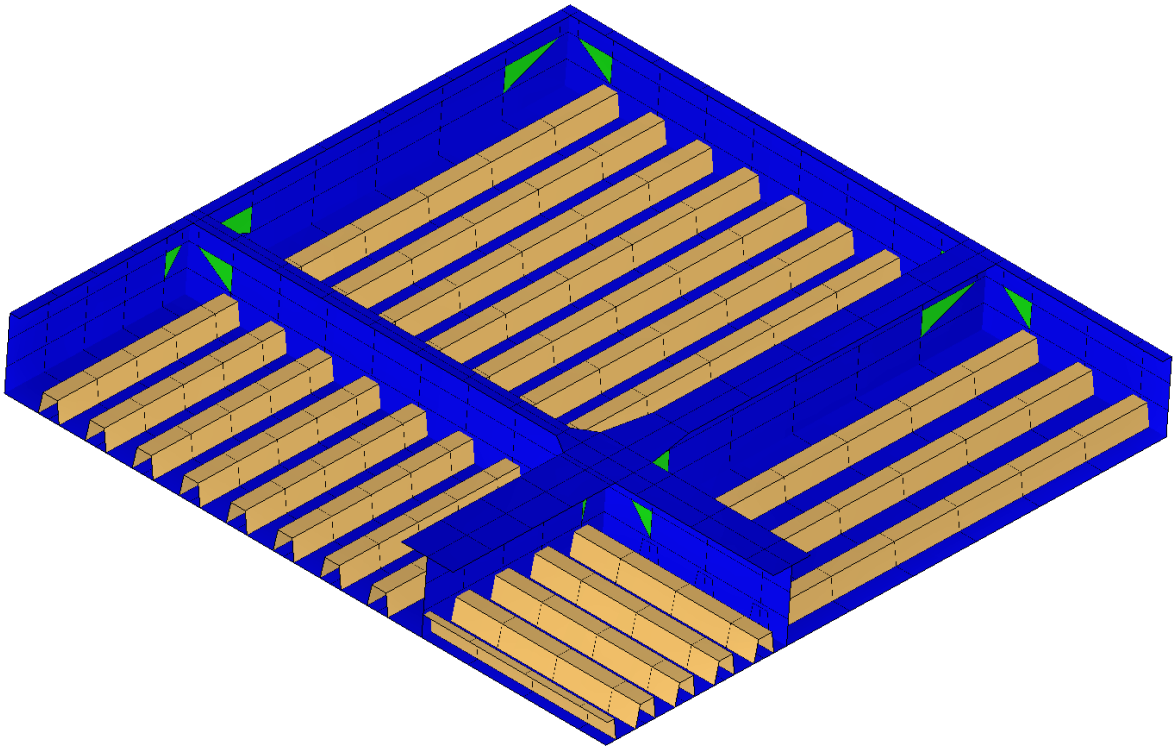


Slika 2-20 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_4`

5.) Ispitna varijanta `hc_var_5`

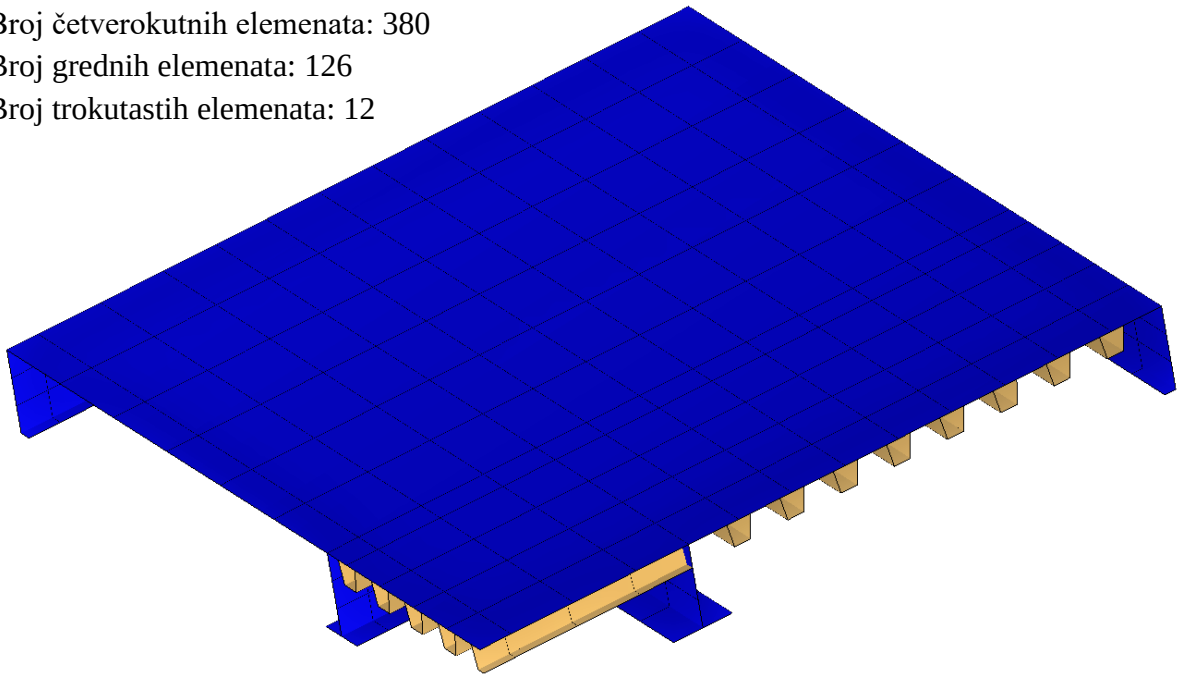


Slika 2-21 Model ispitne varijante konstrukcije `hc_var_5`



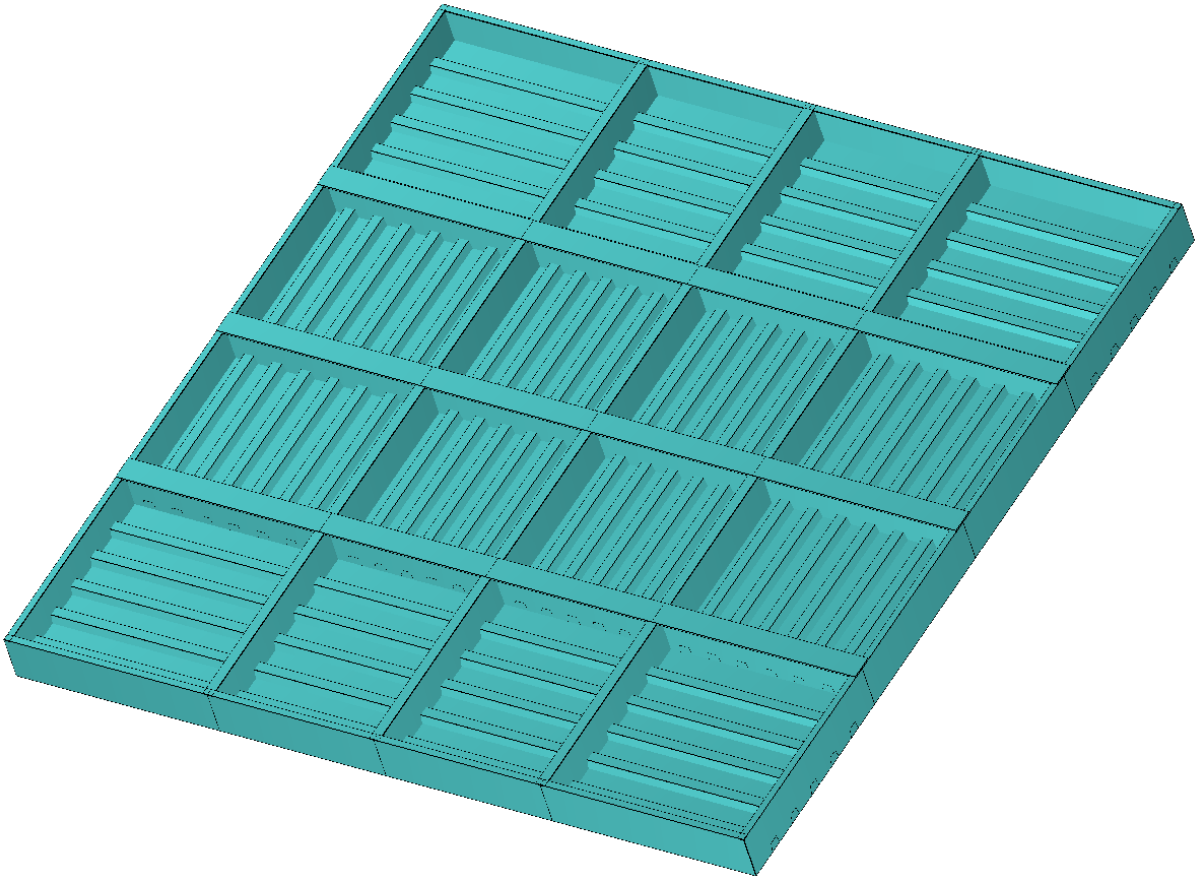
Slika 2-22 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_5`

Varijanta mreže *V2*
Broj čvorova: 411
Broj elemenata: 518
Broj četverokutnih elemenata: 380
Broj grednih elemenata: 126
Broj trokutastih elemenata: 12

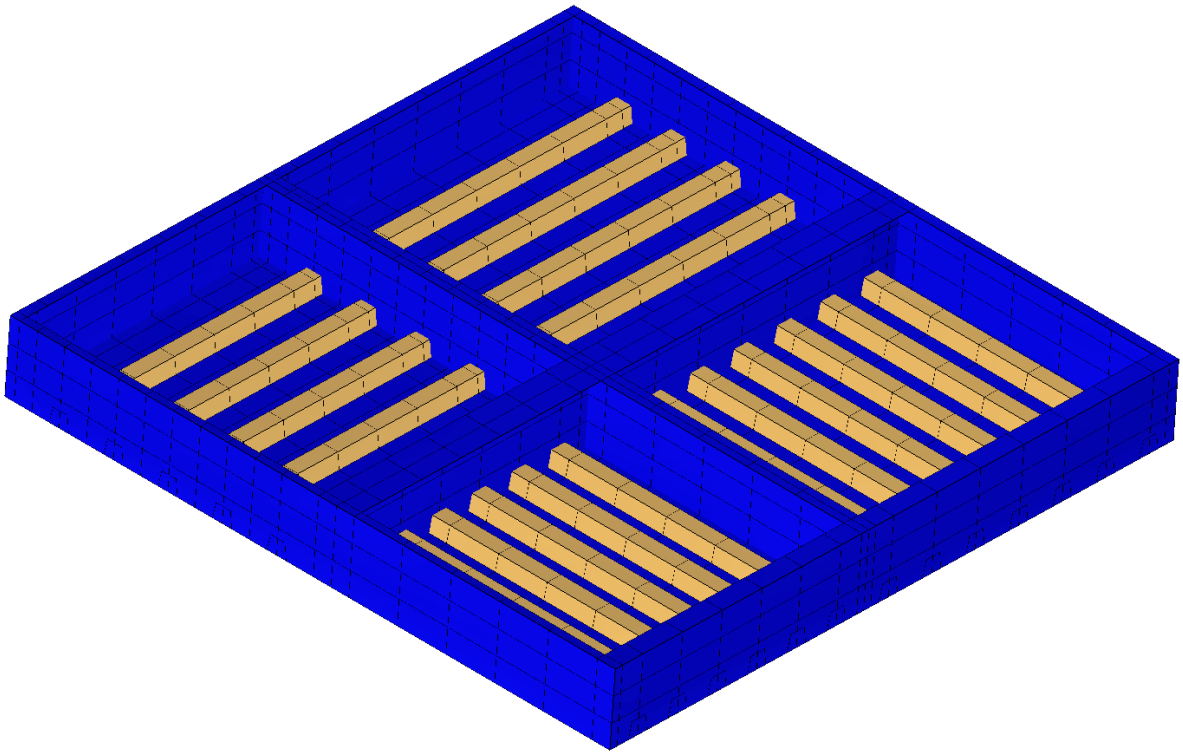


Slika 2-23 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_5`

6.) Ispitna varijanta `hc_var_6`

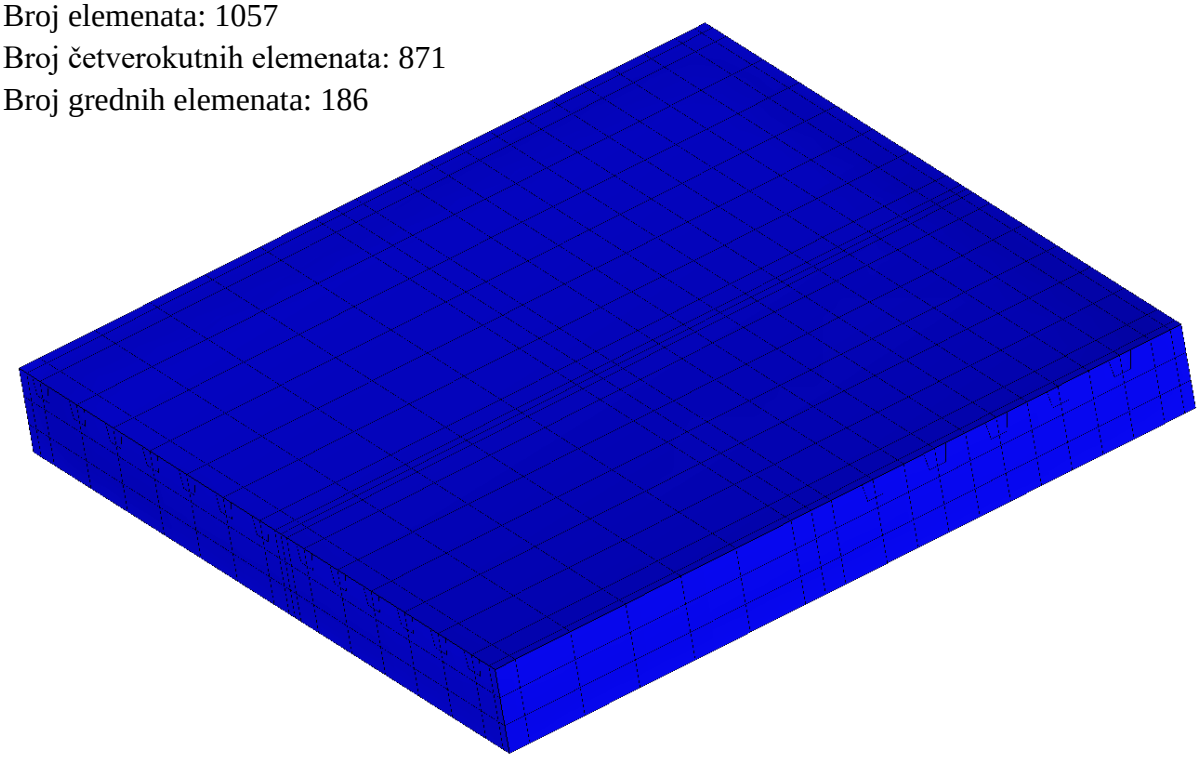


Slika 2-24 Model ispitne varijante konstrukcije `hc_var_6`

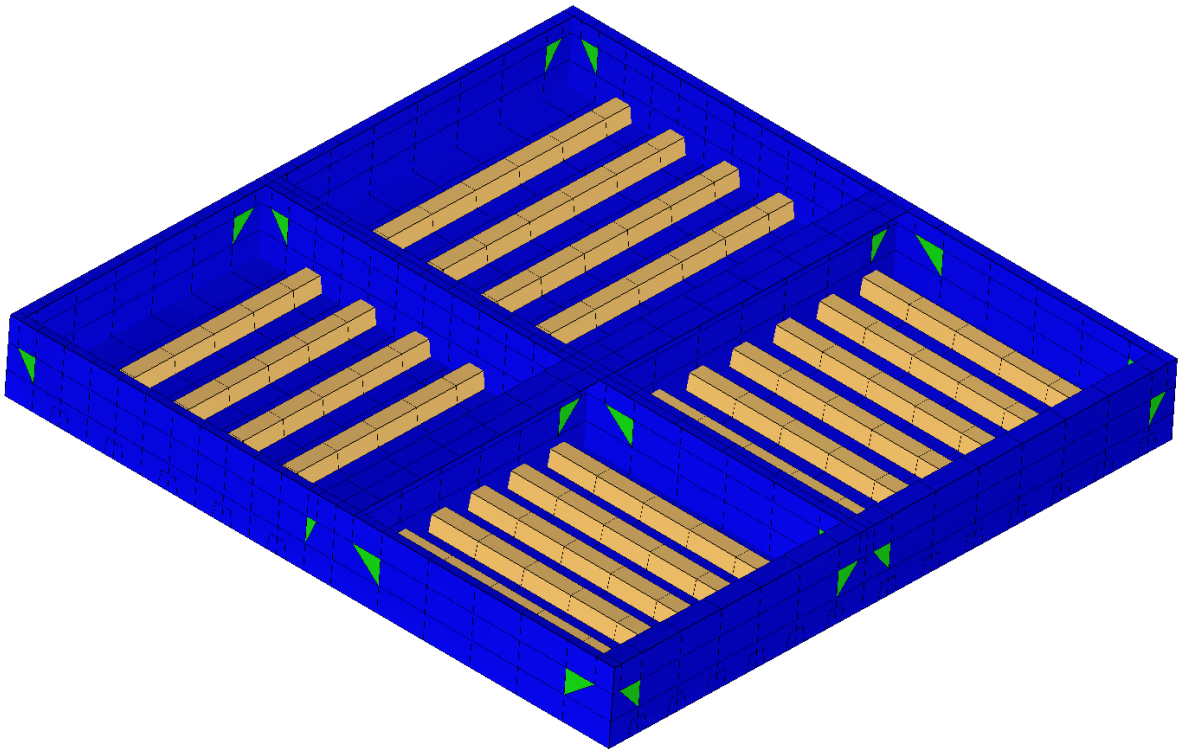


Slika 2-25 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_6`

Varijanta mreže *V1*
Broj čvorova: 886
Broj elemenata: 1057
Broj četverokutnih elemenata: 871
Broj grednih elemenata: 186

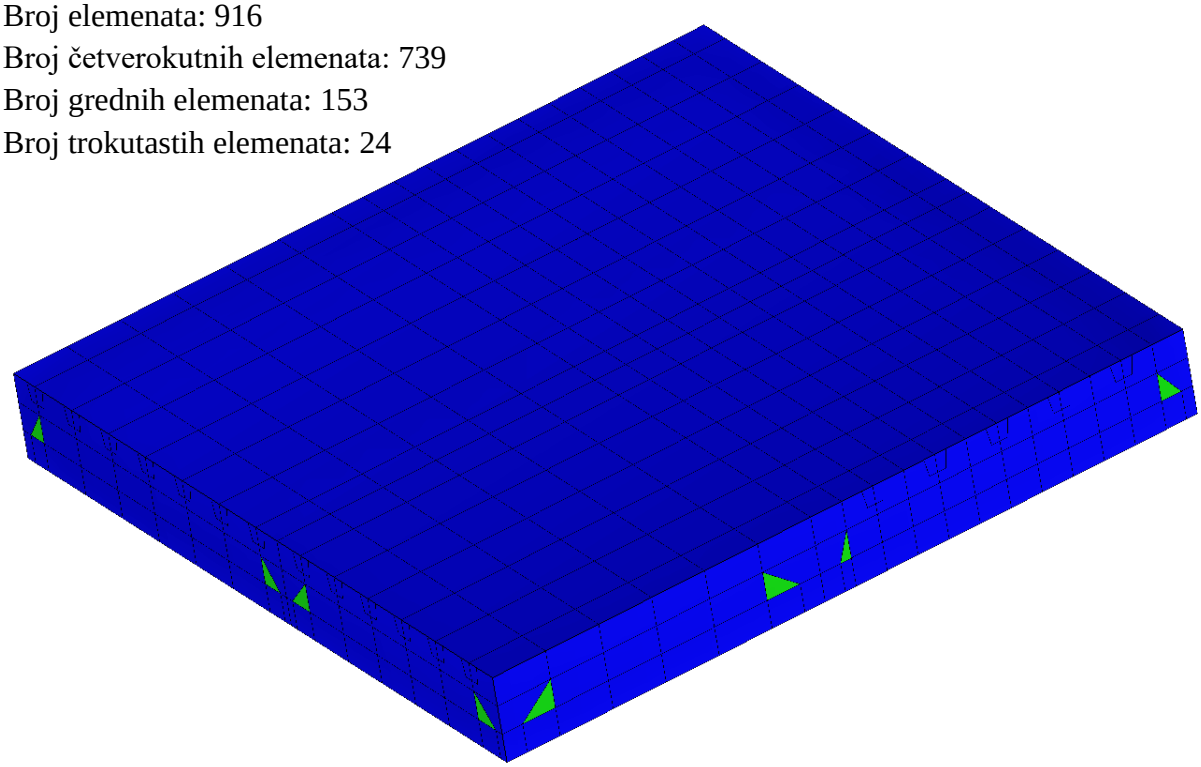


Slika 2-26 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_6`



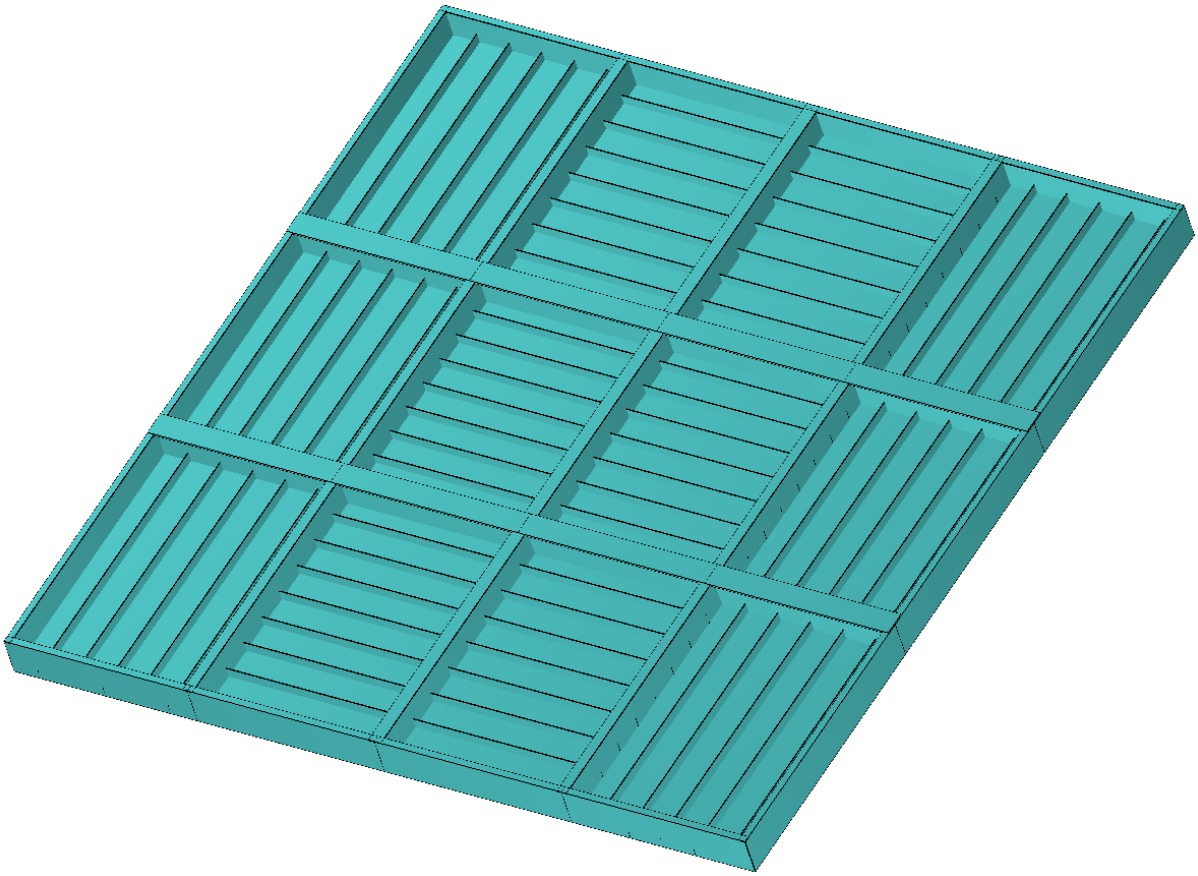
Slika 2-27 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_6`

Varijanta mreže *V2*
Broj čvorova: 772
Broj elemenata: 916
Broj četverokutnih elemenata: 739
Broj grednih elemenata: 153
Broj trokutastih elemenata: 24

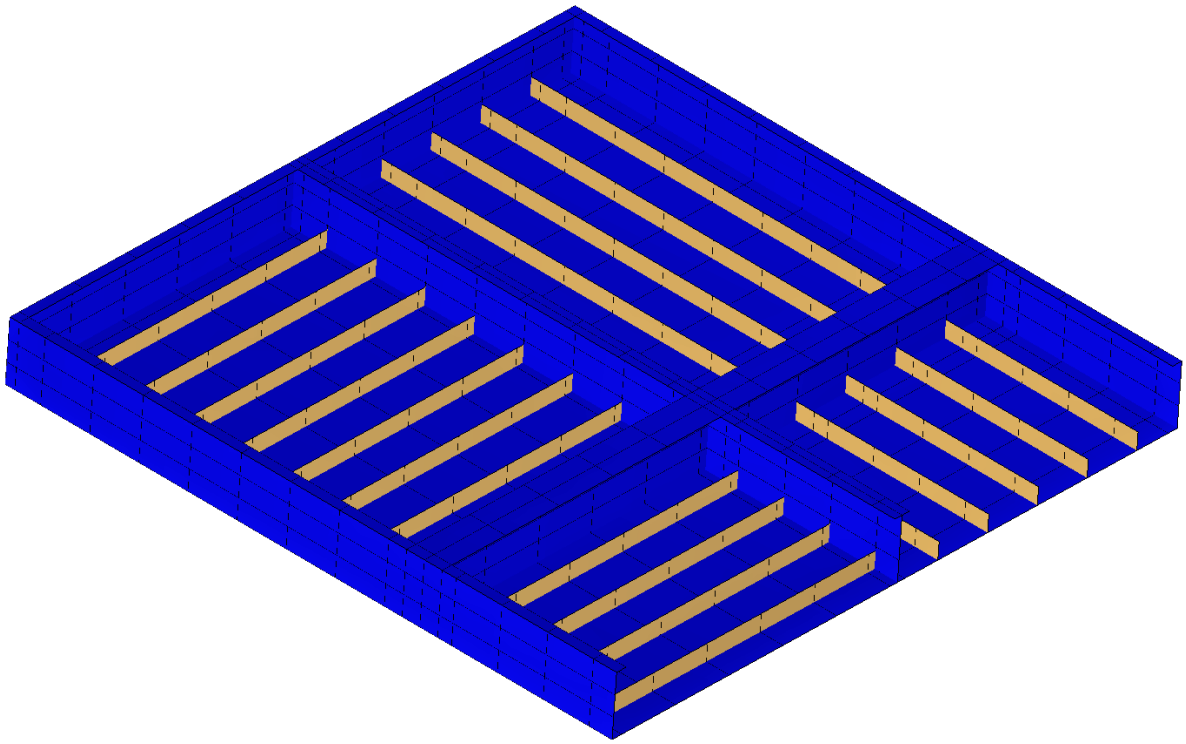


Slika 2-28 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_6`

7.) Ispitna varijanta `hc_var_7`

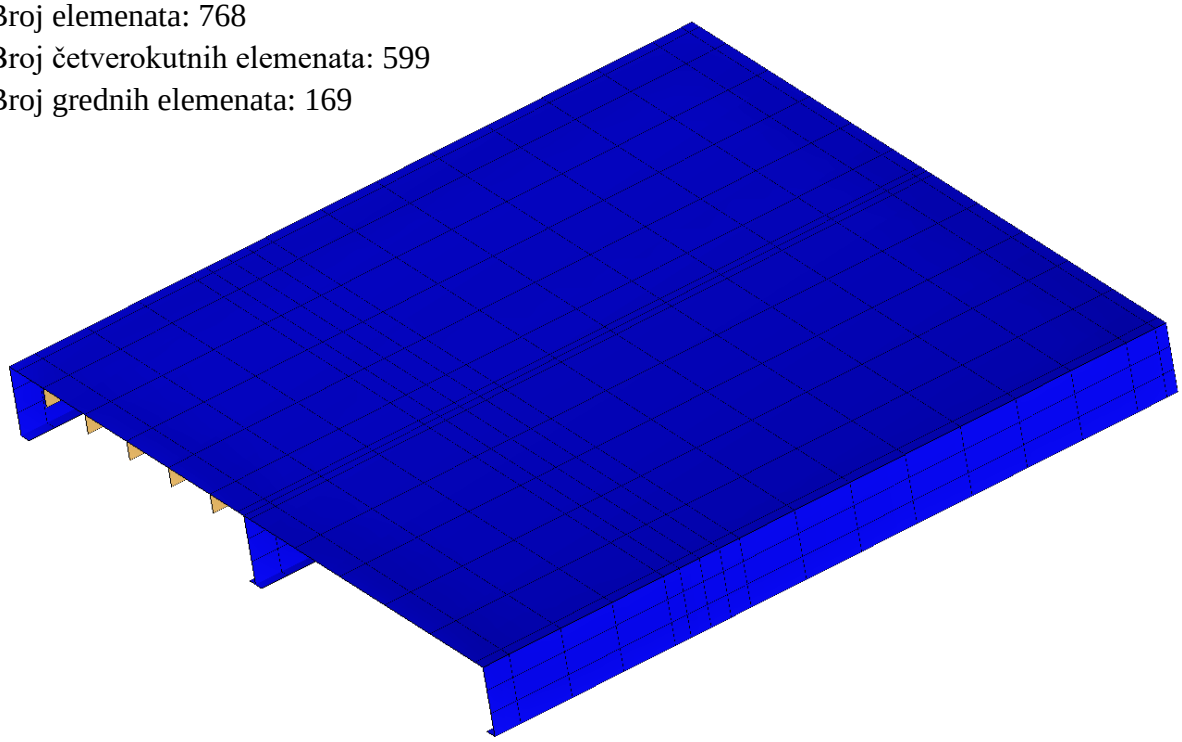


Slika 2-29 Model ispitne varijante konstrukcije `hc_var_7`

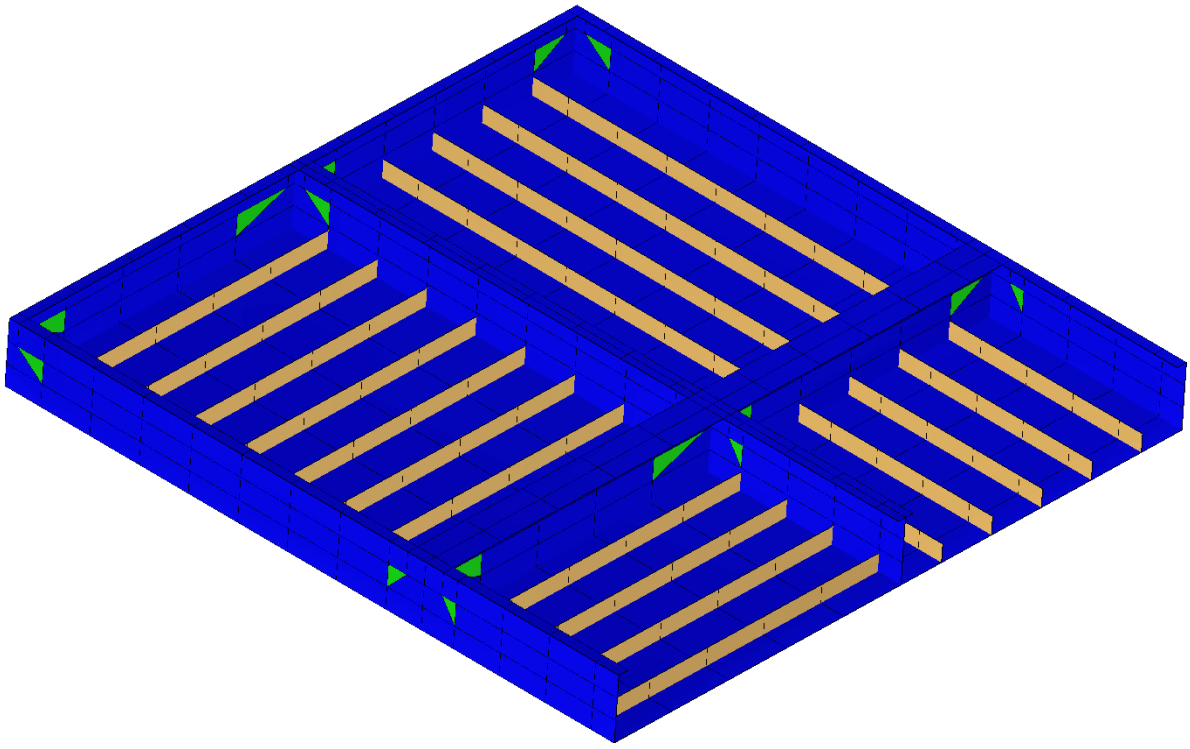


Slika 2-30 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_7`

Varijanta mreže *V1*
Broj čvorova: 619
Broj elemenata: 768
Broj četverokutnih elemenata: 599
Broj grednih elemenata: 169

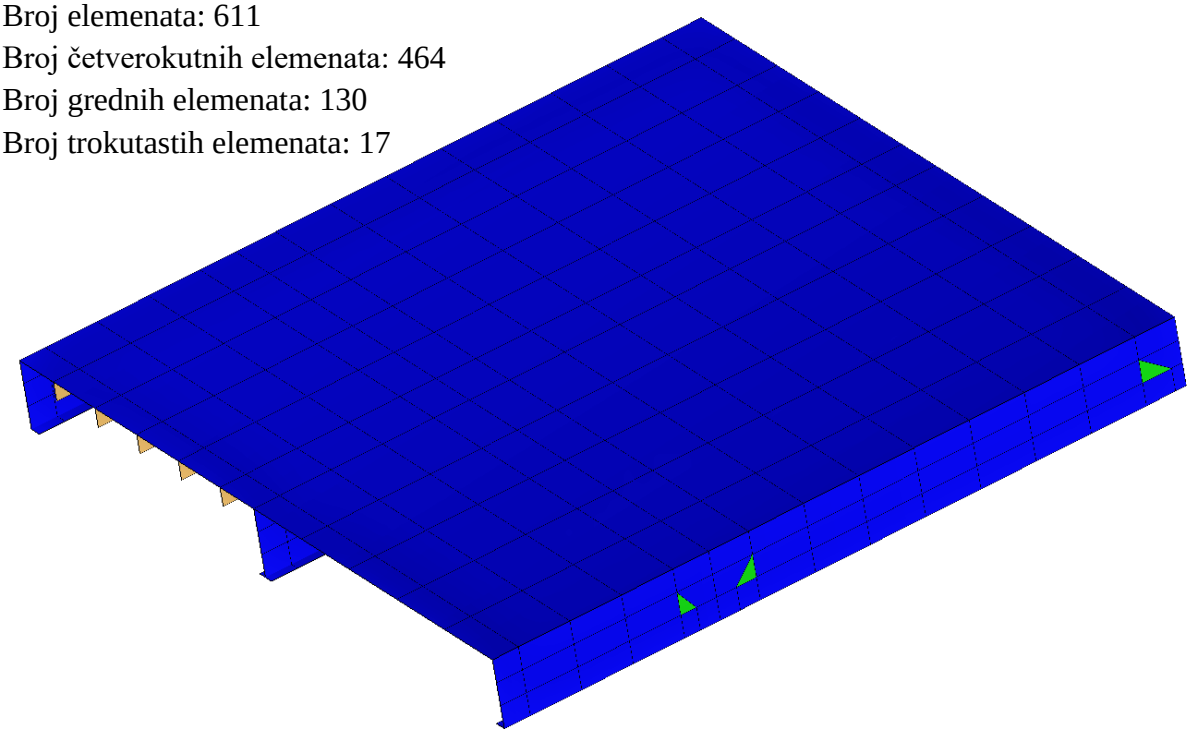


Slika 2-31 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_7`



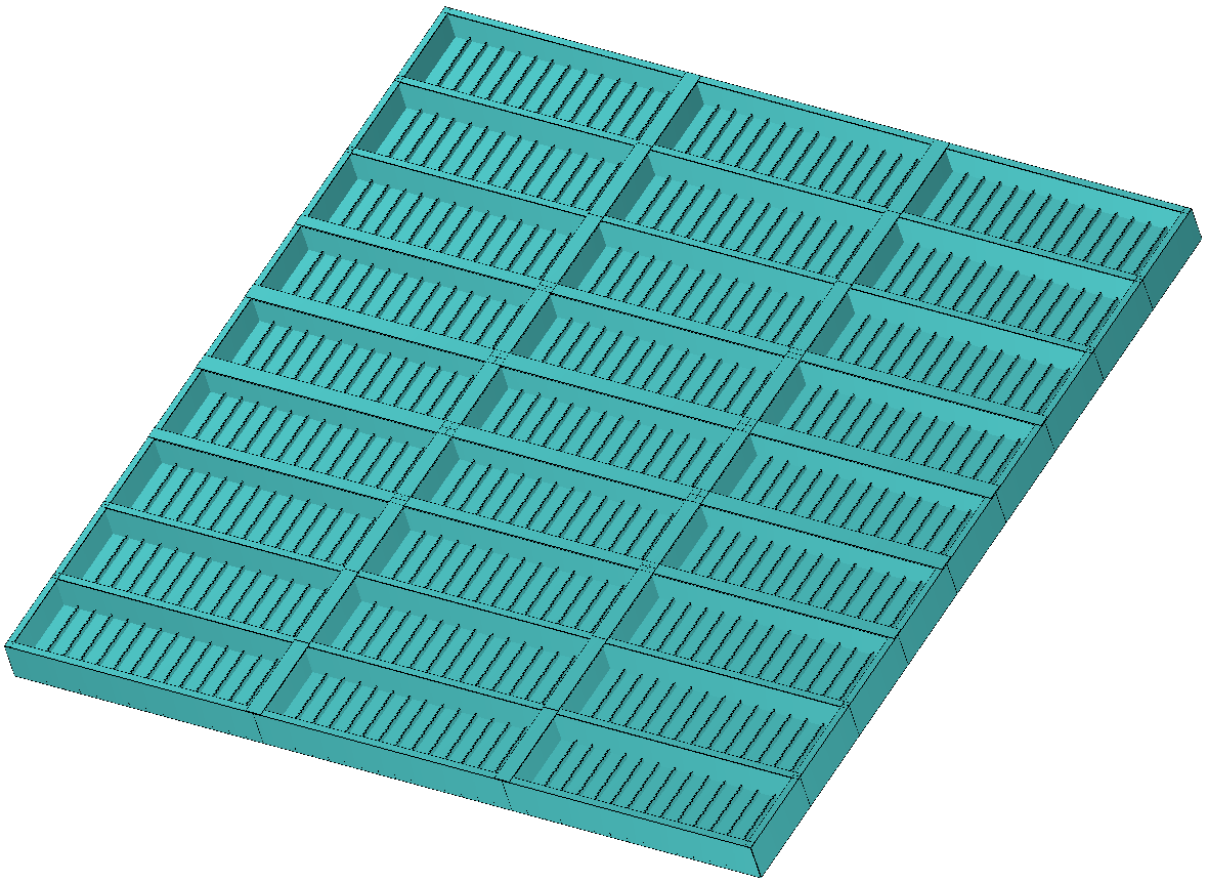
Slika 2-32 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_7`

Varijanta mreže *V2*
Broj čvorova: 494
Broj elemenata: 611
Broj četverokutnih elemenata: 464
Broj grednih elemenata: 130
Broj trokutastih elemenata: 17

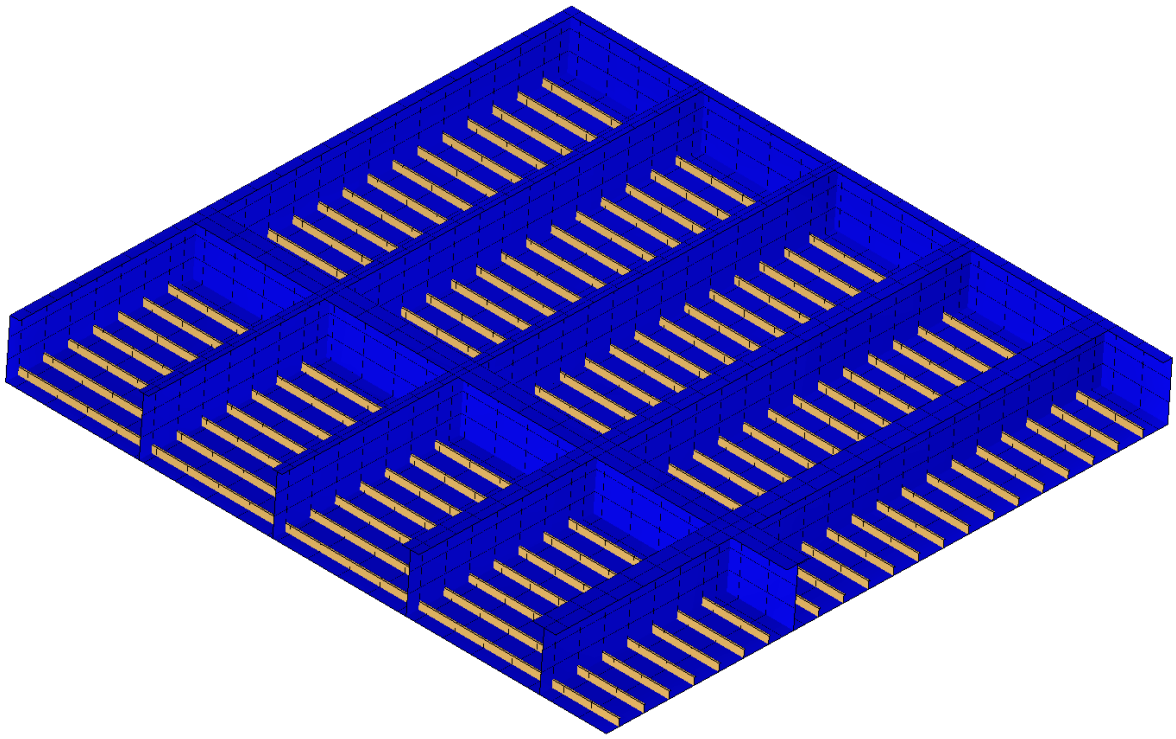


Slika 2-33 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_7`

8.) Ispitna varijanta `hc_var_8`

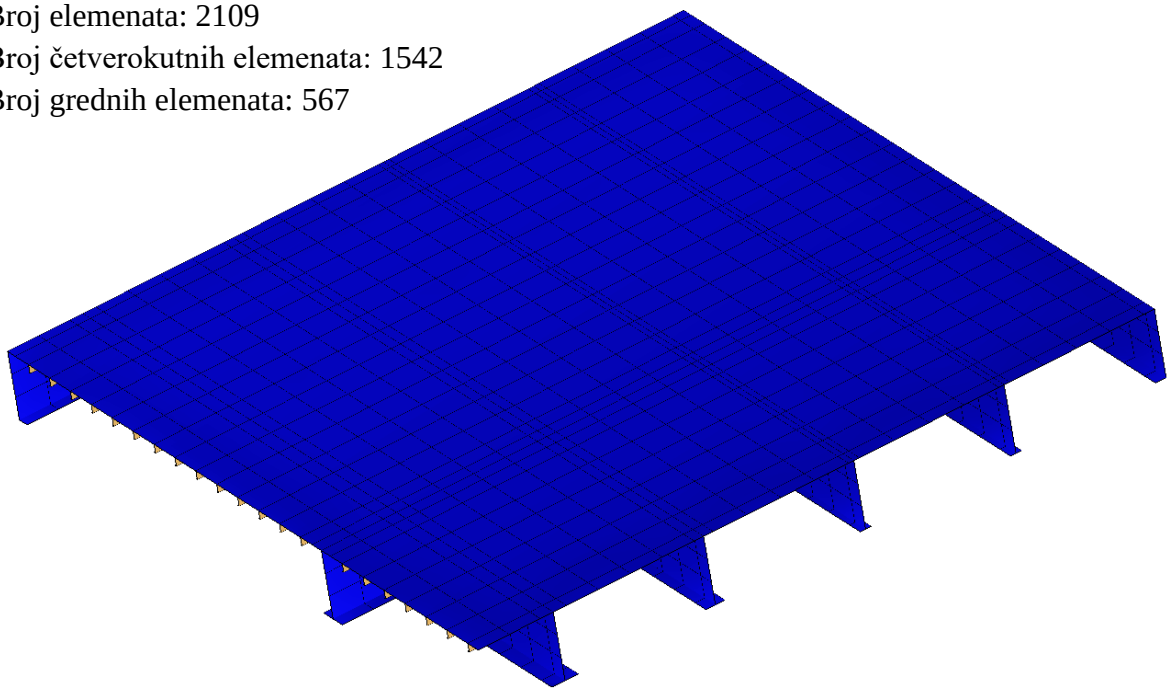


Slika 2-34 Model ispitne varijante konstrukcije `hc_var_8`

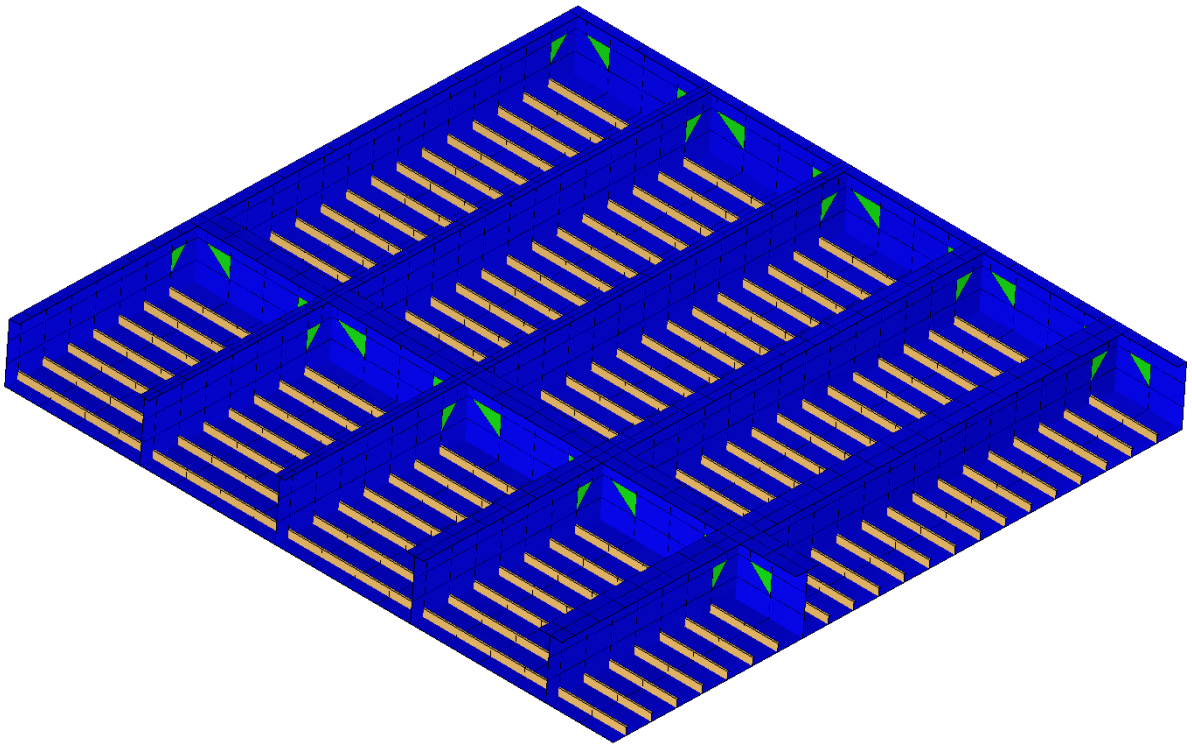


Slika 2-35 Varijanta mreže *V1*, ispitna varijanta konstrukcije `hc_var_8`

Varijanta mreže *V1*
Broj čvorova: 1566
Broj elemenata: 2109
Broj četverokutnih elemenata: 1542
Broj grednih elemenata: 567

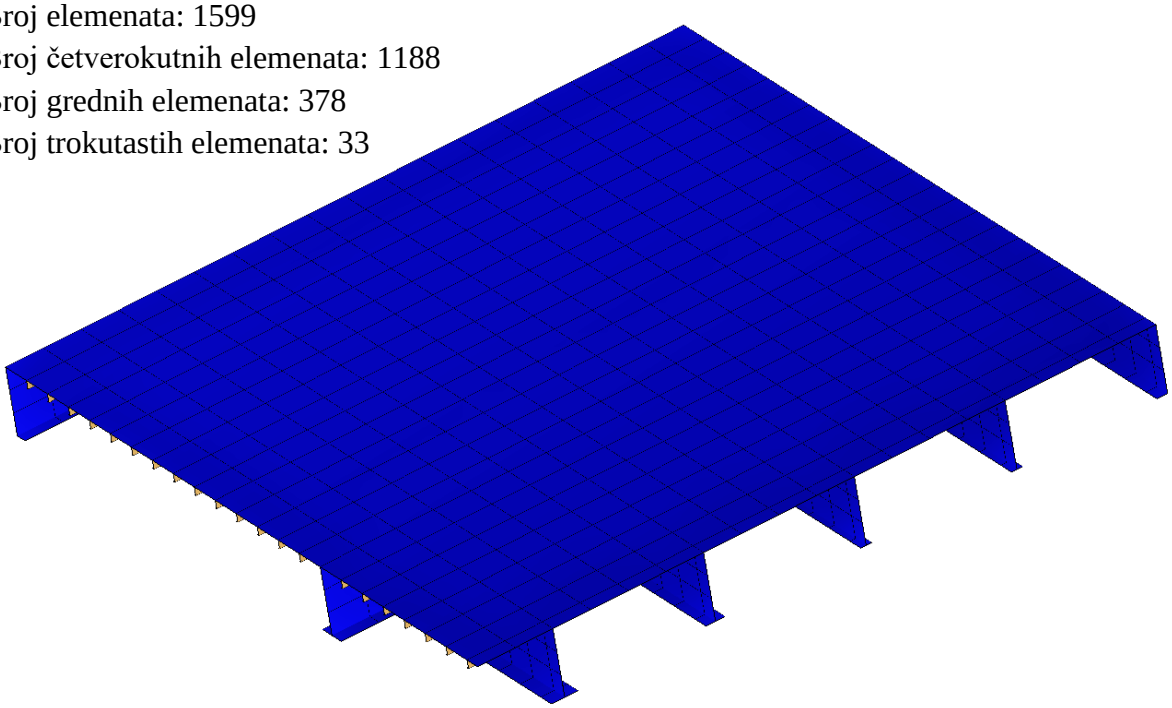


Slika 2-36 Varijanta mreže *V1*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_8`



Slika 2-37 Varijanta mreže *V2*, ispitna varijanta konstrukcije `hc_var_8`

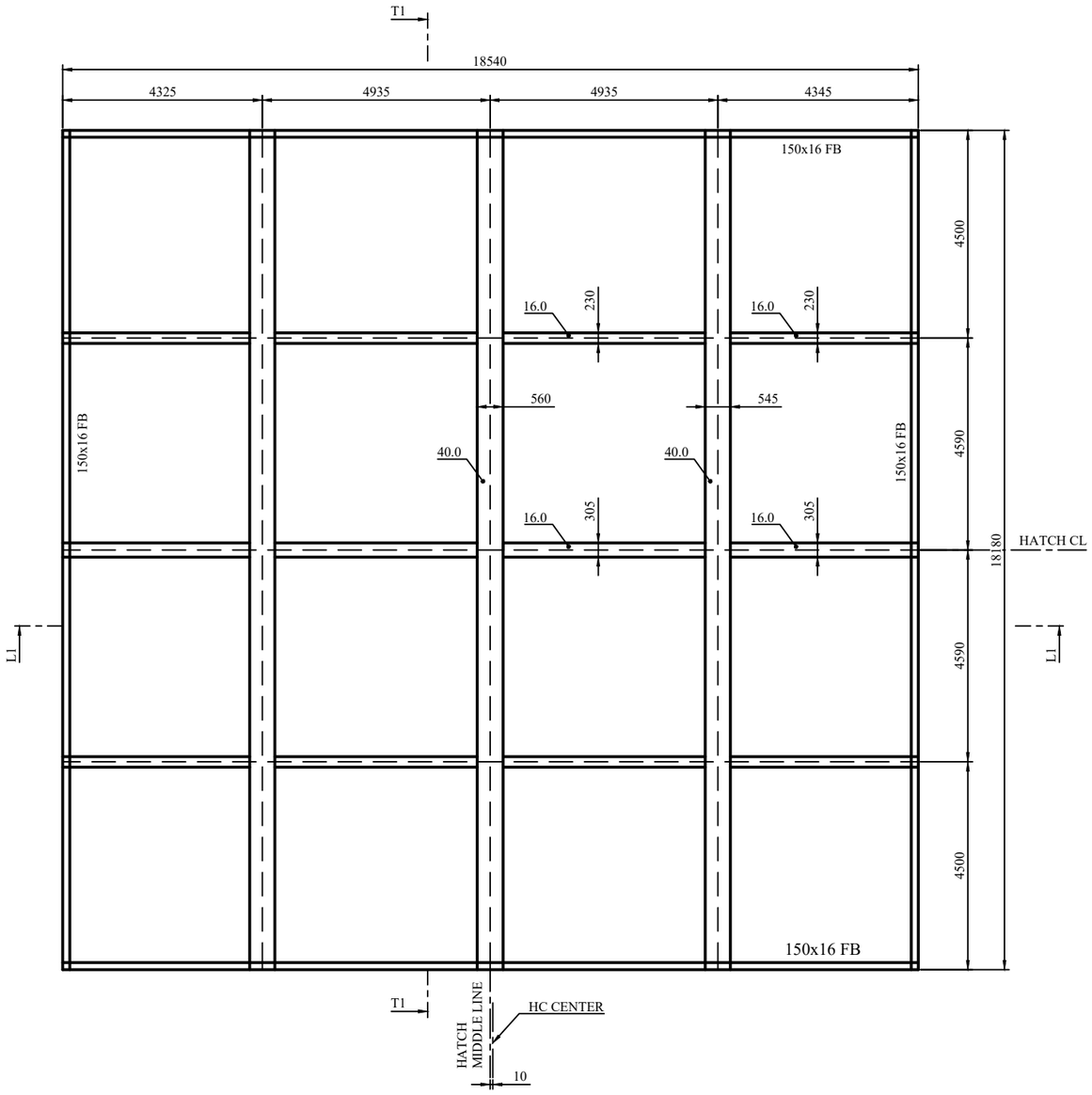
Varijanta mreže *V2*
Broj čvorova: 1233
Broj elemenata: 1599
Broj četverokutnih elemenata: 1188
Broj grednih elemenata: 378
Broj trokutastih elemenata: 33

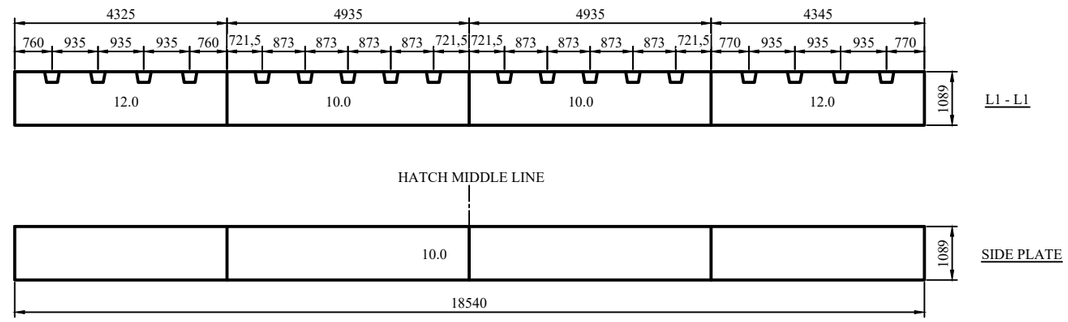


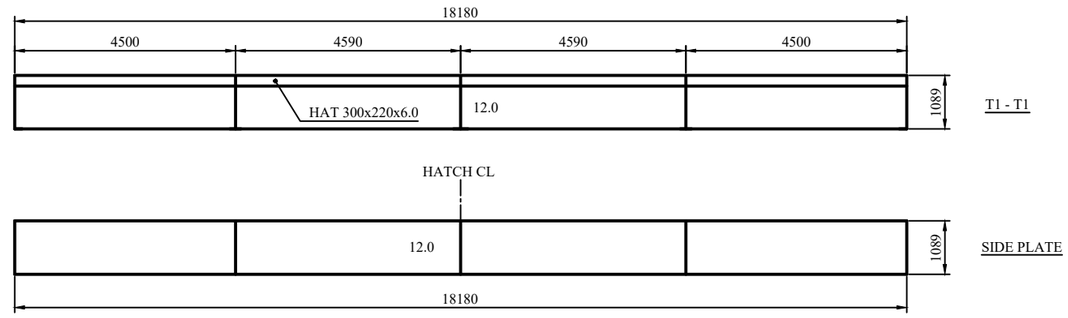
Slika 2-38 Varijanta mreže *V2*, prikaz oplate, ispitna varijanta konstrukcije `hc_var_8`

PRILOG III

Nacrti ispitnih varijanti
konstrukcija grotlenih poklopaca





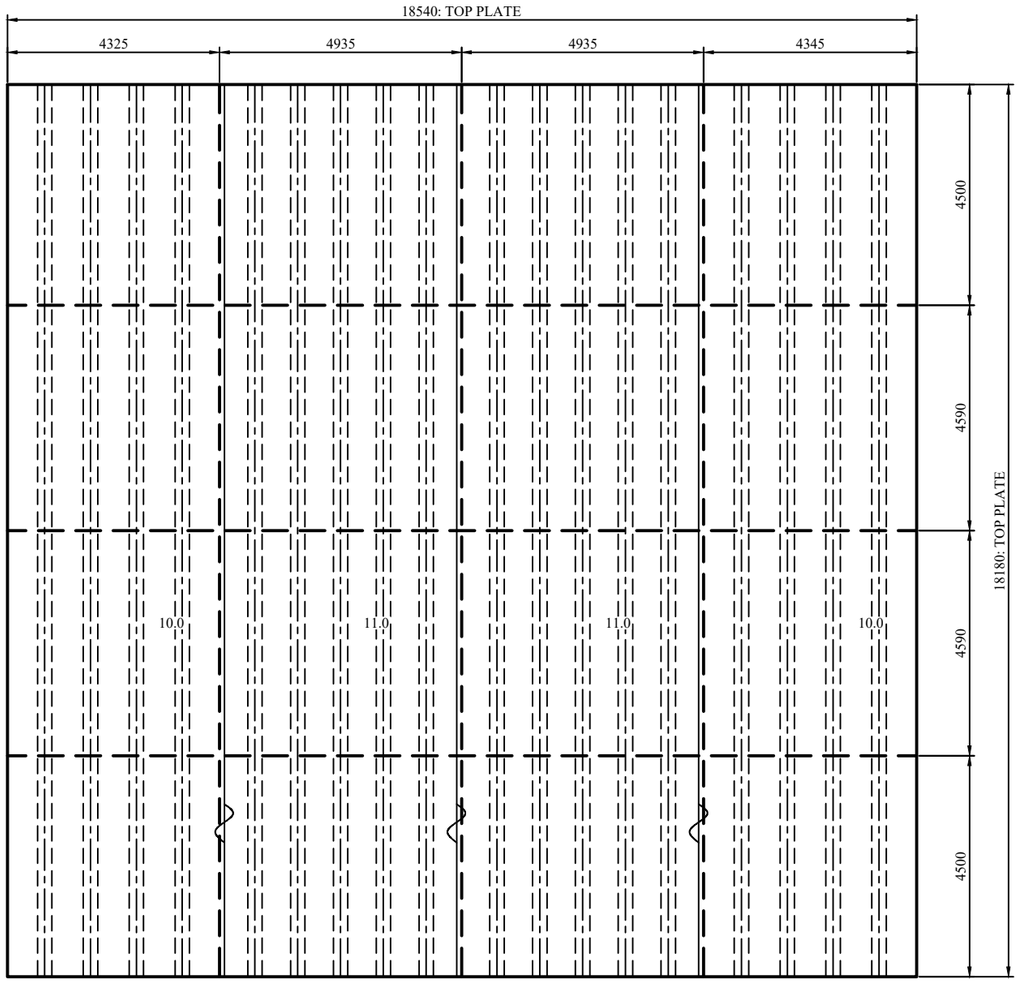


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_1		Objekt broj: 1
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 1	Format: A3
		Listova: 14
		List: 1

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

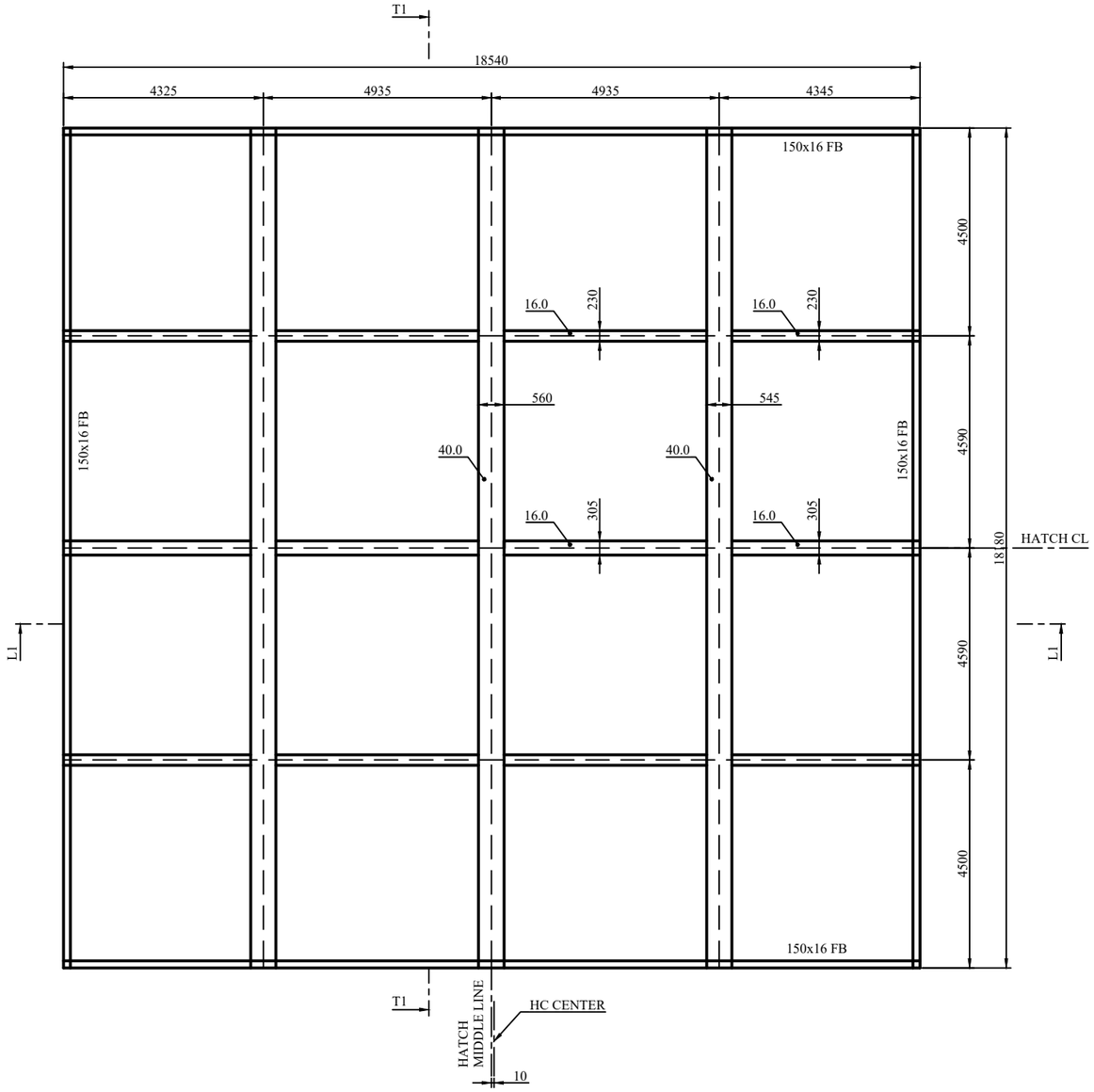
HAT 300x220x6.0 AH36

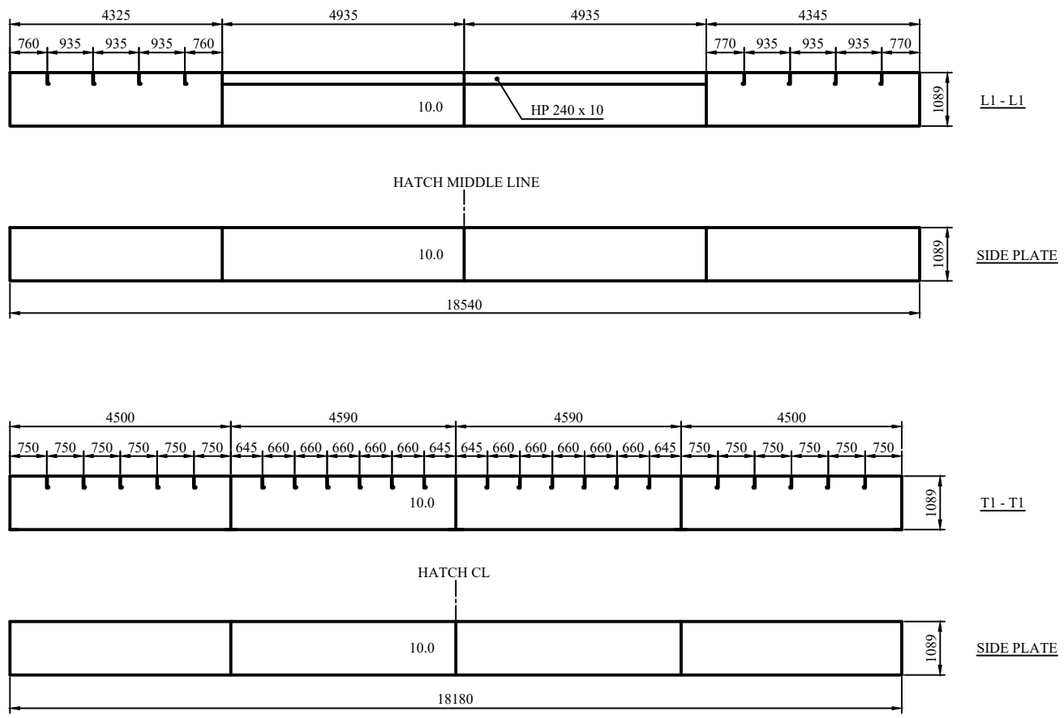
	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_1		Objekt broj: 1	
		R. N. broj:	
Napomena:			Kopija
Materijal:	Masa:	Diplomski rad	
Mjerilo originala 1:125	Naziv: Oplata	Pozicija:	Format: A3
			Listova: 14
	Crtež broj: 2		List: 2

0 10 20 30 40 50 60 70 80 90 100



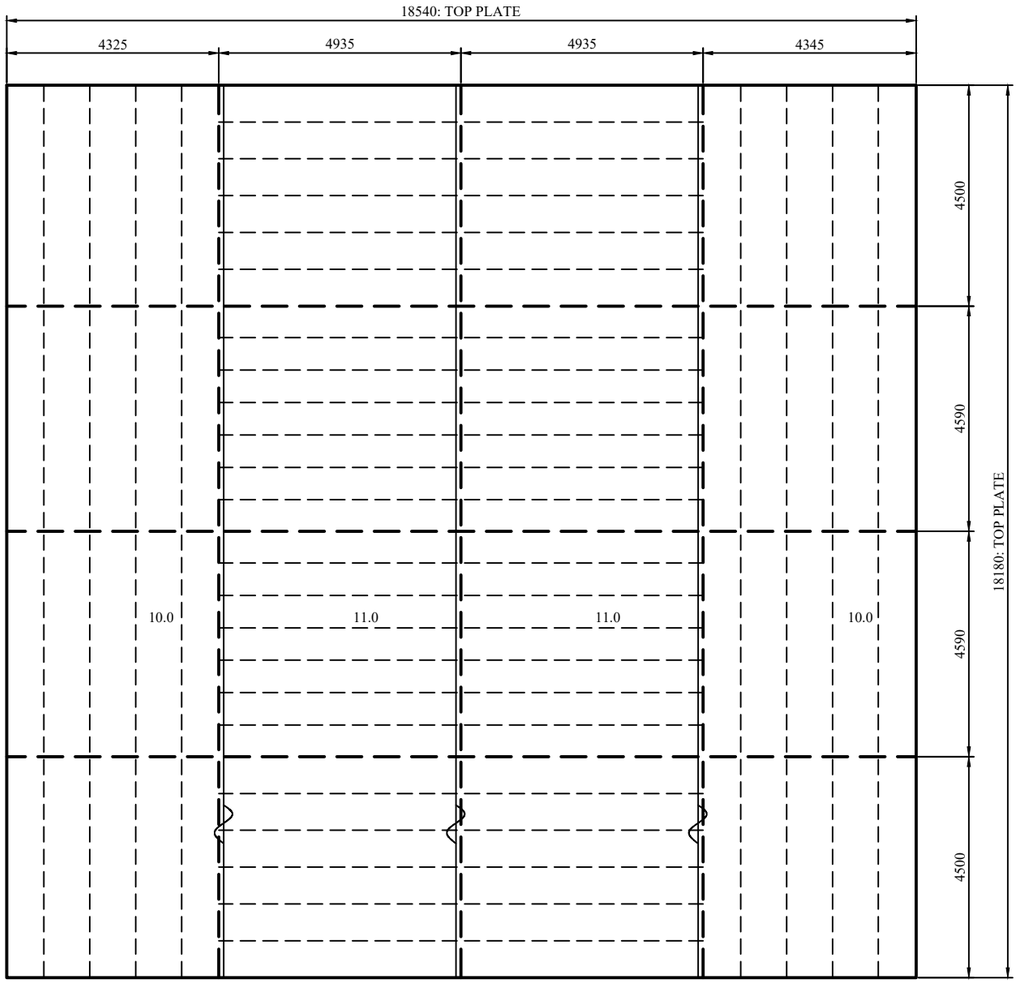


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_2		Objekt broj: 2
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 3	Format: A3
		Listova: 14
		List: 3

0 10 20 30 40 50 60 70 80 90 100



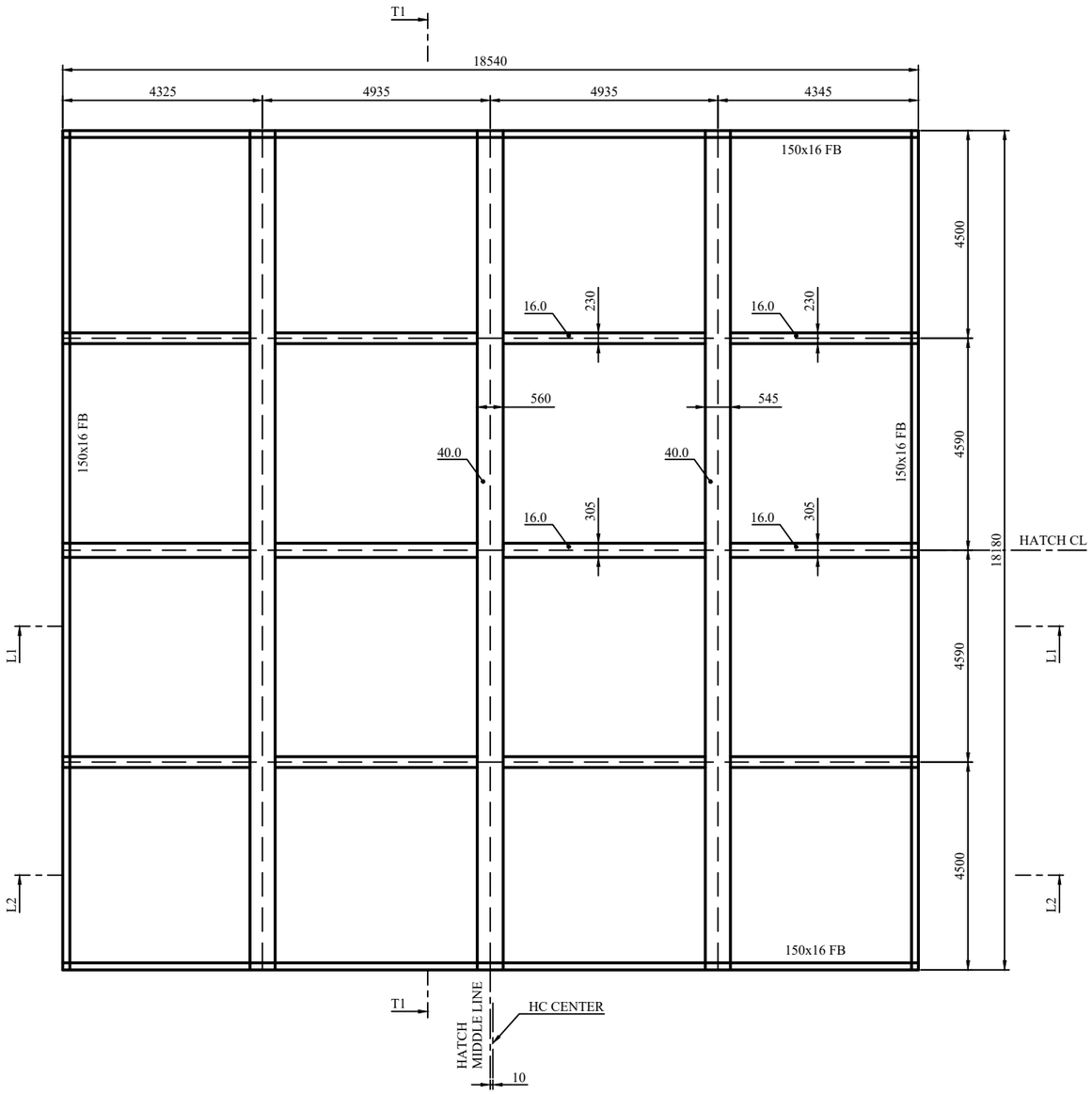
STIFFENERS:

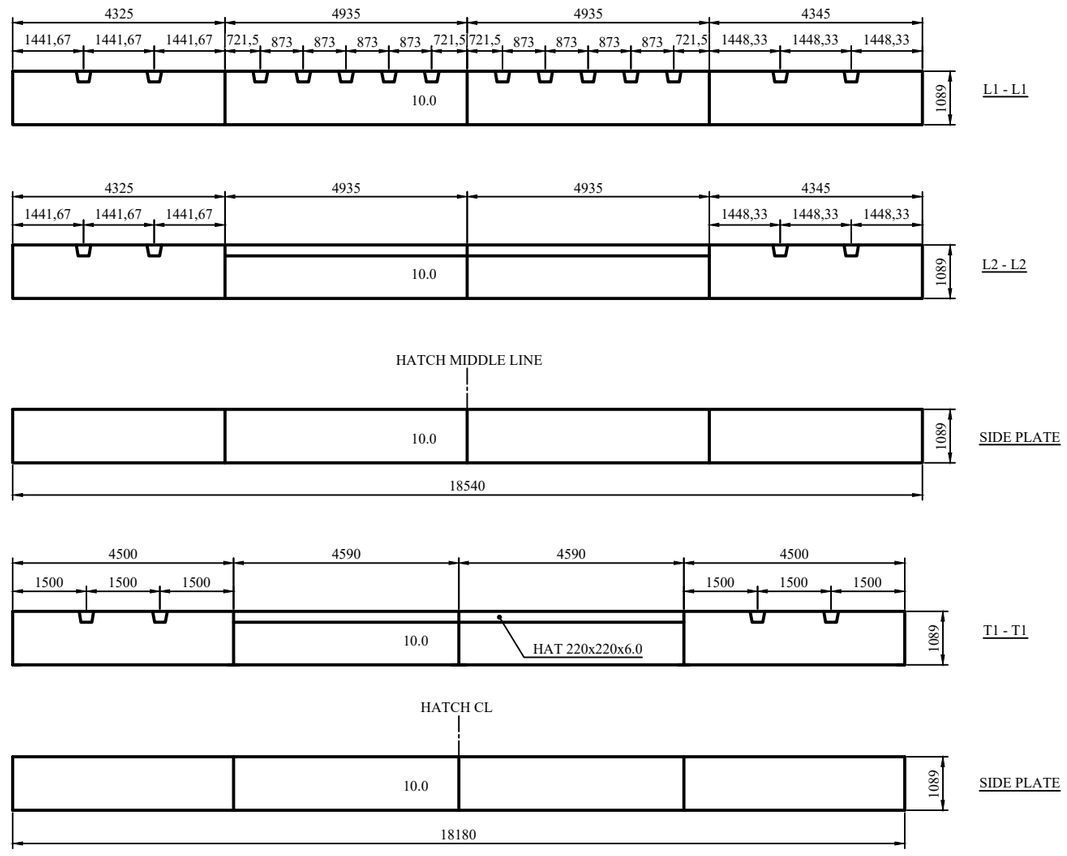
HP 240x10 AH36

	Datum	Ime i prezime	Potpis	FSB Zagreb
Projektirao				
Razradio				
Crtao	19.11.2022.	Gordan Kos		
Pregledao				

Objekt: Ispitna varijanta konstrukcije hc_var_2		Objekt broj: 2
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Oplata	Pozicija:
	Crtež broj: 4	Format: A3
		Listova: 14
		List: 4

0 10 20 30 40 50 60 70 80 90 100



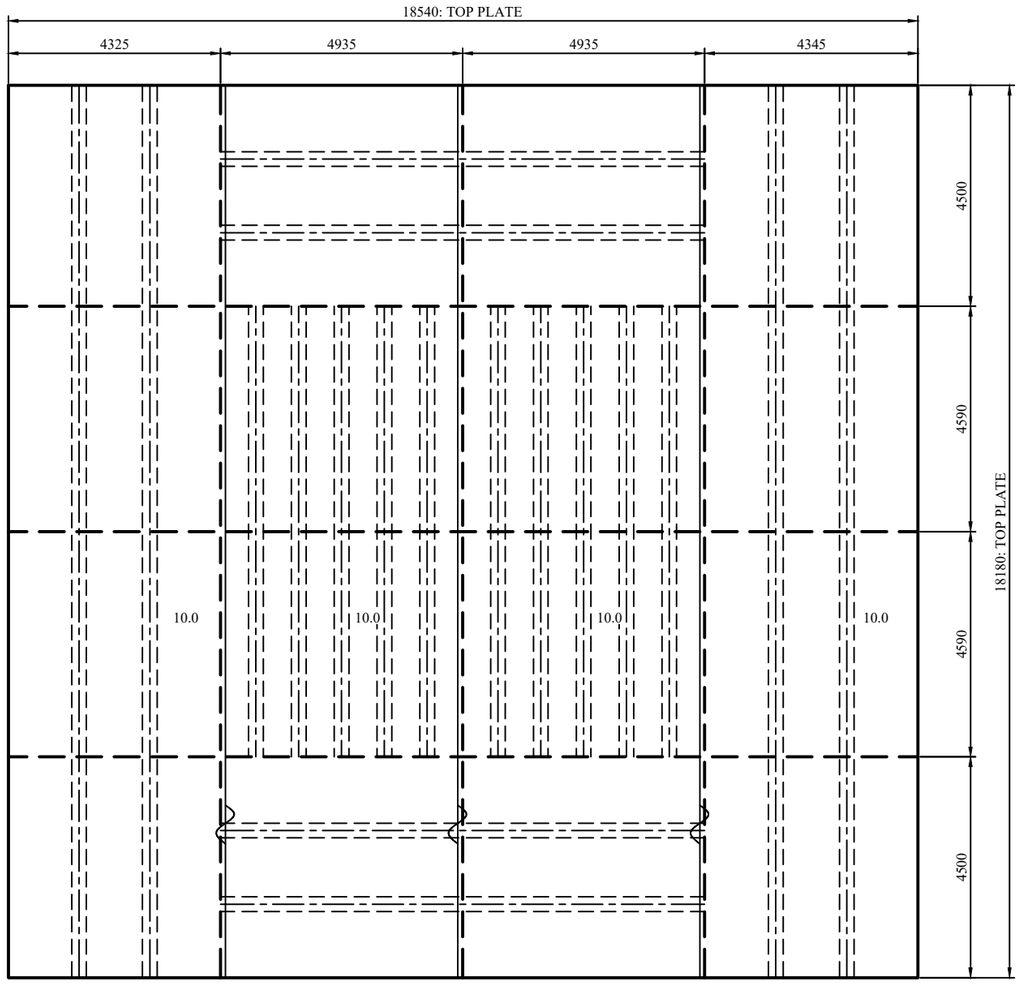


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_3		Objekt broj: 3
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 5	Format: A3
		Listova: 14
		List: 5

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

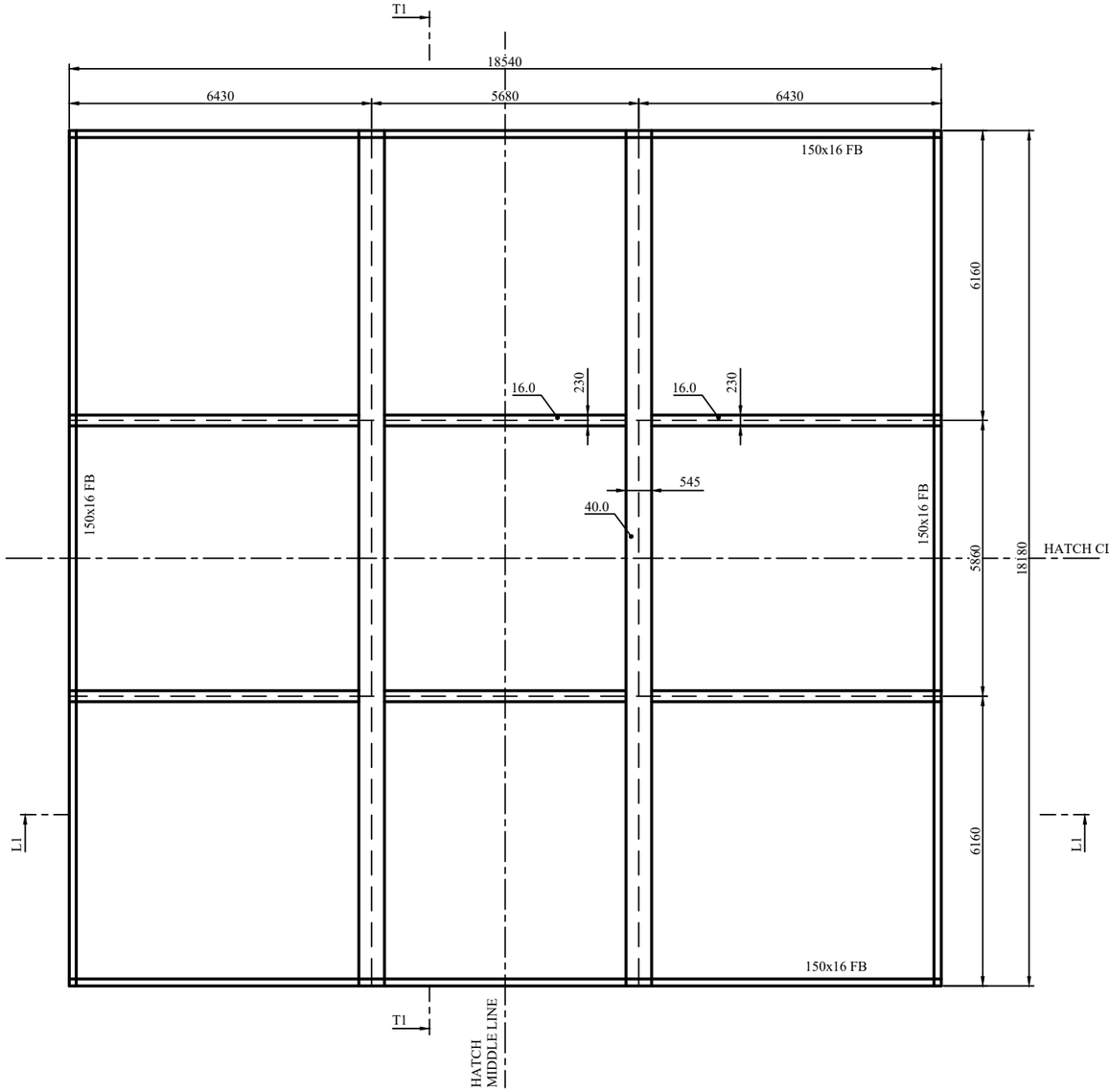
HAT 220x220x6.0 AH36

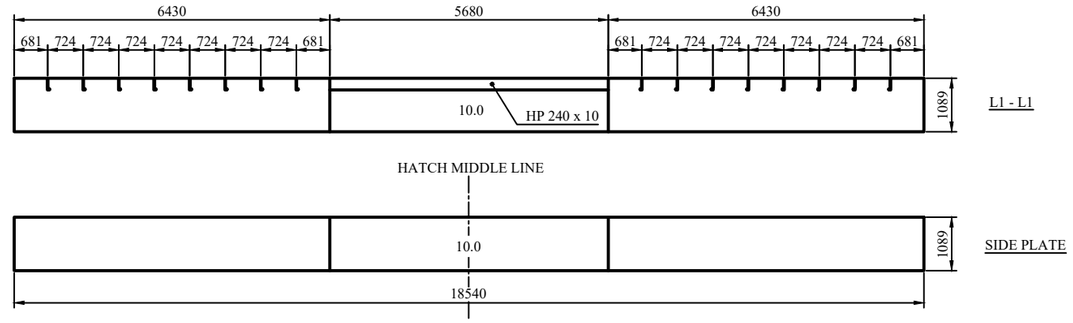
	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

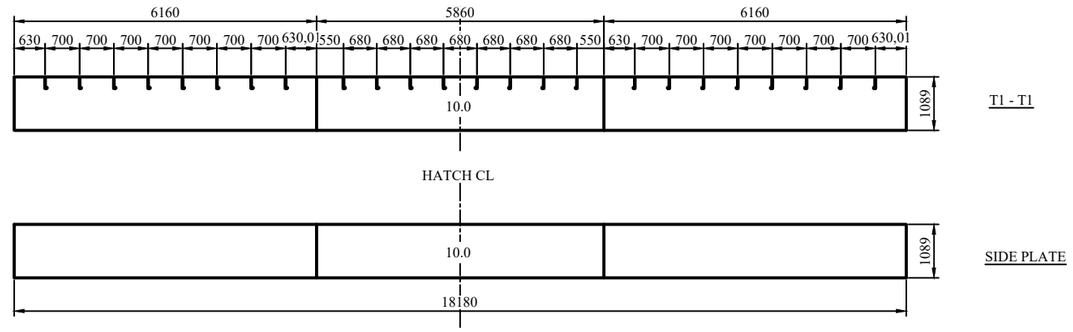
FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_3		Objekt broj: 3
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Oplata	Pozicija:
	Crtež broj: 6	Format: A3
		Listova: 14
		List: 6

0 10 20 30 40 50 60 70 80 90 100





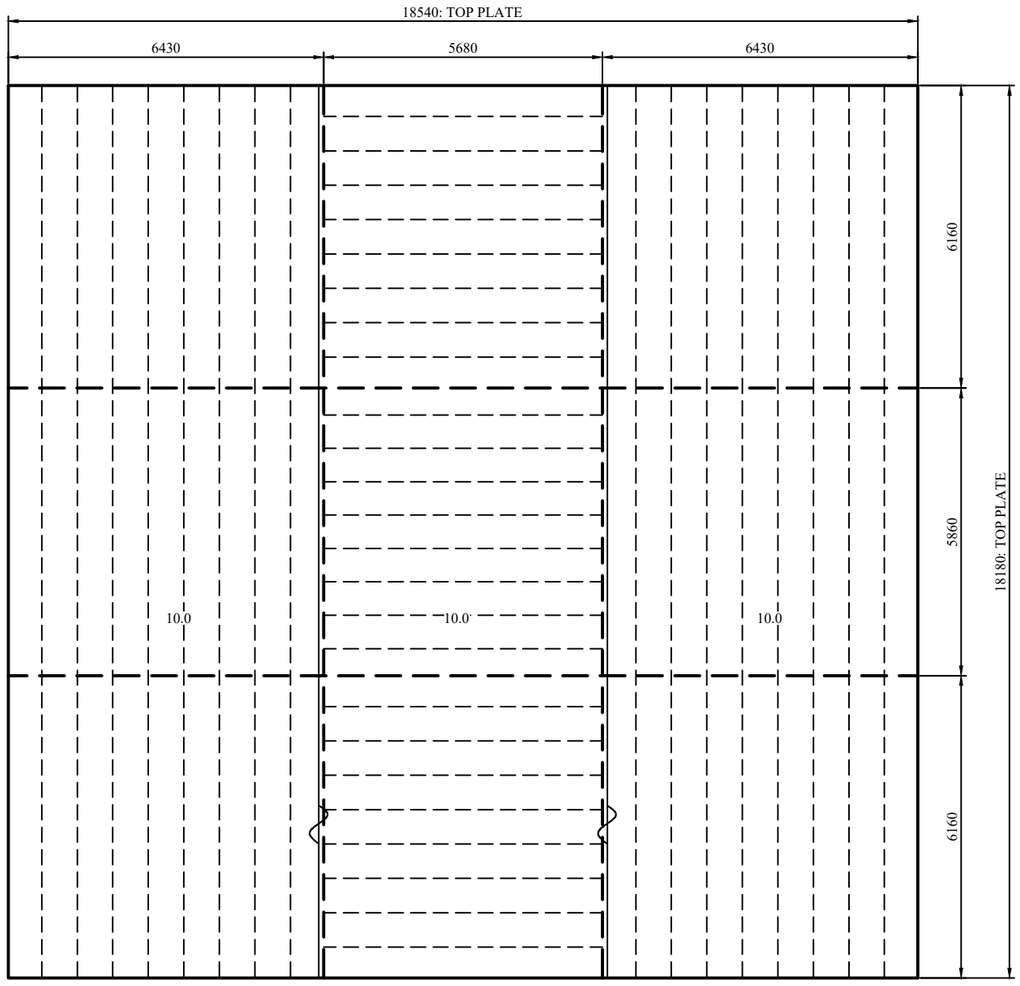


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_4		Objekt broj: 4 R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija: Format: A3 Listova: 14 List: 7
	Crtež broj: 7	

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

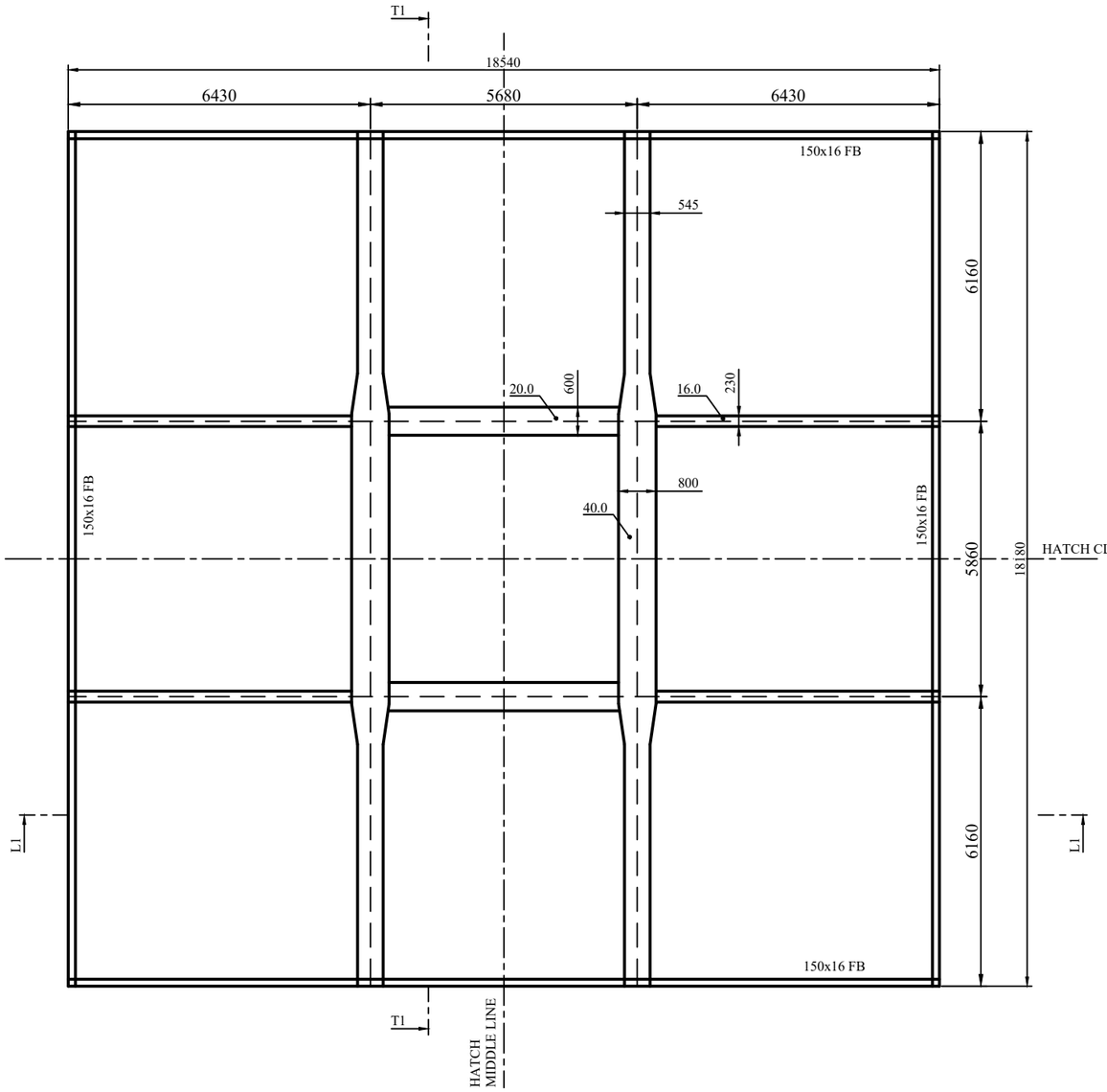
HP 240x10 AH36

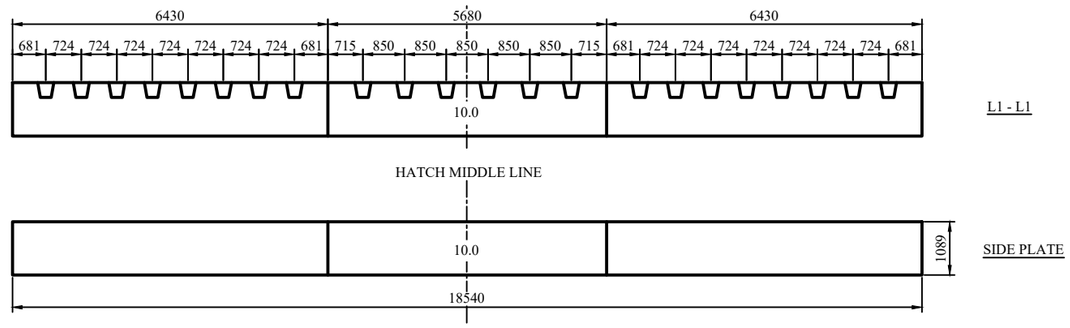
	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

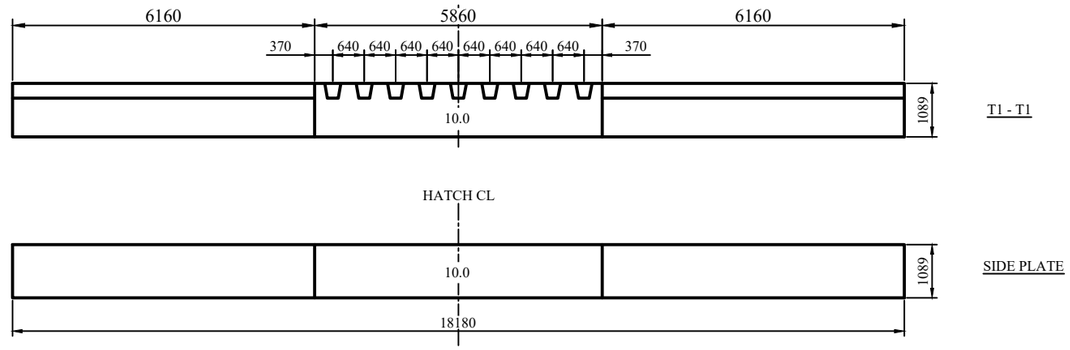
FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_4	Objekt broj: 4
	R. N. broj:
Napomena:	Kopija
Materijal: / Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Oplata / Pozicija:
	Format: A3
	Listova: 14
Crtež broj: 8	List: 8

0 10 20 30 40 50 60 70 80 90 100





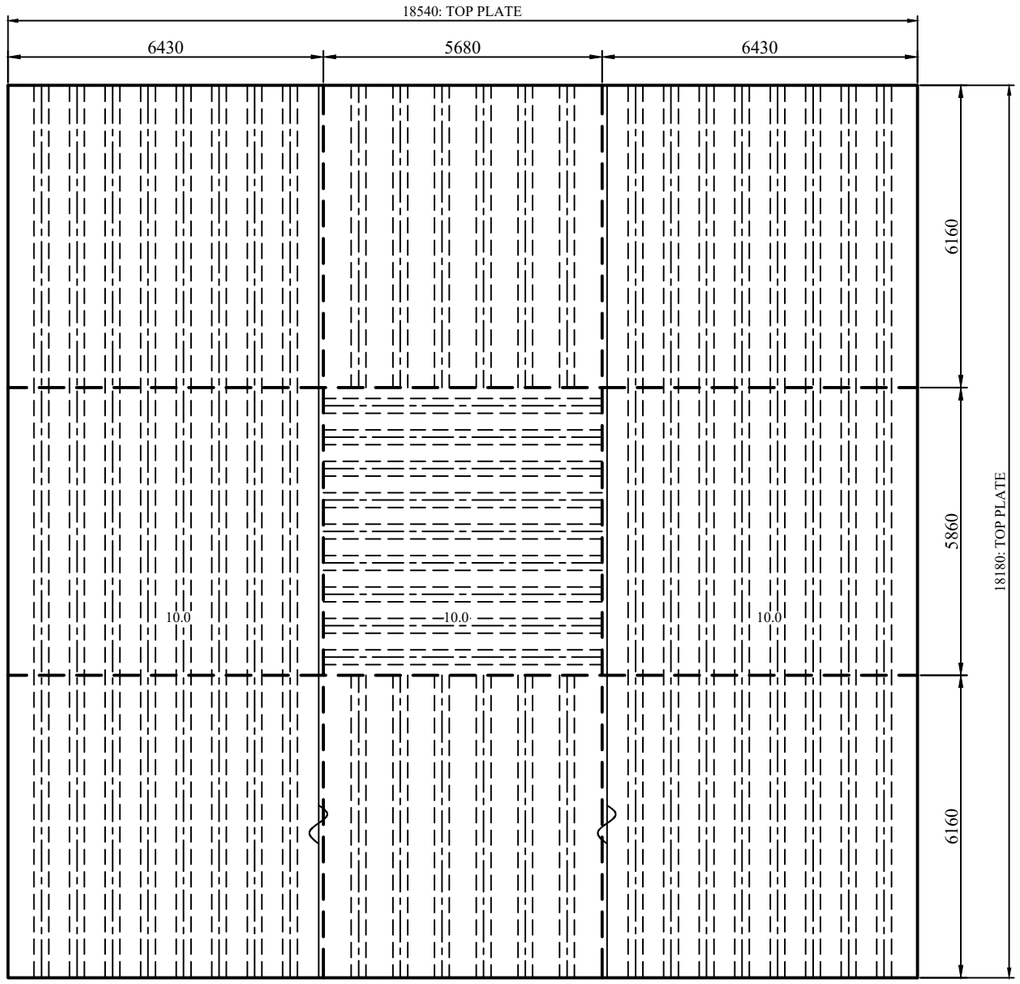


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_5		Objekt broj: 5
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 9	Format: A3
		Listova: 14
		List: 9

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

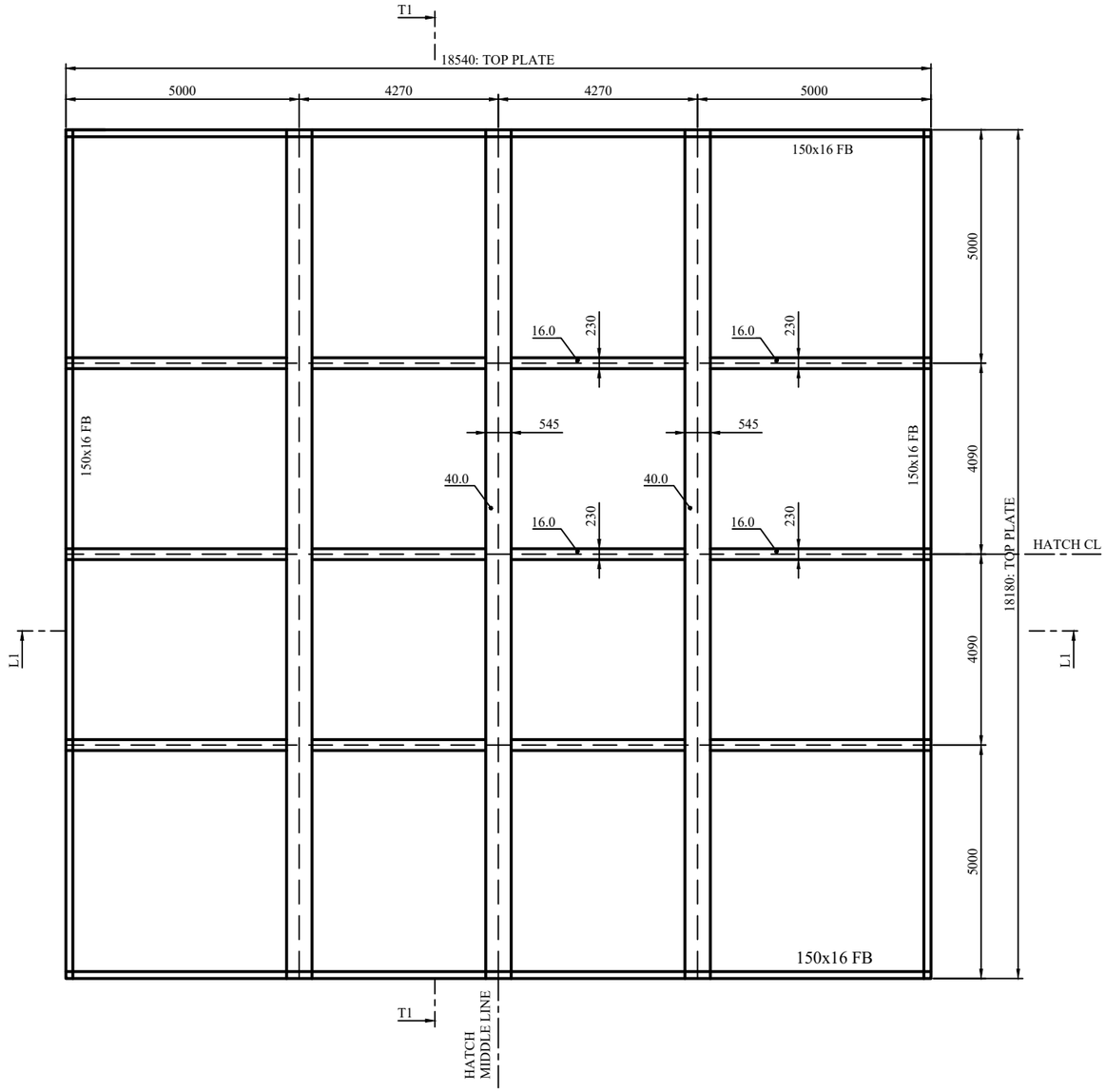
HAT 300x220x8.0 AH36

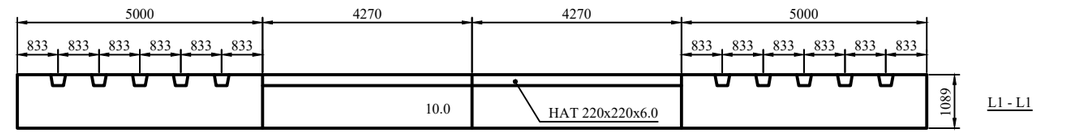
	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

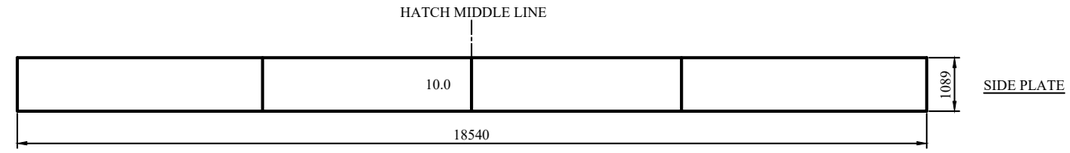
FSB Zagreb

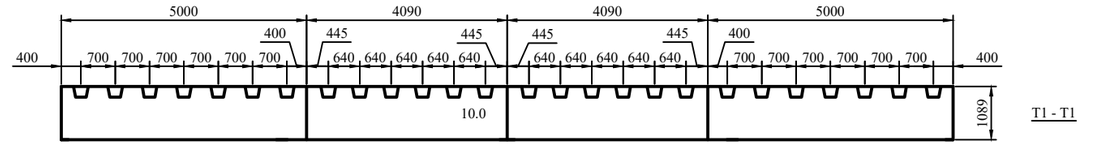
Objekt: Ispitna varijanta konstrukcije hc_var_5		Objekt broj: 5 R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Oplata	Pozicija:
	Crtež broj: 10	Format: A3 Listova: 14 List: 10

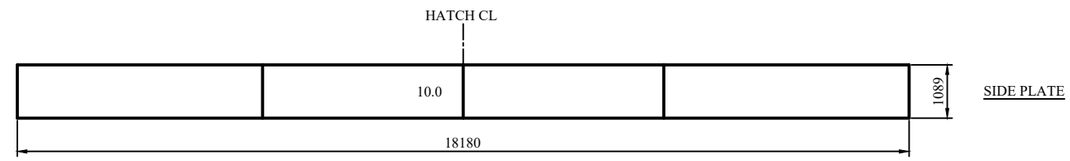
0 10 20 30 40 50 60 70 80 90 100









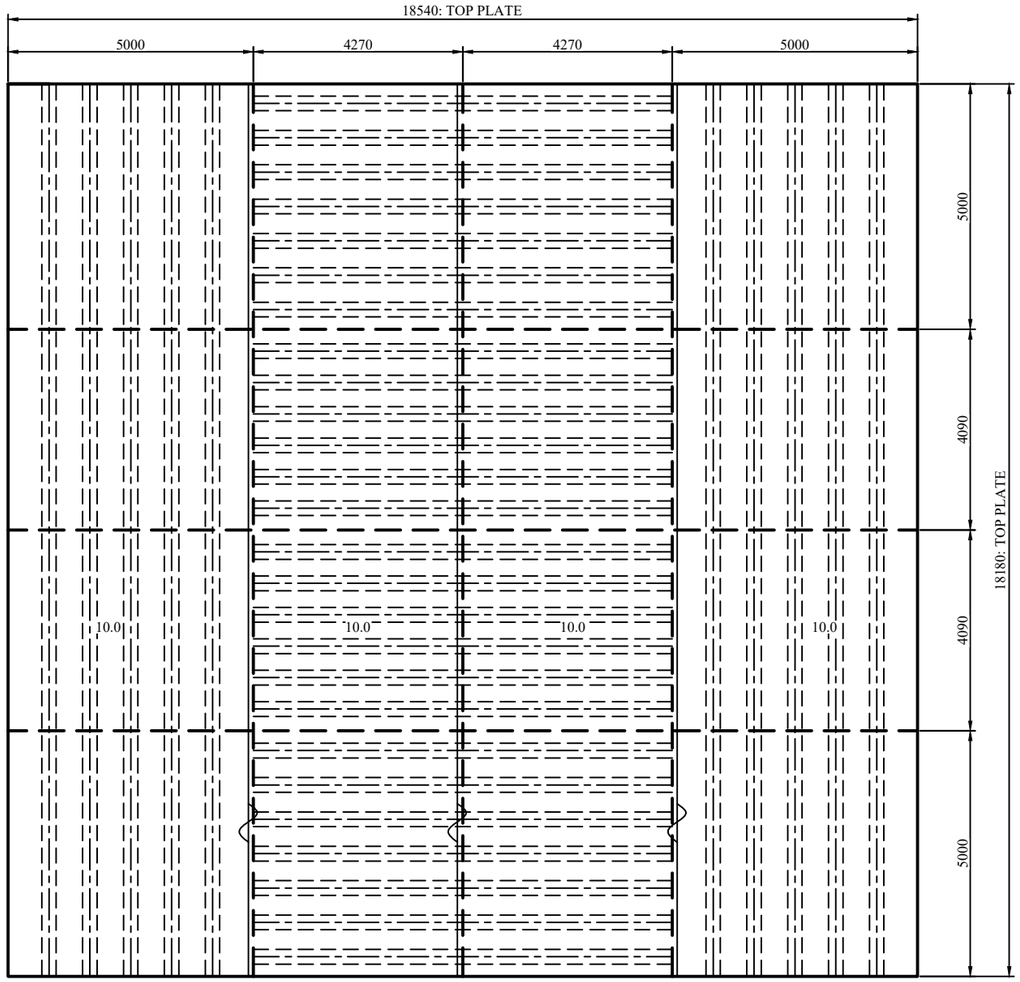


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_6		Objekt broj: 6 R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 11	Format: A3 Listova: 14 List: 11

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

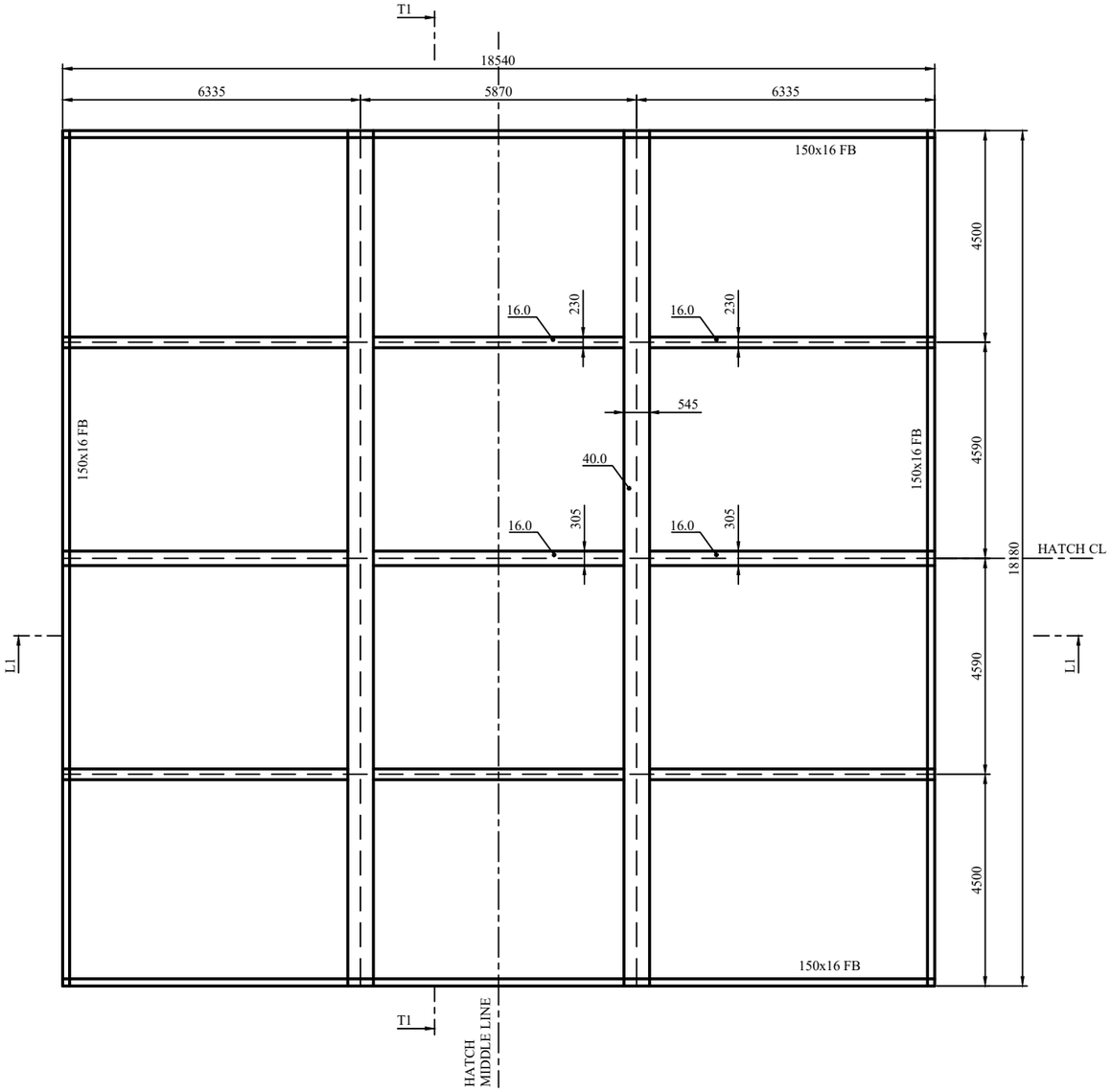
HAT 220x220x6.0 AH36

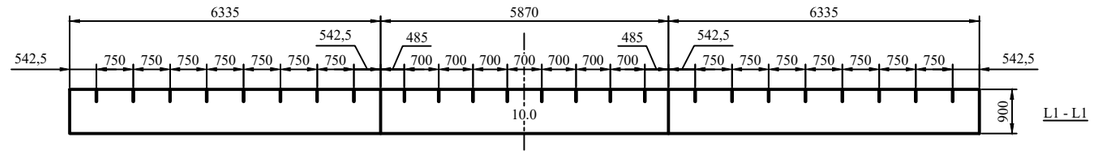
	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

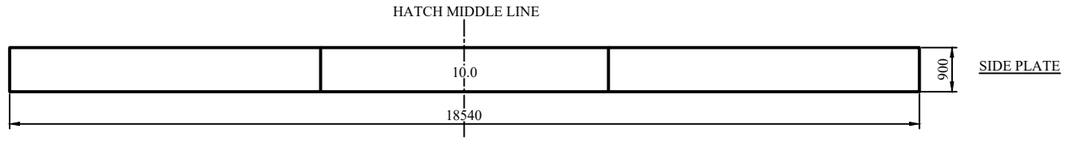
FSB Zagreb

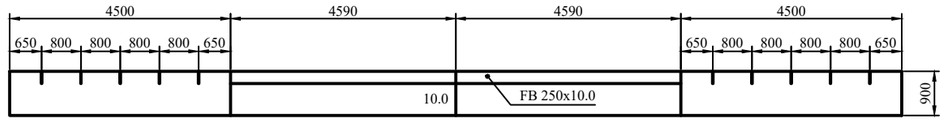
Objekt: Ispitna varijanta konstrukcije hc_var_6	Objekt broj: 6
	R. N. broj:
Napomena:	Kopija
Materijal: / Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Oplata / Pozicija:
	Format: A3
	Listova: 14
Crtež broj: 12	List: 12

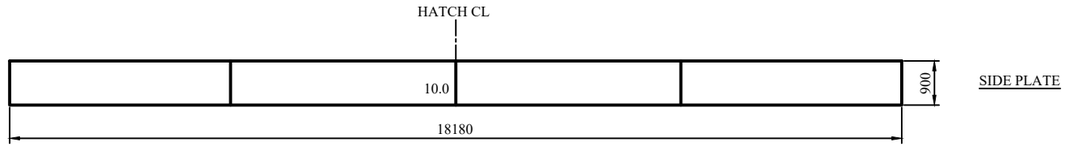
0 10 20 30 40 50 60 70 80 90 100









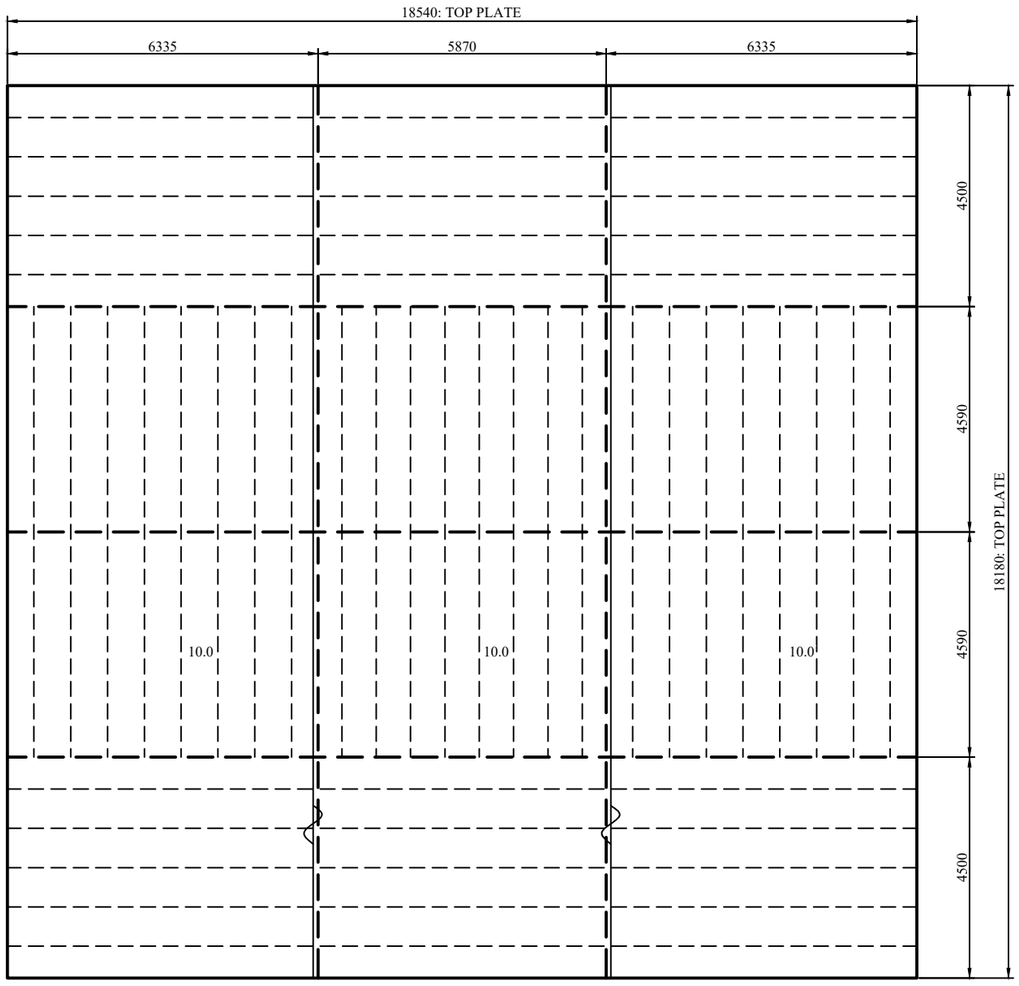


	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt: Ispitna varijanta konstrukcije hc_var_7		Objekt broj: 7
		R. N. broj:
Napomena:		Kopija
Materijal:	Masa:	Diplomski rad
Mjerilo originala 1:125	Naziv: Jaki nosači	Pozicija:
	Crtež broj: 13	Format: A3
		Listova: 14
		List: 13

0 10 20 30 40 50 60 70 80 90 100



STIFFENERS:

FB 250x10.0 AH36

	Datum	Ime i prezime	Potpis
Projektirao			
Razradio			
Crtao	19.11.2022.	Gordan Kos	
Pregledao			

FSB Zagreb

Objekt:
Ispitna varijanta konstrukcije hc_var_7

Objekt broj: 7

R. N. broj:

Napomena:

Kopija

Materijal:

Masa:

Diplomski rad

Mjerilo originala
1:125

Naziv:
Oplata

Pozicija:

Format: A3

Listova: 14

Crtež broj: 14

List: 14

0 10 20 30 40 50 60 70 80 90 100

PRILOG IV

Programski kod modula grillage_mesher.py

```python
"""
University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture
Department of Naval Architecture and Ocean Engineering

Master's thesis project

    Gordan Kos, univ.bacc.ing.nav.arch.
    Dr.sc. Pero Prebeg, dipl.ing.

Module for grillage finite element mesh generation
Design Visualizer for Ship Grillage Design d3v-sgd

"""
import itertools
from grillage.grillage_model import *
from femdir.custom_exceptions import *
from grillage.grillage_fem import GeoGrillageFEM


class MeshVariant(Enum):
    V1 = 1
    V2 = 2


class ModelCheck:
    def __init__(self, grillage: Grillage):
        """
        Class for checking the grillage model for symmetry
        and mesh generation feasibility.

        Symmetry checks include:
            Relative distances of symmetric primary supporting members.
            Central primary supporting member position.
            Plate property, stiffener layout and stiffener direction.
            Beam property of segments.

        :param grillage: Input grillage model.
        """
        self._grillage = grillage

    def longitudinal_psm_symmetry(self):
        """
        :return: True if relative distances of longitudinal primary supporting
            members are symmetric.
        """
        test = False
        for member in self._grillage.longitudinal_members().values():
            if member.rel_dist < 0.5:
                y1 = member.rel_dist
                y2 = member.symmetric_member.rel_dist
                test = np.isclose(y1, 1 - y2)
                if not test:
                    break
        return test

    def transverse_psm_symmetry(self):
        """
        :return: True if relative distances of transverse primary supporting
            members are symmetric.
        """
```

```
        test = False
        for member in self._grillage.transverse_members().values():
            if member.rel_dist < 0.5:
                x1 = member.rel_dist
                x2 = member.symmetric_member.rel_dist
                test = np.isclose(x1, 1 - x2)
                if not test:
                    break
        return test

    def central_longitudinal(self):
        """
        :return: True if central longitudinal primary supporting member exists
            and has relative distance coordinate 0.5
        """
        n_long = self._grillage.N_longitudinal
        if np.mod(n_long, 2) != 0:
            member_id = int(np.ceil(n_long / 2))
            central_member = self._grillage.longitudinal_members()[member_id]
            if np.isclose(central_member.rel_dist, 0.5):
                return True
            else:
                return False
        else:
            return True

    def central_transversal(self):
        """
        :return: True if central transverse primary supporting member exists
            and has relative distance coordinate 0.5
        """
        n_tran = self._grillage.N_transverse
        if np.mod(n_tran, 2) != 0:
            member_id = int(np.ceil(n_tran / 2))
            central_member = self._grillage.transverse_members()[member_id]
            if np.isclose(central_member.rel_dist, 0.5):
                return True
            else:
                return False
        else:
            return True

    def plate_zone_ID_array(self):
        """
        :return: 2D array of all plating zone IDs arranged to represent
            relative placement of zones on the entire grillage model.
        """
        total_rows = self._grillage.N_longitudinal - 1
        total_columns = self._grillage.N_transverse - 1
        total_zones = total_rows * total_columns

        id_list = np.arange(1, total_zones + 1, 1)
        plating_zone_array = np.reshape(id_list, [total_rows, total_columns])
        return plating_zone_array

    def longitudinal_plate_symmetry(self):
        """
        :return: True if longitudinally symmetric plating zones have the same
            plate property, stiffener layout and stiffener direction.
        """
```

```
        same_plate_prop = True
        same_stiff_layout = True
        same_stiff_dir = True

        plate_ref_array = self.plate_zone_ID_array()
        long_symm_ref_array = np.flip(plate_ref_array, axis=0)

        row_limit = int(np.floor((self._grillage.N_longitudinal - 1) / 2))
        column_limit = self._grillage.N_transverse - 1

        for row in range(0, row_limit):
            for column in range(0, column_limit):
                plate_id = plate_ref_array[row, column]
                symm_plate_id = long_symm_ref_array[row, column]
                plate = self._grillage.plating()[plate_id]
                symm_plate = self._grillage.plating()[symm_plate_id]

                if plate.plate_prop is not symm_plate.plate_prop:
                    same_plate_prop = False
                    break

                if plate.stiff_layout is not symm_plate.stiff_layout:
                    same_stiff_layout = False
                    break

                if plate.stiff_dir is not symm_plate.stiff_dir:
                    same_stiff_dir = False
                    break

        tests = [same_plate_prop, same_stiff_layout, same_stiff_dir]
        if all(tests):
            return True
        else:
            return False

    def transverse_plate_symmetry(self):
        """
        :return: True if transversely symmetric plating zones have the same
            plate property, stiffener layout and stiffener direction.
        """
        same_plate_prop = True
        same_stiff_layout = True
        same_stiff_dir = True

        plate_ref_array = self.plate_zone_ID_array()
        tran_symm_ref_array = np.flip(plate_ref_array, axis=1)

        row_limit = self._grillage.N_longitudinal - 1
        column_limit = int(np.floor((self._grillage.N_transverse - 1) / 2))

        for row in range(0, row_limit):
            for column in range(0, column_limit):
                plate_id = plate_ref_array[row, column]
                symm_plate_id = tran_symm_ref_array[row, column]
                plate = self._grillage.plating()[plate_id]
                symm_plate = self._grillage.plating()[symm_plate_id]

                if plate.plate_prop is not symm_plate.plate_prop:
                    same_plate_prop = False
                    break
```

```
181
182                 if plate.stiff_layout is not symm_plate.stiff_layout:
183                     same_stiff_layout = False
184                     break
185
186                 if plate.stiff_dir is not symm_plate.stiff_dir:
187                     same_stiff_dir = False
188                     break
189
190         tests = [same_plate_prop, same_stiff_layout, same_stiff_dir]
191         if all(tests):
192             return True
193         else:
194             return False
195
196     def longitudinal_segment_symmetry(self):
197         """
198         :return: True if symmetric longitudinal segments have the same
199             beam property.
200         """
201         same_beam_prop = True
202
203         for member in self._grillage.longitudinal_members().values():
204             if member.rel_dist < 0.5:
205                 for segment_id in range(1, self._grillage.N_transverse):
206                     segment = member.segments[segment_id - 1]
207                     symm_seg = member.symmetric_member.segments[segment_id - 1]
208
209                     if segment.beam_prop is not symm_seg.beam_prop:
210                         same_beam_prop = False
211                         break
212         return same_beam_prop
213
214     def transverse_segment_symmetry(self):
215         """
216         :return: True if symmetric transverse segments have the same
217             beam property.
218         """
219         same_beam_prop = True
220
221         for member in self._grillage.transverse_members().values():
222             if member.rel_dist < 0.5:
223                 for segment_id in range(1, self._grillage.N_longitudinal):
224                     segment = member.segments[segment_id - 1]
225                     symm_seg = member.symmetric_member.segments[segment_id - 1]
226
227                     if segment.beam_prop is not symm_seg.beam_prop:
228                         same_beam_prop = False
229                         break
230         return same_beam_prop
231
232     def long_symmetry_tests(self):
233         """
234         :return: True if model passes all longitudinal symmetry tests.
235         """
236         tests = [self.longitudinal_psm_symmetry(),
237                  self.central_longitudinal(),
238                  self.longitudinal_segment_symmetry(),
239                  self.longitudinal_plate_symmetry()]
240         if all(tests):
```

```
            return True
        else:
            return False

    def tran_symmetry_tests(self):
        """
        :return: True if model passes all transverse symmetry tests.
        """
        tests = [self.transverse_psm_symmetry(),
                 self.central_transversal(),
                 self.transverse_segment_symmetry(),
                 self.transverse_plate_symmetry()]
        if all(tests):
            return True
        else:
            return False

    def assign_symmetry(self):
        self._grillage.assign_symmetric_members()

        if self.long_symmetry_tests() and self.tran_symmetry_tests():
            return AOS.BOTH
        elif self.long_symmetry_tests():
            return AOS.LONGITUDINAL
        elif self.tran_symmetry_tests():
            return AOS.TRANSVERSE
        else:
            return AOS.NONE

    def mesh_feasibility(self):
        """
        :return: Calls custom exception if input grillage model
            can not be meshed.

        Method stops calculation of limits of grillage mesh generation if
        grillage model does not meet the following criteria:
            1.) Plating zones between two adjacent Primary Supporting Members
                may not have the same stiffener orientation and different
                stiffener spacing.
            2.) Insert another impossibility here
        """
        # 1.)
        # Between longitudinal primary supporting members:
        for psm_id in range(1, self._grillage.N_longitudinal):
            psm_1 = self._grillage.longitudinal_members()[psm_id]
            psm_2 = self._grillage.longitudinal_members()[psm_id + 1]
            plate_list = self._grillage.plating_zones_between_psm(psm_1, psm_2)
            plate_combinations = itertools.combinations(plate_list, 2)

            for plate in list(plate_combinations):
                plate1 = plate[0]  # First plating zone of plate combinations
                plate2 = plate[1]  # Second plating zone of plate combinations
                spacing1 = Plate.get_stiffener_spacing(plate1)
                spacing2 = Plate.get_stiffener_spacing(plate2)

                if plate1.stiff_dir is BeamDirection.LONGITUDINAL and \
                        plate2.stiff_dir is BeamDirection.LONGITUDINAL and \
                        spacing1 != spacing2:
                    raise FeasibilityTestFailLong(psm_1.id, psm_2.id, plate1.id,
                                                  plate2.id, spacing1, spacing2)
```

```
301
302         # Between transverse primary supporting members:
303         for psm_id in range(1, self._grillage.N_transverse):
304             psm_1 = self._grillage.transverse_members()[psm_id]
305             psm_2 = self._grillage.transverse_members()[psm_id + 1]
306             plate_list = self._grillage.plating_zones_between_psm(psm_1, psm_2)
307             plate_combinations = itertools.combinations(plate_list, 2)
308
309             for plate in list(plate_combinations):
310                 plate1 = plate[0]  # First plating zone of plate combinations
311                 plate2 = plate[1]  # Second plating zone of plate combinations
312                 spacing1 = Plate.get_stiffener_spacing(plate1)
313                 spacing2 = Plate.get_stiffener_spacing(plate2)
314
315                 if plate1.stiff_dir is BeamDirection.TRANSVERSE and \
316                         plate2.stiff_dir is BeamDirection.TRANSVERSE and \
317                         spacing1 != spacing2:
318                     raise FeasibilityTestFailTran(psm_1.id, psm_2.id, plate1.id,
319                                                   plate2.id, spacing1, spacing2)
320
321     @staticmethod
322     def symmetry_override_check(discovered: AOS, override: AOS):
323         """
324         Method checks if grillage model is actually symmetric about the chosen
325             Axis of Symmetry override, based on automatically discovered AOS.
326         :return: Raises InvalidAxisOfSymmOverride custom exception.
327         """
328         if discovered is AOS.NONE:
329             invalid_overrides = [AOS.LONGITUDINAL, AOS.TRANSVERSE, AOS.BOTH]
330             if override in invalid_overrides:
331                 raise InvalidAxisOfSymmOverride(discovered, override)
332
333         elif discovered is AOS.LONGITUDINAL:
334             invalid_overrides = [AOS.TRANSVERSE, AOS.BOTH]
335             if override in invalid_overrides:
336                 raise InvalidAxisOfSymmOverride(discovered, override)
337
338         elif discovered is AOS.TRANSVERSE:
339             invalid_overrides = [AOS.LONGITUDINAL, AOS.BOTH]
340             if override in invalid_overrides:
341                 raise InvalidAxisOfSymmOverride(discovered, override)
342
343     @staticmethod
344     def mesh_V2_tr_check(plate_edge_nodes, flange_edge_nodes):
345         """
346         Method checks transition row feasibility based on segment web finite
347         element dimensions at the top and bottom row of elements.
348
349         :param plate_edge_nodes: Distances between plate nodes.
350         :param flange_edge_nodes: Distances between flange nodes.
351         :return: Calls custom exception if grillage model can not be meshed using
352             MeshVariantV2 because the input mesh control parameters result in a
353             finite element mesh with elements that are too small.
354         """
355         p1_end1 = plate_edge_nodes[1]
356         p2_end1 = plate_edge_nodes[2]
357         p_end1_total = p1_end1 + p2_end1
358
359         p1_end2 = plate_edge_nodes[len(plate_edge_nodes)]
360         p2_end2 = plate_edge_nodes[len(plate_edge_nodes) - 1]
```

```
361         p_end2_total = p1_end2 + p2_end2
362
363         f1_end1 = flange_edge_nodes[1]
364         f1_end2 = flange_edge_nodes[len(flange_edge_nodes)]
365
366         if f1_end1 > p_end1_total:
367             raise MeshV2FeasibilityFail(p_end1_total, f1_end1)
368
369         if f1_end2 > p_end2_total:
370             raise MeshV2FeasibilityFail(p_end2_total, f1_end2)
371
372         if len(plate_edge_nodes) > 6:
373             p3_end1 = plate_edge_nodes[3]
374             p3_end2 = plate_edge_nodes[len(plate_edge_nodes) - 2]
375             f2_end1 = flange_edge_nodes[2]
376             f2_end2 = flange_edge_nodes[len(flange_edge_nodes) - 1]
377
378             p_end1_total = p1_end1 + p2_end1 + p3_end1
379             p_end2_total = p1_end2 + p2_end2 + p3_end2
380
381             if np.isclose(f1_end1 + f2_end1, p_end1_total):
382                 raise MeshV2FeasibilityFailGeneric()
383
384             if np.isclose(f1_end2 + f2_end2, p_end2_total):
385                 raise MeshV2FeasibilityFailGeneric()
386
387     def same_flange_width_test(self):
388         """
389         Method checks if Primary Supporting Members have different flange width
390         on their segments. Raises acustom exception for MeshVariantV1.
391         """
392         for member in self._grillage.longitudinal_members().values():
393             bf_set = set()
394             for segment in member.segments:
395                 beam_type = segment.beam_prop.beam_type
396                 if beam_type is BeamType.T or beam_type is BeamType.L:
397                     bf = segment.beam_prop.bf
398                     bf_set.add(bf)
399                 if beam_type is BeamType.FB:
400                     bf_set.add(0)
401             if len(bf_set) > 1:
402                 raise DifferentFlangeWidth(member.id, member.direction.name)
403
404         for member in self._grillage.transverse_members().values():
405             bf_set = set()
406             for segment in member.segments:
407                 beam_type = segment.beam_prop.beam_type
408                 if beam_type is BeamType.T or beam_type is BeamType.L:
409                     bf = segment.beam_prop.bf
410                     bf_set.add(bf)
411                 if beam_type is BeamType.FB:
412                     bf_set.add(0)
413             if len(bf_set) > 1:
414                 raise DifferentFlangeWidth(member.id, member.direction.name)
415
416
417 class MeshExtent:
418     def __init__(self, grillage: Grillage, axis_of_symm_override):
419         """
420         Class for calculating FE mesh extents for the selected grillage model
```

```
        and Axis of Symmetry. Contains dictionaries of all plating zones and
        segments to be fully or partially meshed. Generates GeoFEM materials,
        plate and beam properties based on identified mesh extents.

        :param grillage: Input grillage model.
        :param axis_of_symm_override: overrides automatic Axis of Symmetry discovery.

        Class contains the following data:

        plating_zones_ref_array - 2D array of plating zone IDs to be meshed,
                                  based on Axis of Symmetry selection (AOS)
        all_plating_zones - All plating zones included in mesh generation
        full_plate_zones - Plating zones to be fully meshed
        long_half_plate_zones - Plating zones to be split with a longitudinal
                                axis of symmetry
        tran_half_plate_zones - Plating zones to be split with a transverse AOS
        quarter_plate_zone - Plating zone to be split with both AOS
        long_e_split_zone - Plating zones with longitudinal axis of symmetry
                            passing between stiffeners
        tran_e_split_zone - Plating zones with transverse axis of symmetry
                            passing between stiffeners

        all_segments - All segments included in mesh generation.
        full_segments - Segments to be fully meshed
        half_segments - Segments split in half by some axis of symmetry
        """
        self._grillage = grillage
        self.model_check = ModelCheck(self._grillage)
        self._axis_of_symm_override = axis_of_symm_override
        self._aos_input = self.model_check.assign_symmetry()

        self.plating_zones_ref_array = []
        self.all_plating_zones = {}
        self.full_plate_zones = {}
        self.long_half_plate_zones = {}
        self.tran_half_plate_zones = {}
        self.quarter_plate_zone = {}
        self.long_e_split_zone = {}
        self.tran_e_split_zone = {}

        self._all_segment_count = 0
        self.all_segments = {}
        self.full_segments = {}
        self.half_segments = {}

    @property
    def grillage(self):
        return self._grillage

    @property
    def axis_of_symm_override(self):
        return self._axis_of_symm_override

    @property
    def aos_input(self):
        return self._aos_input

    @property
    def axis_of_symm(self):
        if self._axis_of_symm_override:
```

```
            return self._axis_of_symm_override
        else:
            return self._aos_input

    def hc_plate_zone_ref_ID_array(self):
        """
        :return: 2D array of all plating zone IDs arranged to represent
            relative placement of zones on the entire grillage model.

        total_rows - Total number of plating zone rows on the grillage model
        total_columns - Total number of plating zone columns on the model
        """
        total_rows = self._grillage.N_longitudinal - 1
        total_columns = self._grillage.N_transverse - 1
        total_zones = total_rows * total_columns

        id_list = np.arange(1, total_zones + 1, 1)
        plating_zone_array = np.reshape(id_list, [total_rows, total_columns])
        return plating_zone_array

    def longitudinal_psm_extent(self):
        """
        :return: Dictionary of longitudinal primary supporting members to be
            considered for mesh generation, based on input Axis of Symmetry.
        """
        longitudinals = {}

        if self.axis_of_symm is AOS.LONGITUDINAL or self.axis_of_symm is AOS.BOTH:
            n_long = int(np.ceil(self._grillage.N_longitudinal / 2))
            for i in range(1, n_long + 1):
                longitudinals[i] = self._grillage.longitudinal_members()[i]
            return longitudinals
        else:
            return self._grillage.longitudinal_members()

    def transverse_psm_extent(self):
        """
        :return: Dictionary of transverse primary supporting members to be
            considered for mesh generation, based on input Axis of Symmetry.
        """
        transversals = {}

        if self.axis_of_symm is AOS.TRANSVERSE or self.axis_of_symm is AOS.BOTH:
            n_tran = int(np.ceil(self._grillage.N_transverse / 2))
            for i in range(1, n_tran + 1):
                transversals[i] = self._grillage.transverse_members()[i]
            return transversals
        else:
            return self._grillage.transverse_members()

    def identify_long_full_segments(self):
        """
        :return: Identifies Segment objects for full mesh generation; grillage
            with a longitudinal axis of symmetry.
            Stores identified segments in full_segments dictionary.

        n_tran_segments - Number of transverse segmenets to be fully meshed
        """
        n_long = self._grillage.N_longitudinal
        n_tran = self._grillage.N_transverse
```

```
        n = 1
        for member in self.longitudinal_psm_extent().values():
            for segment_id in range(0, n_tran - 1):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

        n_tran_segments = int(np.floor((n_long - 1) / 2))
        for member in self._grillage.transverse_members().values():
            for segment_id in range(0, n_tran_segments):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

    def identify_tran_full_segments(self):
        """
        :return: Identifies Segment objects for full mesh generation; grillage
            with a transverse axis of symmetry.
            Stores identified segments in full_segments dictionary.

        n_long_segments - Number of longitudinal segments to be fully meshed
        """
        n_long = self._grillage.N_longitudinal
        n_tran = self._grillage.N_transverse
        n = 1
        n_long_segments = int(np.floor((n_tran - 1) / 2))
        for member in self._grillage.longitudinal_members().values():
            for segment_id in range(0, n_long_segments):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

        for member in self.transverse_psm_extent().values():
            for segment_id in range(0, n_long - 1):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

    def identify_both_full_segments(self):
        """
        :return: Identifies Segment objects for full mesh generation; grillage
            with both axis of symmetry.
            Stores identified segments in full_segments dictionary.

        n_long_segments - Number of longitudinal segments to be fully meshed
        n_tran_segments - Number of transverse segmenets to be fully meshed
        """
        n_long = self._grillage.N_longitudinal
        n_tran = self._grillage.N_transverse

        n = 1
        n_long_segments = int(np.floor((n_tran - 1) / 2))
        for member in self.longitudinal_psm_extent().values():
            for segment_id in range(0, n_long_segments):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

        n_tran_segments = int(np.floor((n_long - 1) / 2))
        for member in self.transverse_psm_extent().values():
```

```
            for segment_id in range(0, n_tran_segments):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

    def identify_none_full_segments(self):
        """
        :return: Identifies Segment objects for full mesh generation;
            grillage with no axis of symmetry.
            Stores identified segments in full_segments dictionary.
        """
        n_long = self._grillage.N_longitudinal
        n_tran = self._grillage.N_transverse

        n = 1
        for member in self._grillage.longitudinal_members().values():
            for segment_id in range(0, n_tran - 1):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

        for member in self._grillage.transverse_members().values():
            for segment_id in range(0, n_long - 1):
                self.full_segments[n] = member.segments[segment_id]
                self.add_to_all_segments(member.segments[segment_id])
                n += 1

    def identify_long_half_segments(self):
        """
        :return: Identifies Segment objects for half mesh generation;
            grillage with a longitudinal axis of symmetry. Center transverse
            segment exists for even number of longitudinal members.
            Stores identified segments in half_segments dictionary.
        """
        n_long = self._grillage.N_longitudinal
        middle_segment_ID = int(np.ceil((n_long - 1) / 2))
        n = 1

        if np.mod(n_long, 2) == 0:
            for member in self._grillage.transverse_members().values():
                self.half_segments[n] = member.segments[middle_segment_ID - 1]
                self.add_to_all_segments(member.segments[middle_segment_ID - 1])

                n += 1

    def identify_tran_half_segments(self):
        """
        :return: Identifies Segment objects for half mesh generation;
            grillage with a transverse axis of symmetry. Center longitudinal
            segment exists for even number of transverse members.
            Stores identified segments in half_segments dictionary.
        """
        n_tran = self._grillage.N_transverse
        middle_segment_ID = int(np.ceil((n_tran - 1) / 2))
        n = 1

        if np.mod(n_tran, 2) == 0:
            for member in self._grillage.longitudinal_members().values():
                self.half_segments[n] = member.segments[middle_segment_ID - 1]
                self.add_to_all_segments(member.segments[middle_segment_ID - 1])
```

```
                n += 1

    def identify_both_half_segments(self):
        """
        :return: Identifies Segment objects for half mesh generation;
            grillage with both axis of symmetry.
            Stores identified segments in half_segments dictionary.
        """
        n_long = self._grillage.N_longitudinal
        n_tran = self._grillage.N_transverse
        long_segment_ID = int(np.ceil((n_tran - 1) / 2))
        tran_segment_ID = int(np.ceil((n_long - 1) / 2))

        n = 1
        if np.mod(n_tran, 2) == 0:
            for member in self.longitudinal_psm_extent().values():
                self.half_segments[n] = member.segments[long_segment_ID - 1]
                self.add_to_all_segments(member.segments[long_segment_ID - 1])
                n += 1

        if np.mod(n_long, 2) == 0:
            for member in self.transverse_psm_extent().values():
                self.half_segments[n] = member.segments[tran_segment_ID - 1]
                self.add_to_all_segments(member.segments[tran_segment_ID - 1])
                n += 1

    def identify_long_full_plate_zones(self):
        """
        :return: Identifies Plate objects for full mesh generation on the entire
            plating zone; grillage with a longitudinal axis of symm.
            Stores identified zones in full_plate_zones dictionary.
        """
        total_rows = self._grillage.N_longitudinal - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()

        n_full_rows = int(np.floor(total_rows / 2))
        zone_arr_split = plating_zone_array[0:n_full_rows, :]
        for i in zone_arr_split:
            for j in i:
                self.full_plate_zones[j] = self._grillage.plating()[j]
                self.all_plating_zones[j] = self._grillage.plating()[j]

    def identify_tran_full_plate_zones(self):
        """
        :return: Identifies Plate objects for full mesh generation on the entire
            plating zone; grillage with a transverse axis of symm.
            Stores identified zones in full_plate_zones dictionary.
        """
        total_columns = self._grillage.N_transverse - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()

        n_full_columns = int(np.floor(total_columns / 2))
        zone_arr_split = plating_zone_array[:, 0:n_full_columns]
        for i in zone_arr_split:
            for j in i:
                self.full_plate_zones[j] = self._grillage.plating()[j]
                self.all_plating_zones[j] = self._grillage.plating()[j]

    def identify_both_full_plate_zones(self):
        """
```

```
        :return: Identifies Plate objects for full mesh generation on the entire
            plating zone; grillage with both axis of symm.
            Stores identified zones in full_plate_zones dictionary.
        """
        total_rows = self._grillage.N_longitudinal - 1
        total_columns = self._grillage.N_transverse - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()

        mid_row_id = int(np.floor(total_rows / 2))
        mid_col_id = int(np.floor(total_columns / 2))
        zone_arr_split = plating_zone_array[0:mid_row_id, 0:mid_col_id]
        for i in zone_arr_split:
            for j in i:
                self.full_plate_zones[j] = self._grillage.plating()[j]
                self.all_plating_zones[j] = self._grillage.plating()[j]

    def identify_long_half_plate_zones(self):
        """
        :return: Identifies Plate objects for half mesh generation; plating
            zones split with a longitudinal axis of symmetry.
            Stores identified zones in long_half_plate_zones dictionary.
        """
        total_rows = self._grillage.N_longitudinal - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()

        if np.mod(self._grillage.N_longitudinal, 2) == 0:
            mid_row_id = int(np.floor(total_rows / 2))
            middle_row = plating_zone_array[mid_row_id, :]
            for i in middle_row:
                self.long_half_plate_zones[i] = self._grillage.plating()[i]
                self.all_plating_zones[i] = self._grillage.plating()[i]

    def identify_tran_half_plate_zones(self):
        """
        :return: Identifies Plate objects for half mesh generation; plating
            zones split with a transverse axis of symmetry.
            Stores identified zones in tran_half_plate_zones dictionary.
        """
        total_columns = self._grillage.N_transverse - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()

        if np.mod(self._grillage.N_transverse, 2) == 0:
            mid_col_id = int(np.floor(total_columns / 2))
            middle_column = plating_zone_array[:, mid_col_id]
            for i in middle_column:
                self.tran_half_plate_zones[i] = self._grillage.plating()[i]
                self.all_plating_zones[i] = self._grillage.plating()[i]

    def identify_both_half_plate_zones(self):
        """
        :return: Identifies Plate objects for half mesh generation; grillage
            with both axis of symm. Stores identified zones in
            long_half_plate_zones and tran_half_plate_zones dictionary.
        """
        total_rows = self._grillage.N_longitudinal - 1
        total_columns = self._grillage.N_transverse - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()
        mid_row_id = int(np.floor(total_rows / 2))
        mid_col_id = int(np.floor(total_columns / 2))

```

```
781         if np.mod(self._grillage.N_longitudinal, 2) == 0:
782             middle_row = plating_zone_array[mid_row_id, 0:mid_col_id]
783             for i in middle_row:
784                 self.long_half_plate_zones[i] = self._grillage.plating()[i]
785                 self.all_plating_zones[i] = self._grillage.plating()[i]
786 
787         if np.mod(self._grillage.N_transverse, 2) == 0:
788             middle_column = plating_zone_array[0:mid_row_id, mid_col_id]
789             for i in middle_column:
790                 self.tran_half_plate_zones[i] = self._grillage.plating()[i]
791                 self.all_plating_zones[i] = self._grillage.plating()[i]
792 
793     def identify_quarter_plate_zone(self):
794         """
795         :return: Identifies Plate object for quarter mesh generation;
796             plating zone split with both axis of symmetry.
797             Only possible if grillage has both axis of symmetry, with even
798             number of longitudinal and transverse primary supporting members.
799             There can be only one plating zone split with both axis of symmetry.
800             Identified zone is saved in quarter_plate_zone dictionary
801             for consistency with other methods.
802         """
803         if np.mod(self._grillage.N_longitudinal, 2) == 0 and \
804                 np.mod(self._grillage.N_transverse, 2) == 0:
805             total_rows = self._grillage.N_longitudinal - 1
806             total_columns = self._grillage.N_transverse - 1
807             plating_zone_array = self.hc_plate_zone_ref_ID_array()
808 
809             mid_row_id = int(np.floor(total_rows / 2))
810             mid_col_id = int(np.floor(total_columns / 2))
811             plateid = plating_zone_array[mid_row_id, mid_col_id]
812 
813             self.quarter_plate_zone[plateid] = self._grillage.plating()[plateid]
814             self.all_plating_zones[plateid] = self._grillage.plating()[plateid]
815 
816     def get_plate_dim(self, plate: Plate, plate_dim):
817         """
818         :param plate: Selected plating zone.
819         :param plate_dim: Full plating zone dimension.
820         :return: Method returns half of the full plating zone dimension if
821             selected plate is split by any axis of symmetry.
822         """
823         if plate.id in self.full_plate_zones:
824             return plate_dim
825 
826         elif plate.id in self.long_half_plate_zones or \
827                 plate.id in self.tran_half_plate_zones or \
828                 plate.id in self.quarter_plate_zone:
829             return plate_dim / 2
830 
831     def get_long_plate_dim(self, plate: Plate):
832         """
833         :param plate: Selected plating zone.
834         :return: Method returns longitudinal dimension of the selected plating
835             zone.Returns half of the original value if the zone is split by
836             transverse axis of symmetry.
837         """
838         if plate.id in self.full_plate_zones or \
839                 plate.id in self.long_half_plate_zones:
840             return plate.plate_longitudinal_dim() * 1000
```

```
841
842         elif plate.id in self.tran_half_plate_zones or \
843                 plate.id in self.quarter_plate_zone:
844             return (plate.plate_longitudinal_dim() / 2) * 1000
845
846     def get_tran_plate_dim(self, plate: Plate):
847         """
848         :param plate: Selected plating zone.
849         :return: Method returns transverse dimension of the selected plating
850             zone.Returns half of the original value if the zone is split by
851             longitudinal axis of symmetry.
852         """
853         if plate.id in self.full_plate_zones or \
854                 plate.id in self.tran_half_plate_zones:
855             return plate.plate_transverse_dim() * 1000
856
857         elif plate.id in self.long_half_plate_zones or \
858                 plate.id in self.quarter_plate_zone:
859             return (plate.plate_transverse_dim() / 2) * 1000
860
861     def aos_between_stiffeners(self, plate: Plate):
862         """
863         :param plate: Selected plating zone.
864         :return: True if some axis of symmetry passes between stiffeners on the
865             selected plating zone. False return does not indicate stiffener is
866             located on some axis of symmetry! If a new stiffener layout
867             definition type would be created with nonsymmetric stiffener
868             placement, this method needs to be modified.
869         """
870         stiff_num = plate.get_stiffener_number()
871         if (plate.id in self.long_half_plate_zones or
872             plate.id in self.quarter_plate_zone) and \
873                 np.mod(stiff_num, 2) == 0 and \
874                 plate.stiff_dir is BeamDirection.LONGITUDINAL:
875             return True
876
877         elif (plate.id in self.tran_half_plate_zones or
878               plate.id in self.quarter_plate_zone) and \
879                 np.mod(stiff_num, 2) == 0 and \
880                 plate.stiff_dir is BeamDirection.TRANSVERSE:
881             return True
882         else:
883             return False
884
885     def aos_on_stiffener(self, plate: Plate):
886         """
887         :param plate: Selected plating zone.
888         :return: True if some axis of symmetry is located on and parallel to a
889             stiffener on the selected plating zone. False return does not
890             indicate axis of symmetry passes between stiffeners! If a new
891             stiffener layout definition type would be created with nonsymmetric
892             stiffener placement, this method needs to be modified.
893         """
894         stiff_num = plate.get_stiffener_number()
895         if (plate.id in self.long_half_plate_zones or
896             plate.id in self.quarter_plate_zone) and \
897                 np.mod(stiff_num, 2) != 0 and \
898                 plate.stiff_dir is BeamDirection.LONGITUDINAL:
899             return True
900
```

```
        elif (plate.id in self.tran_half_plate_zones or
              plate.id in self.quarter_plate_zone) and \
                np.mod(stiff_num, 2) != 0 and \
                plate.stiff_dir is BeamDirection.TRANSVERSE:
            return True
        else:
            return False

    def aos_on_segment(self, segment: Segment):
        """
        :param segment: Segment of a primary supporting member.
        :return: True if some axis of symmetry is located on and parallel to a
            segment. Used to identify which segments should be assigned half of
            their original web thickness and half of their flange.
        """
        rel_dist = segment.primary_supp_mem.rel_dist
        direction = segment.primary_supp_mem.direction
        aos = self.axis_of_symm
        if (aos is AOS.LONGITUDINAL or aos is AOS.BOTH) and \
                direction is BeamDirection.LONGITUDINAL and \
                np.isclose(rel_dist, 0.5):
            return True

        elif (aos is AOS.TRANSVERSE or aos is AOS.BOTH) and \
                direction is BeamDirection.TRANSVERSE and \
                np.isclose(rel_dist, 0.5):
            return True
        else:
            return False

    def identify_tran_split_zones(self):
        """
        :return: Identifies Plate objects where transverse axis of symmetry
            passes between stiffeners. Stores all identified zones located on
            the column of plating zones into tran_e_split_zone dictionary.
        """
        plating_zone_array = self.hc_plate_zone_ref_ID_array()
        for plate in self.all_plating_zones.values():
            if self.aos_between_stiffeners(plate) \
                    and plate.stiff_dir is BeamDirection.TRANSVERSE:
                column_id = np.where(plating_zone_array == plate.id)[1]
                split_element_zones = plating_zone_array[:, column_id]
                for i in split_element_zones:
                    for j in i:
                        self.tran_e_split_zone[j] = self._grillage.plating()[j]

    def identify_long_split_zones(self):
        """
        :return: Identifies Plate objects where longitudinal axis of symmetry
            passes between stiffeners. Stores all identified zones located on
            the row of plating zones into long_e_split_zone dictionary.
        """
        plating_zone_array = self.hc_plate_zone_ref_ID_array()
        for plate in self.all_plating_zones.values():
            if self.aos_between_stiffeners(plate) \
                    and plate.stiff_dir is BeamDirection.LONGITUDINAL:
                row_id = np.where(plating_zone_array == plate.id)[0]
                split_element_zones = plating_zone_array[row_id, :]
                for i in split_element_zones:
                    for j in i:
```

```
                        self.long_e_split_zone[j] = self._grillage.plating()[j]

    def longitudinal_symm_plate_ref_array(self):
        """
        :return: 2D array of plating zone IDs to be meshed for longitudinal
            Axis of Symmetry, arranged to represent their relative placement
            on the grillage model.
        """
        total_rows = self._grillage.N_longitudinal - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()
        mid_row_id = int(np.ceil(total_rows / 2))
        self.plating_zones_ref_array = plating_zone_array[0:mid_row_id, :]

    def transverse_symm_plate_ref_array(self):
        """
        :return: 2D array of plating zone IDs to be meshed for transverse
            Axis of Symmetry, arranged to represent their relative placement
            on the grillage model.
        """
        total_columns = self._grillage.N_transverse - 1
        plating_zone_array = self.hc_plate_zone_ref_ID_array()
        mid_col_id = int(np.ceil(total_columns / 2))
        self.plating_zones_ref_array = plating_zone_array[:, 0:mid_col_id]

    def both_symm_plate_ref_array(self):
        """
        :return: 2D array of plating zone IDs to be meshed for both longitudinal
            and transverse Axis of Symmetry, arranged to represent their
            relative placement on the grillage model.
        """
        total_rows = self._grillage.N_longitudinal - 1
        total_columns = self._grillage.N_transverse - 1
        ref_array = self.hc_plate_zone_ref_ID_array()
        mid_row_id = int(np.ceil(total_rows / 2))
        mid_col_id = int(np.ceil(total_columns / 2))
        self.plating_zones_ref_array = ref_array[0:mid_row_id, 0:mid_col_id]

    def grillage_plate_extent(self):
        """
        :return: Determines limits of plate mesh generation based on input
            Axis of Symmetry value. Calls specific methods for identifying which
            plating zones will be fully or partially meshed. If grillage has no
            axis of symmetry, all plating zones inside grillage.plating() will
            be meshed. Saves a reference ID array of all plating zones to be
            meshed into plating_zones_ref_array.
        """
        if self.axis_of_symm is AOS.LONGITUDINAL:
            self.identify_long_full_plate_zones()
            self.identify_long_half_plate_zones()
            self.longitudinal_symm_plate_ref_array()
            self.identify_long_split_zones()

        elif self.axis_of_symm is AOS.TRANSVERSE:
            self.identify_tran_full_plate_zones()
            self.identify_tran_half_plate_zones()
            self.transverse_symm_plate_ref_array()
            self.identify_tran_split_zones()

        elif self.axis_of_symm is AOS.BOTH:
            self.identify_both_full_plate_zones()
```

```
            self.identify_both_half_plate_zones()
            self.identify_quarter_plate_zone()
            self.both_symm_plate_ref_array()
            self.identify_long_split_zones()
            self.identify_tran_split_zones()

        else:
            self.full_plate_zones = self._grillage.plating()
            self.all_plating_zones = self._grillage.plating()
            self.plating_zones_ref_array = self.hc_plate_zone_ref_ID_array()

    def add_to_all_segments(self, segment):
        self._all_segment_count += 1
        self.all_segments[self._all_segment_count] = segment

    def grillage_segment_extent(self):
        """
        :return: Determines limits of segment mesh generation based on selected
            Axis of Symmetry. Calls specific methods for identifying which
            segments will be fully or partially meshed. If grillage has no axis
            of symmetry, all segments on the grillage model will be meshed.
        """
        if self.axis_of_symm is AOS.LONGITUDINAL:
            self.identify_long_full_segments()
            self.identify_long_half_segments()

        elif self.axis_of_symm is AOS.TRANSVERSE:
            self.identify_tran_full_segments()
            self.identify_tran_half_segments()

        elif self.axis_of_symm is AOS.BOTH:
            self.identify_both_full_segments()
            self.identify_both_half_segments()

        else:
            self.identify_none_full_segments()

    def add_unique_plate_prop(self, prop: PlateProperty, fem: GeoGrillageFEM):
        """
        Method adds plating zone plate finite element property.
        Checks for duplicate PlateProperty ID in plate_property_IDs.
        If no duplicate exists, a new GeoFEM plate property is created.
        :param prop: Grillage model PlateProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.plate_property_IDs.keys():
            tp = prop.tp_net(corr_add, prop.tp)
            mat_id = prop.plate_mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_plate_property(gfe_prop_id, tp, mat)
            fem.plate_property_IDs[prop.id] = gfe_prop_id

    def add_unique_web_prop(self, prop: TBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds segment web plate finite element property.
        Checks for duplicate BeamProperty ID in web_property_IDs.
        If no duplicate exists, a new GeoFEM plate property is created.
        :param prop: Grillage model TBeamProperty object.
```

```
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.web_property_IDs.keys():
            tw = prop.tw_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_plate_property(gfe_prop_id, tw, mat)
            fem.web_property_IDs[prop.id] = gfe_prop_id

    def add_unique_half_web_prop(self, prop: TBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds segment web plate finite element property with half tw.
        Checks for duplicate BeamProperty ID in half_web_property_IDs.
        If no duplicate exists, a new GeoFEM plate property is created.
        :param prop: Grillage model TBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_web_property_IDs.keys():
            tw = prop.tw_net(corr_add) / 2
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_plate_property(gfe_prop_id, tw, mat)
            fem.half_web_property_IDs[prop.id] = gfe_prop_id

    def add_unique_flange_prop(self, prop: TBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds segment flange plate finite element property.
        Checks for duplicate BeamProperty ID in flange_property_IDs.
        If no duplicate exists, a new GeoFEM plate property is created.
        :param prop: Grillage model TBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.flange_property_IDs.keys():
            tf = prop.tf_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_plate_property(gfe_prop_id, tf, mat)
            fem.flange_property_IDs[prop.id] = gfe_prop_id

    def add_unique_T_beam_prop(self, prop: TBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model TBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.stiff_beam_prop_IDs.keys():
            name_str = "T_" + str(prop.hw) + "x" + str(prop.tw)
            name_str += "/" + str(prop.bf) + "x" + str(prop.tf)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
```

```
            bf = prop.bf_net(corr_add)
            tf = prop.tf_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_T_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_half_T_beam_prop(self, prop: TBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model TBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_stiff_beam_prop_IDs.keys():
            name_str = "T_" + str(prop.hw) + "x" + str(prop.tw)
            name_str += "/" + str(prop.bf) + "x" + str(prop.tf)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
            bf = prop.bf_net(corr_add)
            tf = prop.tf_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_half_T_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.half_stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_L_beam_prop(self, prop: LBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model LBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.stiff_beam_prop_IDs.keys():
            name_str = "L_" + str(prop.hw) + "x" + str(prop.tw)
            name_str += "/" + str(prop.bf) + "x" + str(prop.tf)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
            bf = prop.bf_net(corr_add)
            tf = prop.tf_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_L_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_half_L_beam_prop(self, prop: LBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model LBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
```

```
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_stiff_beam_prop_IDs.keys():
            name_str = "L_" + str(prop.hw) + "x" + str(prop.tw)
            name_str += "/" + str(prop.bf) + "x" + str(prop.tf)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
            bf = prop.bf_net(corr_add)
            tf = prop.tf_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_half_L_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.half_stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_FB_beam_prop(self, prop: FBBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model FBBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.stiff_beam_prop_IDs.keys():
            name_str = "FB_" + str(prop.hw) + "x" + str(prop.tw)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_FB_beam_property(name_str, hw, tw, mat)
            fem.stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_half_FB_beam_prop(self, prop: FBBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model FBBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_stiff_beam_prop_IDs.keys():
            name_str = "FB_" + str(prop.hw) + "x" + str(prop.tw)
            hw = prop.hw_net(corr_add, 0.0)
            tw = prop.tw_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_half_FB_beam_property(name_str, hw, tw, mat)
            fem.half_stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_Bulb_beam_prop(self, prop: BulbBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model BulbBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
```

```
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.stiff_beam_prop_IDs.keys():
            name_str = "HP_" + str(prop.hw_HP) + "x" + str(prop.tw_HP)
            hw = prop.hw_ekv_net(corr_add)
            tw = prop.tw_ekv_net(corr_add)
            bf = prop.bf_ekv_net(corr_add)
            tf = prop.tf_ekv_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_Bulb_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_half_Bulb_beam_prop(self, prop: BulbBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model BulbBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_stiff_beam_prop_IDs.keys():
            name_str = "HP_" + str(prop.hw_HP) + "x" + str(prop.tw_HP)
            hw = prop.hw_ekv_net(corr_add)
            tw = prop.tw_ekv_net(corr_add)
            bf = prop.bf_ekv_net(corr_add)
            tf = prop.tf_ekv_net(corr_add)
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_half_Bulb_beam_property(name_str, hw, tw, bf, tf, mat)
            fem.half_stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_Hat_beam_prop(self, prop: HatBeamProperty, fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model HatBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.stiff_beam_prop_IDs.keys():
            name_str = "Hat_" + str(prop.h) + "x" + str(prop.t)
            name_str += "x" + str(prop.bf) + "x" + str(prop.fi) + "°"
            h = prop.h_net(corr_add)
            t = prop.t_net(corr_add)
            bf = prop.bf_net(corr_add)
            fi = prop.fi
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_Hat_beam_property(name_str, h, t, bf, fi, mat)
            fem.stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def add_unique_half_Hat_beam_prop(self, prop: HatBeamProperty,
                                      fem: GeoGrillageFEM):
        """
        Method adds stiffener layout beam element property.
        Checks for duplicate BeamProperty ID in stiff_beam_prop_IDs.
```

```
        If no duplicate exists, a new GeoFEM Beam property is created.
        :param prop: Grillage model HatBeamProperty object.
        :param fem: Grillage FEM model.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        gfe_prop_id = fem.id_prop_count
        if prop.id not in fem.half_stiff_beam_prop_IDs.keys():
            name_str = "Hat_" + str(prop.h) + "x" + str(prop.t)
            name_str += "x" + str(prop.bf) + "x" + str(prop.fi) + "°"
            h = prop.h_net(corr_add)
            t = prop.t_net(corr_add)
            bf = prop.bf_net(corr_add)
            fi = prop.fi
            mat_id = prop.mat.id
            mat = fem.getMaterial(mat_id)
            fem.add_half_Hat_beam_property(name_str, h, t, bf, fi, mat)
            fem.half_stiff_beam_prop_IDs[prop.id] = gfe_prop_id

    def generate_FEM_material(self, fem: GeoGrillageFEM):
        """
        :return: Generates GeoFEM materials from grillage model materials.
        """
        for mat in self._grillage.material_props().values():
            fem.add_material(mat)

    def generate_FEM_plate_property(self, fem: GeoGrillageFEM):
        """
        :return: Generates GeoFEM plate property from grillage BeamProperty.
        """
        for plate in self.all_plating_zones.values():
            self.add_unique_plate_prop(plate.plate_prop, fem)

        for segment in self.all_segments.values():
            beam_prop = segment.beam_prop
            beam_type = beam_prop.beam_type

            if self.aos_on_segment(segment):
                self.add_unique_half_web_prop(beam_prop, fem)
            else:
                self.add_unique_web_prop(beam_prop, fem)

            if beam_type is BeamType.T or beam_type is BeamType.L:
                self.add_unique_flange_prop(beam_prop, fem)

    def generate_FEM_beam_property(self, fem: GeoGrillageFEM):
        """
        :return: Generates GeoFEM beam property from grillage BeamProperty.
            Does not recognize stiffeners on Axis Of Symmetry. Stiffener on
            AOS identification is done at the moment of beam element generation.
        """
        for plate in self.all_plating_zones.values():
            stiff_prop = plate.stiff_layout.beam_prop

            if stiff_prop.beam_type is BeamType.T:
                self.add_unique_T_beam_prop(stiff_prop, fem)

            elif stiff_prop.beam_type is BeamType.L:
                self.add_unique_L_beam_prop(stiff_prop, fem)

            elif stiff_prop.beam_type is BeamType.FB:
```

```
                self.add_unique_FB_beam_prop(stiff_prop, fem)

            elif stiff_prop.beam_type is BeamType.Bulb:
                self.add_unique_Bulb_beam_prop(stiff_prop, fem)

            elif stiff_prop.beam_type is BeamType.Hat:
                self.add_unique_Hat_beam_prop(stiff_prop, fem)

    def generate_half_FEM_beam_property(self, fem: GeoGrillageFEM):
        for plate in self.all_plating_zones.values():
            stiff_prop = plate.stiff_layout.beam_prop
            if self.aos_on_stiffener(plate):
                if stiff_prop.beam_type is BeamType.T:
                    self.add_unique_half_T_beam_prop(stiff_prop, fem)

                elif stiff_prop.beam_type is BeamType.L:
                    self.add_unique_half_L_beam_prop(stiff_prop, fem)

                elif stiff_prop.beam_type is BeamType.FB:
                    self.add_unique_half_FB_beam_prop(stiff_prop, fem)

                elif stiff_prop.beam_type is BeamType.Bulb:
                    self.add_unique_half_Bulb_beam_prop(stiff_prop, fem)

                elif stiff_prop.beam_type is BeamType.Hat:
                    self.add_unique_half_Hat_beam_prop(stiff_prop, fem)

    def grillage_mesh_extent(self):
        """
        :return: Determines limits of grillage mesh generation for plating zones
            and segments based on Axis of Symmetry value. Method stops mesh
             generation if the model does not pass the feasibility test.
        """
        self.model_check.mesh_feasibility()
        self.grillage_plate_extent()
        self.grillage_segment_extent()

    def identify_both_psm_ends(self):
        """
        Identifies coordinates for pinned boundary conditions on both ends of
        all Primary Supporting Members and returns list of end_coords.
        """
        symmetry = self.axis_of_symm
        end_coords = []

        for member in self.longitudinal_psm_extent().values():
            end1_coords, end2_coords = member.end_nodes
            if symmetry is AOS.TRANSVERSE or symmetry is AOS.BOTH:
                end_coords.append(end1_coords * [1, 1, 0])
            else:
                end_coords.append(end1_coords * [1, 1, 0])
                end_coords.append(end2_coords * [1, 1, 0])

        for member in self.transverse_psm_extent().values():
            end1_coords, end2_coords = member.end_nodes
            if symmetry is AOS.LONGITUDINAL or symmetry is AOS.BOTH:
                end_coords.append(end1_coords * [1, 1, 0])
            else:
                end_coords.append(end1_coords * [1, 1, 0])
                end_coords.append(end2_coords * [1, 1, 0])
```

```
        return end_coords


class MeshSize:
    def __init__(self, mesh_extent: MeshExtent,
                 min_num_ebs: int,
                 min_num_eweb: int,
                 num_eaf: int,
                 flange_aspect_ratio: float,
                 plate_aspect_ratio: float,
                 des_plate_aspect_ratio: float):
        """
        Class for calculating finite element dimensions on the selected
        grillage model.

        Final result of the following methods are distances between edge nodes
        of all structural elements, along both x and y axis, which will be used
        for all node and element generation.

        Mesh control parameters:
        min_num_ebs - Minimum number of elements between stiffeners; default = 1
        min_num_eweb - Minimum number of elements representing the web of a
            primary supporting member along its height; default = 3
        num_eaf - Number of elements across primary supporting member flange;
            default = 1
        flange_aspect_ratio - Maximum aspect ratio value for primary supporting
            member flange quad elements; default = 8
        plate_aspect_ratio - Maximum aspect ratio value for plating and primary
            supporting member web quad elements; default = 3
        des_plate_aspect_ratio - Desirable plating aspect ratio value less than
            the maximum; default = 2

        Mesh dimensions are saved into:
        mesh_dim_x - Final base mesh x dimensions (dim_x) for each column of
            plating zones on the grillage model
        mesh_dim_y - Final base mesh y dimensions (dim_y) for each row of
            plating zones on the grillage model

        :param mesh_extent: FE mesh extents for the selected grillage model
            and Axis of Symmetry.
        """
        self._mesh_extent = mesh_extent

        self._grillage = self._mesh_extent.grillage
        self._axis_of_symm = self._mesh_extent.axis_of_symm

        self._min_num_ebs = min_num_ebs
        self._min_num_eweb = min_num_eweb
        self._num_eaf = num_eaf
        self._flange_aspect_ratio = flange_aspect_ratio
        self._plate_aspect_ratio = plate_aspect_ratio
        self._des_plate_aspect_ratio = des_plate_aspect_ratio

        self._mesh_dim_x = {}
        self._mesh_dim_y = {}

        self.start_nodes = {}

    @property
    def mesh_extent(self):
```

```
        return self._mesh_extent

    @property
    def min_num_ebs(self):
        return self._min_num_ebs

    @min_num_ebs.setter
    def min_num_ebs(self, value):
        self._min_num_ebs = value

    @property
    def min_num_eweb(self):
        return self._min_num_eweb

    @min_num_eweb.setter
    def min_num_eweb(self, value):
        self._min_num_eweb = value

    @property
    def num_eaf(self):
        return self._num_eaf

    @num_eaf.setter
    def num_eaf(self, value):
        self._num_eaf = value

    @property
    def flange_aspect_ratio(self):
        return self._flange_aspect_ratio

    @flange_aspect_ratio.setter
    def flange_aspect_ratio(self, value):
        self._flange_aspect_ratio = value

    @property
    def plate_aspect_ratio(self):
        return self._plate_aspect_ratio

    @plate_aspect_ratio.setter
    def plate_aspect_ratio(self, value):
        self._plate_aspect_ratio = value

    @property
    def des_plate_aspect_ratio(self):
        return self._des_plate_aspect_ratio

    @property
    def mesh_dim_x(self):
        return self._mesh_dim_x

    @mesh_dim_x.setter
    def mesh_dim_x(self, value):
        self._mesh_dim_x = value

    @property
    def mesh_dim_y(self):
        return self._mesh_dim_y

    @mesh_dim_y.setter
    def mesh_dim_y(self, value):
```

```
        self._mesh_dim_y = value

    @des_plate_aspect_ratio.setter
    def des_plate_aspect_ratio(self, value):
        self._des_plate_aspect_ratio = value
        if value > self._plate_aspect_ratio:
            raise InvalidDesiredAspectRatio(value, self._plate_aspect_ratio)

    @staticmethod
    def save_node_spacing(dictionary: dict, n_dims, dimension):
        """
        Method used for saving edge node spacing dimensions to a dictionary,
        for any number of dimensions of equal value.

        :param dictionary: Dictionary for all node spacing (element) dimensions
            on a plating zone.
        :param n_dims: Number of dimensions to be stored in the Dictionary.
        :param dimension: Element dimension to be stored.
        """
        if dictionary:  # If the dictionary is not empty
            dimension_id = len(dictionary) + 1
        else:
            dimension_id = 1

        end = int(dimension_id + n_dims)
        for element in range(dimension_id, end):
            dictionary[dimension_id] = dimension
            dimension_id += 1

    @staticmethod
    def find_closest_divisor(length: int, spacing: int):
        """
        Method can be used as a separate calculator for equal stiffener spacing
        calculation along some length L and approximate desired spacing.

        :param length: Length L which should be divided into n equal parts, each
            with length x.
        :param spacing: Value to which length x should be closes to.
        :return: Closest divisor of length L, which results in a length x
            closest to given spacing value. If a divisor does not exist,
            this method returns None.
        """
        if np.mod(length, spacing) == 0:
            return length / spacing

        else:
            i = 1
            res = []
            while i <= length:
                if np.mod(length, i) == 0:
                    res.append(i)
                i += 1

            min_diff = spacing
            min_div_id = 1

            if not res:
                return None
            else:
                for i in range(0, len(res)):
```

```
                    if min_diff > abs((length / res[i]) - spacing):
                        min_diff = abs((length / res[i]) - spacing)
                        min_div_id = i
                n = res[min_div_id]
                return n

    @staticmethod
    def find_largest_divisor(length, max_val):
        """
        :param length: Length L which should be divided into n equal parts,
            each with length x.
        :param max_val: Maximum value which length x can not exceed.
        :return: Largest divisor of length L, which results in a length x
            smaller or equal to the given maximum value. If a divisor does not
            exist, such as when length L is decimal, this method returns None.
        """
        i = 1
        divisors = []
        while i <= length:
            if np.mod(length, i) == 0:
                divisors.append(i)
            i += 1

        curr_x_max = 0  # Current maximum value of x
        n = None

        if not divisors:
            return None
        else:  # If a divisor exists
            for i in divisors:
                curr_x = length / i  # Current dimension x value
                if curr_x_max < curr_x <= max_val:
                    curr_x_max = curr_x
                    n = i
            return n

    @staticmethod
    def element_aspect_ratio(dim_x, dim_y):
        """
        :param dim_x: Dimension of the quad element along x axis.
        :param dim_y: Dimension of the quad element along y axis.
        :return: Aspect ratio of the quad element.
        """
        if dim_x > dim_y != 0:
            ar = dim_x / dim_y
        elif dim_x < dim_y and dim_x != 0:
            ar = dim_y / dim_x
        else:
            ar = 1
        return ar

    @staticmethod
    def refine_plate_element(length, dim_limit):
        """
        :param length: Dimension which is to be equally divided.
        :param dim_limit: Maximum dimension allowed for the element along the
            given length, defined by aspect ratio or other limits.
        :return: Element dimension value that is less than the maximum allowed
            and equally divides given length.
        """
```

```
        if dim_limit == 0:
            return length
        else:
            n_elements = np.ceil(length / dim_limit)
            dim = length / n_elements
            return dim

    def element_size_perp_to_stiffeners(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Quad element dimension perpendicular to stiffener direction
            based only on number of elements between stiffeners, in [mm].
        """
        stiff_spacing = plate.get_stiffener_spacing() * 1000
        perp_dim = stiff_spacing / self._min_num_ebs
        return perp_dim

    def element_size_para_to_stiffeners(self, plate: Plate, plate_dim):
        """
        :param plate: Selected plating zone.
        :param plate_dim: Plating zone dimension parallel to stiffener
            direction, takes into account Axis of Symmetry.
        :return: Quad element dimension parlallel to stiffener direction, based
            on quad element dimension perpendicular to stiffener direction and
            desired aspect ratio.
        """
        y = self.element_size_perp_to_stiffeners(plate)
        des_x_val = y * self._des_plate_aspect_ratio  # Desired element dim
        x = self.refine_plate_element(plate_dim, des_x_val)
        return x

    def get_web_el_height(self, segment: Segment):
        """
        :param segment: Segment of a primary supporting member.
        :return: Quad element dimension along the height of a primary
            supporting member.
        """
        hw = segment.beam_prop.hw
        dim = hw / self._min_num_eweb
        return dim

    def get_flange_el_width(self, segment: Segment):
        """
        :param segment: Segment of a primary supporting member.
        :return: Flange quad element dimension across the width of the flange
            of the selected segment (perpendicular to the segment direction)
            based on the number of elements across the flange. For longitudinal
            segments this method returns dimension dim_yf and for transverse
            segments returns dimension dim_xf.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        bf_net = TBeamProperty.bf_net(segment.beam_prop, corr_add)

        if segment.beam_prop.beam_type is BeamType.T:
            return bf_net / (self._num_eaf * 2)

        elif segment.beam_prop.beam_type is BeamType.L:
            return bf_net / self._num_eaf

        elif segment.beam_prop.beam_type is BeamType.FB:
```

```
            return 0

    def get_end1_max_flange_width(self, segment: Segment):
        beam_type = segment.beam_prop.beam_type
        corr_add = self._grillage.corrosion_addition()[1]
        eaf = self._num_eaf
        if beam_type is BeamType.T:
            self_bf_net = segment.beam_prop.bf_net(corr_add) / (eaf + 1)
        elif beam_type is BeamType.L:
            self_bf_net = segment.beam_prop.bf_net(corr_add) / eaf
        else:
            self_bf_net = 0

        seg_end1 = self._grillage.get_parallel_segments(segment)[0]

        if seg_end1:
            end1_type = seg_end1.beam_prop.beam_type
            if end1_type is BeamType.T:
                end1_bf_net = seg_end1.beam_prop.bf_net(corr_add) / (eaf + 1)
            elif end1_type is BeamType.L:
                end1_bf_net = seg_end1.beam_prop.bf_net(corr_add) / eaf
            else:
                end1_bf_net = 0
        else:
            end1_bf_net = 0

        bf_net = np.amax([self_bf_net, end1_bf_net])
        return bf_net

    def get_end2_max_flange_width(self, segment: Segment):
        beam_type = segment.beam_prop.beam_type
        corr_add = self._grillage.corrosion_addition()[1]
        eaf = self._num_eaf
        if beam_type is BeamType.T:
            self_bf_net = segment.beam_prop.bf_net(corr_add) / (eaf + 1)
        elif beam_type is BeamType.L:
            self_bf_net = segment.beam_prop.bf_net(corr_add) / eaf
        else:
            self_bf_net = 0

        seg_end2 = self._grillage.get_parallel_segments(segment)[1]

        if seg_end2:
            end2_type = seg_end2.beam_prop.beam_type
            if end2_type is BeamType.T:
                end2_bf_net = seg_end2.beam_prop.bf_net(corr_add) / (eaf + 1)
            elif end2_type is BeamType.L:
                end2_bf_net = seg_end2.beam_prop.bf_net(corr_add) / eaf
            else:
                end2_bf_net = 0
        else:
            end2_bf_net = 0

        bf_net = np.amax([self_bf_net, end2_bf_net])
        return bf_net

    def get_flange_el_length(self, segment: Segment):
        """
        :param segment: Segment of a primary supporting member.
        :return: Maximum flange quad element length based on flange element
```

```
            width and maximum flange aspect ratio (parallel to the segment
            direction). For longitudinal segments this method returns dimension
            dim_xf, and for transverse segments returns dimension dim_yf.
        """
        if self.get_flange_el_width(segment) != 0:
            return self._flange_aspect_ratio * self.get_flange_el_width(segment)
        else:
            return 0

    def get_min_fl_el_len(self, segment1: Segment, segment2: Segment):
        """
        :param segment1: First selected segment.
        :param segment2: Second selected segment.
        :return: Minimum value of two segment flange element lengths.
            If both segments do not have a flange, method returns 0.
        """
        dim_f1 = self.get_flange_el_length(segment1)
        dim_f2 = self.get_flange_el_length(segment2)

        if dim_f1 != 0 and dim_f2 != 0:
            min_dim = np.minimum(dim_f1, dim_f2)
        elif dim_f1 != 0 and dim_f2 == 0:
            min_dim = dim_f1
        elif dim_f1 == 0 and dim_f2 != 0:
            min_dim = dim_f2
        else:
            min_dim = 0

        return min_dim

    def get_min_fl_el_len_between_psm(self, member1: PrimarySuppMem,
                                      member2: PrimarySuppMem):
        """
        :param member1: First selected Primary supporting member.
        :param member2: Second selected Primary supporting member.
        :return: Method identifies all segments between two adjacent primary
            supporting members and returns the minimum flange element length of
            all identified segments. If all flange element length values are 0,
            the el_length_list will be empty and this method will return 0.
        """
        segments_list = self._grillage.segments_between_psm(member1, member2)
        el_length_list = [self.get_flange_el_length(segment)
                          for segment in segments_list
                          if self.get_flange_el_length(segment) != 0]

        if not el_length_list:
            return 0
        else:
            return np.amin(el_length_list)

    def element_size_plating_zone_perp(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Quad element dimension perpendicular to stiffener direction
            based on two criteria:
                1.) Number of elements between stiffeners
                2.) Maximum element dimension
        """
        y = self.element_size_perp_to_stiffeners(plate)

```

```
        if plate.stiff_dir is BeamDirection.LONGITUDINAL:
            max_dim = self.get_min_fl_el_len(plate.trans_seg1, plate.trans_seg2)
        else:
            max_dim = self.get_min_fl_el_len(plate.long_seg1, plate.long_seg2)

        if y > max_dim != 0:
            stiff_spacing = plate.get_stiffener_spacing() * 1000
            y = self.refine_plate_element(stiff_spacing, max_dim)
        return y

    def element_size_plating_zone_para(self, plate: Plate, plate_dim):
        """
        :param plate: Selected plating zone.
        :param plate_dim: Plating zone dimension parallel to stiffener direction.
        :return: Quad element dimension parallel to stiffener direction based
            on two criteria:
                1.) Element dimensiom: method element_size_para_to_stiffeners
                2.) Maximum element dimension: method get_min_flange_el_length
        """
        x = self.element_size_para_to_stiffeners(plate, plate_dim)
        if plate.stiff_dir is BeamDirection.LONGITUDINAL:
            max_dim = self.get_min_fl_el_len(plate.long_seg1, plate.long_seg2)
        else:
            max_dim = self.get_min_fl_el_len(plate.trans_seg1, plate.trans_seg2)

        if x > max_dim != 0:
            x = self.refine_plate_element(plate_dim, max_dim)
        return x

    def element_size_plating_zone(self, plate: Plate, plate_dim):
        """
        Method for local consideration of base mesh dimensions dim_x and dim_y,
            for each plating zone individually.

        :param plate: Selected plating zone.
        :param plate_dim: Plating zone dimension parallel to stiffener
            direction. For MeshV1 this value should be calculated using method
            get_reduced_plate_dim and for others using method
            plate_dim_parallel_to_stiffeners.
        :return: Base mesh dimensions x and y that are below the maximum values
            based on plate aspect ratio and flange dimensions,
            considering each plating zone individually.
        """
        if plate.stiff_dir is BeamDirection.LONGITUDINAL:
            dim_x = self.element_size_plating_zone_para(plate, plate_dim)
            dim_y = self.element_size_plating_zone_perp(plate)
        else:
            dim_x = self.element_size_plating_zone_perp(plate)
            dim_y = self.element_size_plating_zone_para(plate, plate_dim)

        if self.element_aspect_ratio(dim_x, dim_y) > self._plate_aspect_ratio:
            dim_xf = self.get_min_fl_el_len(plate.long_seg1, plate.long_seg2)
            dim_yf = self.get_min_fl_el_len(plate.trans_seg1, plate.trans_seg2)

            dim_x_limit = np.minimum(dim_xf, dim_y * self._plate_aspect_ratio)
            dim_y_limit = np.minimum(dim_yf, dim_x * self._plate_aspect_ratio)

            if plate.stiff_dir is BeamDirection.LONGITUDINAL:
                if dim_y > dim_x:
                    stiff_spacing = plate.get_stiffener_spacing() * 1000
```

```
                    dim_y = self.refine_plate_element(stiff_spacing, dim_y_limit)
                else:
                    dim_x = self.refine_plate_element(plate_dim, dim_x_limit)

            elif plate.stiff_dir is BeamDirection.TRANSVERSE:
                if dim_y > dim_x:
                    dim_y = self.refine_plate_element(plate_dim, dim_y_limit)
                else:
                    stiff_spacing = plate.get_stiffener_spacing() * 1000
                    dim_x = self.refine_plate_element(stiff_spacing, dim_x_limit)

        return dim_x, dim_y

    def calc_element_base_size(self):
        """
        :return: Calculates the quad element size based on stiffener spacing
            and maximum allowed aspect ratio for all plating zones.
            Returns dictionaries of x (dim_x) and y (dim_y) base dimensions
            for all plating zones.
        """
        mesh_dim_x = {}  # Dimension x for all plating zones
        mesh_dim_y = {}  # Dimension y for all plating zones

        for plate in self._mesh_extent.all_plating_zones.values():
            parallel_dim = plate.plate_dim_parallel_to_stiffeners() * 1000
            plate_dim = self._mesh_extent.get_plate_dim(plate, parallel_dim)
            dim_x, dim_y = self.element_size_plating_zone(plate, plate_dim)
            mesh_dim_x[plate.id] = dim_x
            mesh_dim_y[plate.id] = dim_y

        return mesh_dim_x, mesh_dim_y

    def assign_base_dim_x(self, mesh_dim_x):
        """
        Method for global consideration of base mesh dimensions dim_x,
            for each column of plating zones on the grillage model.

        :param mesh_dim_x: Input dictionary of quad element x dimensions based
            on stiffener spacing and maximum allowed aspect ratio.
        :return: Assigns dimension x for each column of plating zones between
            transverse primary supporting members, identified using plating
            zones reference array based on Axis of Symmetry input.
            If there are no transverse stiffeners on the column of plating
            zones, a minimum value of all saved dim_x for plating zones
            between transverse primary supporting members will be used.
        """
        ref_array = self._mesh_extent.plating_zones_ref_array
        n_rows, n_columns = np.shape(ref_array)
        final_mesh_dim_x = {}

        for column in range(1, n_columns + 1):
            plating_zone_IDs = ref_array[:, column - 1]
            plating_zones = [self._grillage.plating()[plate_id]
                             for plate_id in plating_zone_IDs]

            x_list = [mesh_dim_x[plate.id] for plate in plating_zones]
            dim_x = np.amin(x_list)
            final_mesh_dim_x[column] = dim_x

            for plate in plating_zones:
```

```
                if plate.stiff_dir is BeamDirection.TRANSVERSE:
                    tran1 = self._grillage.transverse_members()[column]
                    tran2 = self._grillage.transverse_members()[column + 1]
                    max_x = self.get_min_fl_el_len_between_psm(tran1, tran2)
                    dim_x = mesh_dim_x[plate.id]
                    if dim_x > max_x != 0:  # If dimension x exceeds the maximum
                        stiff_spacing = plate.get_stiffener_spacing() * 1000
                        dim_x = self.refine_plate_element(stiff_spacing, max_x)
                        final_mesh_dim_x[column] = dim_x
                    else:
                        final_mesh_dim_x[column] = dim_x
                    break  # Stop after finding transverse stiffeners

        return final_mesh_dim_x

    def assign_base_dim_y(self, mesh_dim_y):
        """
        Method for global consideration of base mesh dimensions dim_y,
            for each row of plating zones on the grillage model.

        :param mesh_dim_y: Input dictionary of quad element y dimensions based
            on stiffener spacing and maximum allowed aspect ratio.
        :return: Assigns dimension y for each row of plating zones between
            longitudinal primary supporting members, identified using plating
            zones reference array based on Axis of Symmetry input.
            If there are no longitudinal stiffeners on the column of plating
            zones, a minimum value of all saved dim_y for plating zones between
            longitudinal primary supporting members will be used.
        """
        ref_array = self._mesh_extent.plating_zones_ref_array
        n_rows, n_columns = np.shape(ref_array)
        final_mesh_dim_y = {}

        for row in range(1, n_rows + 1):
            plating_zone_IDs = ref_array[row - 1, :]
            plating_zones = [self._grillage.plating()[plate_id]
                             for plate_id in plating_zone_IDs]

            y_list = [mesh_dim_y[plate.id] for plate in plating_zones]
            dim_y = np.amin(y_list)
            final_mesh_dim_y[row] = dim_y

            for plate in plating_zones:
                if plate.stiff_dir is BeamDirection.LONGITUDINAL:
                    long1 = self._grillage.longitudinal_members()[row]
                    long2 = self._grillage.longitudinal_members()[row + 1]
                    max_y = self.get_min_fl_el_len_between_psm(long1, long2)
                    dim_y = mesh_dim_y[plate.id]
                    if dim_y > max_y != 0:  # If dimension y exceeds the maximum
                        stiff_spacing = plate.get_stiffener_spacing() * 1000
                        dim_y = self.refine_plate_element(stiff_spacing, max_y)
                        final_mesh_dim_y[row] = dim_y
                    else:  # If dim_y does not exceed the maximum max_y
                        final_mesh_dim_y[row] = dim_y
                    break  # Stop after finding longitudinal stiffeners

        return final_mesh_dim_y

    def get_base_dim_x(self, plate: Plate):
        """
```

```
        :param plate: Selected plating zone.
        :return: Base quad element x dimension for any plating zone. Returns the
            value based on longitudinal segment ID from the list of all x
            dimensions: mesh_dim_x.
        """
        if self._mesh_dim_x:
            segment_id = plate.long_seg1.id
            return self._mesh_dim_x[segment_id]
        else:
            raise BaseMeshDimensionX

    def get_base_dim_y(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Base quad element y dimension for any plating zone. Returns the
            value based on transverse segment ID from the list of all y
            dimensions: mesh_dim_y.
        """
        if self._mesh_dim_y:
            segment_id = plate.trans_seg1.id
            return self._mesh_dim_y[segment_id]
        else:
            raise BaseMeshDimensionY

    def get_flange_element_num(self, segment: Segment):
        """
        :param segment: Selected segment of a primary supporting member.
        :return: Number of elements across the flange.
            Method returns 0 for FB beam type.
        """
        flange_element_width = self.get_flange_el_width(segment)
        if flange_element_width == 0:
            return 0
        else:
            return self._num_eaf

    def transition_dim_x(self, plate: Plate):
        """
        Method for local consideration of transition mesh dimensions dim_tr_x
        for each plating zone individually.

        If stiffener direction is transverse, there are transition elements
        next to transverse segments.

        :param plate: Selected plating zone.

        n_elem - Number of elements with dimension dim_x that fit inside
            the remaining distance
        """
        dim_tr_x = 0

        if plate.stiff_dir is BeamDirection.TRANSVERSE:
            dim_x = self.get_base_dim_x(plate)
            dim_y = self.get_base_dim_y(plate)
            stiff_offset = plate.get_equal_stiffener_offset() * 1000

            if stiff_offset < 0:
                raise NegativeRemainingDistance(plate.id)

            n_elem = np.floor(stiff_offset / dim_x)
```

```
            if n_elem == 0:
                dim_tr_x = stiff_offset
            else:
                dim_tr_x = stiff_offset - n_elem * dim_x

            if dim_tr_x != 0:
                ar = self.element_aspect_ratio(dim_tr_x, dim_y)
                if ar > self._plate_aspect_ratio and stiff_offset > dim_x:
                    dim_tr_x += dim_x

        return dim_tr_x, dim_tr_x

    def transition_dim_y(self, plate: Plate):
        """
        Method for local consideration of transition mesh dimensions dim_tr_y
        for each plating zone individually.

        If stiffener direction is longitudinal, there are transition elements
        next to longitudinal segments.

        :param plate: Selected plating zone.

        n_elem - Number of elements with dimension dim_y that fit inside
            the remaining distance
        """
        dim_tr_y = 0

        if plate.stiff_dir is BeamDirection.LONGITUDINAL:
            dim_x = self.get_base_dim_x(plate)
            dim_y = self.get_base_dim_y(plate)
            stiff_offset = plate.get_equal_stiffener_offset() * 1000

            if stiff_offset < 0:
                raise NegativeRemainingDistance(plate.id)

            n_elem = np.floor(stiff_offset / dim_y)
            if n_elem == 0:
                dim_tr_y = stiff_offset
            else:
                dim_tr_y = stiff_offset - n_elem * dim_y

            if dim_tr_y != 0:
                ar = self.element_aspect_ratio(dim_tr_y, dim_x)
                if ar > self._plate_aspect_ratio and stiff_offset > dim_y:
                    dim_tr_y += dim_y

        return dim_tr_y, dim_tr_y

    def assign_transition_dim_x(self):
        """
        Method for global consideration of transition mesh dimension x, for each
            column of plating zones.

        :return: Assigns transition elemenet x dimension for each column of
            plating zones between transverse primary supporting members,
            identified using plating zones reference array based on
            Axis of Symmetry.
        """
        ref_array = self._mesh_extent.plating_zones_ref_array
        n_rows, n_columns = np.shape(ref_array)
```

```
        tr_el_dim_x = np.zeros((2, n_columns))

        for column in range(1, n_columns + 1):
            plating_zone_IDs = ref_array[:, column - 1]
            plating_zones = [self._grillage.plating()[plate_id] for
                             plate_id in plating_zone_IDs]
            for plate in plating_zones:
                tr_dim_x1, tr_dim_x2 = self.transition_dim_x(plate)
                tr_el_dim_x[0, column - 1] = tr_dim_x1
                tr_el_dim_x[1, column - 1] = tr_dim_x2
                if plate.stiff_dir is BeamDirection.TRANSVERSE:
                    break

        return tr_el_dim_x

    def assign_transition_dim_y(self):
        """
        Method for global consideration of transition mesh dimension y, for
            each row of plating zones.

        :return: Assigns transition elemenet y dimension for each row of plating
            zones between longitudinal primary supporting members,
            identified using plating zones reference array based on
            Axis of Symmetry.
        """
        ref_array = self._mesh_extent.plating_zones_ref_array
        n_rows, n_columns = np.shape(ref_array)
        tr_el_dim_y = np.zeros((2, n_rows))

        for row in range(1, n_rows + 1):
            plating_zone_IDs = ref_array[row - 1, :]
            plating_zones = [self._grillage.plating()[plate_id] for
                             plate_id in plating_zone_IDs]
            for plate in plating_zones:
                tr_dim_y1, tr_dim_y2 = self.transition_dim_y(plate)
                tr_el_dim_y[0, row - 1] = tr_dim_y1
                tr_el_dim_y[1, row - 1] = tr_dim_y2
                if plate.stiff_dir is BeamDirection.LONGITUDINAL:
                    break

        return tr_el_dim_y

    def get_tr_dim_x(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Transition quad element x dimensions for any plating zone.
            Returns the value based on longitudinal segment ID. First value
            (dim_1) represents the element closest to the first transverse
            segment (trans_seg_1) and the second (dim_2) represents the element
            closest to the second transverse segment (trans_seg_2)
            that define the plating zone.
        """
        segment_id = plate.long_seg1.id
        dim_id = segment_id - 1
        dim = self.assign_transition_dim_x()
        dim_1 = dim[0][dim_id]
        dim_2 = dim[1][dim_id]
        return dim_1, dim_2

    def get_tr_dim_y(self, plate: Plate):
```

```
        """
        :param plate: Selected plating zone.
        :return: Transition quad element x dimensions for any plating zone.
            Returns the value based on transverse segment ID. First value
            (dim_1) represents the element closest to the first longitudinal
            segment (long_seg_1) and the second (dim_2) represents the element
            closest to the second longitudinal segment (long_seg_2)
            that define the plating zone.
        """
        segment_id = plate.trans_seg1.id
        dim_id = segment_id - 1
        dim = self.assign_transition_dim_y()
        dim_1 = dim[0][dim_id]
        dim_2 = dim[1][dim_id]
        return dim_1, dim_2

    def long_split_element(self):
        """
        :return: True if a quad element between stiffeners is split in half by a
            longitudinal Axis of Symmetry on plating zones in long_e_split_zone
        """
        for plate in self.mesh_extent.long_e_split_zone.values():
            stiff_spacing = plate.get_stiffener_spacing() * 1000
            base_dim = self.get_base_dim_y(plate)
            n_elem = np.round(stiff_spacing / base_dim)
            if np.mod(n_elem, 2) != 0:
                return True
            else:
                return False
        else:
            return False

    def tran_split_element(self):
        """
        :return: True if a quad element between stiffeners is split in half by a
            transverse Axis of Symmetry on plating zones in tran_e_split_zone.
        """
        for plate in self.mesh_extent.tran_e_split_zone.values():
            stiff_spacing = plate.get_stiffener_spacing() * 1000
            base_dim = self.get_base_dim_x(plate)
            n_elem = np.round(stiff_spacing / base_dim)
            if np.mod(n_elem, 2) != 0:
                return True
            else:
                return False
        else:
            return False

    def get_base_element_number(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Number of base size elements along x and y dimension of
            the plating zone.

        Index reference:
            x - number of elements or dimension in the longitudinal direction
            y - number of elements or dimension in the transverse direction
        """
        L = self._mesh_extent.get_long_plate_dim(plate)
        B = self._mesh_extent.get_tran_plate_dim(plate)
```

```
2281
2282        dim_x = self.get_base_dim_x(plate)
2283        dim_y = self.get_base_dim_y(plate)
2284
2285        tr_el_dim_x1, tr_el_dim_x2 = self.get_tr_dim_x(plate)
2286        tr_el_dim_y1, tr_el_dim_y2 = self.get_tr_dim_y(plate)
2287
2288        if plate.id in self._mesh_extent.long_half_plate_zones:
2289            tr_el_dim_y2 = 0
2290
2291        elif plate.id in self._mesh_extent.tran_half_plate_zones:
2292            tr_el_dim_x2 = 0
2293
2294        elif plate.id in self._mesh_extent.quarter_plate_zone:
2295            tr_el_dim_x2 = 0
2296            tr_el_dim_y2 = 0
2297
2298        dim_x_range = L - tr_el_dim_x1 - tr_el_dim_x2
2299        dim_y_range = B - tr_el_dim_y1 - tr_el_dim_y2
2300
2301        n_elem_x = dim_x_range / dim_x
2302        n_elem_y = dim_y_range / dim_y
2303
2304        if plate in self.mesh_extent.tran_e_split_zone.values() \
2305                and self.tran_split_element():
2306            n_dim_x = np.round(n_elem_x - 0.5)
2307        else:
2308            n_dim_x = np.round(n_elem_x)  # Number of elements with dim_x
2309
2310        if plate in self.mesh_extent.long_e_split_zone.values() \
2311                and self.long_split_element():
2312            n_dim_y = np.round(n_elem_y - 0.5)
2313        else:
2314            n_dim_y = np.round(n_elem_y)  # Number of elements with dim_y
2315
2316        return n_dim_x, n_dim_y
2317
2318    def get_long_split_element_num(self, plate: Plate):
2319        """
2320        :param plate: Selected plating zone.
2321        :return: Number of base size elements being split in half by
2322            longitudinal Axis of Symmetry.
2323        """
2324        if plate in self._mesh_extent.long_e_split_zone.values() \
2325                and self.long_split_element():
2326            return 1
2327        else:
2328            return 0
2329
2330    def get_tran_split_element_num(self, plate: Plate):
2331        """
2332        :param plate: Selected plating zone.
2333        :return: Number of base size elements being split in half by
2334            transverse Axis of Symmetry.
2335        """
2336        if plate in self._mesh_extent.tran_e_split_zone.values() \
2337                and self.tran_split_element():
2338            return 1
2339        else:
2340            return 0
```

```
2341
2342    def get_long_tr_element_num(self, plate: Plate, segment: Segment):
2343        """
2344        :param plate: Selected plating zone.
2345        :param segment: Selected segment of a primary supporting member.
2346        :return: Number of transition elements between longitudinal segment
2347            and base elements.
2348        """
2349        tr_element_dim_1, tr_element_dim_2 = self.get_tr_dim_y(plate)
2350        if segment is plate.long_seg1 and tr_element_dim_1 != 0:
2351            return 1
2352        elif segment is plate.long_seg2 and tr_element_dim_2 != 0:
2353            return 1
2354        else:
2355            return 0
2356
2357    def get_tran_tr_element_num(self, plate: Plate, segment: Segment):
2358        """
2359        :param plate: Selected plating zone.
2360        :param segment: Selected segment of a primary supporting member.
2361        :return: Number of transition elements between transverse segment
2362            and base elements.
2363        """
2364        tr_element_dim_1, tr_element_dim_2 = self.get_tr_dim_x(plate)
2365        if segment is plate.trans_seg1 and tr_element_dim_1 != 0:
2366            return 1
2367        elif segment is plate.trans_seg2 and tr_element_dim_2 != 0:
2368            return 1
2369        else:
2370            return 0
2371
2372    def plate_edge_node_spacing_x(self, plate: Plate):
2373        """
2374        :param plate: Selected plating zone.
2375        :return: Distance between plate nodes along longitudinal edges
2376         (along x axis), in order, for the selected plating zone.
2377        """
2378        base_dim_x = self.get_base_dim_x(plate)
2379        tr_element_dim_x1, tr_element_dim_x2 = self.get_tr_dim_x(plate)
2380
2381        tr_el_num_seg_1 = self.get_tran_tr_element_num(plate, plate.trans_seg1)
2382        tr_el_num_seg_2 = self.get_tran_tr_element_num(plate, plate.trans_seg2)
2383        base_mesh_element_num = self.get_base_element_number(plate)[0]
2384        split_element_number = self.get_tran_split_element_num(plate)
2385
2386        if plate.id in self._mesh_extent.tran_half_plate_zones or \
2387                plate.id in self._mesh_extent.quarter_plate_zone:
2388            tr_el_num_seg_2 = 0
2389
2390        x_spacing = {}
2391        self.save_node_spacing(x_spacing, tr_el_num_seg_1, tr_element_dim_x1)
2392        self.save_node_spacing(x_spacing, base_mesh_element_num, base_dim_x)
2393        self.save_node_spacing(x_spacing, split_element_number, base_dim_x / 2)
2394        self.save_node_spacing(x_spacing, tr_el_num_seg_2, tr_element_dim_x2)
2395
2396        return x_spacing
2397
2398    def plate_edge_node_spacing_y(self, plate: Plate):
2399        """
2400        :param plate: Selected plating zone.
```

```
        :return: Distance between plate nodes along transverse edges
            (along y axis), in order, for the selected plating zone.
        """
        base_dim_y = self.get_base_dim_y(plate)
        tr_element_dim_y1, tr_element_dim_y2 = self.get_tr_dim_y(plate)

        tr_el_num_seg_1 = self.get_long_tr_element_num(plate, plate.long_seg1)
        tr_el_num_seg_2 = self.get_long_tr_element_num(plate, plate.long_seg2)
        base_mesh_element_num = self.get_base_element_number(plate)[1]
        split_element_number = self.get_long_split_element_num(plate)

        if plate.id in self._mesh_extent.long_half_plate_zones or \
                plate.id in self._mesh_extent.quarter_plate_zone:
            tr_el_num_seg_2 = 0

        y_spacing = {}
        self.save_node_spacing(y_spacing, tr_el_num_seg_1, tr_element_dim_y1)
        self.save_node_spacing(y_spacing, base_mesh_element_num, base_dim_y)
        self.save_node_spacing(y_spacing, split_element_number, base_dim_y / 2)
        self.save_node_spacing(y_spacing, tr_el_num_seg_2, tr_element_dim_y2)

        return y_spacing

    def opposite_flange_width(self, segment: Segment):
        """
        :param segment: Selected segment of a primary supporting member.
        :return: Maximum net flange width of both perpendicular segments
            connected at the intersection of primary supporting members.
            Returns value bf_net at both ends of the selected segment.
            Includes checks for beam type and flange direction at both ends.
        """
        corr_add = self._grillage.corrosion_addition()[1]
        end1, end2 = self._grillage.get_perpendicular_segments(segment)
        segment_at_end1 = end1[0]
        segment_at_end2 = end2[0]

        bf_end1 = [segment.beam_prop.bf_net(corr_add) for segment in end1]
        bf_end2 = [segment.beam_prop.bf_net(corr_add) for segment in end2]
        bf_max1 = np.amax(bf_end1)
        bf_max2 = np.amax(bf_end2)

        t_beam_at_end1 = [segment.beam_prop.beam_type for segment in end1 if
                          segment.beam_prop.beam_type is BeamType.T]

        t_beam_at_end2 = [segment.beam_prop.beam_type for segment in end2 if
                          segment.beam_prop.beam_type is BeamType.T]

        if any(t_beam_at_end1):
            bf_max1 = bf_max1 / 2
        if any(t_beam_at_end2):
            bf_max2 = bf_max2 / 2

        l_beam_at_end1 = [segment.beam_prop.beam_type for segment in end1
                          if segment.beam_prop.beam_type is BeamType.L]
        l_beam_at_end2 = [segment.beam_prop.beam_type for segment in end2
                          if segment.beam_prop.beam_type is BeamType.L]

        flange_dir1 = segment_at_end1.primary_supp_mem.flange_direction
        flange_dir2 = segment_at_end2.primary_supp_mem.flange_direction
        central_segment = self._grillage.central_segment(segment)
```

```
2461
2462        # End 1
2463        if flange_dir1 is FlangeDirection.INWARD \
2464                and any(l_beam_at_end1) and not t_beam_at_end1 \
2465                and segment_at_end1.primary_supp_mem.rel_dist > 0.5\
2466                and not central_segment:
2467            bf_max1 = 0
2468
2469        if flange_dir1 is FlangeDirection.OUTWARD \
2470                and any(l_beam_at_end1) and not t_beam_at_end1 \
2471                and segment_at_end1.primary_supp_mem.rel_dist < 0.5 \
2472                and not central_segment:
2473            bf_max1 = 0
2474
2475        # End 2
2476        if flange_dir2 is FlangeDirection.INWARD \
2477                and any(l_beam_at_end2) and not t_beam_at_end2 \
2478                and segment_at_end2.primary_supp_mem.rel_dist < 0.5 \
2479                and not central_segment:
2480            bf_max2 = 0
2481
2482        if flange_dir2 is FlangeDirection.OUTWARD \
2483                and any(l_beam_at_end2) and not t_beam_at_end2 \
2484                and segment_at_end2.primary_supp_mem.rel_dist > 0.5 \
2485                and not central_segment:
2486            bf_max2 = 0
2487
2488        return bf_max1, bf_max2
2489
2490    def opposite_flange_element_num(self, segment: Segment):
2491        """
2492        :param segment: Selected segment of a primary supporting member.
2493        :return: Number of elements across the flange of a perpendicular
2494            segment that influences the FE mesh of the selected segment.
2495            Method returns values for both ends of the selected segment.
2496            Returns 0 for FB beam type.
2497        """
2498        bf_max1, bf_max2 = self.opposite_flange_width(segment)
2499        if bf_max1 == 0 and bf_max2 == 0:
2500            return 0, 0
2501        elif bf_max1 == 0 and bf_max2 != 0:
2502            return 0, self._num_eaf
2503        elif bf_max1 != 0 and bf_max2 == 0:
2504            return self._num_eaf, 0
2505        else:
2506            return self._num_eaf, self._num_eaf
2507
2508    def transition_dim(self, base, plate_tr, fl_width):
2509        if plate_tr == 0:
2510            if base > fl_width:
2511                flange_tr_dim = base - fl_width
2512            else:
2513                flange_tr_dim = 2 * base - fl_width
2514        else:
2515            if plate_tr > fl_width:
2516                flange_tr_dim = plate_tr - fl_width
2517            else:
2518                flange_tr_dim = plate_tr + base - fl_width
2519
2520        if flange_tr_dim != 0:
```

```
            ar = self.element_aspect_ratio(flange_tr_dim, fl_width)
            if ar > self._flange_aspect_ratio and flange_tr_dim < fl_width:
                flange_tr_dim += base

        return flange_tr_dim

    def flange_transition_dim(self, segment: Segment):
        """
        :param segment: Selected segment.
        :return: Transition element dimensions for any segment on both ends.
        """
        direction = segment.primary_supp_mem.direction
        el_width_1, el_width_2 = self.opposite_flange_width(segment)
        plate_list = self._grillage.segment_common_plates(segment)
        plate = plate_list[0]

        if direction is BeamDirection.LONGITUDINAL:
            base_dim = self.get_base_dim_x(plate)
            plate_tr_1, plate_tr_2 = self.get_tr_dim_x(plate)
        else:
            base_dim = self.get_base_dim_y(plate)
            plate_tr_1, plate_tr_2 = self.get_tr_dim_y(plate)

        fl_tr_1 = self.transition_dim(base_dim, plate_tr_1, el_width_1)
        fl_tr_2 = self.transition_dim(base_dim, plate_tr_2, el_width_2)

        return fl_tr_1, fl_tr_2

    def flange_base_element_num(self, segment: Segment):
        """
        :param segment: Selected segment of a primary supporting member.
        :return: Number of base size elements the axial direction of the
            selected segment.

        Index reference:
            x - number of elements or dimension in the longitudinal direction
            y - number of elements or dimension in the transverse direction
        """
        plate_list = self._grillage.segment_common_plates(segment)
        plate = plate_list[0]
        direction = segment.primary_supp_mem.direction
        segment_len = segment.segment_len() * 1000
        fl_tr_1, fl_tr_2 = self.flange_transition_dim(segment)
        fl_el_width_1, fl_el_width_2 = self.opposite_flange_width(segment)

        if direction is BeamDirection.LONGITUDINAL:
            base_dim = self.get_base_dim_x(plate)
            split_num = self.get_tran_split_element_num(plate)
        else:
            base_dim = self.get_base_dim_y(plate)
            split_num = self.get_long_split_element_num(plate)

        if segment in self.mesh_extent.half_segments.values():
            segment_len = segment_len / 2
            fl_tr_2 = 0
            fl_el_width_2 = 0

        base_range = segment_len - fl_el_width_1 - fl_tr_1 - fl_tr_2
        base_range -= fl_el_width_2 + split_num * base_dim * 0.5
        base_dim_num = np.round(base_range / base_dim)
```

```
        return base_dim_num

    def get_flange_transition_num(self, segment: Segment):
        """
        :param segment: Selected segment.
        :return: Number of transition elements on the segment flange.
            Returns value for both ends of the selected segment.
        """
        tr_el_dim_1, tr_el_dim_2 = self.flange_transition_dim(segment)

        if tr_el_dim_1 == 0:
            tr_num_edge1 = 0
        else:
            tr_num_edge1 = 1

        if tr_el_dim_2 == 0:
            tr_num_edge2 = 0
        else:
            tr_num_edge2 = 1

        return tr_num_edge1, tr_num_edge2

    def flange_edge_node_spacing(self, segment: Segment):
        pass

    def calc_plate_start_node_ids(self):
        """
        :return: Calculates starting node ID for all meshed plating zone and
            saves first node ID into start_nodes.
        """
        node_id_counter = 1
        for plate in self._mesh_extent.all_plating_zones.values():
            edge_nodes_x = self.plate_edge_node_spacing_x(plate)
            edge_nodes_y = self.plate_edge_node_spacing_y(plate)
            column_limit = len(edge_nodes_x) + 1
            row_limit = len(edge_nodes_y) + 1
            total = row_limit * column_limit
            self.start_nodes[plate.id] = node_id_counter
            node_id_counter += total

    def calculate_mesh_dimensions(self):
        """
        :return: Calculates all element dimensions and saves them to
            dictionaries mesh_dim_x, mesh_dim_y. These values need to be
            calculated only once and will be used for all node and element
            generation. Contains check of chosen Axis of Symmetry override.
        """
        discovered = self._mesh_extent.aos_input
        override = self._mesh_extent.axis_of_symm_override
        mc = self._mesh_extent.model_check
        mc.symmetry_override_check(discovered, override)

        if override:
            print("Selected Axis of Symmetry override:", override.name)
            print("Automatic symmetry discovery would have selected:",
                  discovered.name)
        else:
            print("Automatically discovered grillage model symmetry:",
                  discovered.name)

```

```
2641         self._mesh_extent.grillage_mesh_extent()
2642         base_dim_x, base_dim_y = self.calc_element_base_size()
2643         self.mesh_dim_x = self.assign_base_dim_x(base_dim_x)
2644         self.mesh_dim_y = self.assign_base_dim_y(base_dim_y)
2645         self.calc_plate_start_node_ids()
2646 
2647 
2648 class ElementSizeV1(MeshSize):
2649     def __init__(self, mesh_extent: MeshExtent,
2650                  min_num_ebs: int,
2651                  min_num_eweb: int,
2652                  num_eaf: int,
2653                  flange_aspect_ratio: float,
2654                  plate_aspect_ratio: float,
2655                  des_plate_aspect_ratio: float):
2656         """
2657         Class for calculating finite element dimensions
2658         specific to meshing variant V1.
2659 
2660         Mesh variant V1 reflects primary supporting member flange elements
2661         onto plating and has the following limitations:
2662 
2663             1.) Flange width of a primary supporting member has to be constant.
2664             2.) All primary supporting members need to have the same web height.
2665             3.) Flange element overlap has to be in the same plane.
2666             4.) Grillage plating can not be defined with any camber.
2667         """
2668         super().__init__(mesh_extent, min_num_ebs, min_num_eweb, num_eaf,
2669                          flange_aspect_ratio, plate_aspect_ratio, des_plate_aspect_ratio)
2670 
2671     def get_reduced_plate_dim(self, plate: Plate):
2672         """
2673         :param plate: Selected plating zone.
2674         :return: Reduced plate dimensions based on plate stiffener orientation
2675             and flange width for mesh variant V1.
2676         """
2677         n_eaf = self._num_eaf
2678         if plate.stiff_dir is BeamDirection.LONGITUDINAL:
2679             L = plate.plate_longitudinal_dim() * 1000
2680             dim_xf1 = self.get_flange_el_width(plate.trans_seg1) * n_eaf
2681             dim_xf2 = self.get_flange_el_width(plate.trans_seg2) * n_eaf
2682             L_red = L - dim_xf1 - dim_xf2
2683             return L_red
2684 
2685         elif plate.stiff_dir is BeamDirection.TRANSVERSE:
2686             B = plate.plate_transverse_dim() * 1000
2687             dim_yf1 = self.get_flange_el_width(plate.long_seg1) * n_eaf
2688             dim_yf2 = self.get_flange_el_width(plate.long_seg2) * n_eaf
2689             B_red = B - dim_yf1 - dim_yf2
2690             return B_red
2691 
2692     def calc_element_base_size(self):
2693         """
2694         Difference in this method specific to mesh variant V1 is in plate
2695         dimension plate_dim used for calculating base mesh dimensions.
2696         If flange elements are reflected onto plating mesh,
2697         a reduced plate dimension should be used.
2698 
2699         :return: Calculates the quad element size based on stiffener spacing
2700             and maximum allowed aspect ratio for all plating zones. Returns
```

```
            dictionaries of x and y base dimensions for all plating zones.
        """
        mesh_dim_x = {}  # Dimension x for all plating zones
        mesh_dim_y = {}  # Dimension y for all plating zones

        for plate in self._mesh_extent.all_plating_zones.values():
            reduced_dim = self.get_reduced_plate_dim(plate)
            plate_dim = self._mesh_extent.get_plate_dim(plate, reduced_dim)
            dim_x, dim_y = self.element_size_plating_zone(plate, plate_dim)
            mesh_dim_x[plate.id] = dim_x
            mesh_dim_y[plate.id] = dim_y

        return mesh_dim_x, mesh_dim_y

    def transition_dim_x(self, plate: Plate):
        """
        Method for local consideration of transition mesh dimensions dim_tr_x
        for each plating zone individually. Specific to meshing variant V1

        If stiffener direction is transverse, there are transition elements
        next to transverse segments.

        :param plate: Selected plating zone.

        n_elem - Number of elements with dimension dim_x that fit inside
            the remaining distance
        """
        dim_tr_x1 = 0
        dim_tr_x2 = 0

        if plate.stiff_dir is BeamDirection.TRANSVERSE:
            dim_x = self.get_base_dim_x(plate)
            dim_y = self.get_base_dim_y(plate)
            stiff_offset = plate.get_equal_stiffener_offset() * 1000
            n_eaf = self._num_eaf

            flange_width1 = self.get_flange_el_width(plate.trans_seg1) * n_eaf
            flange_width2 = self.get_flange_el_width(plate.trans_seg2) * n_eaf
            remaining_dist1 = stiff_offset - flange_width1
            remaining_dist2 = stiff_offset - flange_width2

            if remaining_dist1 < 0 or remaining_dist2 < 0:
                raise NegativeRemainingDistance(plate.id)

            n_elem1 = np.floor(remaining_dist1 / dim_x)
            n_elem2 = np.floor(remaining_dist2 / dim_x)

            if n_elem1 == 0:
                dim_tr_x1 = stiff_offset - flange_width1
            else:
                dim_tr_x1 = stiff_offset - n_elem1 * dim_x - flange_width1

            if n_elem2 == 0:
                dim_tr_x2 = stiff_offset - flange_width2

            else:
                dim_tr_x2 = stiff_offset - n_elem2 * dim_x - flange_width2

            if dim_tr_x1 != 0:
                ar = self.element_aspect_ratio(dim_tr_x1, dim_y)
```

```
                if ar > self._plate_aspect_ratio and remaining_dist1 > dim_x:
                    dim_tr_x1 += dim_x

            if dim_tr_x2 != 0:
                ar = self.element_aspect_ratio(dim_tr_x2, dim_y)
                if ar > self._plate_aspect_ratio and remaining_dist2 > dim_x:
                    dim_tr_x2 += dim_x

        return dim_tr_x1, dim_tr_x2

    def transition_dim_y(self, plate: Plate):
        """
        Method for local consideration of transition mesh dimensions dim_tr_y
        for each plating zone individually. Specific to meshing variant V1.

        If stiffener direction is longitudinal, there are transition elements
        next to longitudinal segments.

        :param plate: Selected plating zone.

        n_elem - Number of elements with dimension dim_y that fit inside
            the remaining distance
        """
        dim_tr_y1 = 0
        dim_tr_y2 = 0

        if plate.stiff_dir is BeamDirection.LONGITUDINAL:
            dim_x = self.get_base_dim_x(plate)
            dim_y = self.get_base_dim_y(plate)
            stiff_offset = plate.get_equal_stiffener_offset() * 1000
            n_eaf = self._num_eaf

            flange_width1 = self.get_flange_el_width(plate.long_seg1) * n_eaf
            flange_width2 = self.get_flange_el_width(plate.long_seg2) * n_eaf
            remaining_dist1 = stiff_offset - flange_width1
            remaining_dist2 = stiff_offset - flange_width2

            if remaining_dist1 < 0 or remaining_dist2 < 0:
                raise NegativeRemainingDistance(plate.id)

            n_elem1 = np.floor(remaining_dist1 / dim_y)
            n_elem2 = np.floor(remaining_dist2 / dim_y)

            if n_elem1 == 0:
                dim_tr_y1 = stiff_offset - flange_width1
            else:
                dim_tr_y1 = stiff_offset - n_elem1 * dim_y - flange_width1

            if n_elem2 == 0:
                dim_tr_y2 = stiff_offset - flange_width2

            else:
                dim_tr_y2 = stiff_offset - n_elem2 * dim_y - flange_width2

            if dim_tr_y1 != 0:
                ar = self.element_aspect_ratio(dim_tr_y1, dim_x)
                if ar > self._plate_aspect_ratio and remaining_dist1 > dim_y:
                    dim_tr_y1 += dim_y

            if dim_tr_y2 != 0:
```

```
                ar = self.element_aspect_ratio(dim_tr_y2, dim_x)
                if ar > self._plate_aspect_ratio and remaining_dist2 > dim_y:
                    dim_tr_y2 += dim_y

        return dim_tr_y1, dim_tr_y2

    def get_base_element_number(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Number of base size elements along x and y dimension of
            the plating zone.

        Index reference:
            x - number of elements or dimension in the longitudinal direction
            y - number of elements or dimension in the transverse direction
        """
        L = self._mesh_extent.get_long_plate_dim(plate)
        B = self._mesh_extent.get_tran_plate_dim(plate)
        n_eaf = self._num_eaf
        dim_x = self.get_base_dim_x(plate)
        dim_y = self.get_base_dim_y(plate)

        tr_el_dim_x1, tr_el_dim_x2 = self.get_tr_dim_x(plate)
        tr_el_dim_y1, tr_el_dim_y2 = self.get_tr_dim_y(plate)

        fl_dim_x1 = self.get_flange_el_width(plate.trans_seg1) * n_eaf
        fl_dim_x2 = self.get_flange_el_width(plate.trans_seg2) * n_eaf
        fl_dim_y1 = self.get_flange_el_width(plate.long_seg1) * n_eaf
        fl_dim_y2 = self.get_flange_el_width(plate.long_seg2) * n_eaf

        if plate.id in self._mesh_extent.long_half_plate_zones:
            fl_dim_y2 = 0
            tr_el_dim_y2 = 0

        elif plate.id in self._mesh_extent.tran_half_plate_zones:
            fl_dim_x2 = 0
            tr_el_dim_x2 = 0

        elif plate.id in self._mesh_extent.quarter_plate_zone:
            fl_dim_x2 = 0
            tr_el_dim_x2 = 0
            fl_dim_y2 = 0
            tr_el_dim_y2 = 0

        dim_x_range = L - tr_el_dim_x1 - tr_el_dim_x2 - fl_dim_x1 - fl_dim_x2
        dim_y_range = B - tr_el_dim_y1 - tr_el_dim_y2 - fl_dim_y1 - fl_dim_y2

        n_elem_x = dim_x_range / dim_x
        n_elem_y = dim_y_range / dim_y

        if plate in self.mesh_extent.tran_e_split_zone.values() \
                and self.tran_split_element():
            n_dim_x = np.round(n_elem_x - 0.5)
        else:
            n_dim_x = np.round(n_elem_x)  # Number of elements with dim_x

        if plate in self.mesh_extent.long_e_split_zone.values() \
                and self.long_split_element():
            n_dim_y = np.round(n_elem_y - 0.5)
        else:
```

```
            n_dim_y = np.round(n_elem_y)  # Number of elements with dim_y

        return n_dim_x, n_dim_y

    def plate_edge_node_spacing_x(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Distance between plate nodes along longitudinal edges
         (along x axis), in order, for the selected plating zone.
        """
        fl_dim_x1 = self.get_flange_el_width(plate.trans_seg1)
        fl_dim_x2 = self.get_flange_el_width(plate.trans_seg2)
        base_dim_x = self.get_base_dim_x(plate)
        tr_element_dim_x1, tr_element_dim_x2 = self.get_tr_dim_x(plate)

        tr_el_num_seg_1 = self.get_tran_tr_element_num(plate, plate.trans_seg1)
        tr_el_num_seg_2 = self.get_tran_tr_element_num(plate, plate.trans_seg2)
        flange_el_num_tran_seg1 = self.get_flange_element_num(plate.trans_seg1)
        flange_el_num_tran_seg2 = self.get_flange_element_num(plate.trans_seg2)
        base_mesh_element_num = self.get_base_element_number(plate)[0]
        split_element_number = self.get_tran_split_element_num(plate)

        if plate.id in self._mesh_extent.tran_half_plate_zones or \
                plate.id in self._mesh_extent.quarter_plate_zone:
            flange_el_num_tran_seg2 = 0
            tr_el_num_seg_2 = 0

        x_spacing = {}
        self.save_node_spacing(x_spacing, flange_el_num_tran_seg1, fl_dim_x1)
        self.save_node_spacing(x_spacing, tr_el_num_seg_1, tr_element_dim_x1)
        self.save_node_spacing(x_spacing, base_mesh_element_num, base_dim_x)
        self.save_node_spacing(x_spacing, split_element_number, base_dim_x / 2)
        self.save_node_spacing(x_spacing, tr_el_num_seg_2, tr_element_dim_x2)
        self.save_node_spacing(x_spacing, flange_el_num_tran_seg2, fl_dim_x2)

        return x_spacing

    def plate_edge_node_spacing_y(self, plate: Plate):
        """
        :param plate: Selected plating zone.
        :return: Distance between plate nodes along transverse edges
            (along y axis), in order, for the selected plating zone.
        """
        fl_dim_y1 = self.get_flange_el_width(plate.long_seg1)
        fl_dim_y2 = self.get_flange_el_width(plate.long_seg2)
        base_dim_y = self.get_base_dim_y(plate)
        tr_element_dim_y1, tr_element_dim_y2 = self.get_tr_dim_y(plate)

        tr_el_num_seg_1 = self.get_long_tr_element_num(plate, plate.long_seg1)
        tr_el_num_seg_2 = self.get_long_tr_element_num(plate, plate.long_seg2)
        flange_el_num_long_seg1 = self.get_flange_element_num(plate.long_seg1)
        flange_el_num_long_seg2 = self.get_flange_element_num(plate.long_seg2)
        base_mesh_element_num = self.get_base_element_number(plate)[1]
        split_element_number = self.get_long_split_element_num(plate)

        if plate.id in self._mesh_extent.long_half_plate_zones or \
                plate.id in self._mesh_extent.quarter_plate_zone:
            flange_el_num_long_seg2 = 0
            tr_el_num_seg_2 = 0

```

```
        y_spacing = {}

        self.save_node_spacing(y_spacing, flange_el_num_long_seg1, fl_dim_y1)
        self.save_node_spacing(y_spacing, tr_el_num_seg_1, tr_element_dim_y1)
        self.save_node_spacing(y_spacing, base_mesh_element_num, base_dim_y)
        self.save_node_spacing(y_spacing, split_element_number, base_dim_y / 2)
        self.save_node_spacing(y_spacing, tr_el_num_seg_2, tr_element_dim_y2)
        self.save_node_spacing(y_spacing, flange_el_num_long_seg2, fl_dim_y2)

        return y_spacing


class ElementSizeV2(MeshSize):
    def __init__(self, mesh_extent: MeshExtent,
                 min_num_ebs: int,
                 min_num_eweb: int,
                 num_eaf: int,
                 flange_aspect_ratio: float,
                 plate_aspect_ratio: float,
                 des_plate_aspect_ratio: float):
        """
        Class for calculating finite element dimensions
        specific to meshing variant V2.

        Mesh variant V2 has a more uniform plating mesh with transition plate
        elements between edges of plating zones and base mesh elements.
        Allows for different number of edge nodes along the top and bottom row
        of segment web nodes by creating a transition mesh on the segment web
        using both quad and triangle elements. Variant is not suitable for
        fine mesh generation because of element size limitations.

        Mesh variant V2 has the following limitations:

            1.) All primary supporting members need to have the same web height.
            2.) Flange element overlap has to be in the same plane.
            3.) Grillage plating can not be defined with any camber.
            4.) Value of num_eaf can not be greater than 1.
            5.) Limitation on finite element size
        """
        super().__init__(mesh_extent, min_num_ebs, min_num_eweb, num_eaf,
                         flange_aspect_ratio, plate_aspect_ratio, des_plate_aspect_ratio)

    def flange_edge_node_spacing(self, segment: Segment):
        """
        :param segment: Selected segmenet.
        :return: Distance between segment nodes in the axial direction of the
            selected segment, in order in the direction of global csy axis.
        """
        bf_max1, bf_max2 = self.opposite_flange_width(segment)
        tr_dim_1, tr_dim_2 = self.flange_transition_dim(segment)
        fl_el_num1, fl_el_num2 = self.opposite_flange_element_num(segment)
        tr_num_1, tr_num_2 = self.get_flange_transition_num(segment)
        direction = segment.primary_supp_mem.direction
        plate_list = self._grillage.segment_common_plates(segment)
        plate = plate_list[0]
        base_el_num = self.flange_base_element_num(segment)

        if direction is BeamDirection.LONGITUDINAL:
            base_dim = self.get_base_dim_x(plate)
            split_element_number = self.get_tran_split_element_num(plate)
```

```
        else:
            base_dim = self.get_base_dim_y(plate)
            split_element_number = self.get_long_split_element_num(plate)

        if segment in self.mesh_extent.half_segments.values():
            fl_el_num2 = 0
            tr_num_2 = 0

        spacing = {}
        self.save_node_spacing(spacing, fl_el_num1, bf_max1)
        self.save_node_spacing(spacing, tr_num_1, tr_dim_1)
        self.save_node_spacing(spacing, base_el_num, base_dim)
        self.save_node_spacing(spacing, split_element_number, base_dim / 2)
        self.save_node_spacing(spacing, tr_num_2, tr_dim_2)
        self.save_node_spacing(spacing, fl_el_num2, bf_max2)
        return spacing


class PlateMesh:
    def __init__(self, mesh_size: MeshSize, plate: Plate):
        """
        Class for generating FE mesh on a selected plating zone.

        :param mesh_size: Selected input mesh size for plate zone mesh generation.
        :param plate: Selected plate for generating Node and Element objects.
            of the selected plating zone based on Axis Of Symmetry.
        :return: Determines node coordinates and generates finite element Node
            and Element objects on the selected plating zone. Returns last node
            and element ID, to continue node and element numeration.
        """
        self._mesh_size = mesh_size
        self._plate = plate

        self._edge_nodes_x = self._mesh_size.plate_edge_node_spacing_x(plate)
        self._edge_nodes_y = self._mesh_size.plate_edge_node_spacing_y(plate)

    def get_mesh_limits(self):
        """
        :return: Row and column limit values for generating plate nodes and
            elements, based on which Axis of Symmetry splits the plating zone.

        row_limit - Number of node rows on the entire plating zone
        column_limit - Number of node columns on the entire plating zone
        """
        row_limit = len(self._edge_nodes_y) + 1
        column_limit = len(self._edge_nodes_x) + 1
        return row_limit, column_limit

    def get_plate_element_property(self, fem: GeoGrillageFEM):
        """
        :return: Quad element GeoFEM Plate property ID used for plating.
        """
        plate_prop_id = self._plate.plate_prop.id
        fem_prop_id = fem.plate_property_IDs[plate_prop_id]
        return fem_prop_id

    def get_stiffener_beam_property(self, fem: GeoGrillageFEM):
        """
        :return: Beam element GeoFEM property ID used for stiffeners.
        """
```

```
3061         beam_prop_id = self._plate.stiff_layout.beam_prop.id
3062         fem_prop_id = fem.stiff_beam_prop_IDs[beam_prop_id]
3063         return fem_prop_id
3064 
3065     def get_half_stiffener_beam_property(self, fem: GeoGrillageFEM):
3066         """
3067         :return: Beam element GeoFEM property ID used for stiffeners located on
3068             Axis Of Symmetry.
3069         """
3070         beam_prop_id = self._plate.stiff_layout.beam_prop.id
3071         fem_prop_id = fem.half_stiff_beam_prop_IDs[beam_prop_id]
3072         return fem_prop_id
3073 
3074     @staticmethod
3075     def reference_node_ID_array(start_id, row_limit, column_limit):
3076         """
3077         :param start_id:
3078         :param row_limit:
3079         :param column_limit:
3080         :return: 2D array of node IDs arranged to represent relative placement
3081             of nodes on the plating zone. Used as a reference for quad element
3082             generation.
3083         """
3084         total = row_limit * column_limit  # Total number of nodes
3085         id_list = np.arange(start_id, start_id + total, 1)
3086         node_id_array = np.reshape(id_list, [row_limit, column_limit])
3087         return node_id_array
3088 
3089     def generate_plate_nodes(self, fem: GeoGrillageFEM):
3090         """
3091         :return: Generates nodes on the entire plating zone and returns last
3092             node ID to continue node numeration on other plating zones.
3093             Overlapping nodes along the edges of the plating zone are saved
3094             into a separate dictionary for node overlap identification.
3095 
3096         row - Row of nodes along x axis.
3097         column - Column of nodes along y axis.
3098         spacing_vector - Node coordinates in the local coordinate system.
3099         """
3100         ref_node1 = Segment.get_segment_node1(self._plate.long_seg1)
3101         ref_node2 = Segment.get_segment_node2(self._plate.long_seg1)
3102         ref_vector = np.subtract(ref_node2, ref_node1)
3103         unit_ref_vector = ref_vector / np.linalg.norm(ref_vector)
3104         normal_vector = np.array((0, 0, 1))  # Vector normal to the plating
3105         perp_vector = np.cross(normal_vector, unit_ref_vector)
3106 
3107         spacing_vector_x = np.zeros(3)
3108         spacing_vector_y = np.zeros(3)
3109         dim_y_index = 1
3110         row_limit, column_limit = self.get_mesh_limits()
3111 
3112         for row in range(0, row_limit):
3113             dim_x_index = 1
3114             if row > 0:
3115                 dim_y = self._edge_nodes_y[dim_y_index]
3116                 spacing_vector_y += dim_y * perp_vector
3117                 dim_y_index += 1
3118             else:
3119                 spacing_vector_y = np.zeros(3)
3120 
```

```
            for column in range(0, column_limit):
                if column > 0:
                    dim_x = self._edge_nodes_x[dim_x_index]
                    spacing_vector_x += dim_x * unit_ref_vector
                    dim_x_index += 1
                else:
                    spacing_vector_x = np.zeros(3)

                spacing_vector = spacing_vector_x + spacing_vector_y
                node_coords = spacing_vector + ref_node1

                node = fem.add_node(node_coords)
                if row == 0 or row == row_limit - 1:
                    fem.add_node_to_node_overlaps(node)
                if column == 0 or column == column_limit - 1:
                    fem.add_node_to_node_overlaps(node)

    def generate_plate_elements(self, fem: GeoGrillageFEM):
        """
        :return: Generates elements on the entire plating zone and returns last
            element ID to continue element numeration on other plating zones.

        row_limit - Row of elements along x axis.
        column_limit - Column of elements along y axis.
        """
        row_limit, column_limit = self.get_mesh_limits()
        start_id = self._mesh_size.start_nodes[self._plate.id]
        node_id_array = self.reference_node_ID_array(start_id, row_limit, column_limit)
        fem_prop_id = self.get_plate_element_property(fem)

        id_el_nodes = [None] * 4
        for row in range(0, row_limit - 1):
            for column in range(0, column_limit - 1):
                id_el_nodes[0] = node_id_array[row, column]
                id_el_nodes[1] = node_id_array[row, column + 1]
                id_el_nodes[2] = node_id_array[row + 1, column + 1]
                id_el_nodes[3] = node_id_array[row + 1, column]

                elem = fem.add_quad_element(fem_prop_id, id_el_nodes)
                fem.add_to_plate_elements(elem)

    def identify_beam_nodes(self):
        """
        :return: Method identifies rows or columns of nodes, depending on
            stiffener orientation, where ordinary stiffeners are located in
            the reference node ID array. Returns a list of row or column
            indexes in the reference node ID array for beam element generation.
        """
        stiff_num = self._plate.get_stiffener_number()
        stiff_spacing = self._plate.get_stiffener_spacing() * 1000
        stiff_offset = self._plate.get_equal_stiffener_offset() * 1000
        id_list = []
        dist = 0
        stiff_counter = 0
        spacing = stiff_spacing
        if self._plate.stiff_dir is BeamDirection.TRANSVERSE:
            edge_nodes = self._edge_nodes_x.items()
        else:
            edge_nodes = self._edge_nodes_y.items()

```

```
        for item in edge_nodes:
            key, val = item
            dist += val
            if np.isclose(dist, stiff_offset):
                id_list.append(key)
                stiff_counter += 1
            if np.isclose(dist, stiff_offset + spacing):
                if stiff_counter == stiff_num:
                    break
                spacing += stiff_spacing
                id_list.append(key)
                stiff_counter += 1

        return id_list

    def generate_beam_elements(self, fem: GeoGrillageFEM):
        """
        :return: Generates beam elements of ordinary stiffeners between nodes
            identified using method identify_beam_nodes. Returns last beam
            element ID to continue beam element numeration on other plating
            zones.
        """
        row_limit, column_limit = self.get_mesh_limits()
        start_id = self._mesh_size.start_nodes[self._plate.id]
        node_id_array = self.reference_node_ID_array(start_id, row_limit, column_limit)
        stiff_dir = self._plate.stiff_dir
        prop_id = self.get_stiffener_beam_property(fem)
        aos_on_stiff = self._mesh_size.mesh_extent.aos_on_stiffener(self._plate)
        node_id_index_list = self.identify_beam_nodes()
        id_el_nodes = [None] * 2
        dir_vector = np.array([0, 0, -1])

        if stiff_dir is BeamDirection.LONGITUDINAL:
            for index in range(0, len(node_id_index_list)):
                if index == len(node_id_index_list) - 1 and aos_on_stiff:
                    prop_id = self.get_half_stiffener_beam_property(fem)

                for i in range(0, column_limit - 1):
                    id_el_nodes[0] = node_id_array[node_id_index_list[index], i]
                    id_el_nodes[1] = node_id_array[node_id_index_list[index], i + 1]
                    fem.add_beam_element(prop_id, id_el_nodes, dir_vector)
        else:
            for index in range(0, len(node_id_index_list)):
                if index == len(node_id_index_list) - 1 and aos_on_stiff:
                    prop_id = self.get_half_stiffener_beam_property(fem)

                for i in range(0, row_limit - 1):
                    id_el_nodes[0] = node_id_array[i, node_id_index_list[index]]
                    id_el_nodes[1] = node_id_array[i + 1, node_id_index_list[index]]
                    fem.add_beam_element(prop_id, id_el_nodes, dir_vector)

    def generate_mesh(self, fem: GeoGrillageFEM):
        """
        :return: Generates all nodes and elements on the selected plating zone.
        """
        self.generate_plate_nodes(fem)
        self.generate_plate_elements(fem)
        self.generate_beam_elements(fem)
```

```python
class SegmentMesh:
    def __init__(self, mesh_size: MeshSize, segment: Segment,
                 start_n_id, start_e_id):
        """
        Class for generating FE mesh on a selected segment.

        :param mesh_size: Selected input mesh size for segment mesh generation.
        :param segment: Selected segment for generating Noodes and Elements.
        :param start_n_id: Starting node ID which allows continued numeration.
        :param start_e_id: Starting element ID which allows continued numeration.
        :return: Determines node coordinates and generates finite element Node
            and Element objects on the selected segment. Returns last node and
            element ID, to continue node and element numbering.

        Class contains the following data:

        edge_plate_nodes - Distances between nodes, at the point of connection
            of primary supporting member web with plating, along the length
            of the selected segment.
        edge_flange_nodes - Distances between nodes, at the point of connection
            of primary supporting member web with its flange, along the length
            of the selected segment.
        edge_nodes_z - Distances between nodes, in order along z axis

        web_node_ref_array - Reference array for generating web nodes.
        """
        self._mesh_size = mesh_size
        self._mesh_extent = self._mesh_size.mesh_extent
        self._segment = segment
        self._start_node_id = start_n_id
        self._start_element_id = start_e_id

        self._edge_plate_nodes = {}
        self._edge_flange_nodes = self._mesh_size.flange_edge_node_spacing(segment)
        self._edge_nodes_z = {}

    def get_plate_edge_nodes(self):
        """
        :return: Identifies a plating zone the segment defines and gets
            distances between edge nodes in the appropriate direction, at the
            connection of segment web and plating zone.
            Loads identified dimensions into dictionary edge_plate_nodes,
            which are necessary for web node generation.
            This method is the first step of generating any segment nodes.
        """
        direction = self._segment.primary_supp_mem.direction
        for plate in self._mesh_extent.all_plating_zones.values():
            segment_defines_plate = plate.test_plate_segment(self._segment)
            if segment_defines_plate:
                if direction is BeamDirection.LONGITUDINAL:
                    edge_nodes = self._mesh_size.plate_edge_node_spacing_x(plate)
                else:
                    edge_nodes = self._mesh_size.plate_edge_node_spacing_y(plate)
                self._edge_plate_nodes = edge_nodes
                break

    def get_inward_flange_vector(self):
        """
        :return: Inward flange direction unit vector, based on Primary
            supporting member direction and relative distance.
```

```
        """
        direction = self._segment.primary_supp_mem.direction
        rel_dist = self._segment.primary_supp_mem.rel_dist

        if direction is BeamDirection.LONGITUDINAL:
            if rel_dist < 0.5:
                return np.array((0, 1, 0))
            else:
                return np.array((0, -1, 0))

        if direction is BeamDirection.TRANSVERSE:
            if rel_dist < 0.5:
                return np.array((1, 0, 0))
            else:
                return np.array((-1, 0, 0))

    def get_outward_flange_vector(self):
        """
        :return: Outward flange direction unit vector, based on Primary
            supporting member direction and relative distance.
        """
        direction = self._segment.primary_supp_mem.direction
        rel_dist = self._segment.primary_supp_mem.rel_dist

        if direction is BeamDirection.LONGITUDINAL:
            if rel_dist < 0.5:
                return np.array((0, -1, 0))
            else:
                return np.array((0, 1, 0))

        if direction is BeamDirection.TRANSVERSE:
            if rel_dist < 0.5:
                return np.array((-1, 0, 0))
            else:
                return np.array((1, 0, 0))

    def get_web_element_property_id(self, fem: GeoGrillageFEM):
        """
        :return: Quad element plate property ID used for primary supporting
            member segment web elements.
        """
        beam_prop_id = self._segment.beam_prop.id
        if self._mesh_extent.aos_on_segment(self._segment):
            fem_prop_id = fem.half_web_property_IDs[beam_prop_id]
        else:
            fem_prop_id = fem.web_property_IDs[beam_prop_id]
        return fem_prop_id

    def get_flange_element_property_id(self, fem: GeoGrillageFEM):
        """
        :return: Quad element plate property ID used for primary supporting
            member segment flange elements.
        """
        beam_prop_id = self._segment.beam_prop.id
        fem_prop_id = fem.flange_property_IDs[beam_prop_id]
        return fem_prop_id

    def reference_web_node_ID_array(self):
        """
        :return: 2D array of node IDs arranged to represent relative placement
```

```
            of nodes on the primary supporting member segment web. Used as a
            reference for quad element generation.
            Version 1 assumes equal number of nodes in each row and quad
            elements with edges parallel to the global coordinate axis.

        row_limit - Number of web nodes along global z axis.
        column_limit - Number of nodes along the local longitudinal segment axis.
        total - Total number of web nodes.
        """
        column_limit = len(self._edge_plate_nodes) + 1
        row_limit = self._mesh_size.min_num_eweb + 1
        total = row_limit * column_limit
        id_list = np.arange(self._start_node_id, self._start_node_id + total, 1)
        web_node_id_array = np.reshape(id_list, [row_limit, column_limit])
        return web_node_id_array

    def plate_node_row_number(self):
        """
        :return: Number of node rows with plate edge spacing along the web.
        """
        eweb = self._mesh_size.min_num_eweb
        if eweb <= 2:
            p_row_limit = 1
        else:
            p_row_limit = eweb - 1
        return p_row_limit

    def flange_node_row_number(self):
        """
        :return: Number of node rows with flange edge spacing along the web.
        """
        eweb = self._mesh_size.min_num_eweb
        if eweb == 1:
            f_row_limit = 1
        else:
            f_row_limit = 2
        return f_row_limit

    def reference_web_node_ID_list(self):
        """
        :return: List of node IDs arranged to represent relative placement
            of nodes on the primary supporting member segment web. Used as a
            reference for quad element generation. For MeshVariantV2.

        row_limit - Number of web nodes along global z axis.
        column_limit - Number of nodes along the local longitudinal segment axis.
        total - Total number of web nodes.
        """
        web_node_id_list = []
        p_column_limit = len(self._edge_plate_nodes) + 1       # Plate col limit
        f_column_limit = len(self._edge_flange_nodes) + 1      # Flange col limit
        p_row_limit = self.plate_node_row_number()             # Plate row limit
        f_row_limit = self.flange_node_row_number()            # Flange row limit

        start_id = self._start_node_id
        for p_row in range(1, p_row_limit + 1):
            id_list = np.arange(start_id, start_id + p_column_limit, 1)
            start_id += p_column_limit
            web_node_id_list.append(id_list)

```

```
        for f_row in range(1, f_row_limit + 1):
            id_list = np.arange(start_id, start_id + f_column_limit, 1)
            start_id += f_column_limit
            web_node_id_list.append(id_list)

        return web_node_id_list

    def generate_web_nodes(self, fem: GeoGrillageFEM) -> int:
        pass

    def generate_web_elements(self, fem: GeoGrillageFEM) -> int:
        pass

    def ref_flange_node_ID_array(self, flange_start_node):
        """
        :return: 2D array of node IDs arranged to represent relative placement
            of nodes on the primary supporting member segment flange of L beam
            type. Used as a reference for quad element generation.
            Method uses common nodes at the connection of web and flange.

        last_web_node_row - List of common node IDs at the connection.
        row_limit - Number of nodes in the direction of flange width.
        column_limit - Number of nodes along the local longitudinal segment axis.
        total - Total number of flange nodes, excluding the middle row.
        """

        web_node_id_list = self.reference_web_node_ID_list()
        last_web_node_row = web_node_id_list[-1]

        column_limit = len(self._edge_flange_nodes) + 1
        row_limit = self._mesh_size.num_eaf + 1
        total = column_limit * (row_limit - 1)
        id_list = np.arange(flange_start_node, flange_start_node + total, 1)
        id_array = np.reshape(id_list, [row_limit - 1, column_limit])
        flange_node_id_array = np.insert(id_array, 0, last_web_node_row, axis=0)
        return flange_node_id_array

    def generate_flange_nodes(self, fem: GeoGrillageFEM,
                              direction: FlangeDirection, start_node_id) -> int:
        pass

    def generate_flange_elements(self, fem: GeoGrillageFEM, flange_start_node,
                                 start_element_id):
        element_id = start_element_id
        node_id_arr = self.ref_flange_node_ID_array(flange_start_node)
        row_limit, column_limit = np.shape(node_id_arr)
        flange_id_array = self.ref_flange_node_ID_array(flange_start_node)
        fem_prop_id = self.get_flange_element_property_id(fem)
        id_el_nodes = [None] * 4

        for row in range(0, row_limit - 1):
            for column in range(0, column_limit - 1):
                id_el_nodes[0] = flange_id_array[row, column]
                id_el_nodes[1] = flange_id_array[row, column + 1]
                id_el_nodes[2] = flange_id_array[row + 1, column + 1]
                id_el_nodes[3] = flange_id_array[row + 1, column]
                elem = fem.add_quad_element(fem_prop_id, id_el_nodes)
                fem.add_element_to_element_overlaps(elem)
                element_id += 1
        return element_id
```

```
3481
3482     def generate_mesh(self, fem: GeoGrillageFEM):
3483         """
3484         :return: Generates all nodes and elements on a segment of a primary
3485             supporting member. Returns last node and element ID to continue
3486             numeration on other segments.
3487         """
3488         nodes = self._start_node_id
3489         elements = self._start_element_id
3490
3491         self.get_plate_edge_nodes()
3492         beam_type = self._segment.beam_prop.beam_type
3493         flange_dir = self._segment.primary_supp_mem.flange_direction
3494
3495         if beam_type is BeamType.T:
3496             web_nodes = self.generate_web_nodes(fem)
3497             web_elements = self.generate_web_elements(fem)
3498             direction = FlangeDirection.INWARD
3499             flange_nodes = self.generate_flange_nodes(fem, direction, web_nodes)
3500             elements = self.generate_flange_elements(fem, web_nodes, web_elements)
3501
3502             if not self._mesh_extent.aos_on_segment(self._segment):
3503                 direction = FlangeDirection.OUTWARD
3504                 nodes = self.generate_flange_nodes(fem, direction, flange_nodes)
3505                 elements = self.generate_flange_elements(fem, flange_nodes, elements)
3506             else:
3507                 nodes = flange_nodes
3508
3509         elif beam_type is BeamType.L:
3510             web_nodes = self.generate_web_nodes(fem)
3511             elements = self.generate_web_elements(fem)
3512             nodes = self.generate_flange_nodes(fem, flange_dir, web_nodes)
3513             elements = self.generate_flange_elements(fem, web_nodes, elements)
3514
3515         elif beam_type is BeamType.FB:
3516             nodes = self.generate_web_nodes(fem)
3517             elements = self.generate_web_elements(fem)
3518
3519         return nodes, elements
3520
3521
3522 class SegmentMeshV1(SegmentMesh):
3523     def __init__(self, mesh_size: MeshSize, segment: Segment,
3524                  start_n_id, start_e_id):
3525         """
3526         CLass for segment mesh generation specific to meshing variant V1.
3527
3528         Distances between flange nodes are equal to plate edge node distances
3529         on meshing variant V1.
3530         """
3531         super().__init__(mesh_size, segment, start_n_id, start_e_id)
3532         self._mesh_size = mesh_size
3533         self._segment = segment
3534         self._start_node_id = start_n_id        # Starting node ID
3535         self._start_element_id = start_e_id     # Starting element ID
3536
3537     def generate_web_nodes(self, fem: GeoGrillageFEM):
3538         """
3539         :return: Generates nodes on the web of one segment of a primary
3540             supporting member and returns last node ID to continue node
```

```
            numeration on other segments.

        row_limit - Number of web nodes along global z axis.
        column_limit - Number of nodes along the local longitudinal segment axis.
        dim_z - Vertical dimension of every web element.
        ref_node1 - Reference node 1 coordinates in [mm], origin of the local csy.
        ref_vector - Reference vector in the direction of the local csy.
        perpendicular_vector - Vector opposite of global z axis.
        long_spacing_vector - Longitudinal vector in the direction of PSM.
        position_vector - Node position vector in the local coordinate system.
        """
        row_limit = self._mesh_size.min_num_eweb + 1
        column_limit = int(len(self._edge_plate_nodes) + 1)
        dim_z = self._mesh_size.get_web_el_height(self._segment)
        mesh_dim = self._edge_plate_nodes
        eaf = self._mesh_size.num_eaf

        node_id = self._start_node_id
        ref_node1 = Segment.get_segment_node1(self._segment)
        ref_node2 = Segment.get_segment_node2(self._segment)
        ref_vector = np.subtract(ref_node2, ref_node1)
        unit_ref_vector = ref_vector / np.linalg.norm(ref_vector)
        perpendicular_vector = np.array((0, 0, -1))
        long_spacing_vector = np.zeros(3)

        for row in range(0, row_limit):
            vertical_spacing_vector = perpendicular_vector * dim_z * row
            long_dim_index = 1
            for column in range(0, column_limit):
                if column > 0:
                    long_mesh_dim = mesh_dim[long_dim_index]
                    long_spacing_vector += long_mesh_dim * unit_ref_vector
                    long_dim_index += 1
                else:
                    long_spacing_vector = np.zeros(3)

                position_vector = long_spacing_vector + vertical_spacing_vector
                node_coords = position_vector + ref_node1
                node = fem.add_node(node_coords)

                if row == 0:
                    fem.add_node_to_node_overlaps(node)
                if row == row_limit - 1 and column <= eaf:
                    fem.add_node_to_node_overlaps(node)
                if row == row_limit - 1 and column >= column_limit - eaf - 1:
                    fem.add_node_to_node_overlaps(node)
                if column == 0 or column == column_limit - 1:
                    fem.add_node_to_node_overlaps(node)
                node_id += 1

        return node_id

    def ref_flange_node_ID_array(self, flange_start_node):
        """
        :return: 2D array of node IDs arranged to represent relative placement
            of nodes on the primary supporting member segment flange of L beam
            type. Used as a reference for quad element generation.
            Method uses common nodes at the connection of web and flange.

        last_web_node_row - List of common node IDs at the connection.
```

```
        row_limit - Number of nodes in the direction of flange width.
        column_limit - Number of nodes along the local longitudinal segment axis.
        total - Total number of flange nodes, excluding the middle row.
        """
        web_node_id_array = self.reference_web_node_ID_array()
        last_web_node_row = web_node_id_array[-1, :]
        column_limit = len(self._edge_plate_nodes) + 1
        n_eaf = self._mesh_size.num_eaf
        row_limit = n_eaf + 1
        total = column_limit * (row_limit - 1)

        id_list = np.arange(flange_start_node, flange_start_node + total, 1)
        id_array = np.reshape(id_list, [row_limit - 1, column_limit])
        flange_node_id_array = np.insert(id_array, n_eaf, last_web_node_row, axis=0)
        return flange_node_id_array

    def generate_flange_nodes(self, fem: GeoGrillageFEM,
                              direction: FlangeDirection, flange_start_node):
        """
        :param fem: Grillage FEM model.
        :param direction: Selected flange direction for node generation.
        :param flange_start_node: Start flange node for continued node
            generation after web nodes.
        :return: Generates nodes on one side of the flange of one segment of a
            primary supporting member. Returns last node ID to continue node
            numeration on other segments.

        row_limit - Number of flange nodes across the width
        column_limit - Number of nodes along the local longitudinal segment axis.
        mesh_dim - Distances between nodes at the connection of web and flange.
        ref_node1 - Reference node 1 coordinates in [mm], origin of the local csy.
        ref_vector - Reference vector in the direction of the local csy.
        perpendicular_vector - Vector opposite of global z axis.
        long_spacing_vector - Longitudinal vector in the direction of PSM.
        position_vector - Node position vector in the local coordinate system.
        """

        if direction is FlangeDirection.INWARD:
            flange_unit_vector = self.get_inward_flange_vector()
        else:
            flange_unit_vector = self.get_outward_flange_vector()
        ref_array = self.ref_flange_node_ID_array(flange_start_node)
        row_limit, column_limit = np.shape(ref_array)
        row_limit -= 1
        eaf = self._mesh_size.num_eaf
        edge_nodes = self._edge_plate_nodes

        ref_node1 = Segment.get_segment_node1(self._segment)
        ref_node2 = Segment.get_segment_node2(self._segment)
        ref_vector = np.subtract(ref_node2, ref_node1)
        unit_ref_vector = ref_vector / np.linalg.norm(ref_vector)

        width_spacing_vector = np.zeros(3)
        long_spacing_vector = np.zeros(3)

        fl_el_width = self._mesh_size.get_flange_el_width(self._segment)
        z_start_offset = ref_node1 * np.array((1, 1, 0))
        width_offset = flange_unit_vector * fl_el_width * eaf
        start_node = z_start_offset + width_offset
        node_id = flange_start_node
```

```
3661         for row in range(0, row_limit):
3662             if row > 0:
3663                 width_spacing_vector -= flange_unit_vector * fl_el_width
3664             else:
3665                 width_spacing_vector = np.zeros(3)
3666 
3667             dim_index = 1
3668             for column in range(0, column_limit):
3669                 if column > 0:
3670                     long_spacing_vector += edge_nodes[dim_index] * unit_ref_vector
3671                     dim_index += 1
3672                 else:
3673                     long_spacing_vector = np.zeros(3)
3674 
3675                 position_vector = long_spacing_vector + width_spacing_vector
3676                 node_coords = position_vector + start_node
3677 
3678                 node = fem.add_node(node_coords)
3679                 if column >= (column_limit - eaf - 1):
3680                     fem.add_node_to_node_overlaps(node)
3681                 if column <= eaf:
3682                     fem.add_node_to_node_overlaps(node)
3683                 node_id += 1
3684 
3685         return node_id
3686 
3687     def generate_web_elements(self, fem: GeoGrillageFEM):
3688         """
3689         :return: Generates elements on the entire segment web and returns last
3690             element ID to continue element numeration on other segments.
3691 
3692         row_limit - Row of elements along x axis.
3693         column_limit - Column of elements along y axis.
3694         """
3695         column_limit = len(self._edge_plate_nodes) + 1
3696         element_id = fem.id_element_count
3697         node_id_array = self.reference_web_node_ID_array()
3698         fem_prop_id = self.get_web_element_property_id(fem)
3699         id_el_nodes = [None] * 4
3700         for row in range(0, self._mesh_size.min_num_eweb):
3701             for column in range(0, column_limit - 1):
3702                 id_el_nodes[0] = node_id_array[row, column]
3703                 id_el_nodes[1] = node_id_array[row, column + 1]
3704                 id_el_nodes[2] = node_id_array[row + 1, column + 1]
3705                 id_el_nodes[3] = node_id_array[row + 1, column]
3706                 fem.add_quad_element(fem_prop_id, id_el_nodes)
3707                 element_id += 1
3708 
3709         return element_id
3710 
3711 
3712 class SegmentMeshV2(SegmentMesh):
3713     def __init__(self, mesh_size: MeshSize, segment: Segment,
3714                  start_n_id, start_e_id):
3715         """
3716         CLass for segment mesh generation specific to meshing variant V2.
3717 
3718         Distances between flange and plate edge nodes do not have to be equal.
3719         Transition mesh on the segment web uses both deformed quad elements and
3720         triangle elements.
```

```
3721         """
3722         super().__init__(mesh_size, segment, start_n_id, start_e_id)
3723         self._mesh_size = mesh_size
3724         self._segment = segment
3725         self._start_node_id = start_n_id
3726         self._start_element_id = start_e_id
3727 
3728     def identify_num_of_tris(self, segment: Segment):
3729         """
3730         :param segment: Selected segment.
3731         :return: Number of triangles at the start and end of transition row.
3732 
3733         Number of triangles depends on the beam type of segments in the
3734         perpendicular direction to the segment being meshed.
3735         """
3736         num_end1, num_end2 = self._mesh_size.opposite_flange_element_num(segment)
3737         n_tri1 = num_end1
3738         n_tri2 = num_end2
3739 
3740         plate_edge_nodes = self._edge_plate_nodes
3741         flange_edge_nodes = self._edge_flange_nodes
3742         if len(plate_edge_nodes) > 4:
3743             model_check = self._mesh_extent.model_check
3744             model_check.mesh_V2_tr_check(plate_edge_nodes, flange_edge_nodes)
3745 
3746         p1_end1 = plate_edge_nodes[1]
3747         p1_end2 = plate_edge_nodes[len(plate_edge_nodes)]
3748         f1_end1 = flange_edge_nodes[1]
3749         f1_end2 = flange_edge_nodes[len(flange_edge_nodes)]
3750 
3751         # No triangle element if first element dimensions are equal
3752         if np.isclose(p1_end1, f1_end1) or f1_end1 > p1_end1:
3753             n_tri1 = 0
3754         if np.isclose(p1_end2, f1_end2) or f1_end2 > p1_end2:
3755             n_tri2 = 0
3756 
3757         # No triangle no.2 if only half of the segment is to be meshed
3758         if segment in self._mesh_size.mesh_extent.half_segments.values():
3759             n_tri2 = 0
3760 
3761         p2_end1 = plate_edge_nodes[2]
3762         p2_end2 = plate_edge_nodes[len(plate_edge_nodes) - 1]
3763         f2_end1 = flange_edge_nodes[2]
3764         f2_end2 = flange_edge_nodes[len(flange_edge_nodes) - 1]
3765 
3766         plate_end1 = p1_end1 + p2_end1
3767         plate_end2 = p1_end2 + p2_end2
3768         flange_end1 = f1_end1 + f2_end1
3769         flange_end2 = f1_end2 + f2_end2
3770 
3771         # No triangle element if the sum of first two element dims are equal
3772         if np.isclose(plate_end1, flange_end1):
3773             n_tri1 = 0
3774         if np.isclose(plate_end2, flange_end2):
3775             n_tri2 = 0
3776 
3777         return n_tri1, n_tri2
3778 
3779     def generate_web_nodes(self, fem: GeoGrillageFEM):
3780         """
```

```
        :return: Generates nodes on the web of one segment of a primary
            supporting member and returns last node ID to continue node
            numeration on other segments.

        row_limit - Number of web nodes along global z axis.
        column_limit - Number of nodes along the local longitudinal segment axis.
        dim_z - Vertical dimension of every web element.
        ref_node1 - Reference node 1 coordinates in [mm], origin of the local csy.
        ref_vector - Reference vector in the direction of the local csy.
        perpendicular_vector - Vector opposite of global z axis.
        long_spacing_vector - Longitudinal vector in the direction of PSM.
        position_vector - Node position vector in the local coordinate system.
        """
        plate_edge_nodes = self._edge_plate_nodes       # Distances between plate nodes
        flange_edge_nodes = self._edge_flange_nodes     # Distances between flange nodes
        p_row_limit = self.plate_node_row_number()      # Plate row limit
        f_row_limit = self.flange_node_row_number()     # Flange row limit
        p_column_limit = int(len(plate_edge_nodes) + 1)   # Plate col limit
        f_column_limit = int(len(flange_edge_nodes) + 1)  # Flange col limit
        dim_z = self._mesh_size.get_web_el_height(self._segment)
        node_id = self._start_node_id
        eaf = self._mesh_size.num_eaf
        ref_node1 = Segment.get_segment_node1(self._segment)
        ref_node2 = Segment.get_segment_node2(self._segment)
        ref_vector = np.subtract(ref_node2, ref_node1)
        unit_ref_vector = ref_vector / np.linalg.norm(ref_vector)
        perpendicular_vector = np.array((0, 0, -1))
        long_spacing_vector = np.zeros(3)

        for row in range(0, p_row_limit):
            vertical_spacing_vector = perpendicular_vector * dim_z * row
            long_dim_index = 1
            for column in range(0, p_column_limit):
                if column > 0:
                    long_mesh_dim = plate_edge_nodes[long_dim_index]
                    long_spacing_vector += long_mesh_dim * unit_ref_vector
                    long_dim_index += 1
                else:
                    long_spacing_vector = np.zeros(3)

                position_vector = long_spacing_vector + vertical_spacing_vector
                node_coords = position_vector + ref_node1

                node = fem.add_node(node_coords)
                if row == 0:
                    fem.add_node_to_node_overlaps(node)
                if column >= (p_column_limit - eaf):
                    fem.add_node_to_node_overlaps(node)
                if column <= eaf - 1:
                    fem.add_node_to_node_overlaps(node)

                node_id += 1

        for row in range(p_row_limit, p_row_limit + f_row_limit):
            vertical_spacing_vector = perpendicular_vector * dim_z * row
            long_dim_index = 1
            for column in range(0, f_column_limit):
                if column > 0:
                    long_mesh_dim = flange_edge_nodes[long_dim_index]
                    long_spacing_vector += long_mesh_dim * unit_ref_vector
```

```
                    long_dim_index += 1
                else:
                    long_spacing_vector = np.zeros(3)

                position_vector = long_spacing_vector + vertical_spacing_vector
                node_coords = position_vector + ref_node1

                node = fem.add_node(node_coords)

                if column >= (f_column_limit - eaf - 1):
                    fem.add_node_to_node_overlaps(node)
                if column <= eaf + 1:
                    fem.add_node_to_node_overlaps(node)

                node_id += 1

        return node_id

    def generate_flange_nodes(self, fem: GeoGrillageFEM,
                              direction: FlangeDirection, flange_start_node):
        """
        :param fem: Grillage FEM model.
        :param direction: Selected flange direction for node generation.
        :param flange_start_node: Start flange node for continued node
            generation after web nodes.
        :return: Generates nodes on one side of the flange of one segment of a
            primary supporting member. Returns last node ID to continue node
            numeration on other segments.

        row_limit - Number of flange nodes across the width
        column_limit - Number of nodes along the local longitudinal segment axis.
        mesh_dim - Distances between nodes at the connection of web and flange.
        ref_node1 - Reference node 1 coordinates in [mm], origin of the local csy.
        ref_vector - Reference vector in the direction of the local csy.
        perpendicular_vector - Vector opposite of global z axis.
        long_spacing_vector - Longitudinal vector in the direction of PSM.
        position_vector - Node position vector in the local coordinate system.
        """

        if direction is FlangeDirection.INWARD:
            flange_unit_vector = self.get_inward_flange_vector()
        else:
            flange_unit_vector = self.get_outward_flange_vector()
        ref_array = self.ref_flange_node_ID_array(flange_start_node)

        row_limit, column_limit = np.shape(ref_array)
        row_limit -= 1
        eaf = self._mesh_size.num_eaf
        mesh_dim = self._edge_flange_nodes

        ref_node1 = Segment.get_segment_node1(self._segment)
        ref_node2 = Segment.get_segment_node2(self._segment)
        ref_vector = np.subtract(ref_node2, ref_node1)
        unit_ref_vector = ref_vector / np.linalg.norm(ref_vector)

        long_spacing_vector = np.zeros(3)

        fl_el_dim = self._mesh_size.get_flange_el_width(self._segment)
        fl_el_dim_end1 = self._mesh_size.get_end1_max_flange_width(self._segment)
        fl_el_dim_end2 = self._mesh_size.get_end2_max_flange_width(self._segment)
```

```
        z_start_offset = ref_node1 * np.array((1, 1, 0))
        start_node = z_start_offset + flange_unit_vector * fl_el_dim
        half_segments = self._mesh_size.mesh_extent.half_segments.values()
        node_id = flange_start_node

        for row in range(0, row_limit):
            dim_index = 1
            for column in range(0, column_limit):
                if row > 0:
                    width_spacing_vector = flange_unit_vector * fl_el_dim
                else:
                    width_spacing_vector = np.zeros(3)

                if column > 0:
                    long_spacing_vector += mesh_dim[dim_index] * unit_ref_vector
                    dim_index += 1
                else:
                    long_spacing_vector = np.zeros(3)

                if column < 2:
                    width_spacing_vector = flange_unit_vector * \
                                           (fl_el_dim_end1 - fl_el_dim)

                elif column > column_limit - 3 \
                        and self._segment not in half_segments:
                    width_spacing_vector = flange_unit_vector * \
                                           (fl_el_dim_end2 - fl_el_dim)

                position_vector = long_spacing_vector + width_spacing_vector
                node_coords = position_vector + start_node
                node = fem.add_node(node_coords)

                if column >= (column_limit - eaf - 1):
                    fem.add_node_to_node_overlaps(node)
                if column <= eaf + 1:
                    fem.add_node_to_node_overlaps(node)

                node_id += 1

        return node_id

    def top_web_element_row(self, fem: GeoGrillageFEM, start_element_id):
        """
        :param fem: Grillage FEM model.
        :param start_element_id: Starting element ID for the row.
        :return: Generates quad elements on the top row.
        """
        plate_edge_nodes = self._edge_plate_nodes
        ref_node_list = self.reference_web_node_ID_list()
        p_row_limit = self.plate_node_row_number()
        element_id = start_element_id
        fem_prop_id = self.get_web_element_property_id(fem)
        range_end = len(plate_edge_nodes)
        for row in range(0, p_row_limit - 1):
            local_id = 0
            for quad_id in range(local_id, range_end):
                node1 = ref_node_list[row][local_id]
                node2 = ref_node_list[row][local_id + 1]
                node3 = ref_node_list[row + 1][local_id + 1]
                node4 = ref_node_list[row + 1][local_id]
```

```
                node_id_list = [node1, node2, node3, node4]
                fem.add_quad_element(fem_prop_id, node_id_list)

                element_id += 1
                local_id += 1
        return element_id

    def add_first_quad_to_row(self, fem: GeoGrillageFEM, row1, row2, l_id, num):
        """
        :param fem:  Grillage FEM model.
        :param row1: Input node ID list, upper row of nodes.
        :param row2: Input node ID list, lower row of nodes.
        :param l_id: Local ID of the element in the transition row.
        :param num: Number of quad elements to be generated.
        """
        fem_prop_id = self.get_web_element_property_id(fem)
        for quad_id in range(l_id, l_id + num):
            node1 = row1[quad_id]
            node2 = row1[quad_id + 1]
            node3 = row2[quad_id + 1]
            node4 = row2[quad_id]
            node_id_list = [node1, node2, node3, node4]
            fem.add_quad_element(fem_prop_id, node_id_list)

    def add_first_tria_to_row(self, fem: GeoGrillageFEM, row1, row2, local_id):
        fem_prop_id = self.get_web_element_property_id(fem)
        node1 = row1[local_id]
        node2 = row2[local_id + 1]
        node3 = row2[local_id]
        node_id_list = [node1, node2, node3]
        fem.add_tria_element(fem_prop_id, node_id_list)

    def add_second_tria_to_row(self, fem: GeoGrillageFEM, row1, row2, local_id):
        # Triangle at ref node 2 (right) for n_tri1 == 1:
        fem_prop_id = self.get_web_element_property_id(fem)
        node1 = row1[local_id - 1]
        node2 = row2[local_id + 1]
        node3 = row2[local_id]
        node_id_list = [node1, node2, node3]
        fem.add_tria_element(fem_prop_id, node_id_list)

    def add_quad_after_tria(self, fem: GeoGrillageFEM, row1, row2, local_id, el_num):
        # Central (non deformed) quad elements after a first tria element
        fem_prop_id = self.get_web_element_property_id(fem)
        for quad_id in range(local_id, local_id + el_num):
            node1 = row1[quad_id - 1]
            node2 = row1[quad_id]
            node3 = row2[quad_id + 1]
            node4 = row2[quad_id]
            node_id_list = [node1, node2, node3, node4]
            fem.add_quad_element(fem_prop_id, node_id_list)

    def add_last_quad_to_row(self, fem: GeoGrillageFEM, row1, row2, local_id, el_num):
        # Last element: deformed quad for n_tria1 == 1
        fem_prop_id = self.get_web_element_property_id(fem)
        for quad_id in range(local_id, local_id + el_num):
            node1 = row1[local_id - 2]
            node2 = row1[local_id - 1]
            node3 = row2[local_id + 1]
            node4 = row2[local_id]
```

```
4021            node_id_list = [node1, node2, node3, node4]
4022            fem.add_quad_element(fem_prop_id, node_id_list)
4023
4024    def add_def_quad_end2(self, fem: GeoGrillageFEM, row1, row2, local_id, el_num):
4025        # Last element: deformed quad for
4026        fem_prop_id = self.get_web_element_property_id(fem)
4027        for quad_id in range(local_id, local_id + el_num):
4028            node1 = row1[local_id - 1]
4029            node2 = row1[local_id]
4030            node3 = row2[local_id + 1]
4031            node4 = row2[local_id]
4032            node_id_list = [node1, node2, node3, node4]
4033            fem.add_quad_element(fem_prop_id, node_id_list)
4034
4035    def tr_web_element_row(self, fem: GeoGrillageFEM, start_element_id):
4036        """
4037        :param fem: Grillage FEM model.
4038        :param start_element_id: Starting element ID for the row.
4039        :return: Generates elements on the transition row using quads and tris.
4040
4041        Method generates elements on the segment transition web row.
4042
4043        flange_edge_nodes - input distances between flange nodes.
4044        p_row_limit - number of node rows with plate edge node spacing.
4045        ref_node_list - list of nodes on the selected segment for each row.
4046        tr_row1 - input node ID list, upper row of nodes.
4047        tr_row2 - input node ID list, lower row of nodes.
4048        n_quad - number of quad elements at the plating.
4049        n_tri1, n_tri2 - number of triangle elements at edge1 and edge2
4050        n_elem - total number of elements in transition row.
4051        n_nd_quad - total number of non deformed quad elements.
4052        """
4053        flange_edge_nodes = self._edge_flange_nodes
4054        p_row_limit = self.plate_node_row_number()
4055        ref_node_list = self.reference_web_node_ID_list()
4056
4057        tr_row1 = ref_node_list[p_row_limit - 1]
4058        tr_row2 = ref_node_list[p_row_limit]
4059
4060        n_tri1, n_tri2 = self.identify_num_of_tris(self._segment)
4061        n_elem = len(flange_edge_nodes)
4062        n_nd_quad = n_elem - 2 * n_tri1 - 2 * n_tri2
4063        end_id = start_element_id + n_elem - 1
4064        local_id = 0
4065
4066        if n_tri1 == 1:
4067            self.add_first_quad_to_row(fem, tr_row1, tr_row2, local_id, 1)
4068            local_id += 1
4069            self.add_first_tria_to_row(fem, tr_row1, tr_row2, local_id)
4070            local_id += 1
4071            self.add_quad_after_tria(fem, tr_row1, tr_row2, local_id, n_nd_quad)
4072        else:
4073            self.add_first_quad_to_row(fem, tr_row1, tr_row2, local_id, n_nd_quad)
4074
4075        if n_tri2 == 1:
4076            local_id += n_nd_quad
4077            if n_tri1 == 1:
4078                self.add_second_tria_to_row(fem, tr_row1, tr_row2, local_id)
4079                local_id += 1
4080                self.add_last_quad_to_row(fem, tr_row1, tr_row2, local_id, 1)
```

```
4081            else:
4082                self.add_first_tria_to_row(fem, tr_row1, tr_row2, local_id)
4083                local_id += 1
4084                self.add_def_quad_end2(fem, tr_row1, tr_row2, local_id, 1)
4085
4086        return end_id
4087
4088    def bot_web_element_row(self, fem: GeoGrillageFEM, start_element_id):
4089        """
4090        :param fem: Grillage FEM model.
4091        :param start_element_id: Starting element ID for the row.
4092        :return: Generates quad elements on the bottom row.
4093        """
4094        flange_edge_nodes = self._edge_flange_nodes
4095        ref_node_list = self.reference_web_node_ID_list()
4096        p_row_limit = self.plate_node_row_number()          # Plate row limit
4097        f_row_limit = self.flange_node_row_number()         # Flange row limit
4098        element_id = start_element_id
4099        range_end = len(flange_edge_nodes)
4100        fem_prop_id = self.get_web_element_property_id(fem)
4101
4102        for row in range(p_row_limit, p_row_limit + f_row_limit - 1):
4103            local_id = 0
4104            for quad_id in range(local_id, range_end):
4105                node1 = ref_node_list[row][local_id]
4106                node2 = ref_node_list[row][local_id + 1]
4107                node3 = ref_node_list[row + 1][local_id + 1]
4108                node4 = ref_node_list[row + 1][local_id]
4109                node_id_list = [node1, node2, node3, node4]
4110                fem.add_quad_element(fem_prop_id, node_id_list)
4111                element_id += 1
4112                local_id += 1
4113        return element_id
4114
4115    def generate_web_elements(self, fem: GeoGrillageFEM):
4116        """
4117        Method for generating web elements one row at a time, using a combination
4118        of quad and triangle elements.
4119
4120        If there are 3 or more rows of elements, the transition row will always
4121        be the second row from the flange.
4122        If there are 2 rows of elements, the transition row is on top.
4123        If there is 1 row of elements, only the transition row exists.
4124
4125        :return: Generates elements on the entire segment web and returns last
4126            element ID to continue element numeration on other segments.
4127        """
4128        if self._mesh_size.min_num_eweb == 1:
4129            end_id = self.tr_web_element_row(fem, self._start_element_id)
4130
4131        elif self._mesh_size.min_num_eweb == 2:
4132            end_id = self.tr_web_element_row(fem, self._start_element_id)
4133            end_id = self.bot_web_element_row(fem, end_id)
4134
4135        else:
4136            end_id = self.top_web_element_row(fem, self._start_element_id)
4137            for row in range(0, self._mesh_size.min_num_eweb - 3):
4138                end_id = self.top_web_element_row(fem, end_id)
4139            end_id = self.tr_web_element_row(fem, end_id)
4140            end_id = self.bot_web_element_row(fem, end_id)
```

```
4141
4142         return end_id
4143
4144
4145 class GrillageMesh:
4146     def __init__(self, grillage: Grillage, axis_of_symm_override: AOS = None):
4147         """
4148         Class for generating FE mesh on the entire grillage model.
4149
4150         :param grillage: Input Grillage model.
4151         :param axis_of_symm_override: Selected Axis of Symmetry override.
4152         """
4153         self._grillage = grillage
4154         self.mesh_extent = MeshExtent(self._grillage, axis_of_symm_override)
4155
4156     def generate_FEM_property(self, fem: GeoGrillageFEM):
4157         self.mesh_extent.generate_FEM_material(fem)
4158         self.mesh_extent.generate_FEM_plate_property(fem)
4159         self.mesh_extent.generate_FEM_beam_property(fem)
4160         self.mesh_extent.generate_half_FEM_beam_property(fem)
4161
4162     def generate_plate_mesh(self, size: MeshSize, fem: GeoGrillageFEM):
4163         for plate in self.mesh_extent.full_plate_zones.values():
4164             pzm = PlateMesh(size, plate)
4165             pzm.generate_mesh(fem)
4166
4167         for plate in self.mesh_extent.long_half_plate_zones.values():
4168             pzm = PlateMesh(size, plate)
4169             pzm.generate_mesh(fem)
4170
4171         for plate in self.mesh_extent.tran_half_plate_zones.values():
4172             pzm = PlateMesh(size, plate)
4173             pzm.generate_mesh(fem)
4174
4175         for plate in self.mesh_extent.quarter_plate_zone.values():
4176             pzm = PlateMesh(size, plate)
4177             pzm.generate_mesh(fem)
4178
4179     def generate_psm_mesh(self, size: MeshSize, fem: GeoGrillageFEM):
4180         pass
4181
4182     def vertical_bc_node_group(self, fem: GeoGrillageFEM):
4183         """
4184         Adds all nodes at the ends of primary supporting members to nodal group
4185         for pinned boundary conditions. Group ID = 1
4186         """
4187         nodes_dict = fem.nodes.values()
4188         end_nodes = self.mesh_extent.identify_both_psm_ends()
4189         coords = [node.p for node in nodes_dict]
4190         id_list = [node.id for node in nodes_dict]
4191         coords_1 = np.expand_dims(coords, 0)
4192         coords_2 = np.expand_dims(end_nodes, 1)
4193         boolean_array = np.isclose(coords_1, coords_2, rtol=1e-3).all(-1)
4194
4195         pinned_boundary_nodes = {}
4196         index_list = np.where(boolean_array)[1]
4197         for index in index_list:
4198             node_id = id_list[index]
4199             node = fem.nodes[node_id]
4200             pinned_boundary_nodes[node.id] = node
```

```
4201
4202            group_id = 1
4203            fem.add_node_group(group_id, pinned_boundary_nodes)
4204
4205        @staticmethod
4206        def origin_bc_node(fem: GeoGrillageFEM):
4207            """
4208            Identifies node at origin and adds it to group for
4209            translation boundary conditions along x and y axis. Group ID: 2
4210            """
4211            nodes_dict = fem.nodes.values()
4212            coords = [node.p for node in nodes_dict]
4213            id_list = [node.id for node in nodes_dict]
4214            coords_1 = np.expand_dims(coords, 0)
4215            coords_2 = np.array([0.0, 0.0, 0.0])
4216            boolean_array = np.isclose(coords_1, coords_2).all(-1)
4217            index_list = np.where(boolean_array)[1]
4218
4219            translation_node = {}
4220            node_id = id_list[index_list[0]]
4221            node = fem.nodes[node_id]
4222            translation_node[node.id] = node
4223
4224            group_id = 2
4225            fem.add_node_group(group_id, translation_node)
4226
4227        @staticmethod
4228        def long_symm_bc_node_group(fem: GeoGrillageFEM):
4229            """
4230            :return: Dictionary of all nodes on the longitudinal Axis of Symmetry
4231                for symmetry boundary conditions. Group ID = 4
4232            """
4233            nodes_dict = fem.nodes.values()
4234            y_coords = [node.p[1] for node in nodes_dict]
4235            id_list = [node.id for node in nodes_dict]
4236
4237            y_max = np.max(y_coords)
4238            boolean_array = np.isclose(y_coords, y_max, rtol=1e-2)
4239            node_index = np.where(boolean_array)
4240            node_index = np.concatenate(node_index)
4241
4242            long_symm_nodes = {}
4243            for index in range(0, len(node_index)):
4244                node_id = id_list[node_index[index]]
4245                node = fem.nodes[node_id]
4246                long_symm_nodes[node.id] = node
4247
4248            group_id = 4
4249            fem.add_node_group(group_id, long_symm_nodes)
4250
4251        @staticmethod
4252        def tran_symm_bc_node_group(fem: GeoGrillageFEM):
4253            """
4254            :return: Dictionary of all nodes on the transverse Axis of Symmetry
4255                for symmetry boundary conditions. Group ID = 5
4256            """
4257            nodes_dict = fem.nodes.values()
4258            x_coords = [node.p[0] for node in nodes_dict]
4259            id_list = [node.id for node in nodes_dict]
4260
```

```
        x_max = np.max(x_coords)
        boolean_array = np.isclose(x_coords, x_max, rtol=1e-2)
        node_index = np.where(boolean_array)
        node_index = np.concatenate(node_index)

        tran_symm_nodes = {}
        for index in range(0, len(node_index)):
            node_id = id_list[node_index[index]]
            node = fem.nodes[node_id]
            tran_symm_nodes[node.id] = node

        group_id = 5
        fem.add_node_group(group_id, tran_symm_nodes)

    def generate_pinned_bc(self, fem: GeoGrillageFEM, symmetry: AOS):
        """
        Method generates nodal groups and boundary conditions on the ends
        of Primary Supporting Members based on Axis of Symmetry.
        """
        self.vertical_bc_node_group(fem)
        bc_id = 1
        lc_id = 1
        dof = [3]       # z axis translation
        dof_val = [0.0]
        nodal_group = 1
        fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)

        self.origin_bc_node(fem)
        if symmetry is AOS.LONGITUDINAL:
            bc_id = 2
            dof = [1]       # x axis translation
            dof_val = [0.0]
            nodal_group = 2
            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)

        elif symmetry is AOS.TRANSVERSE:
            bc_id = 2
            dof = [2]       # y axis translation
            dof_val = [0.0]
            nodal_group = 2
            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)

        elif symmetry is AOS.NONE:
            bc_id = 2
            dof = [1, 2]       # x and y axis translation
            dof_val = [0.0, 0.0]
            nodal_group = 2
            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)

    def generate_symm_bc(self, fem: GeoGrillageFEM, symmetry: AOS):
        """
        Method generates nodal groups and symmetry boundary conditions based on
        Axis of Symmetry.
        """
        lc_id = 1

        if symmetry is AOS.LONGITUDINAL:
            self.long_symm_bc_node_group(fem)
            bc_id = 4
            nodal_group = 4
```

```
4321            dof = [2, 4, 6]     # y axis translation, x and z axis rotation
4322            dof_val = [0.0, 0.0, 0.0]
4323            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)
4324
4325        elif symmetry is AOS.TRANSVERSE:
4326            self.tran_symm_bc_node_group(fem)
4327            bc_id = 4
4328            nodal_group = 5
4329            dof = [1, 5, 6]     # x axis translation, y and z axis rotation
4330            dof_val = [0.0, 0.0, 0.0]
4331            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)
4332
4333        elif symmetry is AOS.BOTH:
4334            self.long_symm_bc_node_group(fem)
4335            self.tran_symm_bc_node_group(fem)
4336
4337            bc_id = 4
4338            nodal_group = 4
4339            dof = [2, 4, 6]     # y axis translation, x and z axis rotation
4340            dof_val = [0.0, 0.0, 0.0]
4341            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)
4342
4343            bc_id = 5
4344            nodal_group = 5
4345            dof = [1, 5, 6]     # x axis translation, y and z axis rotation
4346            dof_val = [0.0, 0.0, 0.0]
4347            fem.add_boundary_condition(bc_id, lc_id, dof_val, dof, nodal_group)
4348
4349    @staticmethod
4350    def generate_pressure_load(fem: GeoGrillageFEM, pressure):
4351        lc_id = 1
4352        fem.addLoadCase(lc_id, "Pressure Load Case")
4353        group_id = 6
4354        fem.add_element_group(group_id, fem.plate_elements)
4355        fem.add_pressure_load(1, lc_id, pressure, group_id)
4356
4357    def generate_loadcase(self, fem: GeoGrillageFEM, symmetry, pressure):
4358        self.generate_pinned_bc(fem, symmetry)
4359        self.generate_symm_bc(fem, symmetry)
4360        self.generate_pressure_load(fem, pressure)
4361
4362    @staticmethod
4363    def generate_self_weight(fem: GeoGrillageFEM, gravity):
4364        lc_id = 1
4365        fem.add_self_weight(2, lc_id, gravity)
4366
4367    def generate_grillage_mesh(self, name, ebs, eweb, eaf, far, par, dpar):
4368        pass
4369
4370
4371 class MeshVariantV1(GrillageMesh):
4372    def __init__(self, grillage: Grillage, axis_of_symm_override: AOS = None):
4373        super().__init__(grillage, axis_of_symm_override)
4374
4375    def generate_psm_mesh(self, size: MeshSize, fem: GeoGrillageFEM):
4376        self.mesh_extent.model_check.same_flange_width_test()
4377
4378        n_id = fem.id_node_count
4379        e_id = fem.id_element_count
4380
```

```
        for segment in self.mesh_extent.full_segments.values():
            sm = SegmentMeshV1(size, segment, n_id, e_id)
            n_id, e_id = sm.generate_mesh(fem)

        for segment in self.mesh_extent.half_segments.values():
            sm = SegmentMeshV1(size, segment, n_id, e_id)
            n_id, e_id = sm.generate_mesh(fem)

    def generate_grillage_mesh(self, name, ebs, eweb, eaf, far, par, dpar):
        """
        :param name: Mesh name.
        :param ebs: Number of elements between stiffeners.
        :param eweb: Number of elements representing the web of a primary
            supporting member along its height.
        :param eaf: Number of elements across primary supporting member flange.
        :param far: Maximum PSM flange aspect ratio.
        :param par: Maximum plate and PSM web aspect ratio.
        :param dpar: Desired plating aspect ratio, less than the maximum.
        :return: GeoGrillageFEM object.
        """
        fem = GeoGrillageFEM(name)
        size = ElementSizeV1(self.mesh_extent, ebs, eweb, eaf, far, par, dpar)
        size.calculate_mesh_dimensions()
        self.generate_FEM_property(fem)
        self.generate_plate_mesh(size, fem)
        self.generate_psm_mesh(size, fem)
        fem.merge_coincident_nodes()
        fem.merge_coincident_elements()

        print("Mesh V1 generation complete.")
        return fem


class MeshVariantV2(GrillageMesh):
    def __init__(self, grillage: Grillage, axis_of_symm_override: AOS = None):
        super().__init__(grillage, axis_of_symm_override)

    def generate_psm_mesh(self, size: MeshSize, fem: GeoGrillageFEM):
        n_id = fem.id_node_count
        e_id = fem.id_element_count

        for segment in self.mesh_extent.full_segments.values():
            sm = SegmentMeshV2(size, segment, n_id, e_id)
            n_id, e_id = sm.generate_mesh(fem)

        for segment in self.mesh_extent.half_segments.values():
            sm = SegmentMeshV2(size, segment, n_id, e_id)
            n_id, e_id = sm.generate_mesh(fem)

    def generate_grillage_mesh(self, name, ebs, eweb, eaf, far, par, dpar):
        """
        :param name: Mesh name.
        :param ebs: Number of elements between stiffeners.
        :param eweb: Number of elements representing the web of a primary
            supporting member along its height.
        :param eaf: Number of elements across primary supporting member flange.
        :param far: Maximum PSM flange aspect ratio.
        :param par: Maximum plate and PSM web aspect ratio.
        :param dpar: Desired plating aspect ratio, less than the maximum.
        :return: GeoGrillageFEM object.
```

```
4441         """
4442         fem = GeoGrillageFEM(name)
4443         size = ElementSizeV2(self.mesh_extent, ebs, eweb, eaf, far, par, dpar)
4444         size.calculate_mesh_dimensions()
4445         self.generate_FEM_property(fem)
4446         self.generate_plate_mesh(size, fem)
4447         self.generate_psm_mesh(size, fem)
4448         fem.merge_coincident_nodes()
4449         fem.merge_coincident_elements()
4450 
4451         print("Mesh V2 generation complete.")
4452         return fem
4453 
```

PRILOG V

Programski kod modula grillage_fem.py

```python
import itertools
from femdir.geofem import *
from timeit import default_timer as timer


class GeoGrillageFEM (GeoFEM):
    def __init__(self, name=''):
        """
        Class for generating GeoFEM grillage FEM model.

        plate_elements - All quad elements of the grillage plating.
        initial_node_overlaps - Dictionary of nodes with expected overlaps.
        flange_element_overlaps - Dictionary of elements with expected overlaps.

        Conversion dictionaries for Grillage model properties [key] into GeoFEM
        properties [value]:

        plate_property_IDs - Model Plate properties into GeoFEM PlateProperty
        stiff_beam_prop_IDs - Model stiffener BeamProperty into GeoFEM Beam property
        half_stiff_beam_prop_IDs - Model stiffener BeamProperty with half original
            stiffeness for beams on Axis Of Symmetry into GeoFEM Beam property.
        web_prop_IDs - Model BeamProperty into GeoFEM PlateProperty
        flange_prop_IDs - Model BeamProperty into GeoFEM PlateProperty
        half_web_property_IDs - Model BeamProperty into GeoFEM PlateProperty,
            for Segments on AOS with half web thickness.
        """
        super().__init__(name)
        self.plate_elements = {}
        self.initial_node_overlaps = {}
        self.flange_element_overlaps = {}

        self.plate_property_IDs = {}
        self.stiff_beam_prop_IDs = {}
        self.half_stiff_beam_prop_IDs = {}
        self.web_property_IDs = {}
        self.flange_property_IDs = {}
        self.half_web_property_IDs = {}

        self.id_node_count = 1
        self.id_element_count = 1
        self.id_prop_count = 1

    def add_node(self, node_coords):
        """
        Add generated node to FEM model.
        :param node_coords:
        """
        node = Node(self.id_node_count, node_coords)
        self.addNode(node)
        self.id_node_count += 1
        return node

    def add_node_to_node_overlaps(self, node):
        """
        :param node:
        :return: Add edge nodes to FEM model overlaps dictionary.
        """
        self.initial_node_overlaps[node.id] = node

    def add_element_to_element_overlaps(self, element):
```

```
        """
        :param element:
        :return: Add flange elements to FEM model overlaps dictionary.
        """
        self.flange_element_overlaps[element.id] = element

    def add_to_plate_elements(self, element):
        """
        :param element:
        :return: Add plate elements to FEM model plate elements dictionary.
        """
        self.plate_elements[element.id] = element

    def add_element(self, elem: Element, idProp, nodeIds):
        """
        Add generated element to FEM model.
        :param elem:
        :param idProp: Element property ID.
        :param nodeIds: Node ID list.
        """
        elem.init(self.id_element_count)
        elem.property = self.getProperty(idProp)
        for idNod in nodeIds:
            node = self.getNode(idNod)
            elem.addNode(node)
        self.addElement(elem)
        self.id_element_count += 1
        return elem

    def add_quad_element(self, idProp, nodeIds):
        """
        Add generated quad element to FEM model.
        :param idProp:
        :param nodeIds:
        """
        elem = QuadElement()
        self.add_element(elem, idProp, nodeIds)
        return elem

    def add_tria_element(self, idProp, nodeIds):
        """
        Add generated triangle element to FEM model.
        :param idProp:
        :param nodeIds:
        """
        elem = TriaElement()
        self.add_element(elem, idProp, nodeIds)
        return elem

    def add_beam_element(self, idProp, nodeIds, vect_orient):
        """
        Add generated beam elemenet to FEM model.
        :param idProp:
        :param nodeIds:
        :param vect_orient:
        """
        elem = BeamElementShipStructure()
        dir_vector = BeamOrientationVector(vect_orient)
        elem.set_beam_orientation(dir_vector)
        self.add_element(elem, idProp, nodeIds)
```

```
        return elem

    def add_material(self, material_property):
        """
        :param material_property: Grillage model MaterialProperty object.
        :return: Add materials from grillage model.
            E - modulus of elasticity, [N/mm2]
            v - Poisson's ratio
            ro - material density, [kg/m3]
            Reh - yield strength, [N/mm2]
        """
        gfe_material = Material()
        gfe_material.init(material_property.id, material_property.name)
        gfe_material.E = material_property.E
        gfe_material.ni = material_property.v
        gfe_material.rho = material_property.ro
        gfe_material.ReH = material_property.Reh
        self.addMaterial(gfe_material)

    def add_property(self, prop):
        prop.id = self.id_prop_count
        self.addProperty(prop)
        self.id_prop_count += 1
        return prop

    def add_plate_property(self, prop_id, tp, mat):
        prop = PlateProperty()
        prop.init(id, 'Plate_property_' + str(prop_id))
        prop.tp = tp
        prop.material = mat
        self.add_property(prop)

    def add_T_beam_property(self, name, hw, tw, bf, tf, mat):
        prop = T_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.bf = bf
        prop.tf = tf
        prop.material = mat
        self.add_property(prop)

    def add_half_T_beam_property(self, name, hw, tw, bf, tf, mat):
        prop = Half_T_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.bf = bf
        prop.tf = tf
        prop.material = mat
        self.add_property(prop)

    def add_L_beam_property(self, name, hw, tw, bf, tf, mat):
        prop = L_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.bf = bf
        prop.tf = tf
        prop.material = mat
```

```
        self.add_property(prop)

    def add_half_L_beam_property(self, name, hw, tw, bf, tf, mat):
        prop = Half_L_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.bf = bf
        prop.tf = tf
        prop.material = mat
        self.add_property(prop)

    def add_FB_beam_property(self, name, hw, tw, mat):
        prop = FB_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.material = mat
        self.add_property(prop)

    def add_half_FB_beam_property(self, name, hw, tw, mat):
        prop = Half_FB_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw = hw
        prop.tw = tw
        prop.material = mat
        self.add_property(prop)

    def add_Hat_beam_property(self, name, h, t, bf, fi, mat):
        prop = Hat_Profile_BeamProperty()
        prop.init(id, name)
        prop.h = h
        prop.t = t
        prop.bf = bf
        prop.fi = fi
        prop.material = mat
        self.add_property(prop)

    def add_half_Hat_beam_property(self, name, h, t, bf, fi, mat):
        prop = Half_Hat_Profile_BeamProperty()
        prop.init(id, name)
        prop.h = h
        prop.t = t
        prop.bf = bf
        prop.fi = fi
        prop.material = mat
        self.add_property(prop)

    def add_Bulb_beam_property(self, name, hw_ekv, tw_ekv, bf_ekv, tf_ekv, mat):
        prop = Bulb_Profile_BeamProperty()
        prop.init(id, name)
        prop.hw_ekv = hw_ekv
        prop.tw_ekv = tw_ekv
        prop.bf_ekv = bf_ekv
        prop.tf_ekv = tf_ekv
        prop.material = mat
        self.add_property(prop)

    def add_half_Bulb_beam_property(self, name, hw_ekv, tw_ekv, bf_ekv, tf_ekv, mat):
        prop = Half_Bulb_Profile_BeamProperty()
```

```
        prop.init(id, name)
        prop.hw_ekv = hw_ekv
        prop.tw_ekv = tw_ekv
        prop.bf_ekv = bf_ekv
        prop.tf_ekv = tf_ekv
        prop.material = mat
        self.add_property(prop)

    @staticmethod
    def check_node_overlap(nodes_dict):
        """
        :return: Identifies coincident nodes in nodes_dict and returns
            overlap_list, where the first element are unique coordinates of
            overlapped nodes and the rest are overlapped geofementity Nodes.
        """
        overlap_list = []
        for node in nodes_dict.values():
            duplicate_coords = False
            for row in overlap_list:
                if np.allclose(node.p, row[0]):
                    row.append(node)
                    duplicate_coords = True
                    break
            if duplicate_coords is False:
                overlap_list.append([node.p, node])
        return overlap_list

    def check_node_overlap_np(self, nodes_dict):
        """
        :return: Identifies coincident nodes in nodes_dict and returns
            overlap_list, where the first element are unique coordinates of
            overlapped nodes and the rest are overlapped geofementity Node
            objects. Optimized version with NumPy functions.
        """
        print("Starting coincident node check...")
        start = timer()

        coords = [node.p for node in nodes_dict.values()]
        id_list = [node.id for node in nodes_dict.values()]

        coords_1 = np.expand_dims(coords, 0)
        coords_2 = np.expand_dims(coords, 1)
        # Relative node merge tolerance = 0.1mm
        boolean_array = np.isclose(coords_1, coords_2, rtol=1e-3).all(-1)
        boolean_array = np.tril(boolean_array)
        np.fill_diagonal(boolean_array, False)
        coincident_pairs = np.where(boolean_array)
        x_index, y_index = coincident_pairs
        n_overlaps = len(x_index)

        overlap_list = []
        for index in range(0, n_overlaps):
            node_1_id = id_list[int(x_index[index])]
            node_2_id = id_list[int(y_index[index])]
            node1 = self.nodes[node_1_id]
            node2 = self.nodes[node_2_id]

            duplicate_coords = False
            for row in overlap_list:
                if np.allclose(node1.p, row[0], rtol=1e-3):
```

```
                    if node1 not in row[1:]:
                        row.append(node1)
                    if node2 not in row[1:]:
                        row.append(node2)
                    duplicate_coords = True
                    break

            if duplicate_coords is False:
                overlap_list.append([node1.p, node1, node2])

        end = timer()
        print("Coincident node identification complete, found",
              len(overlap_list), "unique coordinates in", end - start, "s")

        return overlap_list

    def full_model_node_overlap_check(self):
        print("Starting full model coincident node check...")
        nodes_dict = self.nodes.values()
        coords = [node.p for node in nodes_dict]

        coords_1 = np.expand_dims(coords, 0)
        coords_2 = np.expand_dims(coords, 1)
        # Relative node merge tolerance = 0.1mm
        boolean_array = np.isclose(coords_1, coords_2, rtol=1e-3).all(-1)
        boolean_array = np.tril(boolean_array)
        np.fill_diagonal(boolean_array, False)
        x_index, y_index = np.where(boolean_array)
        n_overlaps = len(x_index)

        if n_overlaps > 1:
            print("Full model coincident node identification complete. "
                  "Node overlaps detected. Total overlaps:", n_overlaps)
        else:
            print("Full model coincident node identification complete."
                  " No node overlaps found.")

    @staticmethod
    def sorted_coincident_nodes(overlap_list):
        """
        Method sorts coincident nodes into remaining nodes and to delete nodes.

        :return: merge_nodes dictionary of nodes to be deleted (key) and
            replaced with (value), delete_list of all nodes to be deleted.
        """
        overlap_counter = 0
        merge_nodes = {}
        delete_list = []
        for row in overlap_list:
            for node_to_delete in row[2:]:
                merge_nodes[node_to_delete] = row[1]   # First node remains
                delete_list.append(node_to_delete)
            # print("Coordinates:", row[0], "nodes:",
            #       [node.id for node in row[1:]])
            overlap_counter += len(row[1:])

        print("Total coincident nodes:", overlap_counter)

        return merge_nodes, delete_list

```

```
    def merge_coincident_nodes(self):
        nodes_dict = self.initial_node_overlaps
        overlap_list = self.check_node_overlap_np(nodes_dict)
        merge_nodes, delete_list = self.sorted_coincident_nodes(overlap_list)

        start = timer()

        # Change overlapped nodes for all elements
        for element in self.elements.values():
            local_node_id = 0
            for node in element.nodes:
                if node in delete_list:
                    element.nodes[local_node_id] = merge_nodes[node]
                local_node_id += 1

        # Delete overlapped nodes
        for node in delete_list:
            del self.nodes[node.id]

        end = timer()
        print("Node merge complete, deleted", len(delete_list),
              "nodes in", end - start, "s")
        print("Total number of nodes:", self.num_nodes)
        print("Total number of elements before merge:", self.num_elements)

    @staticmethod
    def check_element_overlap(element_dict):
        overlap_list = []
        element_combos = itertools.combinations(element_dict.values(), 2)
        for elements in element_combos:
            element_1, element_2 = elements
            nodes1 = [node.id for node in element_1.nodes]
            nodes2 = [node.id for node in element_2.nodes]

            if set(nodes1) == set(nodes2):
                overlap_list.append(elements)

        return overlap_list

    def merge_coincident_elements(self):
        element_dict = self.flange_element_overlaps
        overlapped_elements = self.check_element_overlap(element_dict)
        for elements in overlapped_elements:
            element_1, element_2 = elements

            tp_e1 = self.properties[element_1.prop_id].tp
            tp_e2 = self.properties[element_2.prop_id].tp

            if tp_e2 > tp_e1:
                del self.elements[element_1.id]
            else:
                del self.elements[element_2.id]

        print("Total number of elements after merge:", self.num_elements)

    def add_node_group(self, group_id: int, nodes: Dict):
        group = NodeGroup()
        group.init(group_id, "Node_group_" + str(group_id))
        for node in nodes.values():
            group.add_item(node)
```

```
        self.addGroup(group_id, group)

    def add_element_group(self, group_id: int, elements: Dict):
        group = ElementGroup()
        group.init(group_id, "Element_group_" + str(group_id))
        for element in elements.values():
            group.add_item(element)
        self.addGroup(group_id, group)

    def add_boundary_condition(self, bc_id: int, lc_id: int, values: List[float],
                               dof: List[int], set_id: int):
        """
        :param bc_id: Boundary condition ID
        :param lc_id: Load case ID
        :param values: Value for the DoF: 1 or 0; list of same length as dof
        :param dof: Degrees of Freedom: [wx=1, wy=2, wz=3, rx=4, ry=5, rz=6]
        :param set_id: GeoFEM nodal group ID
        """
        group = self.getGroup(set_id)
        bc = GroupNodalBC(bc_id, group, values, dof)
        self.addBoundaryCondition(bc)
        self.addBoundaryConditionToLoadcase(lc_id, bc)

    def add_pressure_load(self, load_id: int, lc_id: int, pressure: float,
                          set_id: int):
        """
        :param load_id: Pressure load ID
        :param lc_id: Load case ID
        :param pressure: External pressure
        :param set_id: GeoFEM element group ID
        :return:
        """
        group = self.getGroup(set_id)
        pressure_load = GroupPressureLoad(load_id, group, pressure, flip=True)
        self.addLoad(pressure_load)
        self.addLoadToLoadcase(lc_id, pressure_load)

    def add_self_weight(self, load_id: int, lc_id: int, gravity_val):
        self_weight = AccelerationLoad(load_id, gravity_val)
        self.addLoad(self_weight)
        self.addLoadToLoadcase(lc_id, self_weight)

```