Vizualizacija modela za izračun granične čvrstoće trupa broda primjenom programa otvorenog koda linaetal-fsb/d3v

Višak, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ivan Višak

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Izv. prof. dr. sc. Pero Prebeg

Student:

Ivan Višak

Zagreb, 2020.

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

Zahvaljujem se svom mentoru izv. prof. dr. sc. Peri Prebegu na pruženoj podršci, pomoći, stručnom vodstvu te susretljivosti prilikom pisanja ovog rada.

Najveću zahvalu dugujem svojoj dragoj obitelji na neizmjernoj podršci tijekom studiranja te svojoj kćeri Eni kojoj posvećujem ovaj rad.

Ivan Višak

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:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Ivan Višak** Mat. br.: 0035197768

Naslov rada na hrvatskom jeziku: **Vizualizacija modela za izračun granične čvrstoće trupa broda primjenom programa otvorenog koda linaetal-fsb/d3v**

Naslov rada na engleskom jeziku: **Use of the open source program linaetal-fsb/d3v for the visualization of the ship hull ultimate strength model**

Opis zadatka:

Zadovoljavanje granične uzdužne čvrstoće trupa broda jedini je globalni kriterij čvrstoće brodske konstrukcije koji se provjerava u konceptualnoj fazi projektiranja konstrukcije tankera za prijevoz nafte i brodova za prijevoz rasutih tereta. Klasifikacijska društva, kao jednu od mogućih metoda izračuna, preporučuju inkrementalno-iterativni pristup pri analizi progresivnog kolapsa, koji je zasnovan na Smithovoj metodi za izračun granične uzdužne čvrstoće trupa broda. Program otvorenog koda linaetal-fsb/d3v temeljen je na *Qt for Python* tehnologiji, koja omogućuje trodimenzijsku (3D) vizualizaciju geometrijskih entiteta primjenom *OpenGL* tehnologije te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. U radu je potrebno, proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python, omogućiti vizualizaciju modela konstrukcije za izračun granične čvrstoće trupa broda Smithovom metodom te vizualizaciju dobivenih rezultata.

Zadatak obuhvaća sljedeće:

- upoznavanje s metodama za izračun uzdužnu granične čvrstoće trupa, koje je moguće primijeniti prema harmoniziranim zajedničkim pravilima za projektiranje konstrukcije brodova za prijevoz rasutog tereta i tankera za prijevoz nafte Međunarodnog udruženja klasifikacijskih društava (IACS CSR BC & OT)
- upoznavanje s modulom za proračun uzdužne granične čvrstoće trupa broda LUSA, koji je razvijen na FSB-u za potrebu provjere tog kriterija prema IACS CSR BC & OT
- upoznavanje s programom otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje 3D vizualizaciju LUSA modela konstrukcije broda u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje 3D vizualizaciju rezultata proračuna dobivenih primjenom modula LUSA u programu otvorenog koda linaetal-fsb/d3v.

U radu treba navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
28. studenog 2019.

Datum predaje rada:
1. rok: 21. veljače 2020.
2. rok (izvanredni): 1. srpnja 2020.
3. rok: 17. rujna 2020.

Predviđeni datumi obrane:
1. rok: 24.2. – 28.2.2020.
2. rok (izvanredni): 3.7.2020.
3. rok: 21.9. - 25.9.2020.

Zadatak zadao:

Doc. dr. sc. Pero Prebeg

Predsjednica Povjerenstva:

Prof. dr. sc. Nastia Degiuli

SADRŽAJ

POPIS SLIKA

SAŽETAK

U radu je proširenjem programa otvorenog koda Lineateal-fsb/d3v koji je temeljen na Qt for Python tehnologiji, izrađen je Python modul koji primjenom OpenGL tehnologije omogućuje trodimezijusku (3D) vizualizaciju geometrijskih entiteta modela za proračun granične čvrstoće brodskog trupa. Modul omogućuje i prikaz rezultata proračuna uzdužne granične čvrstoće trupa broda modulom LUSA koji koristi inkrementalno itereativnu Smithovu metodu.

Ključne riječi: Linaetal-fsb/d3v, OpenGl, Qt, LUSA, Python, granične čvrstoće brodskog trupa

SUMMARY

Abstract (style: TEKST)

The module for visualization of the main rib of the ship is made in the paper, the moments of bending on the main rib of the ship are shown with the help of the graph. By extending the open-source program Lineateal-fsb / d3v based on Qt for Python technology, a hullultstrength model was created that enables three-dimensional (3D) visualization of the geometric entity using OpenGL technology. To show the results of the bending moments in the form of a graph, the calculation of the longitudinal limiting strength of the ship's hull by the LUSA module using the incrementally iterative Smith method was used.

Key words: Linaetal-fsb/d3v, OpenGl, Qt, LUSA, Python, hullultstrength

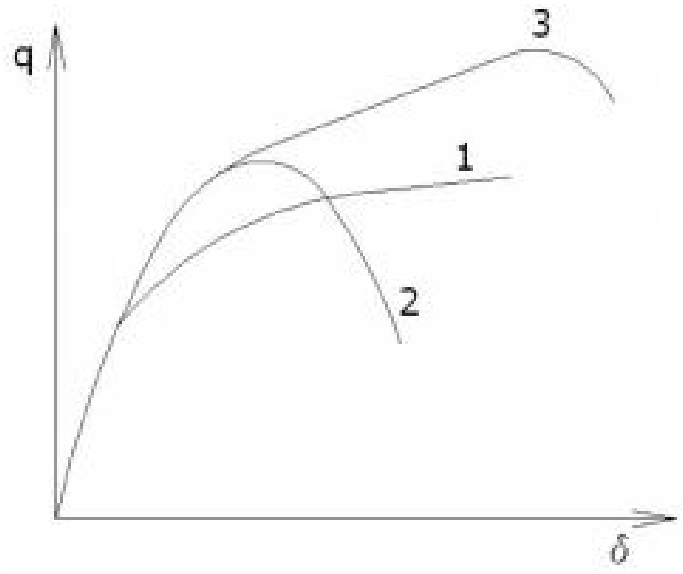
1. UVOD

Zadovoljavanje granične uzdužne čvrstoće trupa broda jedini je globalni kriterij čvrstoće brodske konstrukcije koji se provjerava u konceptualnoj fazi projektiranja konstrukcije tankera za prijevoz nafte i brodova za prijevoz rasutih tereta. Klasifikacijska društva, kao jednu od mogućih metoda izračuna, preporučuju inkrementalno-iterativni pristup pri analizi progresivnog kolapsa, koji je zasnovan na Smithovoj metodi za izračun granične uzdužne čvrstoće trupa broda. Program otvorenog koda linaetal-fsb/d3v temeljen je na Qt for Phyton tehnologiji, koja omogućuje trodimenzijsku (3D) vizualizaciju geometrijskih entiteta primjenom OpenGL tehnologije te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. Cilj ovog rada je proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python, omogućiti vizualizacija modela za proračun granične čvrstoće brodskog trupa te samih rezultata tog proračuna.

2. UZDUŽNA GRANIČNA ČVRSTOĆA TRUPA BRODA

Uzdužna granična čvrstoća broda može se opisati kao stanje naprezanja i deformacije na razini trupa koje odgovara maksimalnom opterećenju (granični moment savijanja) koje broski trup može izdržati. Svako daljnje povećanje momenta savijanja dovodi do kolapsa trupa broda [1].

Klasifikacijska društva sve više vode računa o graničnoj čvrstoći brodskoga trupa. Društva okupljena u IACS-u podupiru razvoj jedinstvenih pravila uzdužne čvrstoće broda uključujući i provjeru granične čvrstoće trupa. Lom pojedinih dijelova i cijeloga trupa broda zbog premašivanja granične čvrstoće je nelinearna pojava, bilo zbog geometrijske nelinearnosti (zbog izvijanja ili drugog znatnog pomaka), bilo zbog nelinearnosti materijala (popuštanje i plastična deformacija). Glavni načini loma čeličnih strukturnih elemenata prikazuju se na osnovi ovisnosti opterećenja q i defl eksije δ za lokalne plastične deformacije (krivulja 1, slika 1), izvijanje nosača (krivulja 2, slika 1) i izvijanje opločenja (krivulja 3, slika 1) [2].



Slika 1. Krivulje ovisnosti opterećenja q i defleksije δ za razne načine loma strukturnih elemenata [2]

Proteklih se godina metoda konačnih elemenata prilagođavala rješavanju nelinearnih problema čvrstoće brodskoga trupa koji uključuju obje nelinearnosti, geometrijsku i materijalnu, uvođenjem inkrementalnog i iterativnog pristupa korištenjem krivulja opterećenje – defleksija [2]. Navedeni postupak je dugotrajan i suviše zahtjevan te iz toga razloga proizlazi potreba za pojednostavljenjem. Nameće se potreba za razvojem adekvatnih alternativnih proračunskih metoda. Metode proračuna uzdužne granične nosivosti složenih tankostijenih konstrukcija:

Pojednostavljene metode:

- Metoda inicijalnog popuštanja
- Elastična analiza
- Metode pretpostavljene raspodjele naprezanja

Sofisticiranije metode:

- NLMKE
- Metode analize progresivnog kolapsa
- Metoda idealiziranih elemenata konstrukcije
- Metoda spregnutih greda[3].

2.1. Metoda inicijalnog popuštanja

Pristup inicijalnog popuštanja implicira mogućnost aproksimacije uzdužne granične nosivosti s vrijednošću čvrstoće inicijalnog popuštanja, izražene u obliku vertikalnog momenta savijanja kao produkta momenta otpora i granice tečenja materijala poprečnog presjeka razmatranog dijela konstrukcije [3].

2.2. Elastična analiza

Elastična analiza podrazumijeva identičan pristup kao metoda inicijalnog popuštanja, pri čemu se umjesto s granicom tečenja materijala moment otpora poprečnog presjeka množi s kritičnim naprezanjem (elastičnog) izvijanja ukrepljene ili neukrepljene oplate dna i/ili palube [3].

2.3. Metode predpostavljene raspodjele naprezanja

Spomenute metode predpostavljene raspodjele naprezanja podrazumijevaju idealizaciju ukrepljenog tankostijenog presjeka razmatrane konstrukcije neukrepljenim presjekom ekvivalentne debljine oplate. Uzimajući u obzir utjecaj izvijanja u tlačnoj zoni presjeka uporabom faktora redukcije naprezanja (koji se množi s granicom tečenja odnosnog materijala) računa se rezultirajući vertikalni moment savijanja, koji se smatra graničnim [3].

2.4. NLMKE

Rezultati primjene NLMKE analize jako ovise o ispravnosti primijenjenih tehnika opisa i idealizacije geometrijskih i materijalnih svojstava diskretiziranog modela konstrukcije i narinutih opterećenja i rubnih uvjeta. Značajniji aspekti primjene:

- Razina razmatranog problema (model cijele konstrukcije ili parcijalni model)
- Dimenzija razmatranog problema (ukupni broj stupnjeva slobode modela)
- Korišteni materijalni modeli (odnos između naprezanja i deformacije za korištene materijale)
- Modeliranje inicijalnih nesavršenosti (deformacije oblika, zaostala naprezanja, korozija) [3].

2.5. Metode analize progresivnog kolapsa

Spomenute metode analize progresivnog kolapsa podrazumijevaju Smithovu metodu i slične. Smithova metoda smatra se rodonačelnom među metodama analize progresivnog kolapsa jer je prva omogućila naprednije razmatranje kolapsne sekvence i poslije-kritične nosivosti elemenata tankostjene konstrukcije opterećene savijanjem, iako je u području zrakoplovstva znatno ranije predložena nešto jednostavnija, ali načelno vrlo slična metoda. Do danas je predloženo nekoliko u osnovi vrlo sličnih metoda zasnovanih na Smithovom pristupu, a među njima i nekolicina kojima se pojednostavljuje izvorna metoda. U području brodogradnje pravila mnogih klasifikacijskih društava, kao i IACS-ova Združena pravila za konstrukciju propisuju korištenje inkrementalno-iterativnih procedura zasnovanih na Smithovoj metodi [4].

2.6. Metoda idealiziranih elemenata konstrukcije

Metoda idealiziranih elemenata konstrukcije predstavlja proračunski pristup temeljen na MKE, ali uz značajno smanjenje broja čvorova (tj. stupnjeva slobode) i iterativnih kalkulacija potrebnih za rješavanje nelinearnih jednadžbi krutosti. Na ovaj način nastoji se značajno smanjiti računalno vrijeme potrebno za analizu i zadržati pritom razumnu razinu točnosti rezultata [3].

2.7. Metoda spregnutih greda

Metoda spregnutih greda razvijena je za proračun uzdužne granične čvrstoće složenih više-palubnih i neprizmatičnih konfiguracija konstrukcije broda s diskontinuitetima. Pri tome se razmatrana konstrukcija diskretizira skupom grednih elemenata koji su međusobno spregnuti posredstvom nelinearnih opruga koje omogućavaju uključivanje efekata smičnog opterećenja u razmatranja [3].

3. SMITHOVA INKREMENTALNO-ITERATIVNA METODA

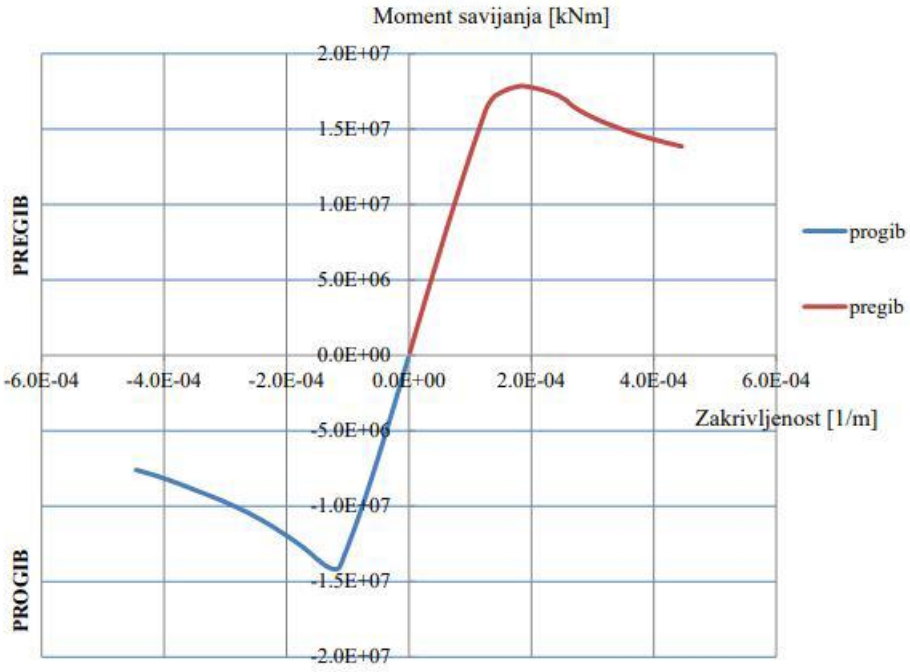
Kod idealiziranog grednog modela globalni odaziv konstrukcije na savojno opterećenje opisuje se Euler-Bernullijevom jednadžbom (1) savijanja grede za složeni tankostjeni presjek:

$$\frac{d^2}{dx^2}\left(EI\frac{d^2w}{dx^2}\right) = -q \;\; ; \quad M = EI\kappa$$

$$\varepsilon = \frac{(\rho - z)d\theta - \rho d\theta}{\rho d\theta} = -\frac{z}{\rho} = -z\kappa \;\; ; \quad \mathrm{K} = \frac{1}{\rho} \qquad \textbf{(1)}$$

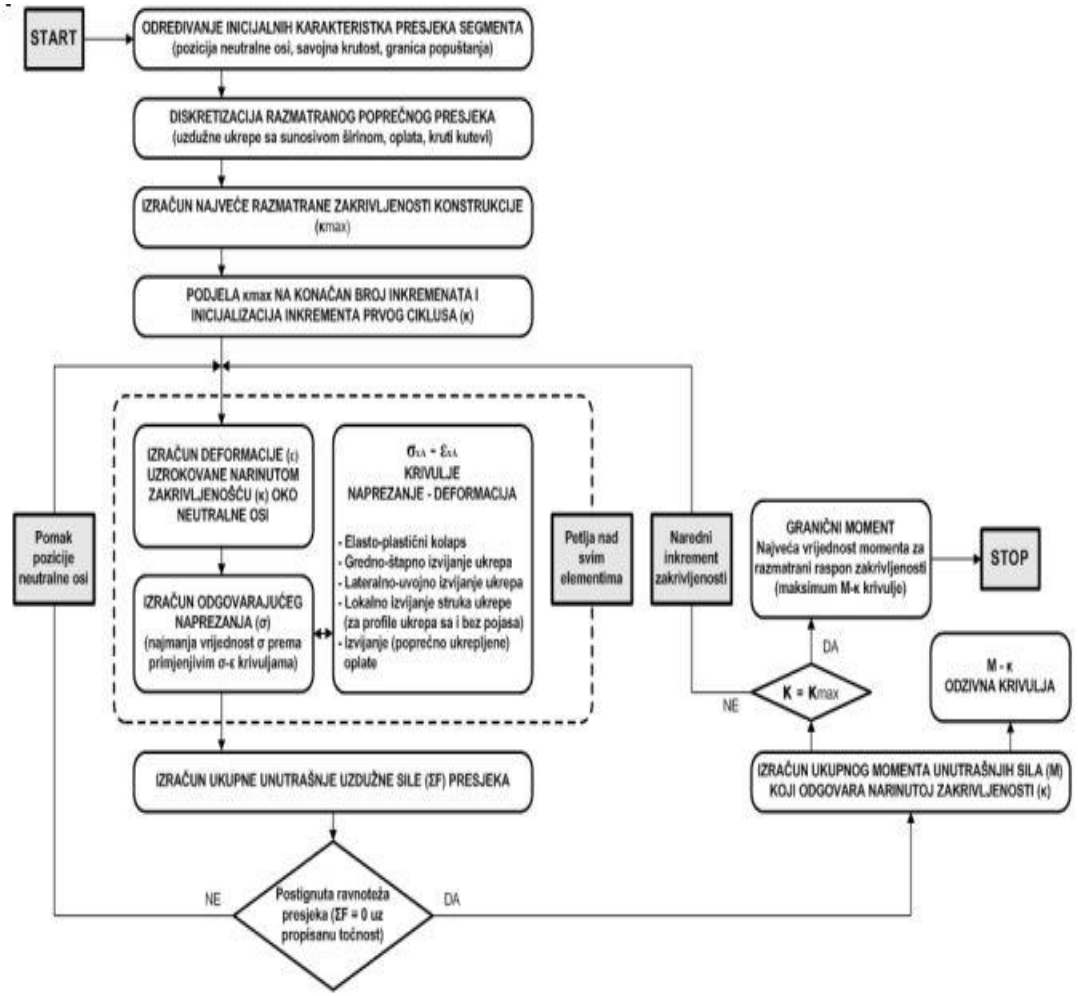
U konačnim inkrementima zakrivljenost se pri savijanju povećava a pri tome uzrokuje različite linearne distribucije uzdužnih deformacija po visini razmatranog presjekaza svaki inkrement.

Uzdužni globalni kolaps konstrukcije može se definirati kao gubitak savojne krutosti progresivnim kolapsom sastavnih elemenata razmatranog uzdužnog segmenta konstrukcije pri graničnom momentu savijanja[3]. Uslijed različitih načina izvijanja kod tlačno opterećenih elemenata te uslijed popuštanja vlačno opterećenih elemenata uzdužnog segmenta konstrukcije u stanju progiba ili pregiba dolazi do progresivnog kolapsa konstrukcije.



Slika 2. $MM{-}\kappa$ dijagram progresivnog kolapsa konstrukcije pri savijanju [11]

Bitno je reći kako se razmatra vertikalno savijanje konstrukcije dok se utjecaj horizontalnog momenta savijanja, posmične sile, torzijskog momenta i lokalnog opterećenja zanemaruje.



Slika 3. Dijagram toka inkrementalno – iterativnog algoritma Smithove metode [11]

Materijal razmatranog uzdužnog segmenta konstrukcije koji sudjeluje u uzdužnoj nosivosti diskretizira se sa tri vrste međusobno raspregnutih diskretnih sastavnih elemenata:

- Grede tankostjenog presjeka - uzdužne ukrpe sa pridruženom širinom oplate
- Kruti kutovi – spojevi uzdužne oplate koja ne leži u istoj ravnini, vrlo kruti dijelovi presjeka za koje se smatra da nosivost mogu izgubiti isključivo popuštanjem materijala
- Poprečno orebrena oplata – uzdužno neukrepljena oplata kojoj poprečno orebrenje određuje uzdužni raspon [3]

Uzdužna granična nosivost diskretnih elemenata:

Razina uzdužne nosivosti određuje se nezavisno za svaki diskretni sastavni element korištenjem različitih fenomenoloških $\sigma\sigma-\varepsilon\varepsilon$ krivulja. Uz pomoć krivulja definira se nosivost elemenata u

linearnoj odnosno nelinearnoj elasto-plastičnoj domeni. Svaka krivulja odgovara nekom od mogućih načina kolapsa diskretnih sastavnih elemenata:

- Elasto-plastični kolaps – popuštanje
- Gredno–štapno izvijanje
- Savojno–uvojno izvijanje
- Lokalno izvijanje struka ukrpe bez pojasa
- Lokalno izvijanje struka ukrpe s pojasom
- Izvijanje oplate [3]

3.1. LUSA

U ovom radu korišten je programski modul LUSA koji služi za proračun uzdužne čvrstoće broda. Lusa model izrađen je na Fakultetu strojarstva i brodogradnje (FSB) gdje se koristi kao jedan od alata za proračun uzdužne čvrstoće broda. Kako bi se krenulo u proračun potrebno je po određenim pravilima koje zahvaća LUSA izraditi model uzdužne konstrukcije primjenom programa MAESTRO. Programski modul LUSA koristi inkrementalno-iterativnu metodu proračuna.

4. PYTHON

Python je jedan od danas najpopularnijih jezika. Radi se o bogatom objektno orijentiranom programskom jeziku prikladnom za razvoj najzahtjevnijih aplikacija. Važna odlika Pythona je njegova jednostavna i jasna sintaksa i dosljednost, koja ga čini lakim za učenje. Uz to, specifična sintaksa tjera programera na urednost, pa je kôd izuzetno čitljiv i pregledan. Autor jezika je Guido van Rossum, a prva inačica jezika predstavljena 1991. godine. Ime jezika ne dolazi od zmije na koju asocira, već od popularne humorističke serije Leteći cirkus Montyja Pythona, pa su i mnogi primjeri u referentnim udžbenicima inspirirani njihovim skečevima. Jezik se neprestano razvija i redovito izlaze nove inačice. Python je jezik koji korisniku/programeru pruža mnogo više prostora za razmišljanje s obzirom na preostale jezike. Python je zanimljiv i zbog svoje jasne i izražajne sintakse, koja je različita od C-a (odnosno njemu sličnih jezika C++, Java, C#, Perl itd.). Stoga je za iskusnog programera u nekom od takvih jezika ispočetka malo neobičan, no izuzetno se lako i brzo uči jer je vrlo dosljedan [5].

Python je besplatan, open-source software, s izuzetno dobrom potporom, literaturom i dokumentacijom. Python kôd živi u tekst datotekama koje završavaju na .py. Program je moguće kompilirati u niz bytecodeova koji se spremaju u .pyc datoteke koje su prenosive na bilo koje platforme gdje se mogu izvoditi interpretacijom tog međukôda. Slično se izvršava Java kôd. Brzina izvođenja Python kôda istog je reda veličine kao u Javi ili Perlu. Python je napisan u ANSI C i raspoloživ za cijeli niz strojeva i operacijskih sustava uključujući Windows, Unix/Linux i Macintosh. Python je pisan u modularnoj C arhitekturi. Zato se može lako proširivati novi značajkama ili APIima. (eng. *application programming interface*) [6].

Python nudi mnoštvo različitih tipova podataka, standardne u smislu brojeva, nizova, znakova i napredne tipove poput lista i riječnika. Pythonova knjižnica (engl. library), koja uključuje standardnu instalaciju, uključuje preko 200 modula, što pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web-servera. Glavni Python web site (www.python.org) nudi sažeti index mnogih Python projekata i različitih drugih knjižnica. Python ima veliku entuzijastičku zajednicu korisnika koja se svake godine udvostručuje [6].

4.1. Varijable

Varijable su jedna od najčešće korištenih struktura unutar bilo kojeg programa [6].Kako bi definirali varijablu potrebno joj je samo dodijeliti određenu vrijednost. Kako bi dodijelili vrijednost nekoj varijabli koristi se matematički znak jednakosti („=“), a ime vrijeble nalazi se uvijek sa lijeve strane znaka jednakosti dok se desna strana znaka jednakosti koristi za vrijednost varijable koju spremamo. Varijablu se nazova i objektom sa svojom pripadajučom vrijednošću te tipom. Osnovni tipovi podataka s kojima varijable rade su: brojevi, nizovi znakova, liste, riječnici, n-terci, skupovi.

4.2. Komentari

Pisanje komentara neophodno je u svakom programskom kodu. Komentari su zapravo vrlo korisni te ih je vrlo praktično koristiti kako bi komentirali neku funkciju, koncept ili slično. Komentari su vrlo korisni kod rada u grupama programera te oni znatno olakšavaju rad. Umetanje komentara je vrlo jednostavno, a provodi se tako da se umeće znak „#“ na početku komentara. Komentari se mogu protezati do kraja jednog ili nekoliko redaka.

4.3. Unos i ispis podataka

Kako bi mogli obavljati različite operacije sa podacima te manipulirati istima neophodno je dopremiti podatke. Da bi omogučili upisivanje brojčanog ili tekstualnog sadržaja potrebno je upotrijebiti naredbu unosa (eng. Input). Funkcija input() radi na način tako da čita s tipkovnice znakove tako dugo dok se ne pritisne tipka "Enter". Nakon pritiska tipke "Enter" čitanje se završava te funkcija konvertira pročitane podatke u tip podataka „niz znakova“ te tako učitane znakove vraća preko povratne vrijednosti funkcije. Također, funkciji input() možemo poslati jedan argument. Sadržaj tog argumenta se ispisuje prije nego što funkcija krene čitati unesene znakove s tipkovnice [7].

Naredba za ispis podataka pokreće se riječju print, nakon te riječi otvaraju se i zatvaraju zagrade. Moguće je ispisati tekst i vrijednosti varijabli, a razlika je u tome što se tekst stavlja unutar navodnika dok se vrijednosti varijable ne stavlja. Unutar jedne naredbe print može se ispisati više izlaznih podataka, pri čemu ih je potrebno odvojiti zarezima.

4.4. Brojevi

Brojevi su većini korisnika tipovi podataka najlakši za shvatiti. Python prepoznaje nekoliko tipova brojeva: integer odnosno cijeli brojevi, floating point su brojevi s decimalnom točkom i complex brojevi koji imaju osim realnog djela i imaginarni dio. Postoje još i skupovi. Python može koristiti i kalkulator te pomoću njega obavljati mnoštvo matematičkih operacija. Može zbrajati, oduzimati, korjenovati i tako dalje.

4.5. Nizovi znakova

U Pythonu niz brojeva definira se kao skup znakova unutar jednostrukih ili dvostrukih navodnika ('...' ili a"..."). Ako se unutar niza znakova nalazi jednostruki navodnik za identificiranje spomenutog skupa potrebno je korištenje dvostrukih navodnika ili u obrnutom slučaju, ukoliko niz znakova sadržava dvostruke navodnike taj niz se identificira korištenjem jednostrukih navodnika. Nizove znakova možemo indeksirati i to sa pozitivnim ili negativnim brojevima na način da prvi znak u nizu ima indeks 0, drugi indeks 1 ili sa negativnim indeksom tako da zadnji znak u nizu ima indeks -1, predzanji -2 i tako dalje. Indeksiranje nizova znakova omogućava ispis podskupa nekog niza pomoću operatora [ ], ako želimo ispisati samo jedan znak niza, ili pomoću operatora [:], ako želimo ispisati više znakova niza. Nizove znakova se isto tako može povezivati pomoću znaka +, može ih se ponavljati pomoću znaka *, a iznos njihove duljine računa se pomoću naredbe len().

4.6. Liste

Liste ili popis su promjenjiv poredani niz znakova objekta. Lista se definira nabrajanjem znakova, članova koji se odjeljuju zarezima te se smještavaju unutar uglatih zagrada ([ ]). Prazna se lista označava parom praznih uglatih zagrada. Kao i kod nizova znakova elemente liste se može indeksirati. Prvi element liste indeksom 0, drugi indeksom 1,... ili zadnji element indeksom -1, predzanji indeksom -2 i tako dalje. Kako bi se određeni elementi liste ispisali koriste se operateri [ ], a ukoliko se želi ispisati više elemenata liste koristi se [:]. Liste se mogu kao i nizovi znakova spajati pomoću znaka *, a duljina liste izračunava se pomoću naredbe len(). Listi se mogu dodati i pojedinačni elementi pomoću naredbe imeliste.append ().

4.7. Riječnici

Rječnici u Pythonu funkcioniraju po principu ključa. Svaki zapis možemo podijeliti na dva dijela: ključ : vrijednost. Preko ključa dohvaća se vrijednost koja pripada zadanom ključu. Uzmimo primjer s mjesecima, ključ je vrijednost koja je predstavljena brojem mjeseca u godini (1. mjesec, 2. mjesec…), a vrijednost je naziv mjeseca (siječanj, veljača…). Sintaksa definiranja rječnika je da unutar vitičastih zagrada navedemo parove ključ : vrijednost. Parovi su međusobno odvojeni zarezom [7].

Ako želimo definirati prazan rječnik to se radi na način imerječnika= {}. Pridruživanje vrijednosti rječniku radi se gotovo na isti način kao i kod pridruživanja vrijednosti listi samo što se kod rječnika koriste vitičaste zagrade {}. Isto tako unutar rječnika svaki element sastoji se od para ključ : vrijednost.

Bitno je obratiti pažnju na sljedeće stvari:

- Vrijednost ključa mora biti jedinstvena. Ako se želi staviti podatak čiji ključ već postoji unutar rječnika, postojeća vrijednost koja pripada tom ključu zamijenit će se novom vrijednosti.
- Ključ može biti bilo koji tip podatka (niz znakova, brojevi, n-torke…), no kao ključ nije moguće koristiti listu podataka. U jednom te istom rječniku kao ključ moguće je kombinirati različite tipove podataka.
- Vrijednost može biti bilo kojega tipa podatka [7].

Dohvatiti elemente rječnika moguće je tako da se prvo napiše naziv rječnika a zatim u uglatim zagradama stavi ključ vrijednosti koju se želi dohvatiti. Korištenjem naredbe keys() može se dohvatiti lista svih ključeva unutar rječnika. Naredba update() služi za umetanje novog para vrijednosti, dok values() naredba dohvaća listu vrijednosti koja je spremljena u rječnik. Ukoliko se neki par želi izbrisati iz rječnika provodi se na sljedeći način: del imeRjecnika[kljuc]. Broj elemenata spremljenih u nekom rječniku dohvaća se pomoću naredbe len ().

4.8. If – else odluke

Odluke u Pythonu su realizirane u obliku grananja ili if-else uvjetovanja što omogućava programu odabir jedne od dvije mogučnosti kretanja u jednom od odabranih smjerova za izvršavanje naredbi. Grananje predstavlja operaciju true/false jer se temelje na odabiru jednog od dva ponuđena izbora. U Pythonu za odluke se koristi naredba if (uvjet)...else... Vrlo je bitno naponuti da u Pythonu odluka koja se sastoji od ključne riječi if i uvjeta mora završiti znakom dvotočke „:“.

4.9. Programska petlja for

Programske petlje u Pythonu su strukture koje omogučavaju ponavljanje određenog dijela. Petlje sadržavaju uvjet kao i odluke (ifelse), također sadržavaju i brojač. Brojač je nešto poput uvjeta, a zadaje se broj ponavljanja određenog programskog koda. Za aktivaciju petlje koristi se riječ for, a prati je i oznaka i koja označava brojač. Sljedeća bitna riječ u petlji je range sa kojom se određije uvjet ponavljanja koda. Unutar te naredbe postoji početna vrijednost brojača od koje petlja počinje ponavljati kod, a potom i završna vrijednost petlje. Korakom petlje se brojaču diktira u kojem smjeru treba brojati i ponavljati petlju. Ukoliko neku naredbu želimo ponoviti točno odredeni broj puta, koristimo petlju for. Sintaksa naredbe for glasi: for i in range(a,b,k): naredba ili skup naredbi. Znak i označaba brojač petlje, a je početna vrijednost brojača, a b završna vrijednost brojača. K je korak brojača. Kada se želi neka naredba ponoviti n puta, a da je n neki prirodan broj, potrebno je napisati: for i in range(n): naredba ili skup naredbi.

4.10. Programska petlja while

Programska petlja while je programska struktura s istom namjenom kao i petlja for. Petlja while omogućava, kao i petlja for, ponavljanje određenog bloka naredbi. Razika između petlji for i while je u tome što petlja while ne sadrži brojač, ali while petlja ima uvjet kao i for petlja. Uvjet omogućava petlji while izvršavanje naredbi. Uvjet u while petlji može biti određen operatorima: usporedbe, logičkim operatorima i aritmetičkim operatorima.

5. PROGRAM LINAETAL-FSB/D3V ZA VIZUALIZACIJU KONSTRUKCIJE

Program Linaetal-fsb/d3v je modularna Python aplikacija otvorenog programskog koda. Linaetal-fsb/d3v namjenjen je primano za 3D vizualizaciju inženjerskih modela u fazi projektiranja. Nastao je kao rezultat višegodišnje suradnje Fakulteta strojarstava i brodogradnje sa USCS.d.o.o te obrtom Linaetal. Struktura programa temeljena je na dvanaestogodišnjem iskustvu razvoja programa ShipExplorer. Program je tek u povojima te sadrži manji broj funkcija, ali vizualizacija već moguća. Linaetal-fsb/d3v je slobodno dostupan na web adresi https://github.com/linaetal-fsb/d3v. Provedba se temelji na okvirima QtForPython (PySide2) i OpenMesh.

5.1. Qt za Python

Qt je razvojni okvir koji koji omogućuje proširavanje funkcija za programski jezik C++. Proširuje ih korištenjem takozvanog “Meta – Object” prevoditelja nazvanog “MOC”. Qt sam po sebi nije programski jezik već je on napisan u programskom jeziku C++. Moguće ga je koristiti na više operacijskih sustava. Razvoj Qt-a započeli su 1990. godine norveški programeri Eirik Chambe-Eng i Haavard Nord. Njihova tvrtka Trolltech koja je prodavala Qt licence i pružala podršku, tijekom godina prošla je nekoliko akvizicija. Qt Company glavni pokretač Qt-a, ali je razvijen uz pomoć više partnera, tvrtki i pojedinaca širom svijeta. Postoji mnogo načina na koje se može doprinijeti Qt projektu, npr. pisanjem koda ili dokumentacije za okvir, izvještavanjem o pogreškama, pomaganju drugim korisnicima na forumima. Qt je dostupan pod različitim licencama: Qt Company prodaje komercijalne licence, ali je Qt besplatno dostupan u nekliko verzija. Qt ima vlastiti izvršni alat, zvani qmake, za stvaranje izvršnih datoteka vezanih za platformu na kojoj se pokreće, čitajući datoteke s nastavkom „pro“. Qt razvojni okvir obuhvaća nekoliko biblioteka: Core framework, Gui framework, SQL framework, Xml framework, Networking framework, OpenGL framework, Multimedia framework, WebKit framework i tako dalje [8].

5.2. QtCreator

Qt Creator je integrirano razvojno okruženje C ++ , JavaScript i QML koje pojednostavljuje razvoj GUI aplikacija te je prilagođeno potrebama Qt programera. Qt Creator uključuje vizualni program za pronalaženje pogrešaka te integrirani raspored i dizajner obrazaca. Qt Creator koristi C++ iz zbirke GNU Compiler na Linuxu i FreeBSD-u. Razvoj Qt Creatora započeo je 2007. godine i to pod imenima Workbench a kasnije Project Greenhouse. Qt Creator uključuje uređivač koda i integrira Qt Designer za dizajniranje i izgradnju grafičkih korisničkih sučelja iz QT datoteka. Pri pisanju kôda ima mogućnosti formatiranja, nadopunjavanje koda zvano "intellisense", upozoravanje na pogreške i u nekim slučajevima automatsko ispravljanje kôda. Qt ima i mogućnost praćenja curenja memorije. Instalacijom QtCreatora dobiva se i mnoštvo aplikacija u Qt-u, najviše grafičkih. Podržava razne formate datoteka i moguće je otvoriti slike, xml, JSON dokumente i tako dalje. Nudi i mogućnost interakcije s Git sustavom. Qt Creator pruža podršku za pokretanje Qt aplikacija za radna okruženja Windows, Linux, FreeBSD i Mac OS, za mobilne uređaje: Android, BlackBerry, iOS, Maemo i MeeGo.

5.3. PySide 2

PySide2 je jezična poveznica Pythona s Qt višeplatformskim programskim alatom za izradu sučelja. Razvila ga je QT kompanija u projektu Qt for a Python na portalu PySide. PySide2 je alternativa standardnoj bibioteci Tkinter. PySide2 je besplatan softver.

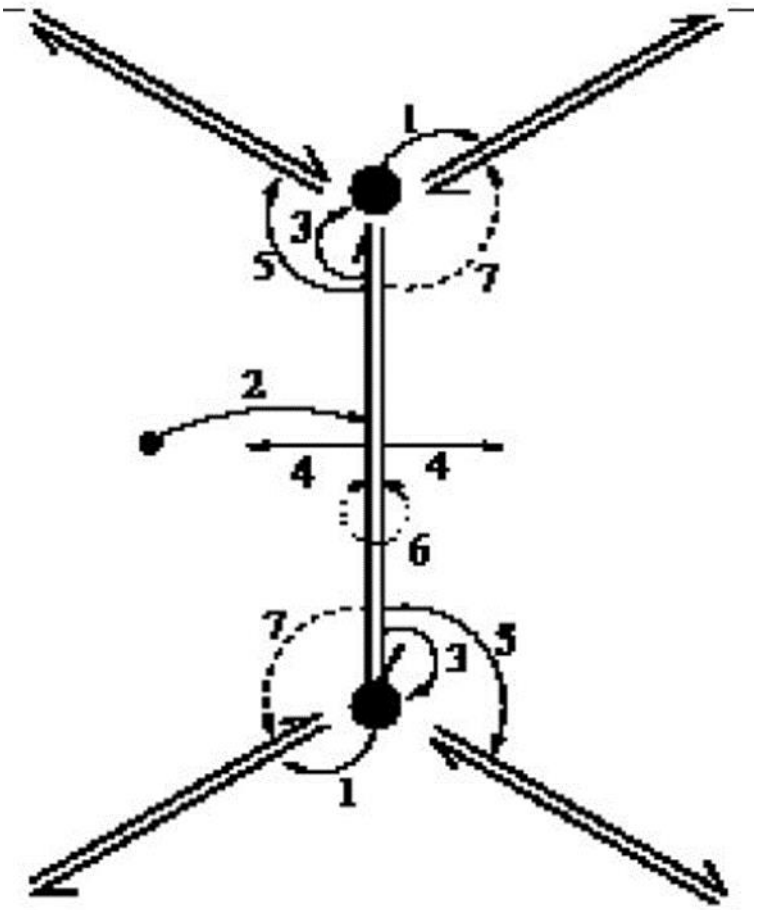
Projekt razvoja PySide biblioteke nastao je kao posljedica neuspjeha u pregovorima Nokije s britanskom tvrtkom Riverbank Computing oko licenciranja PyQt biblioteke, također poveznice Qt-a s Pythonom, pri čemu je zahtjev Nokije bio da PyQt bude licenciran i pod LGPL licencom pored postojeće GPLv3 i komercijalne licence. LGPL licenca omogućuje primjenu PySide biblioteke u razvoju besplatnih programa otvorenog koda, ali i komercijalnog softvera. PySide2 razvoj je započeo tijekom 2014. godine. PySide podržava rad na Linux, Mac OS X, Windows i Maemo operacijskim sustavima, dok je podrška za Android i Symbian u postupku uspostavljanja [9].

5.4. OpenMesh

OpenMesh je generička i učinkovita biblioteka koja nudi strukture podataka za predstavljanje i manipulianje poligonom mreža. Moćan je alat za rukovanje poligonalnim mrežama. Zahvaljujući svojstvenoj generatovnoj strukturi korisniku omogućuje stvaranje mreža prilagođenih specifičnim potrebama aplikacije[10]. OpenMesh razvijen je na Sveučilištu u Aachenu uz financijsku pomoć njemačkog Ministarstva za istraživanje i obrazovanje.

5.5. Polubridna struktura zapisa mreže [14]

Ovdje se opisuje temeljna struktura podataka kojemu je primarna svrha pohranjivati mrežne entitete u smislu vrhova, bridova, poligonskih ploha te podataka o povezivanju istih. Za prikazivanje poligonalnih mreža postoji velik brojstruktura podataka, a u ovome radu korištena je podatkpvna struktura polubridova. Ako postoje dva vrha A i B te rub koji ih povezuje, tada su A i B dva usmjerena polubrida i obratno. Slika 11. prikazuje način na koji se povezanost pohranjuje u spomenutu strukturu.



Slika 4. Pohranjivanje povezanosti [14]

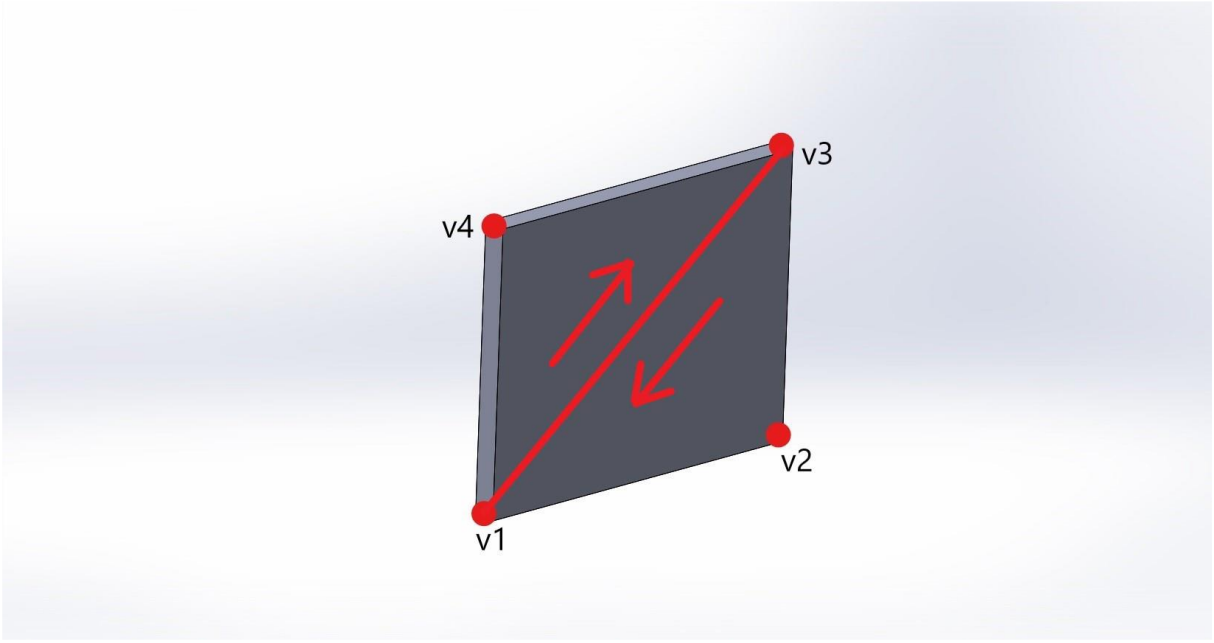
Svaki vrh povezan je na jedan odlazni polubrid. Svaka ploha referira jedan polubrid te ga ograničava. Svaki polubrid daje pokazivač za vrh na koji upućuje, za plohu kojoj pripada, sljedećem polubridu unutar plohe te suprotnom polubridu. Ukoliko su ove veze postojeće, moguće je kružiti oko ploha, te pristupiti njihovim vrhovima, polubrdovima i susjedni plohama. Ukoliko krenemo sa polubridom te ga iteriramo na supritno od prethodnog, jednostavno je moguće kružiti oko tog vrha te dobiti sve susjede; polubridove i plohe.

Prednosti strukture temeljene na polubridovima su te da je moguće: direktno pristupiti vrhovima, plohama i polubridovima, miješati plohe proizvoljnog broja vrhova u mreži, kružno pristupiti entitetima oko vrhova. Posljednje navedena prednost važna je operacija kod velikog broja algoritama na poligonalnim mrežama tako i u polubridnoj strukturi [14].

5.6. Vizualizacija konstrukcije broda u programu Linaetal-fsb/d3v [14]

5.6.1. Viualizacija četverokutnih 2D elemenata

Četverokutni 2D element sastoji se od četiri vrha. Nastaje spajanjem dva trokutna 2D elementa čija orijentacija mora biti međusobno suprotnog smjera. Trokutni element neće se generirati ako je orijentacija dvaju trokuta ista.

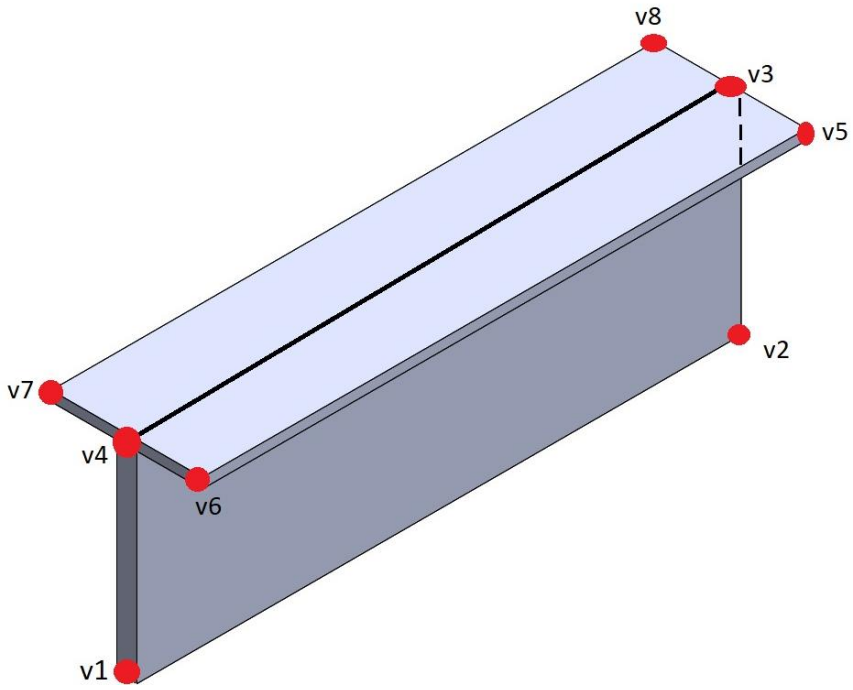


Slika 5. Četverokutni 2D element [14]

5.6.2 Vizualizacija Vizualizacija grednih elemenata s T poprečnim presjekom

U ovom programu vizualizacija grednog elementa sa T poprečnim presjekom i sunosivom širinom izvedena je pomoću šest elemenata, odnosno pripadajućih 12 vrhova. Generira se zadavanjem dvaju čvorova, usmjerenjem struka i karakteristikama grednog elementa (visina struka, širina prirubnice, širina pokrova dna). Koordinate prvih dvaju vrhova su koordinate čvorova grednog elementa.

Uz pomoć usmjerenja i visine struka se dobiju još dva čvora i to na način da se zbroji vektor struka s početnim točkama. Preostalih 4 vrhova odredi se pomoću vektora prirubnice, koje je okomito na struk i to uz pomoću vektorskog produkta vektora struka i vektora koji je dobiven kao razlika dviju početnih točaka. Poznavajući vektor prirubnice dobiju se preostale točke grednog modela dodavanjem ili oduzimanjem pola širine prirubnice. Nakon što se izračunaju sve potrebne koordinate vrhova, vrhovi se generiraju s funkcijom *mesh.add_vertex*, a pomoću tih vrhova generiraju se svi potrebni trokutni elementi funkcijom *mesh.add_face* spajanjem vrhova pazeći na orijentaciju zbog navedenog openmesh ograničenja. Struk i prirubnica grednog elementa se sastoje svaki od 2 trokutna elementa. Prvi trokutni element struka se generira pomoću 3 vrha (v1, v2, v3), dok se drugi sastoji od vrhova v3, v4 i v1. Trokutni elementi prirubnice se sastoje od vrhova v5, v6, v7, odnosno v7, v5, v8.



Slika 6. Gredni element

5.7. Vizualizacija modela konstrukcije bojama [14]

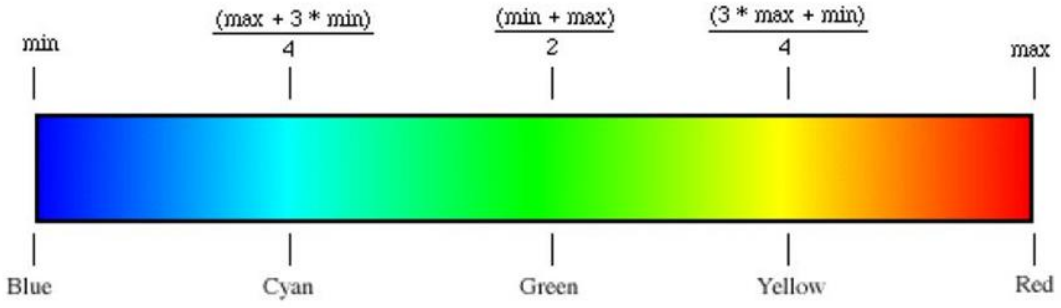
Kako bi vizualizirali značajke modela primjenjeni su dva načina određivanja boja a to su: mapa sa konačnim brojem različitih boja i kontinuirana skala boja.

5.7.1. Mapa sa konačnim brojem različitih boja

Mapa sa konačnim brojem različitih boja koristi se za prikaz diskretnih, konačnih setova. Svaka boja ima svoj r,g,b kanal. (Naprimjer tamno zelena boja s oznakom [0,100,0] kod koje prvi član (0) prestavlja udio crvene boje, drugi član (100) udio zelene boje, a treći član (0) udio plave boje u konačnoj boji koju predstavlja taj kanal. Na internet stranici koja je navedena u literaturi pod rednim brojem [12] moguće je naći r,g,b vrijednosti za različite preddefinirane boje. Od velikog broja boja odabrano je 20 boja koje su dovoljno različite da su njihove razlike jasno uočljive, te su spremljene u listu. U konačnici defini ras e funkcija koja za integerski ulaz vraća jednu boju iz pripremljene liste, ta funkcija naziva *GetDiscreteColor*. Svi predmeti koji sadrže isti ID bivaju obojeni u istoj boji, dok onim različitima dodjeljuje se prva sljedeća slobodna boja iz liste.

5.7.2. Kontinuirana skala boja

Kontinuirana skala boja koristi se za prikazivanje rezultata pomaka; naprezanja. Nadalje, postoje dvije mogučnosti prikaza od kojih je jedna konstantna boja po elementu a druga mogučnost je da je boja određena po čvorovima. Kako bi dobili skalu boja potrebno je definirati metodu *Get Continuous Color* koja uzima jedan float argument normirane vrijednosti od 0.0 do 1.0, a vraća jednu boju. Način određivanja boje ovisi o tome kakva se paleta boja želi prikazati. Paleta boja koja je primjenjena u ovo radu prikazana je na slici niže.



Slika 7. Paleta boja[13]

6. VIZUALIZACIJA MODELA ZA PRORAČUN GRANIČNE ČVRSTOĆE BRODSKOG TRUPA

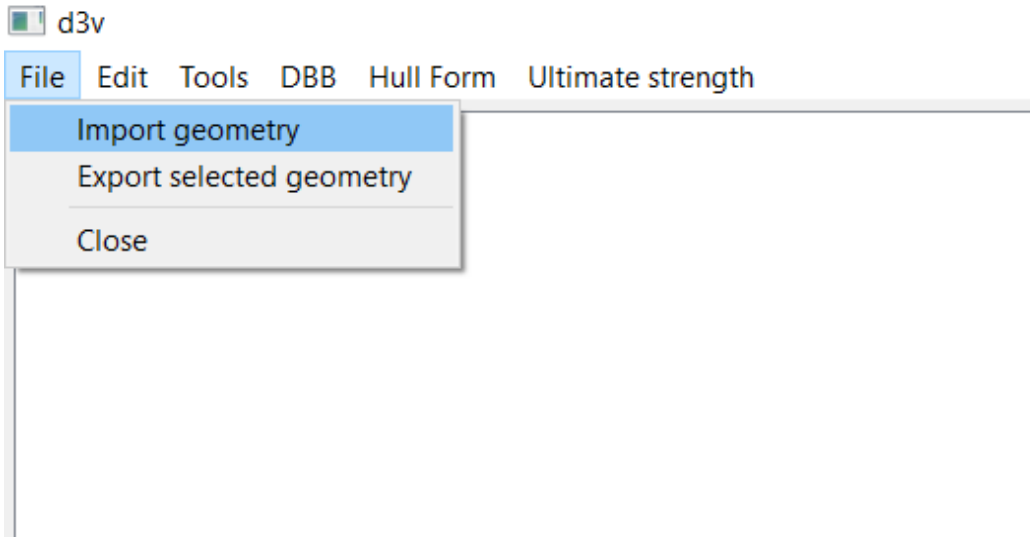
Izrada vizulizacije modela za proračun granične čvrstoće brodskog trupa rađena je u koracima:

- Izradom koda za učitavanje veze između datoteke s ekstenzijom hus i xml datoteke. Naziv koda je hullultstrength.py
- Izradom koda za učitavanje podataka o modelu glavnog rebra nazvanim readxml.py.
- Izradom koda za obradu učitanih podataka i vizualizaciju nazvnim geofem.py.
- Izrada koda za učitanja rezultata proračuna LUSA i njegova vizualizacija pomoću grafa.

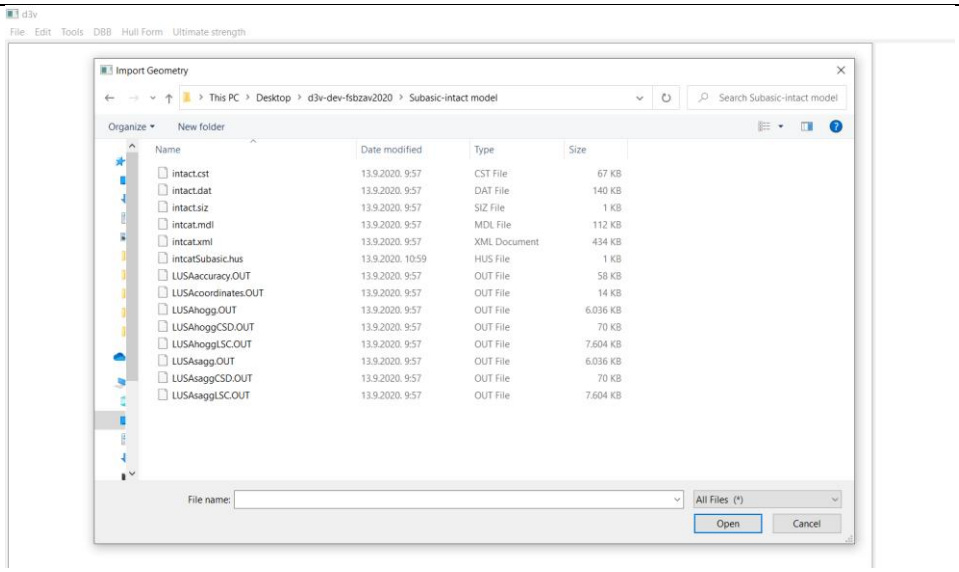
6.1. Hullultstrength.py

Hullultstrength.py je u Pythonu napisan programski kod koji služi za povezivanje izvršne datoteke hullultstrength modela i datoteke s ekstenzijom xml. Program Linaetal-fsb/d3v pokreće se u gornjem ljevom uglu gdje se nalazi izbornik File (slika 8) koji sadrži Import geometry.

Pritiskom na Import geometry, program traži da se odabere datoteka (slika 9) koja u ovom slučaju kako bi se aktivirao modul Ultimate strength mora imati ekstenziju hus. Prilikom odabira datoteke s ekstenzijom hus, hullultrstength.py stvara vezu programa s xml datotekom.



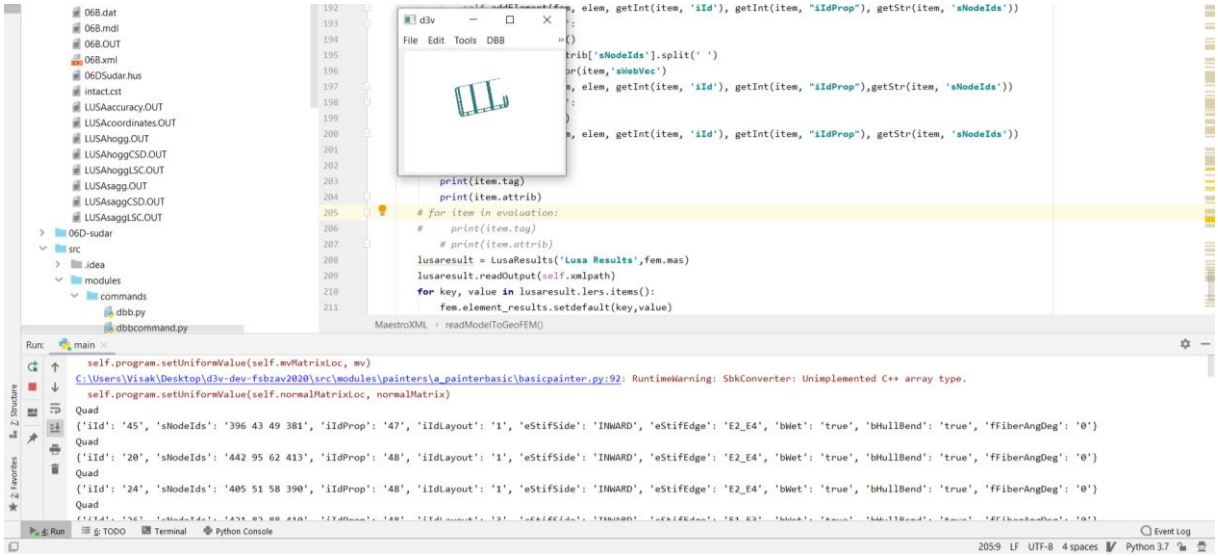
Slika 8. Izbornik File



Slika 9. Import geometry

6.2. Readxml.py

Readxml.py je u Pythonu napisan programski kod za očitanje podataka iz datoteke s etstenzijom xml. One datoteke koja je učitana pomoću hullultstrength.py koda. Brodarski program MAESTRO u kojem je izrađen model glavnog rebra može spremiti taj model kao datoteku s ekstenzijom xml. Upravo taj model glavnoga rebra, napravljen u MAESTRU služi kao ulazna datoteka za hullultstrength model. Iz datoteke se očitaju točke oplate potrebne za vizualizaciji i točke potrebne za vizualizaciju grednih elemenata (slika 10).



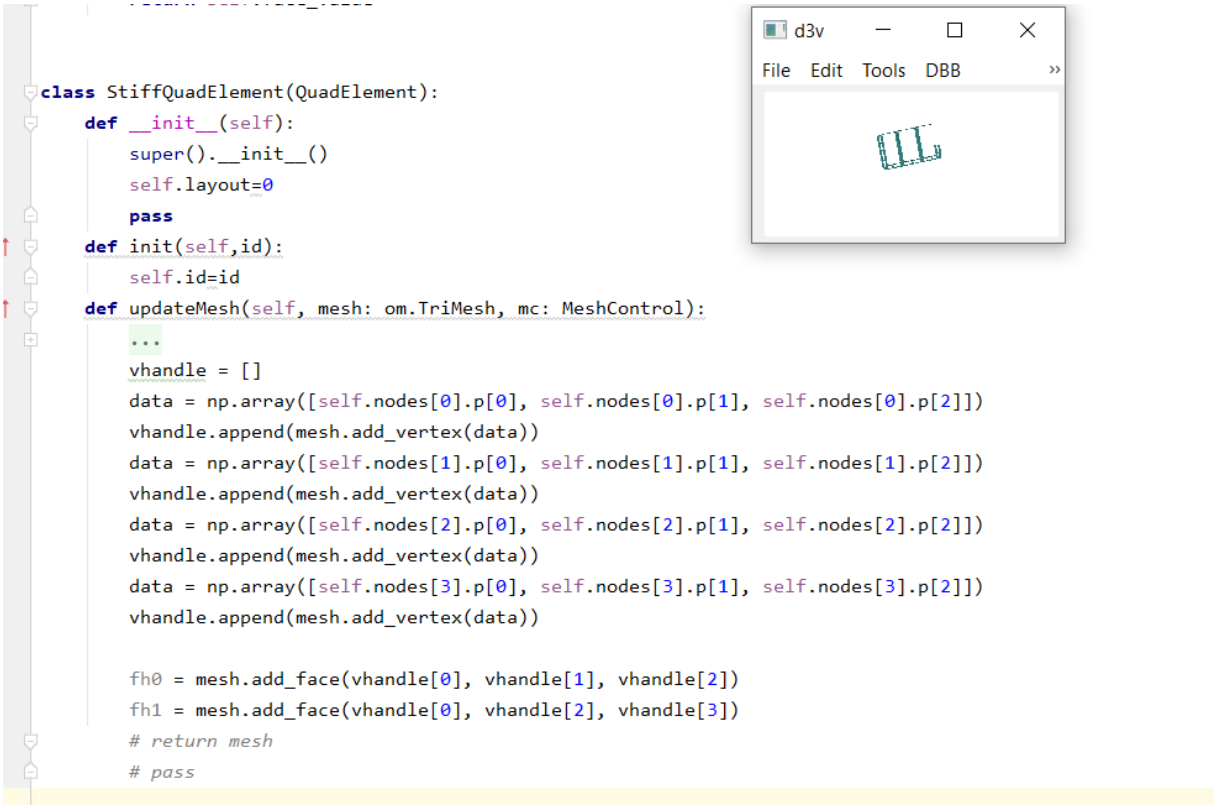
Slika 10. Prikaz točaka oplate

6.3. Geofem.py

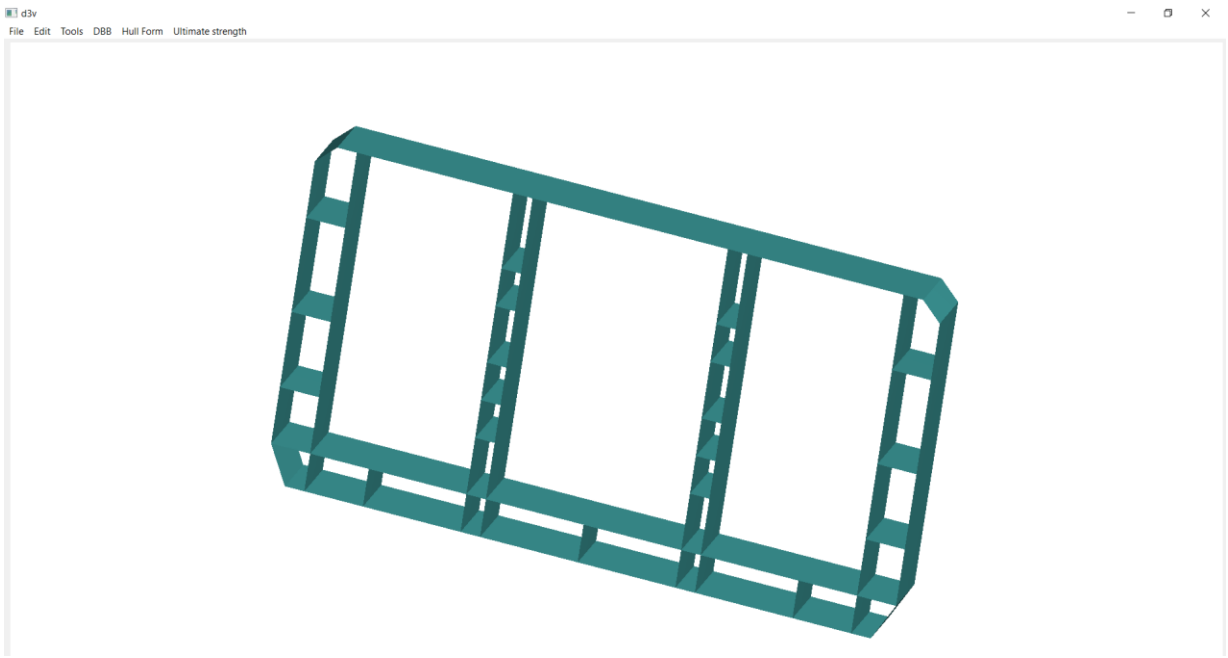
Geofem.py je u Pythonu napisan pogramski kod pomoću kojeg se obrađuju učitane točke da bi se vizualizirala oplata i gredni elementi glavnog rebra.

6.3.1. Vizualizacija elemenata oplate

Elementi oplate se vizualizira programskim kodom prikazanim na slici 11. U poglavlju pod točkom 5.6.1. opisan je postupak vizalizacije četverokuta koji ovdje predstavlja oplatu glavnog rebra broda. Slika 12 prikazuje vizualizirani presjek glavnog rebra broda.



Slika 11. Porgramski kod vizalizacije oplate



Slika 12. Vizualizirani presjek glavnog rebra broda

6.3.2. Vizualizacija grednih elemenata

Oplata je ukrućena grednim elementima koji se vizualiziraju programskim kodom prikazanim na slici 13. U poglavlju pod točkom 5.6.2. opisan je postupak vizualizacije grednog elementa s T poprečnim presjekom koji predstavljaju ukrućenje oplate. Sika 14 prikazuje vizualizaciju ukruta oplate.

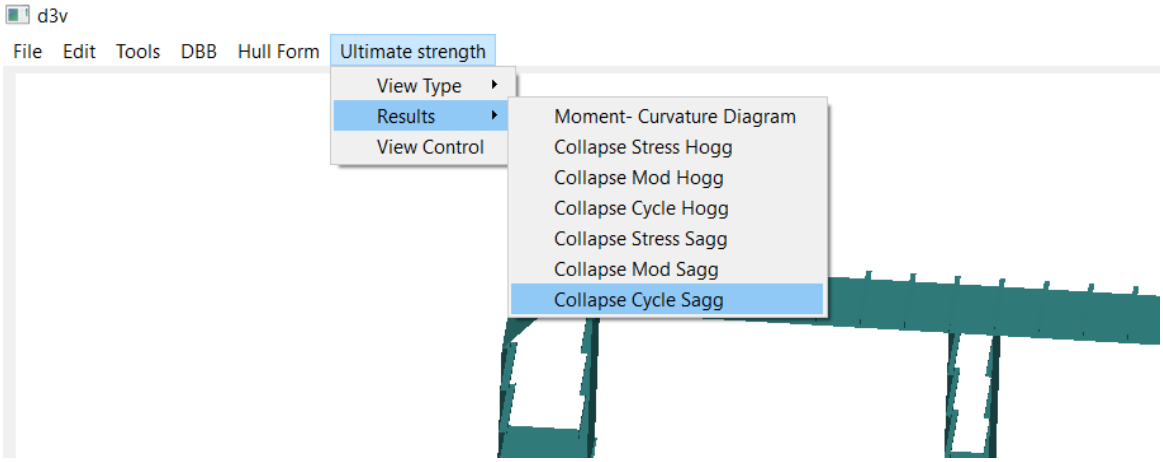
Slika 13. Porgramski kod vizalizacije grednog elementa s T poprečnim prejekom

Slika 14. Vizualizacija ukrepa oplate

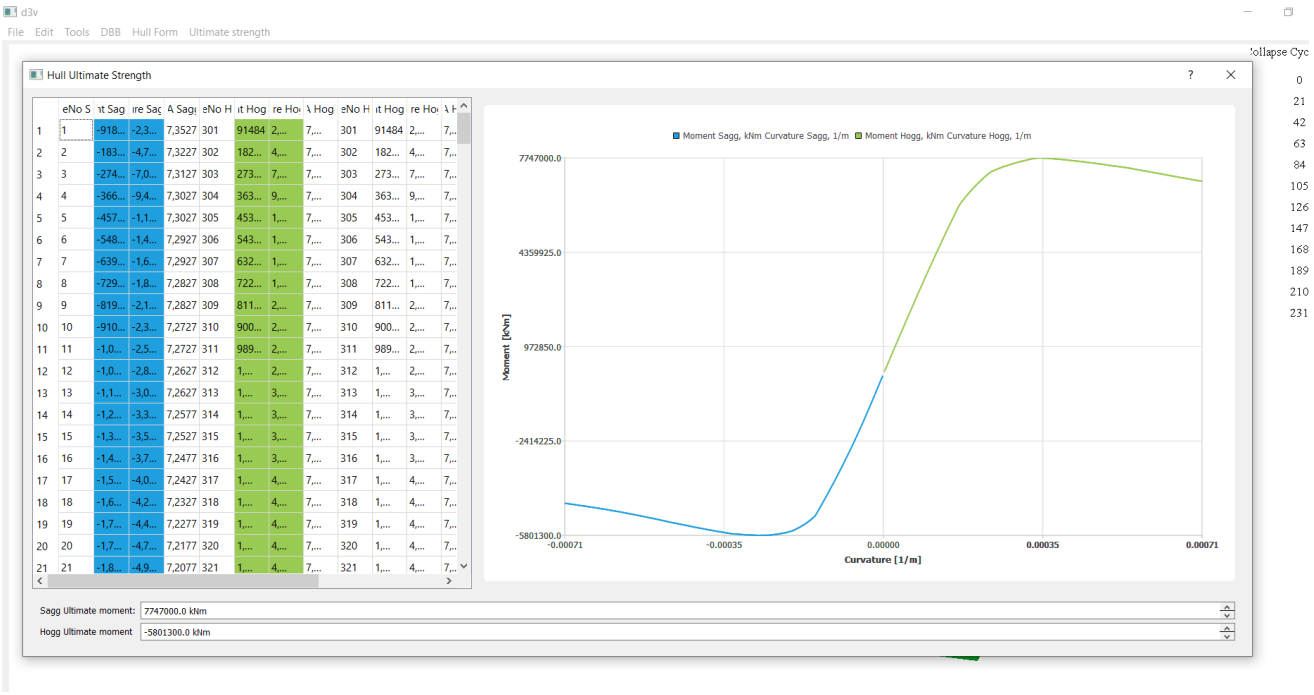
6.4. Grafički prikaz rezultata graničnog momenta savijanja

Pomoću programa imenom LUSA za izračun graničnog momenta savijanja, izrađen je programski kod za učitavanje rezultata tog proračuna. Da bi se prikazali rezultati proračuna potrebno je u izborniku odabrati Ultimate strength koji otvara podizbornik u kojem je potrebno odabrati Result, te jednu od ponuđenih opcija(slika 15).

Rezultati Moment-Curvature Diagrama su prikazani u obliku grafa koji prikazuje moment savijanja na osi y i zakrivljenost na osi x (slika 16). Collapse rezultati prikazani su bojama po elementima.



Slika 15. Moment-Curvature Diagram

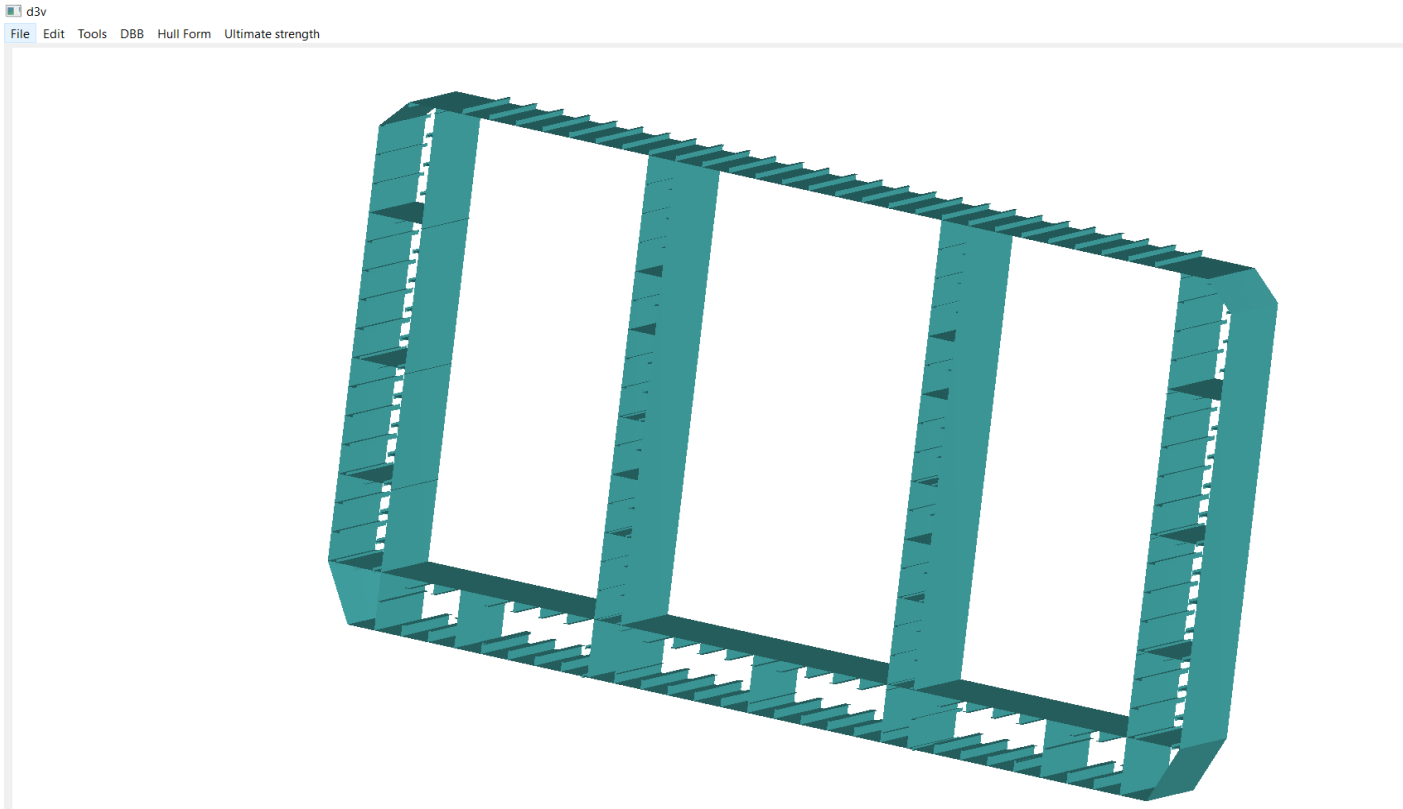


Slika 16. Moment-Curvature Diagram

6.5. Primjena modula za vizualizaciju na primjerima

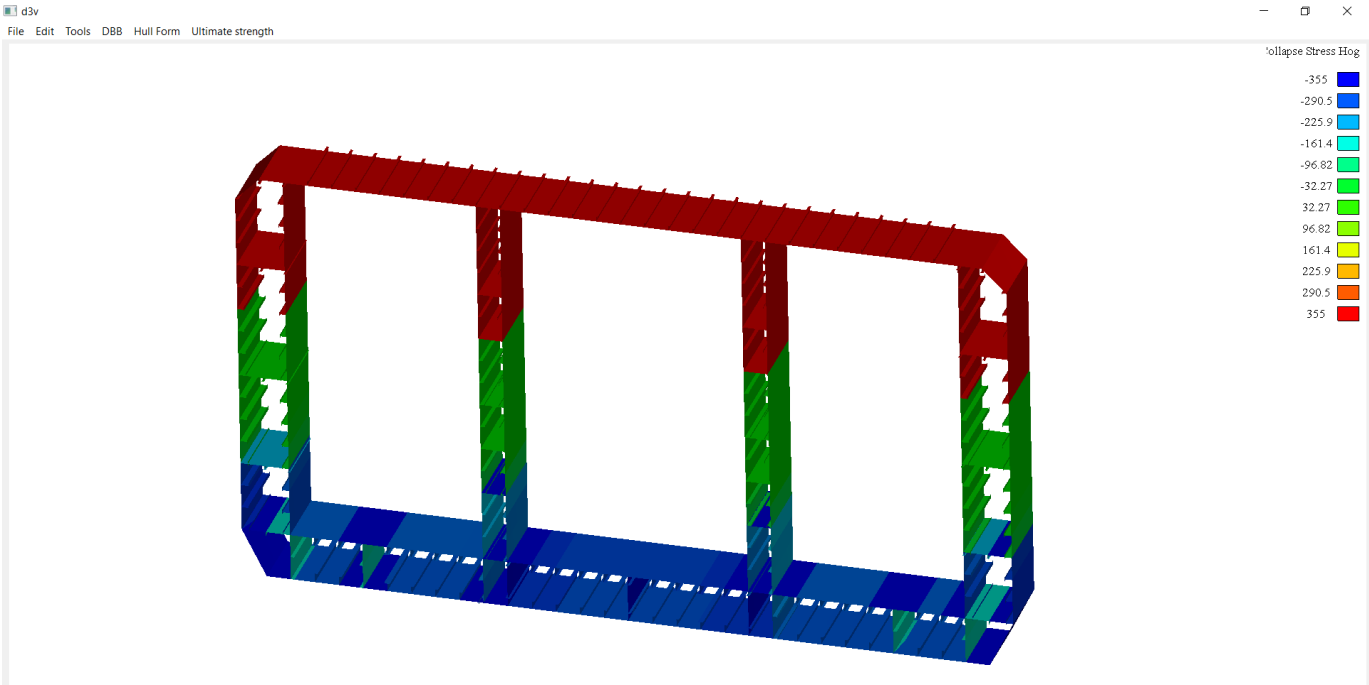
U primjerima je vidljiva razlika maksimalnog momenata savijanja usred oštećenog i neoštećenog stanja glavnog rebra broda.

6.5.1. Primjer neoštećenoga broda, njegov graf momenata savijanja i zakrivljenosti, prikaz raspodjele naprezanja po elementu, mod kolapsa, ciklusu kojem nastaje collaps elementa

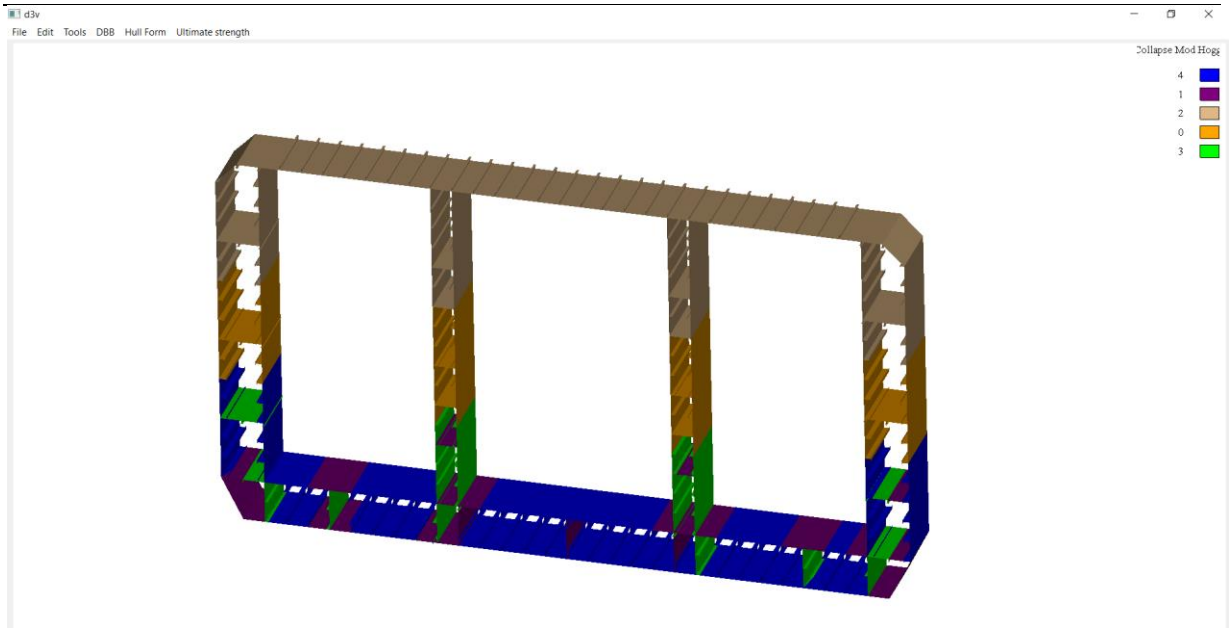


Slika 17. Vizualizacija neoštećenoga broda

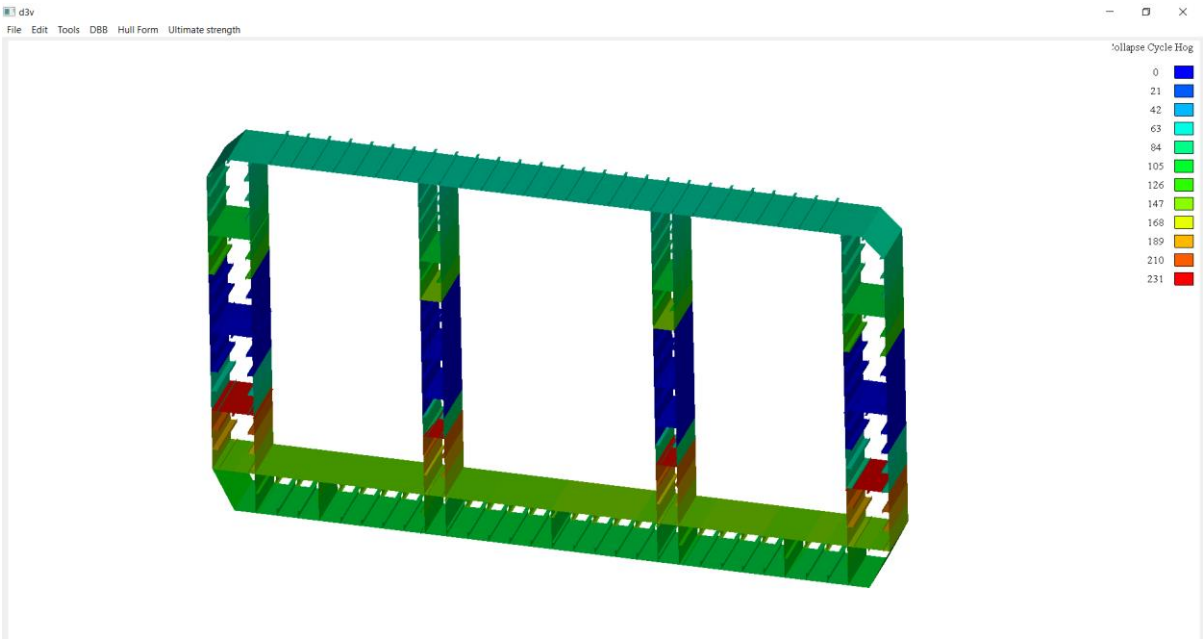
Slika 18. Moment-Curvature Diagram neoštećenoga broda



Slika 19. Prikaz naprezanja pri kojem je došlo do kolapsa pojedinog elementa

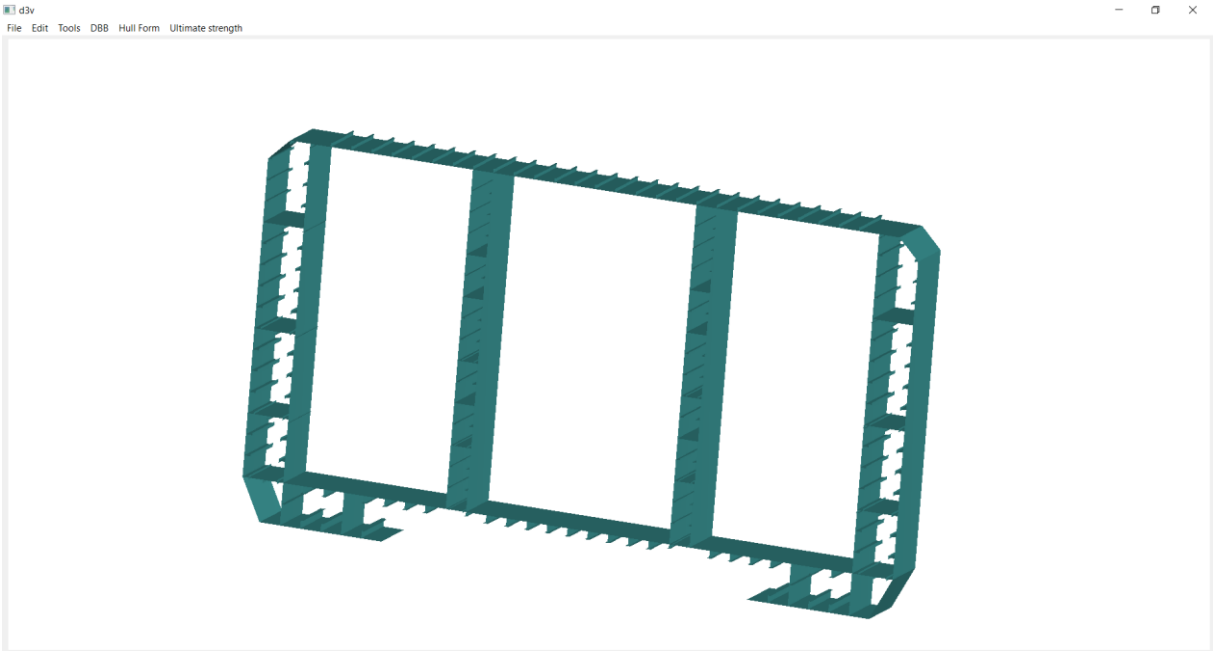


Slika 20. Prikaz moda oštećenja pri kojem je došlo do kolapsa pojedinog elementa

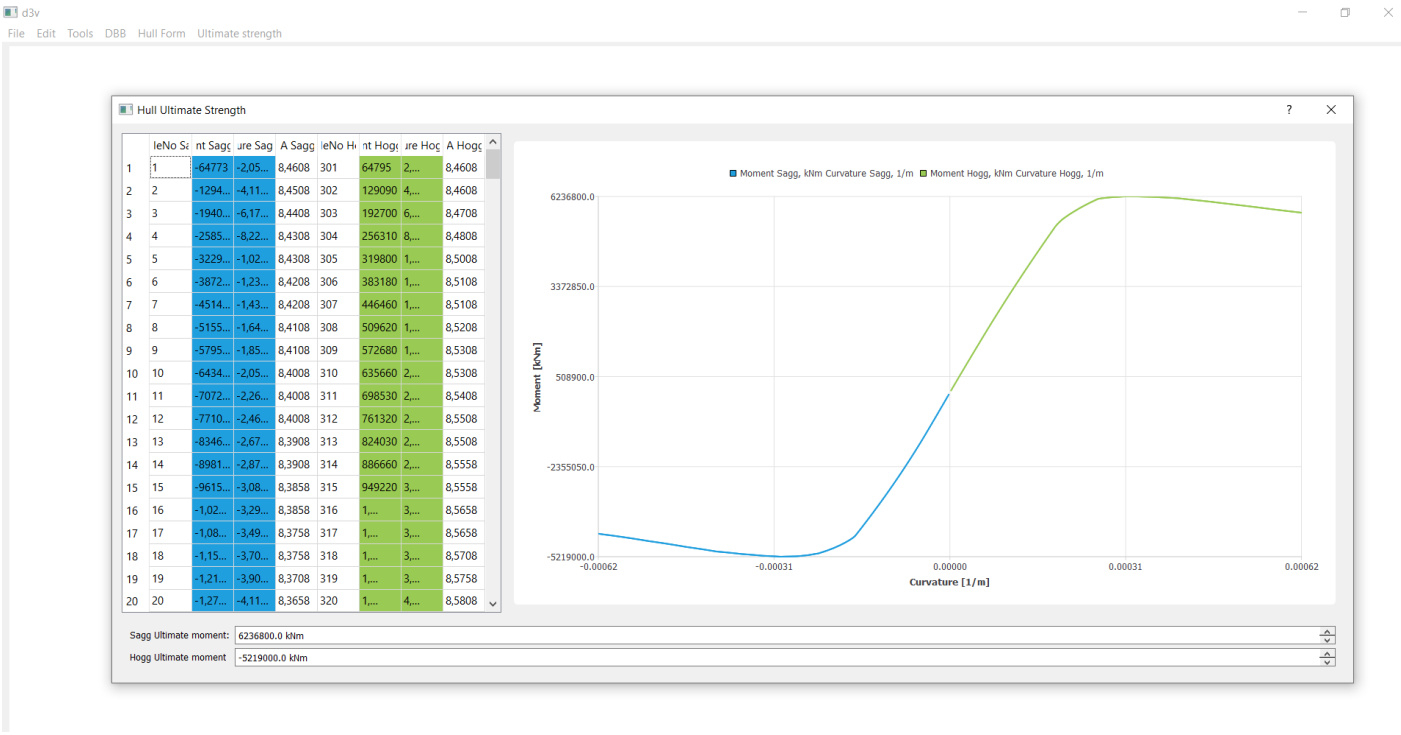


Slika 21. Ciklus u kojem nastupa kolaps elementa

6.5.2. Primjer oštećenja broda nasukavanjem, njegov graf momenata savijanja i zakrivljenosti

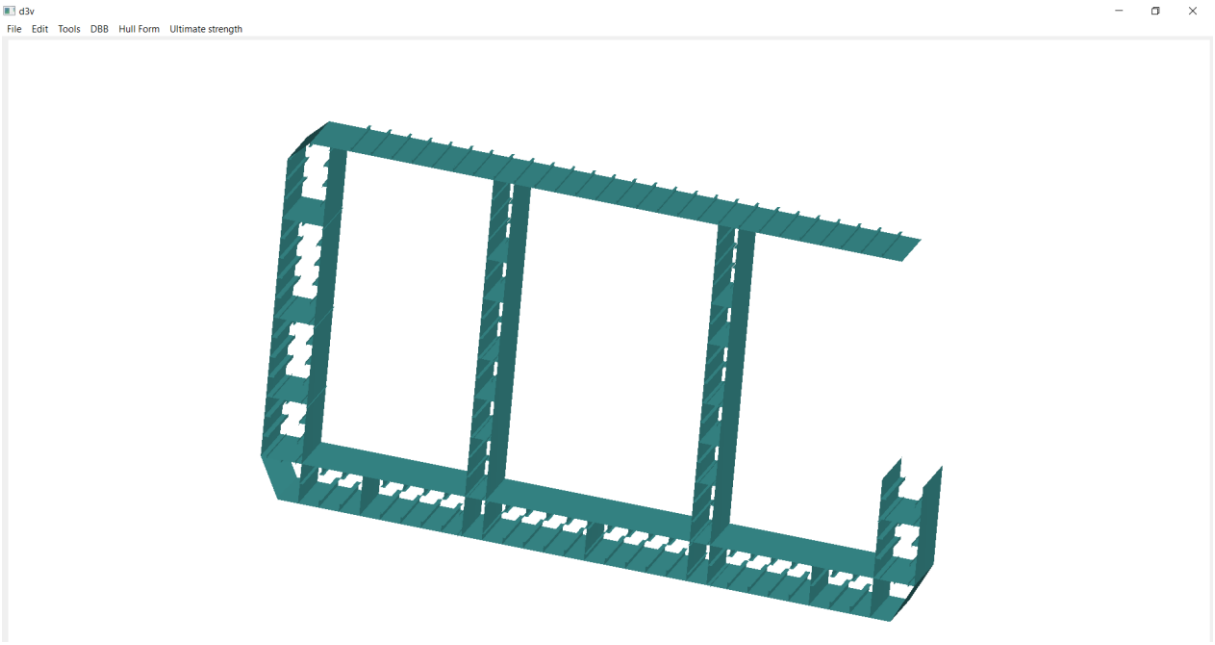


Slika 22. Vizualizacija oštećenja broda nasukavanjem

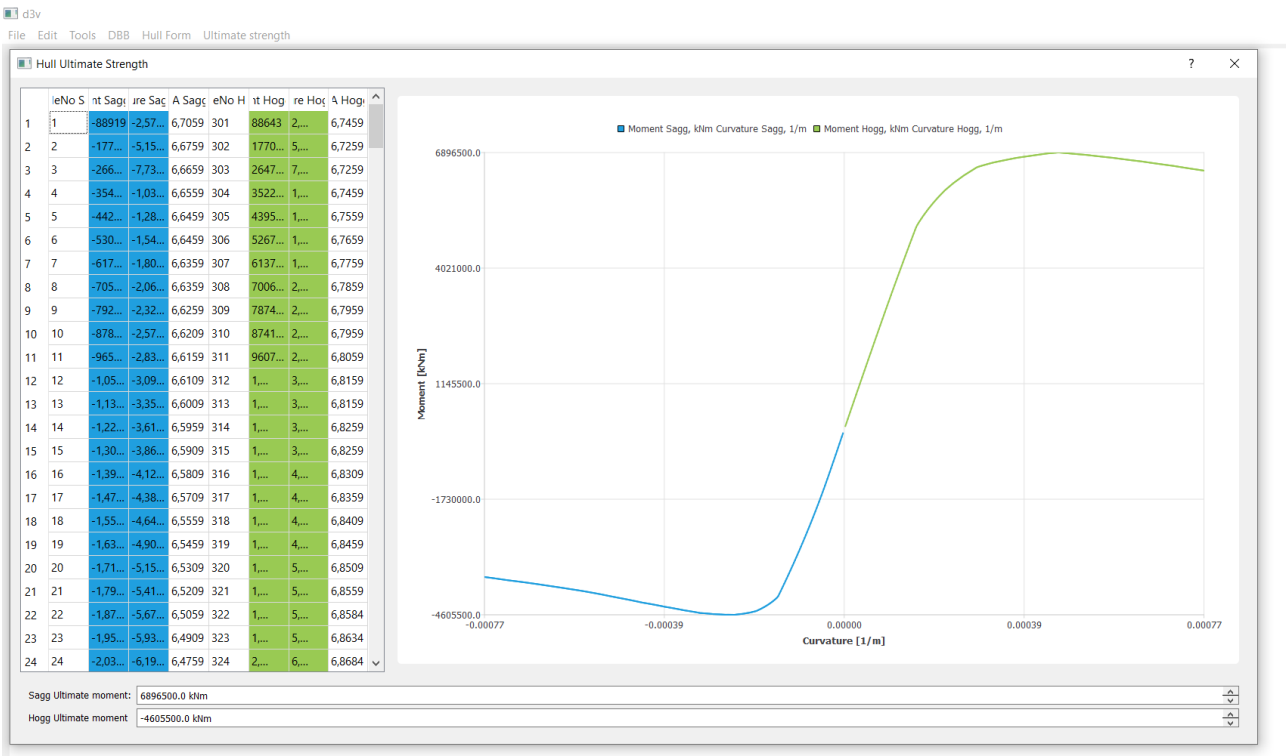


Slika 23. Moment-Curvature Diagram oštećenog broda nasukavanjem

6.5.3. Primjer oštećenja broda nastalog sudaranjem, njegov graf momenata savijanja i zakrivljenosti



Slika 24. Vizualizacija oštećenja broda nastalog sudaranjem



Slika 25. Moment-Curvature Diagram oštećenog broda nastalog sudranjem

7. ZAKLJUČAK

Proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python omogučena je vizualizacija glavnog rebra broda. Sučelje vizualizacije hullultstrength modela jednostavno je za korištenje ali trenutno s malim brojem respoloživih funkcija za prikaz. Uz osnovno poznavanje načina rada programskog jezika Python, moguće je vrlo jednostavno dodavanjem novih funkcija unaprijediti modul. Program koristi Openmesh koji ima ograničenje da ne može više od dva puta koristi isti polubrid, što na spojevima boka oplate i međupalubana stvara problem vizualizacije, riješenje tog problema je korištenjem dodatnih verteksa na istim pozicijama s već postojećim. Korištenje programa jednostavno je za korisnika gdje se jednostavnim odabirom datoteke koja sadrži podatke o glavnom rebru, datoteka učita u program i automatizirano prikaže vizualizaciju, te odabirom u izborniku bira jedna od mogćnosti vizualizacije rješenja proraračuna

LITERATURA

[1] Andrić, J., Metodologija konceptualnog projektiranja brodskih konstrukcija s interakcijom trupnadgrađe, Doktorski rad, Fakultet strojarstva i brodogradnje, Zagreb, 2007.

[2] Žiha, K., Parunov, J, Tušek, B., Granična čvrstoća brodskog trupa, stručni rad, časopis Brodogradnja 58 (1), pp 29-41, 2007.

[3] https://eucenje.fsb.hr/pluginfile.php/30573/mod_resource/content/1/PrezentacijaUS2015.pdf

[4] https://urn.nsk.hr/urn:nbn:hr:235:348479

[5] https://www.torkildson.com/wpcontent/uploads/2015/12/Skriptni_jezici_2012.pdf?fbclid=IwAR3x1l3PGmfK4St0V7ITQGOwY3iKbvLaogx4pV6r1k49HKaLZ-balczzsmw

[6] http://os22lipnjask.skole.hr/upload/os22lipnjask/images/static3/1082/File/python__osnove.pdf

[7] https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/d450_polaznik.pdf

[8] About Qt., url:https://wiki.qt.io/About_Qt

[9] Komaromi, S., QT grafičko korisničko sučelje u Pythonu, Završni rad, Sveučilište Josipa Jurja Strossmayera, Osijek, 2017.

[10] https://www.graphics.rwth-aachen.de/media/openmesh_static/Documentations/OpenMesh-6.2-Documentation/a00030.html

[11] http://repozitorij.fsb.hr/3023/1/Diplomski%2Bzadatak.pdf

[12] http://www.w3.org/TR/SVG11/types.html%23ColorKeywords

[13] http://paulbourke.net/miscellaneous/colourspace/

[14] M. Buconić, Vizualizacija OOFEM modela brodske konstrukcije primjenom programa otvorenog koda linaetal-fsb/d3v, Završni rad, Fakultet strojarstva i brodogradnje, Zagreb, 2020.

PRILOZI

I. CD-R disc

II. Python skripte geofem.py, readxml.py, hullultstrength.py

Prilog II.

geofem.py skripta

```
import openmesh as om
from enum import Enum
import numpy as np
from geometry import Geometry
import pathlib

class ViewType(Enum):
    """!
    ViewType
    """

    constant_color = 0
    face_colors = 1
    face_vertex_colors = 2

class MeshControl():
    def __init__(self):
        self.viewtype=ViewType.constant_color
        self.useviewtreshold=False
        self.uppertreshold=0
        self.lowertreshold = 0
        pass
    def getUpperTresholdColor(self):
        return [0.1,0.1,0.1,1.0]
    def getLowerTresholdColor(self):
        return [0.3,0.3,0.3,1.0]
class Property ():
    def __init__(self):
        self.id = 0
        self.name= ''
        pass
    def init(self,id,name):
        self.id = id
        self.name = name
class GeoEntity():
    def __init__(self):
        self.id=0
        pass
    def appendMesh(self, mesh:om.TriMesh,mc:MeshControl):
        pass
    def init(self,id):
        self.id=id

class Group():
    def __init__(self):
        self.entities = []
        pass


class Material (Property):
    def __init__(self):
        super().__init__()
        self.ReH=0
        self.E=0
        self.ni=0
        pass
```

```
    def init(self, id, name):
        super().init(id,name)

class RodProperty (Property):
    def __init__(self):
        super().__init__()
        pass
    def init(self, id, name):
        super().init(id,name)

class BeamProperty (Property):
    def __init__(self):
        super().__init__()
        self.hw=0
        self.tw=0
        self.bf=0
        self.tf=0
        self.secType=''
        self.material=0
        pass
    def init(self, id, name):
        super().init(id,name)
class StiffLayoutProperty (Property):
    def __init__(self):
        self.beam = 0
        pass
    def init(self, id, name):
        super().init(id,name)
class PlateProperty (Property):
    def __init__(self):
        self.tp=0
        self.material =0
        pass
    def init(self, id, name):
        super().init(id,name)




class Node(GeoEntity):
    def __init__(self):
        super().__init__()
        self.p = np.array([0, 0, 0])
        pass
    def x(self):
        return self.p[0]
    def y(self):
        return self.p[1]
    def z(self):
        return self.p[2]
    def init(self,id,x,y,z):
        super().init(id)
        self.p[0]= x
        self.p[1] = y
        self.p[2] = z


class Element(GeoEntity):
```

```
    def __init__(self):
        super().__init__()
        self.property = 0
        self.nodes = []
        self.face_value=0
        self.vertex_based_values=[]
        self.value_name=""
        pass

    def setFaceValueUsingElementID(self,result:dict):
        self.face_value=result.get(self.id)
        return self.face_value

    def getColorForFaceResult(self,fun_getcolor, minvalue, maxvalue):
        # define color

        if fun_getcolor != None:
            color = fun_getcolor(self.face_value, minvalue, maxvalue)
        else:
            color=[]
        return color

    def getColorForFaceVertexResult(self,fun_getcolor, minvalue, maxvalue):
        # define color
        colors = []
        if fun_getcolor != None:
            for i in range(len(self.vertex_based_values)):
                colors.append(fun_getcolor(self.vertex_based_values[i], minvalue,
maxvalue))

    def addNode(self,node):
        self.nodes.append(node)
        self.vertex_based_values.append(0)
    def init(self,id):
        super().init(id)
    def updateMesh(self,mesh:om.TriMesh,mc:MeshControl, const_color = [0.4, 1.0,
1.0, 1.0],
                   fun_getcolor=None,minvalue = 0,maxvalue = 1 ):
        pass

    def onTPLValue(self):
        self.face_value = 0
        return self.face_value

    def onMatID(self):
        self.face_value = self.property.material.id
        return self.face_value

    def onPropID(self):
        self.face_value = self.property.id
        return self.face_value


class RodElement(Element):
    def __init__(self):
        super().__init__()
        pass
```

```
class BeamElement(Element):
    def __init__(self):
        super().__init__()
        self.wo = np.array([0, 0, 0]) #web orientation

    def onTPLValue(self):
        self.face_value = self.property.tw
        return self.face_value

    def updateMesh(self,mesh:om.TriMesh,mc:MeshControl, const_color = [0.4, 1.0,
1.0, 1.0],
                   fun_getcolor=None,minvalue = 0,maxvalue = 1 ):
        color = const_color
        vhandle = []
        handleColorIndex = []
        fhs = []

        hw=self.property.hw
        tw=self.property.tw
        bf=self.property.bf
        tf = self.property.tf
        x = self.nodes[0].p - self.nodes[1].p
        y = self.wo
        v = np.cross(x, y)
        # z = self.nodes[0].p + self.wo*hw - (v*bf*0.5)

        vhandle.append(mesh.add_vertex(self.nodes[0].p)) #0 točka 1
        handleColorIndex.append(0)
        vhandle.append(mesh.add_vertex(self.nodes[1].p)) #1 točka 2
        handleColorIndex.append(1)

        data = self.nodes[0].p + self.wo*hw  #2 točka 3
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(0)
        data = self.nodes[1].p + self.wo * hw  # 3 točka 4
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(1)
        data = self.nodes[0].p + self.wo*hw - (v*bf*0.5)  #4 točka 5
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(0)
        data = self.nodes[1].p + self.wo*hw - (v*bf*0.5)  #5 točka 6
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(1)
        data = self.nodes[1].p + self.wo*hw + (v*bf*0.5)  #6 točka 7
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(1)
        data = self.nodes[0].p + self.wo*hw + (v*bf*0.5)  #7 točka 8
        vhandle.append(mesh.add_vertex(data))
        handleColorIndex.append(0)

        # data = np.array([self.nodes[0].p]) + np.array(self.wo)*self.hw -
np.cross(np.array([self.nodes[1].p]),np.array(self.wo))*(self.bf*0.5) #4 točak 5
        # vhandle.append(mesh.add_vertex(data))

        fhs.append(mesh.add_face(vhandle[0], vhandle[1], vhandle[2])) #1-2-3
        fhs.append(mesh.add_face(vhandle[2], vhandle[1], vhandle[3])) #3-2-4
        fhs.append(mesh.add_face(vhandle[4], vhandle[5], vhandle[6])) #5-6-7
        fhs.append(mesh.add_face(vhandle[4], vhandle[6], vhandle[7])) # 5-7-8
```

```
        # define color
        if fun_getcolor != None:
            if mc.viewtype == ViewType.face_colors:
                color = self.getColorForFaceResult(fun_getcolor, minvalue,
maxvalue)
                for fh in fhs:
                    mesh.set_color(fh, color)
            if mc.viewtype == ViewType.face_vertex_colors:
                colors = self.getColorForFaceVertexResult(fun_getcolor, minvalue,
maxvalue)
                for ivh in range(len(vhandle)):
                    mesh.set_color(vhandle[ivh], color[handleColorIndex[ivh]])

class TriaElement(Element):
    def __init__(self):
        super().__init__()
        pass

    def onTPLValue(self):
        self.face_value = self.property.tp
        return self.face_value

class StiffTriaElement(TriaElement):
    def __init__(self):
        super().__init__()
        self.layout=0
        pass
    def init(self,id):
        self.id=id


class QuadElement(Element):
    def __init__(self):
        super().__init__()
        pass
    def init(self,id):
        self.id=id

    def onTPLValue(self):
        self.face_value = self.property.tp
        return self.face_value


class StiffQuadElement(QuadElement):
    def __init__(self):
        super().__init__()
        self.layout=0
        pass
    def init(self,id):
        self.id=id
    def updateMesh(self,mesh:om.TriMesh,mc:MeshControl, const_color = [0.4, 1.0,
1.0, 1.0],
                   fun_getcolor=None,minvalue = 0,maxvalue = 1 ):

        color = const_color
        vhandle = []
        fhs = []
        data = np.array([self.nodes[0].p[0], self.nodes[0].p[1],
self.nodes[0].p[2]])
```

```
        vhandle.append(mesh.add_vertex(data))
        data = np.array([self.nodes[1].p[0], self.nodes[1].p[1],
self.nodes[1].p[2]])
        vhandle.append(mesh.add_vertex(data))
        data = np.array([self.nodes[2].p[0], self.nodes[2].p[1],
self.nodes[2].p[2]])
        vhandle.append(mesh.add_vertex(data))
        data = np.array([self.nodes[3].p[0], self.nodes[3].p[1],
self.nodes[3].p[2]])
        vhandle.append(mesh.add_vertex(data))

        fhs.append(mesh.add_face(vhandle[0], vhandle[1], vhandle[2]))
        fhs.append(mesh.add_face(vhandle[0], vhandle[2], vhandle[3]))

        # define color
        if fun_getcolor != None:
            if mc.viewtype == ViewType.face_colors:
                color =self.getColorForFaceResult(fun_getcolor, minvalue,
maxvalue)
                for fh in fhs:
                    mesh.set_color(fh, color)
            elif mc.viewtype == ViewType.face_vertex_colors:
                colors =self.getColorForFaceVertexResult(fun_getcolor, minvalue,
maxvalue)
                for ivh in range(len(vhandle)):
                    mesh.set_color(vhandle[ivh], color[ivh])



class Units():
    def __init__(self):
        self.user2si_length=1
        self.user2si_force=1
        self.name_length='m'
        self.name_force = 'N'
        pass
class MaestroElementAssociation():
    def __init__(self):
        self.strakeGirder2fe = {}
        self.strakePlate2fe = {}
        self.strakeFrame2fe = {}
        self.endPoint2node = {}
        self.endPointStrakes = {}

    def addStrakePlate(self,key:int,elList:[]):
        self.strakePlate2fe[key]=elList

    def addStrakeGirder(self,key:int,elList:[]):
        self.strakeGirder2fe[key]=elList

    def addStrakeFrame(self,key:int,elList:[]):
        self.strakeFrame2fe[key]=elList

    def addEndPointNode(self,key:int,nodeFeTag:int):
        feTagList = self.endPoint2node.setdefault(key,[])
        feTagList.append(nodeFeTag)

    def addEndPointStrake(self, key:int, strakeID:int):
```

```
        strakeList = self.endPointStrakes.setdefault(key,[])
        strakeList.append(strakeID)

    def getPlateElsForEnpoint(self, endpointID):
        connectedElements=[]
        strakeList = self.endPointStrakes.get(endpointID)
        if strakeList is not None:
            for idStrake in strakeList:
                for idEl in self.strakePlate2fe[idStrake]:
                    connectedElements.append(idEl)
        return connectedElements

    def getPlateElemForStrake(self,strakeID):
        return self.strakePlate2fe.get(strakeID)

    def getGirderBeamElemForStrake(self,strakeID):
        return self.strakeGirder2fe.get(strakeID)


class LusaElementAssociation():
    def __init__(self):
        self.spc2fe = {}
        self.plate2fe = {}
        self.hc2fe = {}

    def addPlate(self, key:int, strakeID):
        self.plate2fe[key]=strakeID
    def addSPC(self,key:int,strakeID):
        self.spc2fe[key]=strakeID
    def addHC(self,key:int,strakeID):
        self.hc2fe[key]=strakeID

class GeoFEM(Geometry):
    def __init__(self):
        super().__init__()
        self.nodes = {}
        self.elements = {}
        self.materials = {}
        self.properties = {}
        self.stiflayouts = {}
        self.groups = {}
        self.meshcontrol = 0
        self.units= Units()
        self.mc = MeshControl()
        self.mas = MaestroElementAssociation()
        self.element_results= {}
        self.element_vertex_results = {}
        self.vertex_results = {}
        self.model_results = {}
        self.result_name=""
        self.is_node_result=False
        self.is_element_result = False
        self.attrib_val_functions ={}
        self.populateAtribValFunctionsDictionary()
        self.minValue=0
        self.maxValue=0
        self.numDiffValues =0
        self.valueIndexColor = {}
```

```
        self.drawLegend = False
        self.legendValues = []
        self.legendColors = []
        self.legendTitle = ""
        self.fixed_color_list = self.initColorList()
        self.max_fixed_colors= len(self.fixed_color_list)

        pass
    def initColorList(self):
        colors = [[0, 0, 255,255],[128, 0, 128,255],[222, 184, 135,255],[255, 165,
0,255],[0, 255, 0,255],
                  [ 0, 128, 0,255],[128, 0, 0,255],[255, 0, 0,255],[255, 192,
203,255],[222, 184, 135,255],
                  [255, 165, 0,255],[255, 127, 80,255],[128, 128, 0,255],[255,
255, 0,255],[245, 245, 220,255],
                  [0, 255, 0,255],[ 0, 128, 0,255],[245, 255, 250,255],[0, 128,
128,255],[0, 255, 255,255],
                  [0, 0, 128,255],[230, 230, 250,255],[255, 0, 255,255],[205, 133,
63,255]]
        floatColors = []
        for color in colors:
            floatColors.append([x / 255 for x in color])
        return floatColors

    def prepareModelForVisualization(self,key):
        self.minValue = float("inf")
        self.maxValue = float("-inf")
        self.numDiffValues = 0
        self.valueIndexColor.clear()

        fatrib= self.attrib_val_functions.get(key)

        if fatrib == None:
            self.doResultValue(key)
        else:
            fatrib(key)
        self.mc.lowertreshold=self.minValue
        self.mc.uppertreshold=self.maxValue
        self.mc.viewtype = ViewType.face_colors
        self.legendTitle=key
        self.regenerateusingcolor()

    def getContinuousColor(self, v, vmin, vmax):
        color = [1.0, 1.0, 1.0, 1.0]
        if v > self.mc.uppertreshold:
            return self.mc.getUpperTresholdColor()
        elif v < self.mc.lowertreshold:
            return self.mc.getLowerTresholdColor()
        vmin   = max(vmin,self.mc.lowertreshold)
        vmax = min(vmax, self.mc.uppertreshold)

        if v < vmin:
            v = vmin
        if v > vmax:
            v = vmax
        dv = vmax - vmin

        if (v < (vmin + 0.25 * dv)):
            color[0] = 0
```

```
            color[1] = 4 * (v - vmin) / dv
        elif (v < (vmin + 0.5 * dv)):
            color[0] = 0
            color[2] = 1 + 4 * (vmin + 0.25 * dv - v) / dv
        elif (v < (vmin + 0.75 * dv)):
            color[0] = 4 * (v - vmin - 0.5 * dv) / dv
            color[2] = 0
        else:
            color[1] = 1 + 4 * (vmin + 0.75 * dv - v) / dv
            color[2] = 0
        return color

    def getColorFromList(self, v, vmin, vmax):
        if v > self.mc.uppertreshold:
            return self.mc.getUpperTresholdColor()
        elif v < self.mc.lowertreshold:
            return self.mc.getLowerTresholdColor()

        index=self.getValueColorIndex(v)
        color = self.fixed_color_list[index]
        return color


    def prepContColorLegend(self,fun_getcolor,minVal, maxVal,nColor):
        self.legendValues.clear()
        self.legendColors.clear()
        self.drawLegend = True
        minVal = max(minVal, self.mc.lowertreshold)
        maxVal = min(maxVal, self.mc.uppertreshold)
        legendValues=np.linspace(minVal,maxVal,nColor)
        for x in legendValues:
            self.legendValues.append(f"{x:.4g}")
        for val in legendValues:
            color = fun_getcolor(val, minVal, maxVal)
            self.legendColors.append(color)

    def prepListColorLegend(self,fun_getcolor):
        self.legendValues.clear()
        self.legendColors.clear()
        self.drawLegend = True
        for key,index in self.valueIndexColor.items():
            if  key < self.mc.lowertreshold or key > self.mc.uppertreshold:
                continue
            self.legendValues.append(f"{key:.4g}")
            color = fun_getcolor(key, 0, self.numDiffValues)
            self.legendColors.append(color)

    # endregion
    # region Attribute Value Functions

    def populateAtribValFunctionsDictionary(self):
        self.addAttValFunc('TPL', self.doTPLValue)
        self.addAttValFunc('Material ID', self.doMaterialIDValue)
        self.addAttValFunc('Property ID', self.doPropertyIDValue)

    def addAttValFunc(self, key, f):
        self.attrib_val_functions[key] = f

    def doTPLValue(self,key):
```

```
        for el in self.elements.values():
            val= el.onTPLValue()
            self.checkMinMax(val)


    def doMaterialIDValue(self,key):
        for el in self.elements.values():
            val = el.onMatID()
            self.checkMinMax(val)

    def doPropertyIDValue(self, key):
        for el in self.elements.values():
            val = el.onPropID()
            self.checkMinMax(val)


    def doResultValue(self,key):
        self.setValueToItemResults(key)

    # endregion


    def checkMinMax(self,val):
        if val > self.maxValue:
            self.maxValue = val
        if val < self.minValue:
            self.minValue = val

        if self.numDiffValues <= self.max_fixed_colors:
            index = self.getValueColorIndex(val)
            if index == None:
                self.valueIndexColor[val]=self.numDiffValues
                self.numDiffValues=self.numDiffValues+1


    def getValueColorIndex(self,value):
        return self.valueIndexColor.get(value)


    def isElementFaceResult(self):
        return self.is_element_result and (not self.is_node_result)

    def isElementNodeResult(self):
        return self.is_element_result and self.is_node_result

    def isNodeResult(self):
        return (not self.is_element_result) and self.is_node_result

    def setValueToItemResults(self, resultName):
        result = self.element_results.get(resultName)
        if result != None:
            self.is_element_result= True
            self.is_node_result = False
            for key, el in self.elements.items():
                val = el.setFaceValueUsingElementID(result.feres)
                self.checkMinMax(val)
            return

        result = self.element_vertex_results.get(resultName)
```

```
        if result != None:
            self.is_element_result= True
            self.is_node_result = True
            return

        result = self.vertex_results.get(resultName)
        if result != None:
            self.is_element_result = False
            self.is_node_result = True
            return


    def addNode(self,item):
        self.nodes[item.id]=item
    def getNode(self,id):
        return self.nodes[id]
    def getMaterial(self,id):
        return self.materials[id]
    def getProperty(self,id):
        return self.properties[id]
    def getStiffLayout(self,id):
        return self.stiflayouts[id]
    def addElement(self,item):
        self.elements[item.id]=item
    def addProperty(self,item):
        self.properties[item.id]=item
    def addMaterial(self,item):
        self.materials[item.id]=item
    def addStiffLayout(self, item):
        self.stiflayouts[item.id] = item

    def regenerate(self):
        mesh= om.TriMesh()

        self.mc.viewtype = ViewType.constant_color
       # mesh.request_face_colors()
        for el in self.elements.values():
            el.updateMesh(mesh,self.mc)
        pass
        self.mesh = mesh

    def regenerateusingcolor(self):
        fun_getcolor = self.getColorFromList
        if self.numDiffValues > self.max_fixed_colors:
            fun_getcolor= self.getContinuousColor

        mesh= om.TriMesh()
        if self.mc.viewtype == ViewType.constant_color or self.mc.viewtype ==
ViewType.face_colors:
            #mesh.release_vertex_colors()
            mesh.request_face_colors()
        elif self.mc.viewtype == ViewType.face_vertex_colors:
            #mesh.release_face_colors()
            mesh.request_vertex_colors()

        const_color = [0.4, 1.0, 1.0, 1.0]

        for el in self.elements.values():
            el.updateMesh(mesh,self.mc,
```

```
const_color,fun_getcolor,self.minValue,self.maxValue)

        self.mesh = mesh

        if self.numDiffValues > self.max_fixed_colors:
            self.prepContColorLegend(fun_getcolor,self.minValue, self.maxValue,
12)
        else:
            self.prepListColorLegend(fun_getcolor)



    def setResultValuesOnElements(self):
        pass



    # def showFaceColorP(self, propDict):
    #     colors = [[0, 0, 255, 255], [128, 0, 128, 255], [222, 184, 135, 255],
[255, 165, 0, 255], [0, 255, 0, 255],
    #               [0, 128, 0, 255], [128, 0, 0, 255], [255, 0, 0, 255], [255,
192, 203, 255], [222, 184, 135, 255],
    #               [255, 165, 0, 255], [255, 127, 80, 255], [128, 128, 0, 255],
[255, 255, 0, 255], [245, 245, 220, 255],
    #               [0, 255, 0, 255], [0, 128, 0, 255], [245, 255, 250, 255], [0,
128, 128, 255], [0, 255, 255, 255],
    #               [0, 0, 128, 255], [230, 230, 250, 255], [255, 0, 255, 255],
[205, 133, 63, 255]]
    #
    #     floatColors = []
    #     for color in colors:
    #         floatColors.append([x / 255 for x in color])
    #     mesh = self.mesh
    #     mesh.request_face_colors()
    #     propColorDict = {}
    #     self.legendValues.clear()
    #     self.legendColors.clear()
    #     self.drawLegend = False
    #     for el in self.element2Face:
    #         idProp=propDict[el]
    #         nuc=len(propColorDict)
    #         indexColor = 0
    #         if idProp in propColorDict:
    #             indexColor=propColorDict[idProp]
    #         else:
    #             propColorDict[idProp]=nuc
    #             indexColor=nuc
    #             self.legendValues.append(str(idProp))
    #             self.legendColors.append(floatColors[indexColor])
    #         for fh in self.element2Face[el]:
    #             mesh.set_color(fh, floatColors[indexColor])
    #         pass
    #     if len(self.legendValues)> 0:
    #         self.drawLegend=True
    #     pass


class Result():
    def __init__(self, name):
```

```
        self.name = name
        pass


class GeneralResultsDictionary(Result):
    def __init__(self, name):
        super().__init__(name)
        self.results = {}
        pass

    def appendValue(self, key, value):
        resultList = self.results.setdefault(key, [])
        resultList.append(value)

    def addValues(self, key, values: []):
        self.results[key] = values

    def addValues2(self, key, values: list):
        self.results[key] = values.copy()

    def getValues(self, key):
        return self.results[key]

    def getValue(self, key, index: int):
        return self.results[key][index]

    def appdendListwithResultData(self, x: list, y: list, iresx: int, iresy: int):
        for res in self.results.values():
            x.append(res[iresx])
            y.append(res[iresy])

    def appdendListwithKeyPairedResultData(self, x: list, y: list, iresy: int):
        for key, res in self.results:
            x.append(key)
            y.append(res[iresy])

class GeneralResultsTableModel(Result):
    def __init__(self, name):
        super().__init__(name)
        self.data =[]
        self.column_names =[]
        pass

    def appendName(self, name: str):
        self.column_names.append(name)

    def addRow(self, values: list):
        self.data.append(values)

    def getRowValues(self, row_index):
        return self.data[row_index]

    def getValue(self, row_index, column_index: int):
        return self.data[row_index][column_index]

class ElementResult(Result):
    def __init__(self, name):
        super().__init__(name)
        self.feres = {}
```

```
        pass

    def getValue(self, feID):
        return self.feres.get(feID)

    def setValue(self, feID, value):
        self.feres[feID]=value

    def keyExist(self,feID):
        return feID in self.feres


class FEMModelResults:
    def __init__(self, name):
        self.name = name
        pass

    def readOutput(self, path):
        pass

    def setResultsToModel(self, fem: GeoFEM):
        pass


class LusaResults(FEMModelResults):
    def __init__(self, name, mas:MaestroElementAssociation):
        super().__init__(name)
        self.las = LusaElementAssociation()
        self.mas=mas
        self.lers = {} #Lusa element results
        self.modres={}
        pass

    def readOutput(self, path):
        abspath1 = '\\'.join(path.split('\\')[0:-1])
        abspath2 = '/'.join(path.split('/')[0:-1])
        if len(abspath2) > len(abspath1):
            abspath=abspath2 + '/'
        else:
            abspath=abspath1 + '\\'

        abspath_hoggCSD = abspath + 'LUSAhoggCSD.OUT'
        abspath_saggCSD=  abspath + 'LUSAsaggCSD.OUT'

        self.readCSDFile(abspath_hoggCSD,False)
        self.readCSDFile(abspath_saggCSD,True)

        abspath_hogg = abspath + 'LUSAhogg.OUT'
        abspath_sagg = abspath + 'LUSAsagg.OUT'

        iterationResults = GeneralResultsTableModel('Lusa iteration results Sagg')
        self.modres[iterationResults.name]=iterationResults


        iterationResults.appendName('CycleNo Sagg')
        iterationResults.appendName('Moment Sagg, kNm')
        iterationResults.appendName('Curvature Sagg, 1/m')
        iterationResults.appendName('y_NA Sagg, m')
```

```
        self.readMainLusaFile(abspath_sagg,True,iterationResults)

        iterationResults = GeneralResultsTableModel('Lusa iteration results Hogg')
        self.modres[iterationResults.name] = iterationResults
        iterationResults.appendName('CycleNo Hogg')
        iterationResults.appendName('Moment Hogg, kNm')
        iterationResults.appendName('Curvature Hogg, 1/m')
        iterationResults.appendName('y_NA Hogg, m')

        self.readMainLusaFile(abspath_hogg, False, iterationResults)

        pass

    def readMainLusaFile(self, path, isSagg,
tableResult:GeneralResultsTableModel):
        file = pathlib.Path(path)
        if not file.exists():
            return
        f = open(path, "r")
        nlines2skip = 0
        isCycleData = False

        for line in f:
            if nlines2skip > 0:
                nlines2skip = nlines2skip - 1
                continue
            line = ' '.join(line.split())
            if line.startswith('*'):
                continue
            if line == "" or line == " ":
                continue

            if 'HULL MODULE RESPONSE DATA' in line:
                nlines2skip = 4
                isCycleData = True
                continue
            if 'ULTIMATE CAPACITY IS' in line:
                f.close()
#                 if isSagg:
#                     tableResult.data.reverse()
                return
            sline = line.split(" ")
            if len(sline) == 0:
                continue
            if isCycleData:
                if len(sline) == 4:
                    rowValues=[0]*4
                    rowValues[0]= int(sline[0])
                    rowValues[1] = float(sline[1])
                    rowValues[2] = float(sline[2])
                    rowValues[3] = float(sline[3])
                    tableResult.addRow(rowValues)

        f.close()

    def readCSDFile(self,path,isSagg):
        file=pathlib.Path(path)
        if not file.exists():
            return
```

```
        f = open(path, "r")
        nlines2skip=0
        isSPCdata=False
        isGPCdata = False
        isHCdata = False
        if isSagg:
            collapse_stress = ElementResult('Collapse Stress Sagg')
            collapse_mod = ElementResult('Collapse Mod Sagg')
            collapse_cycle = ElementResult('Collapse Cycle Sagg')
        else:
            collapse_stress = ElementResult('Collapse Stress Hogg')
            collapse_mod = ElementResult('Collapse Mod Hogg')
            collapse_cycle = ElementResult('Collapse Cycle Hogg')

        self.lers[collapse_stress.name]=collapse_stress
        self.lers[collapse_mod.name] = collapse_mod
        self.lers[collapse_cycle.name] = collapse_cycle
        for line in f:
            if  nlines2skip > 0:
                nlines2skip=nlines2skip-1
                continue
            line = ' '.join(line.split())
            if line.startswith('*'):
                continue
            if line == "" or line == " ":
                continue

            if 'Stiffener - Plate Combinations (SPCs)' in line:
                nlines2skip=4
                isSPCdata=True
                continue
            if 'Girder - Plate Combinations (GPCs)' in line:
                nlines2skip=4
                isGPCdata=True
                isSPCdata=False
                continue
            if 'Hard Corners (HCs)' in line:
                nlines2skip = 4
                isGPCdata = False
                isHCdata = True
                continue
            sline = line.split(" ")
            if  len(sline)== 0:
                continue
            el_no_lusa=-1
            if isSPCdata:
                if len(sline) > 5:
                    strakeNo    = int(sline[0])
                    el_no_lusa        = int(sline[1])
                    self.las.addPlate(el_no_lusa, strakeNo)

                    elIDs=self.mas.getPlateElemForStrake(strakeNo)
                    cc_new = int(sline[4])
                    for id_el in elIDs:
                        bAddNew=True
                        cc_old= collapse_cycle.getValue(id_el)
                        if cc_old != None and cc_old < cc_new:
                            pass
                        else:
```

```
                            collapse_stress.setValue(id_el, float(sline[2]))
                            collapse_mod.setValue(id_el, float(sline[3]))
                            collapse_cycle.setValue(id_el, cc_new)
            elif isGPCdata:
                if len(sline) > 5:
                    strakeNo    = int(sline[0])
                    el_no_lusa        = int(sline[1])
                    self.las.addSPC(el_no_lusa, strakeNo)

                    elIDs = self.mas.getGirderBeamElemForStrake(strakeNo)
                    elIDsPlate = self.mas.getPlateElemForStrake(strakeNo)
                    if elIDs == None:
                        elIDs = elIDsPlate
                    elif elIDsPlate != None:
                        for el in elIDsPlate:
                            elIDs.append(el)
                    cc_new = int(sline[4])
                    for id_el in elIDs:
                        bAddNew = True
                        cc_old = collapse_cycle.getValue(id_el)
                        if cc_old != None and cc_old < cc_new:
                            pass
                        else:
                            collapse_stress.setValue(id_el, float(sline[2]))
                            collapse_mod.setValue(id_el, float(sline[3]))
                            collapse_cycle.setValue(id_el, cc_new)
            elif isHCdata:
                if len(sline) > 5:
                    endPtNo    = int(sline[0])
                    el_no_lusa        = int(sline[1])
                    self.las.addHC(el_no_lusa, endPtNo)
                    elIDs = self.mas.getPlateElsForEnpoint(endPtNo)
                    cc_new = int(sline[4])
                    for id_el in elIDs:
                        bAddNew = True
                        cc_old = collapse_cycle.getValue(id_el)
                        if cc_old != None and cc_old < cc_new:
                            pass
                        else:
                            collapse_stress.setValue(id_el, float(sline[2]))
                            collapse_mod.setValue(id_el, float(sline[3]))
                            collapse_cycle.setValue(id_el, cc_new)

        f.close()
        pass

    def setResultsToModel(self, fem: GeoFEM):
```

readxml.py skripta

```
from geofem import GeoFEM,Node,StiffQuadElement, Material,
BeamProperty,StiffLayoutProperty,PlateProperty
from geofem import StiffTriaElement, BeamElement, RodElement,Element,LusaResults
import  numpy as np
import uuid
import xml.etree.ElementTree as ET
def getFloat(item, key):
    return  float(item.attrib[key])
def getInt(item,key):
    return  int(item.attrib[key])
def getGuid(item,key):
    return  uuid.UUID(item.attrib[key])
def getStr(item, key):
    return  (item.attrib[key])

def getNPVector(item, key):
    npVec=  (np.array(item.attrib[key].split(','))).astype(np.float)
    return  npVec

def getIntList(item, key):
    intList=  (np.array(item.attrib[key].split(','))).astype(np.int)
    intList = intList.tolist()
    return  intList


class MaestroXML:

    def __init__(self, xmlPath):
        self.xmlpath = xmlPath
    def addElement(self,fem:GeoFEM,elem:Element,id,idProp,nodeIds):
        sNodeIds = nodeIds.split(' ')
        elem.init(id)
        elem.property = fem.getProperty(idProp)
        for idNod in sNodeIds:
            node = fem.getNode(int(idNod))
            elem.addNode(node)
        fem.addElement(elem)
    def processMaestroModule(self,module,fem:GeoFEM):
        for item in module:
            if item.tag =='StrakeList':
                for strake in item:
                    strakeID = getInt(strake, 'iTag')
                    ep1=getInt(strake,'EndPt0')
                    ep2=getInt(strake,'EndPt1')
                    fem.mas.addEndPointStrake(ep1,strakeID)
                    fem.mas.addEndPointStrake(ep2, strakeID)
                    for att in strake:
                        if 'sFeTag' not in att.attrib:
                            continue
                        if att.tag == 'Plate':
                            idList=getIntList(att,'sFeTag')
                            fem.mas.addStrakePlate(strakeID,idList)
                        elif att.tag == 'Frame':
                            idList=getIntList(att,'sFeTag')
                            fem.mas.addStrakeFrame(strakeID, idList)
                        elif att.tag == 'Girder':
                            idList=getIntList(att,'sFeTag')
```

```
                            fem.mas.addStrakeGirder(strakeID, idList)
                    pass
            elif item.tag =='CompaundList':
                print ('Maestro Structural Element Type not implemented:' +
item.tag)
            elif item.tag =='BarList':
                print ('Maestro Structural Element Type not implemented:' +
item.tag)
            elif item.tag =='QuadList':
                print ('Maestro Structural Element Type not implemented:' +
item.tag)
            elif item.tag =='RodList':
                print ('Maestro Structural Element Type not implemented:' +
item.tag)

    def readModelToGeoFEM(self,fem:GeoFEM):
        fname = self.xmlpath
        tree = ET.parse(fname)
        root = tree.getroot()
        jobctrl = 0
        nodes = 0
        elements = 0
        properties = 0
        stiflayouts = 0
        materials=0
        bars=0
        plates=0
        groups = 0
        evaluation = 0
        units =0
        coarseMesh=0
        for child in root:
            tag0=child.tag
            for ch in child:
                #print(ch.tag, ch.attrib)
                sname=(ch.attrib['sName'])
                if sname == 'Maestro':
                    units = ch[0][0]
                    coarseMesh =ch[1][1][0]
                elif sname=='FeModel':
                    jobctrl=ch[0][0][0]
                    nodes = ch[0][0][1]
                    elements = ch[0][0][2]
                    properties = ch[0][0][3]
                    #stiflayouts = ch[0][0][4]
                    groups = ch[0][0][5]
                    #evaluation = ch[0][0][5][0]
                if sname=='FeStifLayout':
                    stiflayouts = ch[0]
                if sname=='MaterialIso':
                    materials = ch[0]
                if sname == 'FePropBar':
                    bars = ch[0]
                if sname == 'FePropPlate':
                    plates = ch[0]


        dbars = {}
        dstifflayprops ={}
```

```
        dmaterials = {}
        dproperties={}
        for item in units:
            unit = Material()
            if item.attrib['sTag']== "length":
                fem.units.name_length=item.attrib['sLabel']
                fem.units.user2si_length = getFloat(item,"dUserToSI")
            if item.attrib['sTag']== "force":
                fem.units.name_force=item.attrib['sLabel']
                fem.units.user2si_force = getFloat(item,"dUserToSI")
        for sub0 in coarseMesh:
            if sub0.tag=='module':
                self.processMaestroModule(sub0)
            else:
                for sub1 in sub0:
                    if sub1.tag == 'module':
                        self.processMaestroModule(sub1,fem)
                    else:
                        for sub2 in sub1:
                            if sub2.tag == 'module':
                                self.processMaestroModule(sub2)
                            else:
                                for sub3 in sub2:
                                    if sub3.tag == 'module':
                                        self.processMaestroModule(sub3)
                                    else:
                                        print ('Nesting of substructures of level
> 4 not implemented')

        for item in materials:
            material = Material()
            material.init(getInt(item, 'iTag'),item.attrib['sName'])
            guid=getGuid(item,'idSelf')
            dmaterials[guid]=material
            fem.addMaterial(material)
            material.E=getFloat(item, 'dYoungs')
            material.ni = getFloat(item, 'dPoisson')
            material.ReH = getFloat(item, 'dYield')
        for item in bars:
            barprop = BeamProperty()
            barprop.init(getInt(item, 'iId'),item.attrib['sName'])
            guid=getGuid(item,'idSelf')
            dbars[guid]=barprop
            fem.addProperty(barprop)
            barprop.hw = getFloat(item,"dWebHeight")
            barprop.tw = getFloat(item,"dWebThick");
            barprop.bf = getFloat(item,"dFlangeBreadth");
            barprop.tf = getFloat(item,"dFlangeThick");
            barprop.secType = getStr(item,"sSecType");
            if item[0].tag == 'Moniker':
                guid=getGuid(item[0],'idrefObj')
                barprop.material=dmaterials[guid]
        for item in stiflayouts:
            stifflay = StiffLayoutProperty()
            stifflay.init(getInt(item, 'iTag'), item.attrib['sName'])
            fem.addStiffLayout(stifflay)
            if item[0].tag == 'Moniker':
                guid = getGuid(item[0], 'idrefObj')
                stifflay.beam = dbars[guid]
```

```
        for item in plates:
            plate = PlateProperty()
            plate.init(getInt(item, 'iId'),item.attrib['sName'])
            fem.addProperty(plate)
            plate.tp = getFloat(item[0][0],"dThick")
            if item[0][0][0].tag == 'Moniker':
                guid=getGuid(item[0][0][0],'idrefObj')
                plate.material=dmaterials[guid]

        for item in nodes:
            node=Node()
            node.init(getInt(item, 'iId'), getFloat(item, 'dX'), getFloat(item,
'dY'), getFloat(item, 'dZ'))
            fem.addNode(node)
        for item in elements:
            if  item.tag == 'Quad':
                elem = StiffQuadElement()
                elem.layout     = fem.getStiffLayout(getInt(item, "iIdLayout"))
                self.addElement(fem, elem, getInt(item, 'iId'), getInt(item,
"iIdProp"), getStr(item, 'sNodeIds'))
            elif  item.tag == 'Tri':
                elem = StiffTriaElement()
                elem.layout     = fem.getStiffLayout(getInt(item, "iIdLayout"))
                self.addElement(fem, elem, getInt(item, 'iId'), getInt(item,
"iIdProp"), getStr(item, 'sNodeIds'))
            elif  item.tag == 'Bar':
                elem = BeamElement()
                sNodeIds = item.attrib['sNodeIds'].split(' ')
                elem.wo= getNPVector(item,'sWebVec')
                self.addElement(fem, elem, getInt(item, 'iId'), getInt(item,
"iIdProp"),getStr(item, 'sNodeIds'))
            elif  item.tag == 'Rod':
                elem = RodElement()
                self.addElement(fem, elem, getInt(item, 'iId'), getInt(item,
"iIdProp"), getStr(item, 'sNodeIds'))

        # for item in groups:
        #     print(item.tag)
            # print(item.attrib)
        # for item in evaluation:
        #     print(item.tag)
            # print(item.attrib)
        lusaresult = LusaResults('Lusa Results',fem.mas)
        lusaresult.readOutput(self.xmlpath)
        for key, value in lusaresult.lers.items():
            fem.element_results.setdefault(key,value)
        for key, value in lusaresult.modres.items():
            fem.model_results.setdefault(key,value)

pass
```

hullultstrength.py skripta

```
from iohandlers import IOHandler
from signals import Signals
import os
import numpy as np
import readxml
import geofem
import csv




class HullUltStrength (geofem.GeoFEM):
    def __init__(self,fileName):
        self.filename = fileName
        super().__init__()
        self.readModel()

    def readModel(self):
        with open(self.filename, newline='') as csvfile:
            hfr = csv.reader(csvfile, delimiter='\t', quotechar='|')
            data=[]
            for row in hfr:
                rown=[]
                for x in row:
                    rown.append(x)
                data.append(rown)
        xmlfile = data[0][1]
        abspath1 = '\\'.join(self.filename.split('\\')[0:-1])
        abspath2 = '/'.join(self.filename.split('/')[0:-1])
        if len(abspath2) > len (abspath1):
            abspath = abspath2 + '/' + xmlfile
        else:
            abspath = abspath1 + '\\' + xmlfile

        m = readxml.MaestroXML(abspath)
        m.readModelToGeoFEM(self)
        self.regenerate()
        return
```