Izrada modula za proračun stabiliteta triangularizirane forme broda u okviru programa otvorenog koda d3v-gsd

Halavanja, Lovro

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Lovro Halavanja

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

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

Student:

Lovro Halavanja

Zagreb, 2023.

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

Zahvaljujem se mentoru izv. prof. dr. sc. Peri Prebegu i mag. ing. Ivanu Muniću na velikoj pomoći i strpljenju tokom izrade ovog rada.

Lovro Halavanja

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 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Lovro Halavanja** JMBAG: **0035217324**

Naslov rada na hrvatskom jeziku: **Izrada modula za proračun stabiliteta triangularizirane forme broda u okviru programa otvorenog koda d3v-gsd**

Naslov rada na engleskom jeziku: **Development of a ship stability module of triangulised hull form as a part of a d3v-gsd open source software**

Opis zadatka:

Pri projektiranju broda i brodske forme neophodno je provesti analizu stabiliteta. Program otvorenog koda d3v-gsd (Design visualizer for General Ship Design) proširenje je programa linaetal-fsb/d3v, koji omogućuje trodimenzijsku vizualizaciju brodske forme, izračun hidrostatičkih karakteristika te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. U radu je potrebno, proširenjem funkcionalnosti programa otvorenog koda d3v-gsd u programskom jeziku Python, omogućiti proračun stabiliteta broda koristeći triangularizirani zapis brodske forme.

Zadatak obuhvaća sljedeće:

- upoznavanje s trenutnom verzijom programa otvorenog koda d3v-gsd
- upoznavanje sa zahtjevima za softver za proračun stabiliteta broda u neoštećenom stanju
- izradu Python modula za izračun istisnine nagnutog broda, pri čemu je brodska forma zadana u mrežom trokuta
- izradu Python modula za izračun težišta istisnine nagnutog broda, pri čemu je brodska forma zadana u mrežom trokuta
- izradu Python modula za izračun ravnotežne vodne linije broda za proizvoljno stanje krcanja, pri čemu je brodska forma zadana u mrežom trokuta
- izradu Python modula za određivanje krivulje poluga statičkog stabiliteta za proizvoljno stanje krcanja
- usporedbu krivulje poluga statičkog stabiliteta dobivene izrađenim Python modulima sa krivuljom generiranom programom Orca3D .

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

Zadatak zadan:
30. 11. 2022.

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Zadatak zadao:
Izv. prof. dr. sc. Pero Prebeg

Predsjednik Povjerenstva:
Izv. prof. dr. sc. Ivan Ćatipović

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
GZ	m	Poluga statičkog stabiliteta
Δ	t	istisnina
heel	°	Poprečni nagib broda
trim	°	uzdužni nagib broda
drought	m	gaz
G	m	Položaj težišta broda
B	m	Položaj težišta istisnine broda

SAŽETAK

U ovom radu napravljen je python modul za proračun statičkog stabiliteta broda u neoštećenom stanju triangulizirane brodske forme u okviru programa otvorenog koda d3v-gsd. To je proširenje programa linaetal-gsb/d3v, koji omogućuje trodimenzijsku vizualizaciju brodske forme, izračun hidrostatičkih karakteristika te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. Izrađeni python modul je primjenjiv za proizvoljno stanje krcanja te za sve brodove zadane trianguliziranom formom. Također izrađen je model za generiranje krivulje poluge statičkog stabiliteta u pythonu za određeno stanje krcanja koja je uspoređena sa krivuljom generiranom programom Orca3D.

Ključne riječi: python modul, triangularizacija forme broda, hidrostatičke karakteristike, stabilitet

SUMMARY

In this paper a python module has been created for the calculation of the static stability of the undamaged ship defined by triangulated form using the open source program d3v-gsd. It is an extension of the linaetal-gsb/d3v program, which has enabled the three-dimensional visualization of the ship's shape, the calculation of hydrostatic characteristics, and the simple defining of the graphical interface for certain specific purposes. The created python module is applicable for an arbitrary loading condition and for all models with a triangulated shape. Additionally, a model has been created to generate the curve of righting arm of a ship in python for a defined loading condition, which has been compared to the curve generated by the Orca3D program.

Key words: python module, triangularization of the ship's form, hydrostatic characteristics, stability

1. UVOD

Triangulizirana forma omogućava precizno modeliranje oblika broda i njegovih hidrostatičkih svojstava, što je važno za osiguravanje stabiliteta broda na moru. U ovom radu prikazat će se kako se za različite forme broda zadovoljavanjem uvjeta plovnosti može odrediti položaj težišta istisnine broda što je ključna točka u daljnjim proračunima stabilitetu broda.

Osnovni cilj rada je izrada Python modula za analizu stabiliteta odnosno poprečnog i uzdužnog stabiliteta broda. Točnost rezultata proračuna stabiliteta dobivenih primjenom izrađenog modula potrebno je usporediti s postojećim prepoznatim softverom

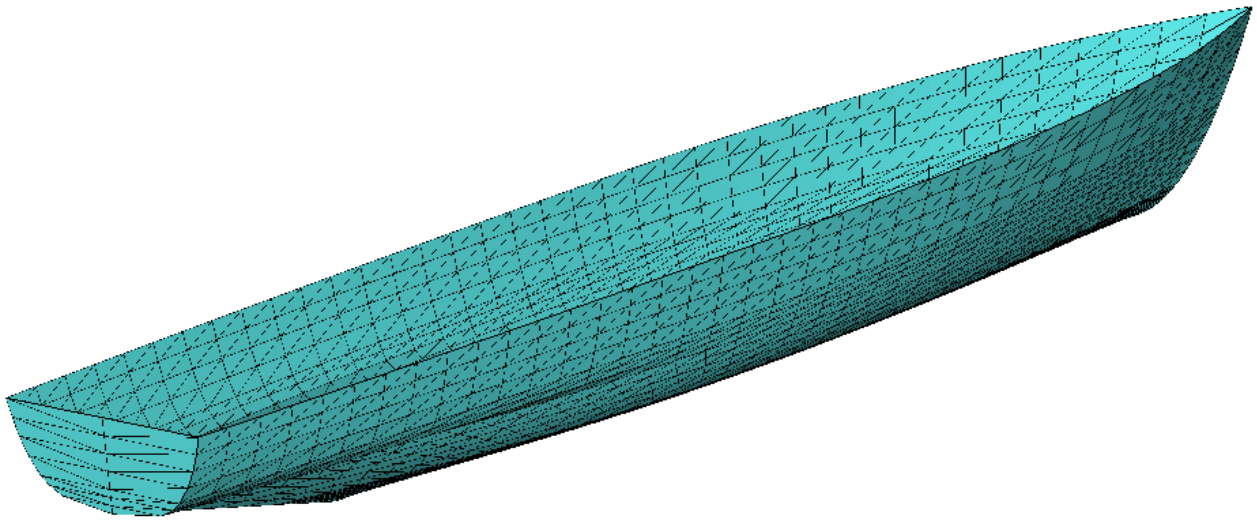
1.1. Osnove o Pythonu

Python je interpreterski, interaktivni, objektno orijentirani programski jezik. Njegova filozofija dizajna naglašava čitljivost koda uz korištenje značajnog uvlačenja. On omogućuje programeruda više razmišlja o problemu koji ima nego o jeziku. On je nešto između tradicionalnih skriptnihjezika (kao što su Tcl, Schema i Perl) i sistemskih jezika (kao što su C, C++ i Java). To znači da nudi jednostavnost i lako korištenje skriptnih jezika, uz napredne programerske alate koji setipično nalaze u sistemskim razvojnim jezicima. Python je besplatan (za akademske ustanove ineprofitnu upotrebu), open-source softver. Osim standardnih tipova podataka (brojevi, nizovi znakova i sl.) python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici. Python nudi sve značajke očekivane u modernom programskom jeziku: objektno orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka, redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena i pakete [1].

1.2. Vizualizacija triangularizirane forme primjenom Linaetal-fsb/d3v programa

Linaetal-fsb d3v (*design visualizer*) je modularna Python aplikacija otvorenog programskog koda, prvenstveno namijenjena za 3D vizualizaciju inženjerskih modela u fazi projektiranja. Inicijalno je nastala kao rezultat dugogodišnje suradnje Fakulteta strojarstva i brodogradnja s USCS.d.o.o te obrtom Linaetal. Struktura programa temeljena je na dvanaestogodišnjem iskustvu razvoja programa ShipExplorer. Program je u ranoj fazi razvoja no već je moguća vizualizacija s ograničenim brojem funkcionalnosti. Implementacije je bazirana na bibliotekama QtForPython (PySide2) i OpenMesh.

Primjenom zapisa brodske forme putem mreže trokuta i programa otvorenog koda d3v realiziran je program d3v-gsd (D3sign Visualizer for General Ship Design) koji omogućuje modificiranje i vizualizaciju forme te izračune hidrostatičkih karakteristika. Program omogućuje jednostavno analitičko zadavanje forme broda, primjenjivo se u ranim fazama projektiranja broda jer omogućuje jednostavno modificiranje forme broda putem promjene manjeg broja parametara. Osim toga tako zadana brodska forma vrlo je pogodna za nastavne potrebe jer omogućuje njenu jednostavnu i intuitivnu promjenu. Osim analitički zadane forme broda, program omogućuje učitavanje brodske forme zadane putem *obj* ili *stl* datoteka [2].



Slika 1. Prikaz triangulizirane forme u d3v-gsd programu

2. OSNOVE O STABILITETU

Stabilitet broda jest svojstvo broda koje mu omogućava da se odupire silama koje ga nastoje pomaknuti iz početnog položaja te da se nakon prestanka djelovanja tih sila vrati u početni položaj. Ona se može podijeliti prema različitim kriterijima i to s obzirom na:

- stanje broda
- vrste sila koje djeluju na brod
- osi oko kojih se brod nagiba

Kako se brod može nalaziti u različitim stanjima, stabilitet pri njima ima određene posebnosti, tako da se stabilitet broda može podijeliti prema stanju u kojem se brod nalazi na:

- stabilitet broda u neoštećenom stanju
- stabilitet broda u oštećenom stanju

Najčešće stanje broda je neoštećeno stanje, pa se stoga najveći dio proračuna stabiliteta broda odnosi na stabilitet broda u neoštećenom stanju. Stabilitet broda u oštećenom stanju najčešće podrazumijeva oštećenje oplate broda i prodor vode u brodski prostor, pri čemu dolazi do povećanja deplasmana broda, stvaranja mogućih slobodnih površina, gubitka uzgona i sl.

Sile koje djeluju na brod i uzrokuju njegovo nagibanje mogu djelovati statički i dinamički. Vrijednost sila koje djeluju statički ne mijenja se, a suprotno tome, vrijednost sila koje djeluju dinamički promjenjiva je u vremenu. Na temelju toga, stabilitet se može

podijeliti prema vrstama sila koje djeluju na brod na:

- statički stabilitet
- dinamički stabilitet

Podjela stabiliteta prema osima oko kojih se brod nagiba je najčešće prva podjela koja se spominje u stručnoj literaturi, a dijeli stabilitet na:

- poprečni stabilitet
- uzdužni stabilitet

2.1. Poprečni stabilitet

Poprečni stabilitet podrazumijeva nagibanje broda oko uzdužne osi (ili kraće uzdužnice) koja dijeli brod na dva simetrična dijela: lijevi i desni bok. Sheme koje služe za pomoć pri razumijevanju poprečnog stabiliteta crtaju se na poprečnom presjeku broda. Za kontrolu stabiliteta broda potrebno je imati konkretan pokazatelj pomoću kojeg se može utvrditi zadovoljava li brod kriterij stabiliteta ili ne. Taj pokazatelj je kod poprečnog stabiliteta početna poprečna metacentarska visina, a ona predstavlja razliku visine početnog poprečnog metacentra iznad kobilice broda i visine sustavnog težišta broda iznad kobilice. Međutim, pri nagibima broda većim od približno 12°, točka metacentra počinje mijenjati svoj položaj i izlazi iz vertikalne simetrale broda, pa stoga početna poprečna metacentarska visina više nije dovoljno dobar pokazatelj stabiliteta broda. Odatle proizlazi sljedeća podjela, a to je podjela poprečnog stabiliteta na:

- početni poprečni stabilitet (kutovi nagiba do 12°)
- poprečni stabilitet pri većim kutovima nagiba (kutovi nagiba veći od 12°)

Pokazatelj stabiliteta pri većim kutovima nagiba jest poluga poprečnog stabiliteta. Budući da je ona različita za svaki kut nagiba, izrađuje se krivulja poluga poprečnog stabiliteta u kojoj je sadržana i početna poprečna metacentarska visina na temelju koje se promatra cjelokupni stabilitet, pri svim kutovima nagiba. Važno je imati na umu da za promatranje stabiliteta broda treba sagledati ukupnu stabilitet broda, tj. i početni poprečni stabilitet i stabilitet pri većim kutovima nagiba.

2.2. Uzdužni stabilitet

Uzdužni stabilitet odnosi se na nagibanje broda oko poprečne osi. Os koja dijeli brod na pramčani i krmeni dio jest glavno rebro, no nagibanje broda u uzdužnom stabilitetu ne događa se oko njega. Razlog tomu je nesimetričnost broda u uzdužnom smislu.

Svakom promjenom rasporeda masa na brodu dolazi do promjene oblika vodene linije, pa se teiste plovne vodene linije rijetko nalazi na glavnom rebru, već mijenja svoj položaj oko njega. Promatrajući uzdužni stabilitet, brod se nagiba oko težišta plovne vodene linije.

Pokazateljem uzdužnog stabiliteta smatra se trim broda koji predstavlja razliku gaza na pramcu i gaza na krmi.

Ako se govori o podjeli stabiliteta prema osima oko kojih se brod nagiba, preostala os je vertikalna, no zakretanje broda oko vertikalne osi ne promatra se u sklopu stabiliteta, iz razloga što je ono zapravo promjena kursa broda i kao takvo spada u upravljivost.

Posebni slučajevi stabiliteta obuhvaćaju:

- stabiliteta nasukanog broda
- stabiliteta broda pri dokovanju
- stabiliteta prevrnutog broda

no oni nisu tema ovog rada**Error! Reference source not found.**.

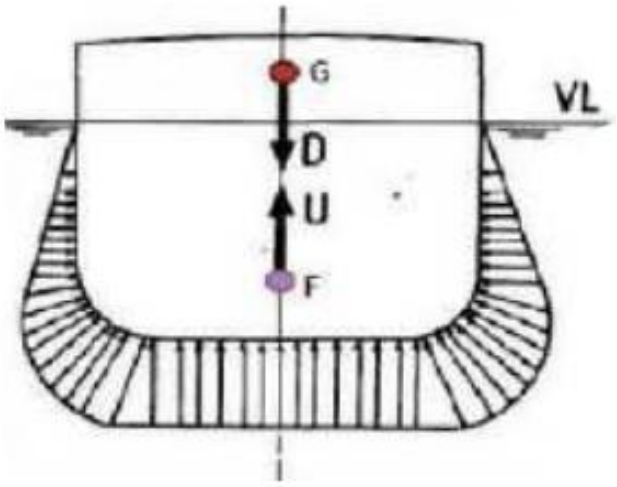
2.3. Uvjeti plovnosti

Arhimedov zakon kaže: Na svako tijelo uronjeno u tekućinu djeluje sila uzgona jednaka težini tekućine istisnute tim tijelom. Plovnost je svojstvo tijela da mirno pluta na tekućini (bez dodira s dnom ili drugim tijelom). Na plovnost utječu karakteristike tijela koje pluta kao i tekućine u kojoj tijelo pluta. Pri tome je tijelo opisano svojim oblikom, masom i težištem, dok je tekućina karakterizirana svojom gustoćom. Primjenom Arhimedova zakona na plovne objekte mogu se formulirati tri uvjeta (zakona) plovnosti:

I. Sila uzgona mora biti jednaka sili težine
II. Sile težine i sile uzgona moraju biti na istom pravcu koji je okomit na teretnu vodenu liniju
III. Potrebno je da brod posjeduju stabilnu ravnotežu

2.3.1. Prvi zakon plovnosti

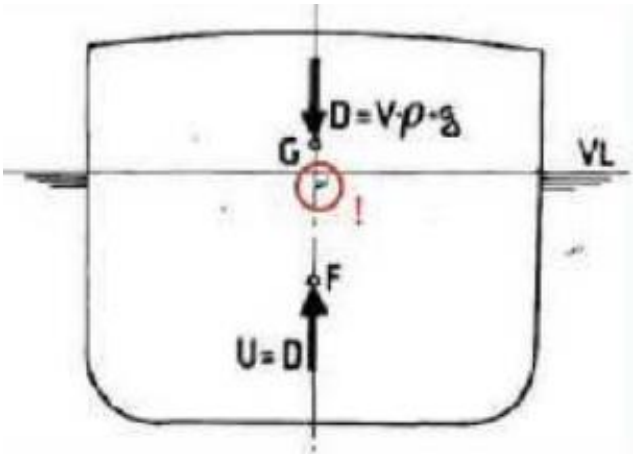
Ako se brod iz bilo kojeg razloga nagne, pri naginjanju javit će se uspravljajući „spreg“ sila koja će vratiti brodu uspravan položaj čim prestane uzrok nagiba. Kada se brod nagne težište istisnine „B“ pomakne se na stranu nagiba jer se promjeni oblik uronjenog dijela broda. Težina broda „G“ i sile uzgona „B“ sastavljaju uspravljeni par sila koje nastoje vratiti brod u uspravan položaj. Na svako tijelo uronjeno u tekućinu djeluje sila uzgona koja odgovara težini istisnute tekućine, što znači da umnožak volumena podvodnog dijela broda i gustoće vode u kojoj brod plovi, mora biti jednak ukupnoj težini broda.



Slika 2. Prvi uvjet plovnosti [4]

2.3.2. Drugi uvjet plovnosti

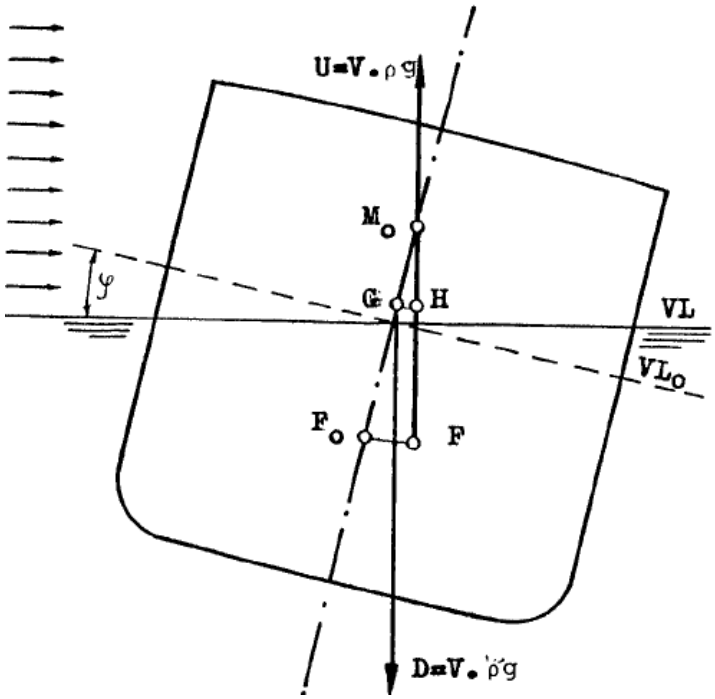
Sila uzgona kao rezultanta svih tlakova koji djeluju na podvodni dio trupa, prolazi težištem istisnute tekućine (F). Težište istisnine F i težište masa G nalaze se na istoj okomici na plovnu vodenu liniju



Slika 3. Drugi uvjet plovnosti [4]

2.3.3. Treći uvjet plovnosti

Brod mora ploviti u stabilnom položaju, tj. Ako se zbog djelovanja neke vanjske sile (vjetra, valova i sl.) brod nagne za neki mali kut, brod se mora vratiti u prvobitni položaj nakon prestanka djelovanja sile koja je izazvala nagib. Metacentar M_0 mora ležati iznad težišta sustava G, odnosno metacentarska visina mora biti pozitivna [4] [5].



Slika 4. Treći uvjet plovnosti [5]

3. HIDROSTATIČKE KARAKTERISTIKE TRIANGULIZIRANE FORME

Triangularizacija plohe podrazumijeva generiranje mreže trokuta koja ju potpuno ili djelomično prekriva. Trokuti se razlikuju oblikom ovisno o njihovom položaju na zakrivljenoj površini kako bi se ona što bolje opisala.

Analitički zadana forma koja se koristi u programu d3v-gsd omogućuje vrlo jednostavnu triangularizaciju, koja se izvodi poprečnim spajanjem točaka između susjednih vodnih linija čime se dobivaju trokuti. Na svakoj vodnoj liniji je jednak broj točaka. Spajanjem točaka dobivaju se trokuti koji naizmjenično imaju po dvije točke na gornjoj vodnoj liniji ili dvije točke na donjoj vodnoj liniji.

Prilikom određivanja stabiliteta broda, jedan od prvih zadataka s kojim se susrećemo je određivanje proračunskog dijela triangulizirane forme za određenu vodnu liniju. Vodna linija zadana je ravninom koju definira jedna točka na toj ravnini te vektor normale što omogućuje zadavanje nagnutih vodnih linija koje su neophodne za proračun stabiliteta. Tako definirana vodna linija dijeli formu na dva dijela. Za proračun stabiliteta relevantan je samo dio mreže ispod vodne linije. U općem slučaju vodna linija presjeca trokute kojima je jedna točka s jedne strane ravnine vodne linije, a druge dvije točke s druge strane vodne linije. Posljedica toga je generiranje novih trokuta neposredno ispod vodne linije koji nastaju na pozicijama tih trokuta [6].

3.1. Definiranje koordinatnog sustava

Prije svega potrebno je definirati koordinatni sustav u prostoru radi boljeg snalaženja te mogućnosti definiranja sljedećih parametara u Python modelu. Koordinatni sustav postavljamo na prednji kraj broda tako da je ishodište smješteno na sjecištu osnovice broda i pramčanog perpendikulara na samom vrhu pramca (krajnja točka broda). Osi su orijentirane gledaju sa pramca na sljedeći način:

- pozitivan smjer x osi u je usmjeren prema krmi
- pozitivan smjer y osi u je usmjeren prema lijevom boku
- pozitivan smjer z osi u je usmjeren prema gore

3.2. Određivanje mreže ispod vodne linije

Cilj ove funkcije je izraditi mrežu trupa ispod vodne linije koja služi za proračun stabiliteta. Za razliku od metode prikazane u koja omogućuje samo zadavanje horizontalne vodne linije, metoda korištena u ovom radu omogućuje zadavanje proizvoljno nagnute vodne linije. Najprije se izračunava ravnina vodne linije koja se definira pomoću točke na ravnini i njene normale. Da bi se utvrdilo je li točka ispod ili iznad ravnine, možemo koristiti jednadžbu ravnine:

$$\mathrm{Ax} + \mathrm{By} + \mathrm{Cz} + \mathrm{D} = 0 \quad \textbf{(1)}$$

gdje su *A, B* i *C* koordinate vektora normale **n**, a *D* je konstanta koju možemo izračunati kao:

$$D = -\mathbf{n} \cdot \mathbf{r}_0 \quad \textbf{(2)}$$

gdje je:

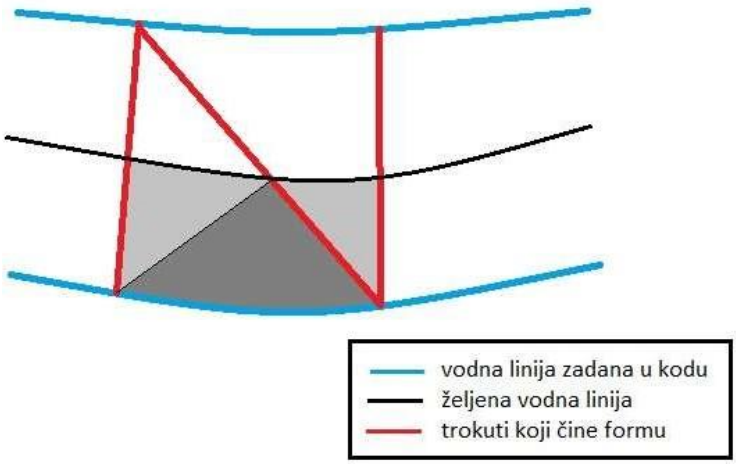
- **n** -vektor normale
- $\mathbf{r}_0$ - pozicija točke na ravnini

Zatim, kako bi provjerili u kojoj se polovici prostora nalazi točka, jednostavno je potrebno izračunati udaljenost te točke od ravnine. Ako je točka ispod ravnine, udaljenost će biti negativna, a ako je iznad ravnine, udaljenost će biti pozitivna.

Trokuti koji su u cijelosti ispod ove ravnine se dodaju u listu trokuta koji se nalaze ispod vodne linije jer ne trebaju biti mijenjani. Trokuti koji su u cijelosti iznad ove ravnine se također ignoriraju jer nisu relevantne za proračun stabiliteta.

3.2.1. Određivanje nove mreže trokuta forme na sjecištu s vodnom linijom

Za svaki trokut koji presijeca vodnu liniju se izračunavaju dvije nove točke koje označavaju točke presjeka te se dodaju na popis točaka. Zatim se dodaju novi trokuti koji će zamijeniti presječeni trokut. Algoritam za određivanje mreže za proračun stabiliteta provjerava moguće slučajeve, pa se za slučaj kad se sva tri vrha trokuta nalaze ispod vodne linije trokut dodaje u mrežu, ako su dva vrha trokuta iznad vodne linije, generira se novi trokut kojem su dvije nove točke na vodnoj liniji, a ako je jedan vrh trokuta iznad vodne linije ispod vodne ostaje trapez kojeg dijelimo na dva trokuta koji se dodaju u mrežu za proračun stabiliteta [6].

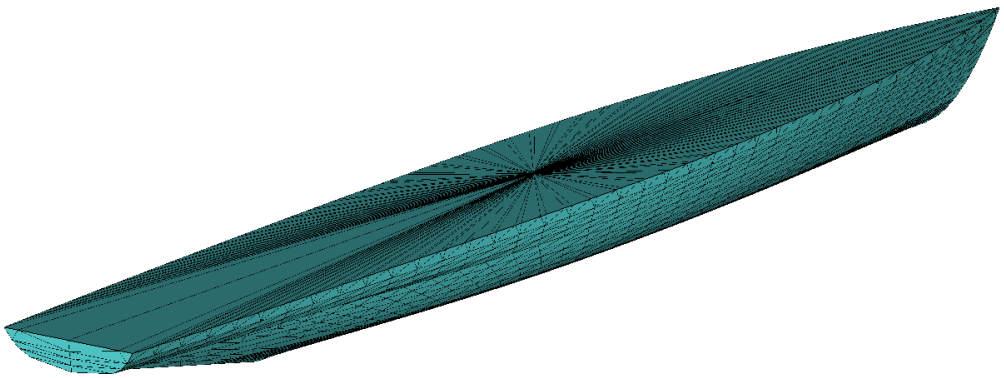


Slika 5. Prikaz presijecanja trokuta i vodne linije. Sivom bojom su označene površine dijelovatrokuta čije projekcije na vodnu liniju pridodajemo površini [6]

3.2.2. Određivanje mreže pokrova

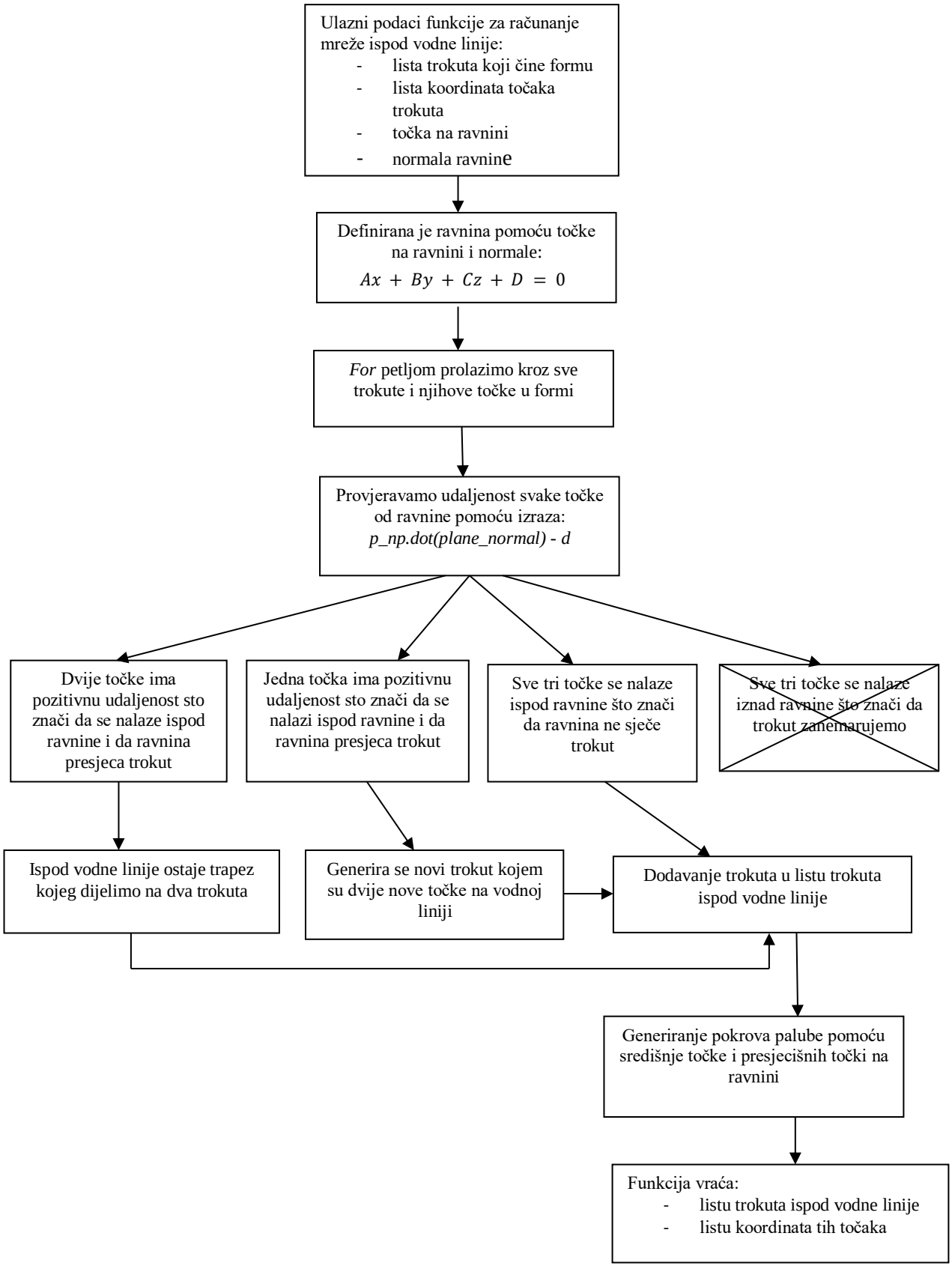
Mreža pokrova nalazi se u ravnini vodne linije, a omeđena je s presjecišnim točkama vodne linije i brodske forme, koje su određene u prethodnom koraku. Ovaj dio mreže neophodan je kako bi se proračunska mreža zatvorila odnosno kako bi definirala zatvoreni volumen.

Mreža pokrova nastaje generiranjem niza trokuta kojima su dvije točke susjedne dvije točke na presjecištu brodske forme i vodne linije, dok je treća točka svima zajednička, a radi se o dodatnoj točki koja se nalazi u ravnini vodne linije na poziciji glavnog rebra. Funkcija u kojoj je ovaj algoritam implementiran vraća listu trokuta ispod vodne linije i na njoj te novi popis koordinata tih točaka koji definiraju mrežu trupa.



Slika 6. Prikaz mreže pokrova broda

Dijagram toka za određivanje mreže ispod vodne linije:



3.3. Određivanje volumena i težišta tetraedra

Za proračun stabiliteta broda nužno je odrediti volumen istisnine te njezino težište za opće postavljenju ravninu vodne linije. Metoda za proračun volumena temelji se na izračunu volumena niza tetraedra čija baza su trokuti na mreži forme i pokrova, a vrh u ishodištu koordinatnog sustava (0,0,0). Generiraju se vektori $\mathbf{p}_1$, $\mathbf{p}_2$ i $\mathbf{p}_3$ koji predstavljaju koordinate vrhova tetraedra. Zatim slijedi računanje volumena svakog tetraedra u paralelepipedu koji proizlazi iz mješovitog produkta 2 vektora na trokutu i ishodišta kao visine. Tih šest članova se koristi u formuli za izračunavanje volumena tetraedra. S obzirom na različit poredak točaka u trokutu i normale na trokutima će biti drugačije orijentirane. Upravo ta orijentacija će odrediti predznak volumena tetraedra. Zbrajanjem volumena svih šest tetraedara te njihovim dijeljenjem s 6 dobivamo traženi volumen. Također, funkcija računa težište tetraedra koristeći jednostavnu formulu za sredinu četiri točke u prostoru.

$\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ = koordinate točaka u trokutu

$$
\begin{aligned}
V_{321} &= \mathrm{p}_3[0] * \mathrm{p}_2[1] * \mathrm{p}_1[2] \\
V_{231} &= \mathrm{p}_2[0] * \mathrm{p}_3[1] * \mathrm{p}_1[2] \\
V_{312} &= \mathrm{p}_3[0] * \mathrm{p}_1[1] * \mathrm{p}_2[2] \\
V_{132} &= \mathrm{p}_1[0] * \mathrm{p}_3[1] * \mathrm{p}_2[2] \\
V_{321} &= \mathrm{p}_2[0] * \mathrm{p}_1[1] * \mathrm{p}_3[2] \\
V_{123} &= \mathrm{p}_1[0] * \mathrm{p}_2[1] * \mathrm{p}_3[2]
\end{aligned}
\qquad \textbf{(4)}
$$

$$
\Delta_t = \frac{1}{6}\left(-v_{321} + v_{231} + v_{312} - v_{132} - v_{213} + v_{123}\right) \qquad \textbf{(5)}
$$

$$
T = \frac{(\boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3)}{4} \qquad \textbf{(6)}
$$

3.4. Određivanje istisnine i težišta istisnine

Ukupna istisnina računa se kao zbroj volumena svih tetraedara, a težište istisnine se računa tako da se dobivena težišta tetraedara pomnože volumenima i sumiraju, a ta suma podijeli sa ukupnom istisninom. Za položaj tog težišta pretpostavljamo da će se nalaziti blago iza glavnog rebra jer je brodska forma punija na krmi nego na pramcu.

Δ_t – volumen pojedinog tetraedra, m^3

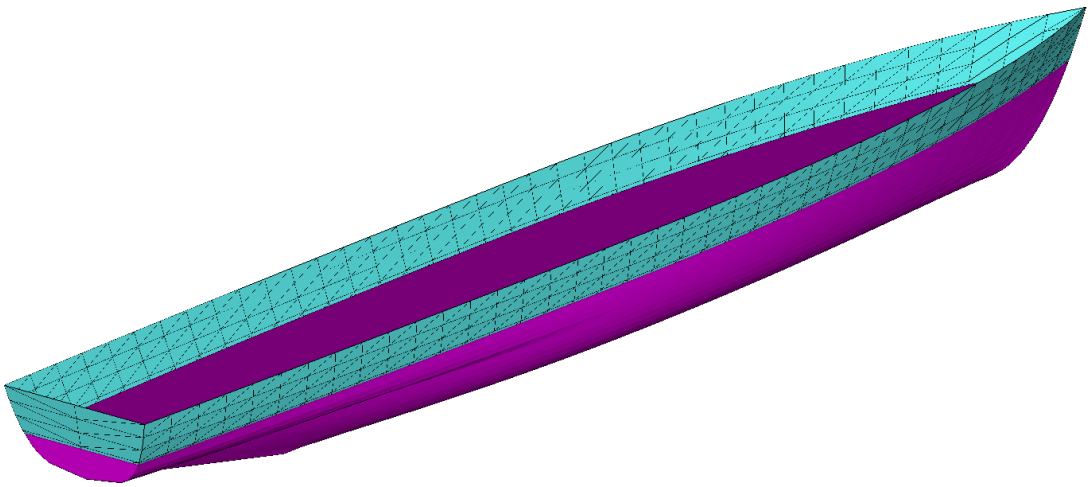
T – položaj težišta pojedinog tetraedra (x1,y1,z1)

Δ – istisnina broda, m^3

CB – položaj težišta istisnine broda (x, y, z)

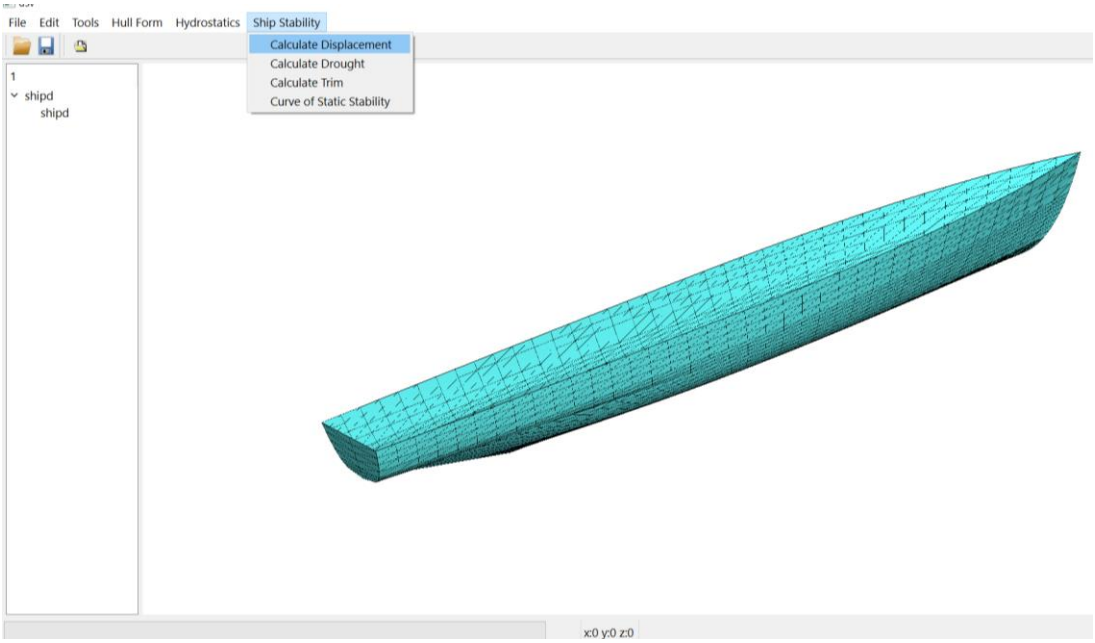
n – broj tetraedara u formi broda

$$CB = \frac{\sum_{i=1}^{n} (\Delta_{t_i} \cdot T_i)}{\Delta} \tag{7}$$

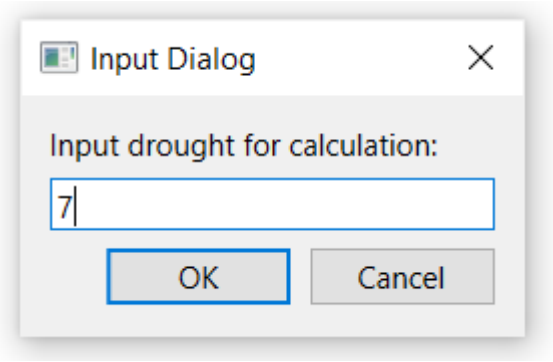


Slika 7. Prikaz istisnine u vizualizatoru

Zadavanjem gaza u prozoru prikazanom na slici program vizualizira istisninu te vrača vrijednosti istisnine i njenog težišta.



Slika 8. Prikaz izračuna istisnine i njenog težišta u programu



Slika 9. Prozor za zadavanje gaza za čiju istisninu želimo dobiti

Za gaz od 7m program ispisuje:

- displacement : 5316.482162118499
- displacement CG : [4.69777365e+01 5.70013154e-16 4.23294715e+00]

4. ODREĐIVANJE VODNE LINIJE ZA ODREĐENA STANJA KRCANJA

4.1. Za ravnu vodnu liniju

U Python modulu stability.py, metoda *determine_horizontal_waterline_for_current_weight* je metoda čiji je cilj određivanje visine ravne vodne linije, odnosno gaza za određenu istisninu. Funkcija kao ulazne podatke uzima težinu broda te težište broda koji su rezultat nekog stanja krcanja. . Prvo se određuje minimalna i maksimalna visina broda kako bi se odredile granice koje funkcija može poprimiti. Zatim se koristi funkcija *scipy.optimize.root()* za traženje nultočke funkcije *fun_opt_root_drought_calculation* da bi dobili gaz za koji je razlika između ukupne mase broda i istisnine nula. Ovim postupkom se zadovoljava prvi uvjet plovnosti, istisnina broda mora bit jednaka njegovoj težini. Gaz koji funkcija pronalazi i vraća je gaza broda za određeno stanje krcanja.

U navedenoj metodi se koriste dvije dodatne funkcije.

1. Funkcija *calculate_displacement_horizontal_waterline* izračunava istisninu broda na ravnoj vodnoj liniji, pri čemu se gaz nalazi na visini T. Funkcija uzima visinu vodne linije T kao ulazni argument, stvara ravninu s tim parametrom i računa istisninu.
2. Funkcija *fun_opt_root_drought_calculation* uzima gaz kao ulazni argument te računa razliku između ukupne mase broda i istisnine broda koji je izračunan prethodno navedenom metodom. Pronalaženjem nultočke ove funkcije dobivamo traženi gaz.

Ulazne podatci težine i težišta broda:

- težina broda: 3160 t i položaj težišta broda: (50, 0, 5.5)

Program ispisuje:

```
message: The solution converged.
success: True
 status: 1
    fun: 0.0
      x: [ 5.334e+00]
   nfev: 7
   fjac: [[-1.000e+00]]
      r: [ 1.032e+03]
    qtf: [ 5.158e-09]
```

Slika 10. Ispis gaza za zadanu težinu broda

4.2. Određivanje uzdužnog trima broda

Ova metoda ima zadatak računanja vodne linije za uzdužne nagibe što znači da se samo x koordinata normale ravnine mijenja ([pl_nv_x, 0.0, -1]). Kao i u slučaju za ravnu vodnu liniju ulazni podatci su isti, težina i težište broda čime je određeno jedinstveno stanje plovidbe broda. Obzirom da se radi o nagnutoj vodnoj liniji, mora se zadovoljiti i drugi uvjet plovnosti, sile težine i sile uzgona moraju biti na istom pravcu koji je okomit na teretnu vodenu liniju. Ova metoda također koristi *SciPy* funkciju *optimize.root() za* numeričko rješavanje jednadžbe koja se temelji na razlici između izračunate istisnine i ukupne mase broda te između koordinata težišta broda i težišta istisnine dobivene projekcijom težišta na vodnu liniju. Nultočka te funkcije vraća ravninu (u obliku točke koja lezi na ravnini i njezine normale) te trim u stupnjevima na kojoj će brod ploviti za određene ulazne parametre.

Umjesto korištenja SciPy funkcije ovaj problem mogao se riješiti i implementacijom metode bisekcije. To je numerička metoda za pronalaženje nultočke funkcije jedne varijable. Osnovna ideja bisekcije je da se interval u kojem se nalazi nultočka funkcije podijeli na dva podintervala jednake duljine. Zatim se provjeri u kojem podintervalu se nalazi nultočka funkcije, te se postupak ponavlja na tom podintervalu. Ovaj postupak se ponavlja sve dok se ne pronađe aproksimacija nultočke funkcije s dovoljnom točnošću.

Prilikom pokretanja proračuna odabirom u izborniku:

Ship stability → Calculate trim

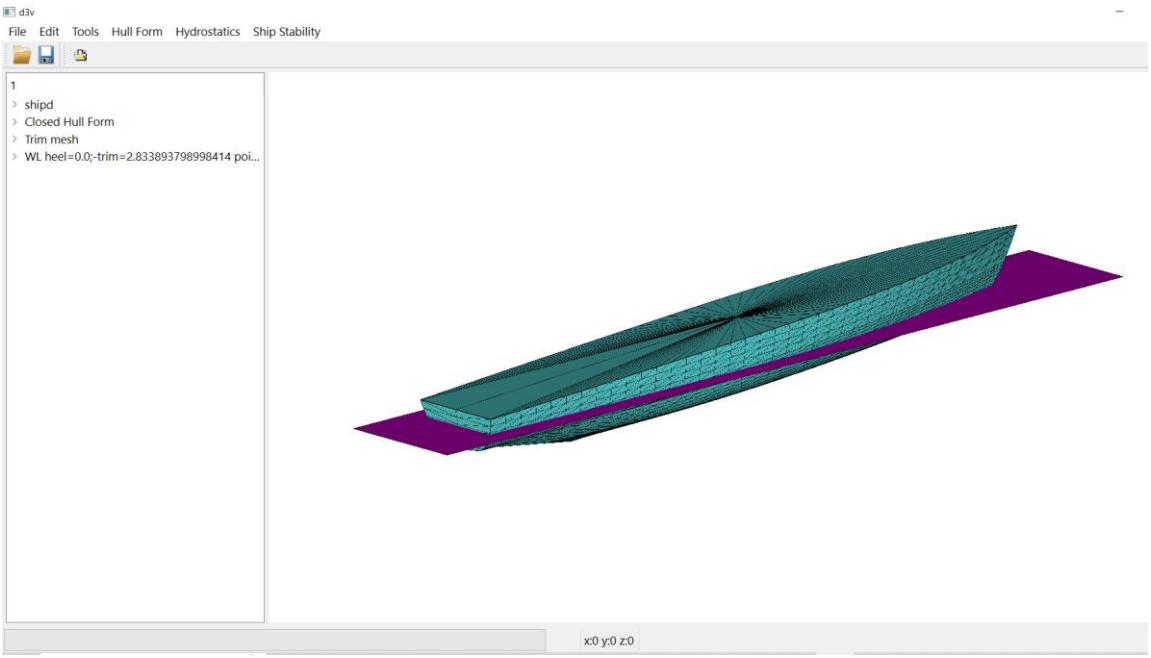
dobivamo sljedeći ispis za određene ulazne parametre težine i težišta broda:

sscalc.set_ship_G_and_CG(3160.0, 50.0, 0.0, 5.5)

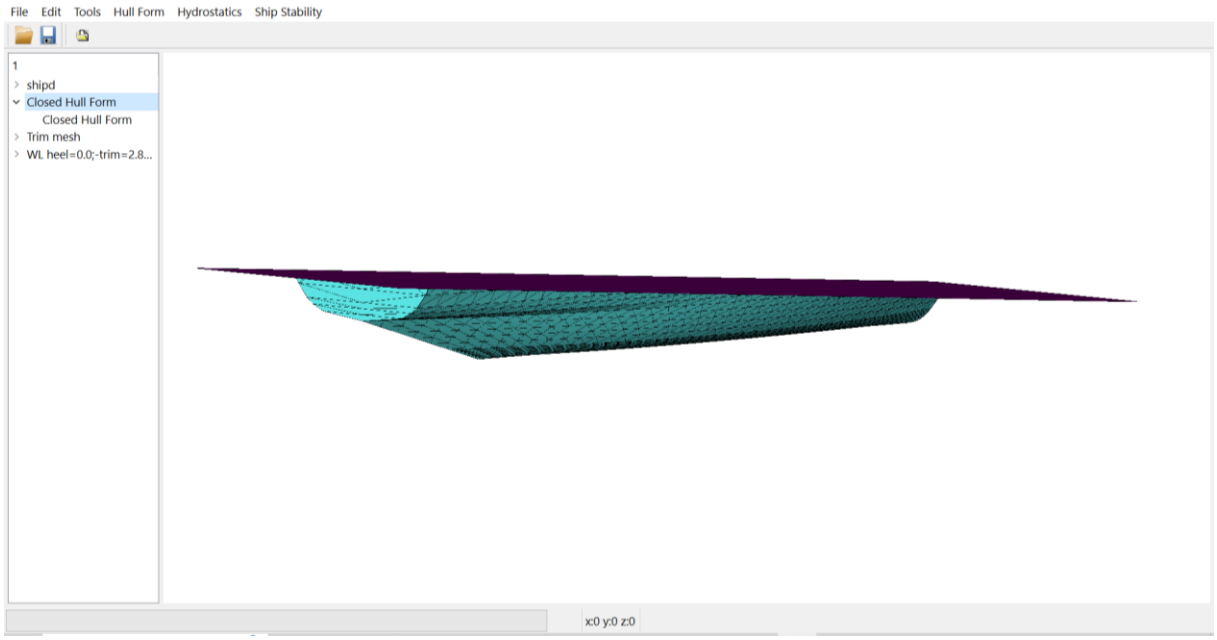
- težina broda: 3160 t
- položaj težišta broda: (50, 0, 5.5) m

Program ispisuje:

- *trim, deg = 2.833893798998414*
- *WL heel=0.0;-trim=0.23264672901494962 point [50. 0. 4.94375534], normal [0.00406047 0. -1.]*



Slika 11. Prikaz uzdužnog nagiba broda u vizualizatoru



Slika 12. Prikaz istisnine broda u vizualizatoru

5. ODREĐIVANJE POLUGE STATIČKOG STABILITETA

Poluga statičkog stabiliteta je udaljenost između dviju sila: sile težine i sile uzgona koje djeluju na brod. Ova udaljenost je izmjerena okomito na smjer vektora sila. Poluga statičkog stabiliteta se koristi za izračunavanje stabiliteta broda i njezina veličina ukazuje koliko će brod biti stabilan kada se nagne. Što je veća poluga statičkog stabilizatora, to će se brod brže i lakše stabilizirati nakon nagnuća.

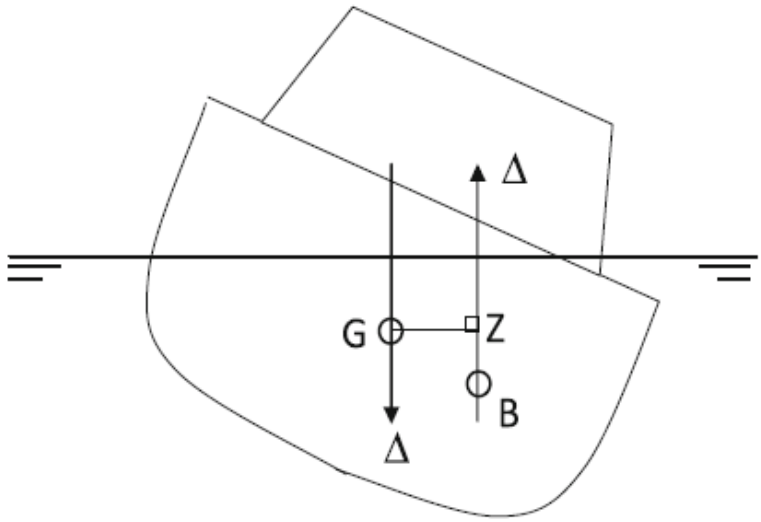
Fizička veličina koja najbolje karakterizira stabilitet broda na mirnoj vodi jest moment statičkog stabiliteta. Njegova veličina ovisi o kutu nagiba broda φ pa se on prikazuje u dijagramu kao funkcija kutova nagiba, obično u području od 0 do 90°. Krivulja momenta statičkog stabiliteta je deformirana sinusoida, jer član NG nije konstantan nego se mijenja s kutom nagiba φ. Kako je deplasman broda Δ neovisan o kutovima nagiba φ, u dijagram se nanose samo promjenljivi članovi jednadžbe za M_{st}, tj. poluge :

$$h = GH = NG\sin(\varphi) \quad \textbf{(8)}$$

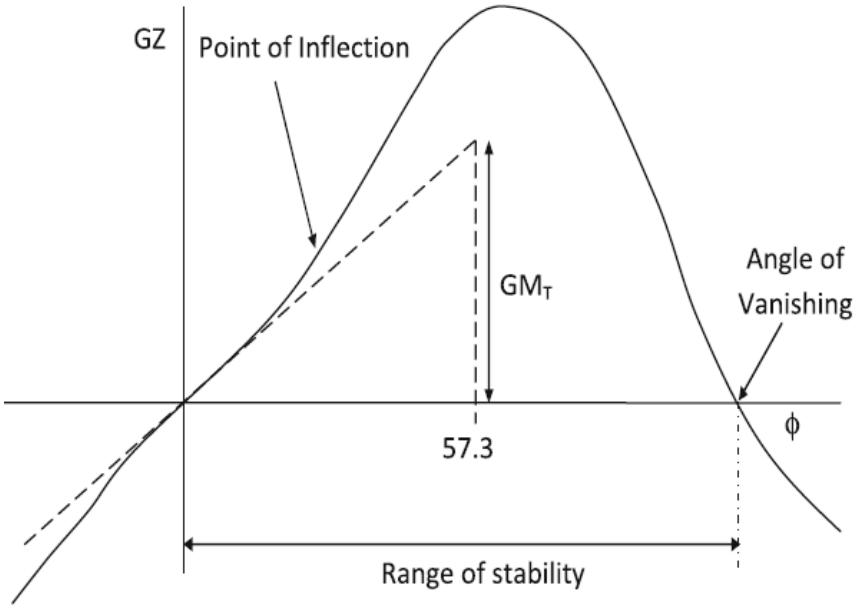
Tako se dobiva krivulja poluga ili Reedov dijagram. Budući da je:

$$Mst = \Delta \cdot h \quad \textbf{(9)}$$

Δ - masa istisnine [t] , konstantan faktor, tok krivulje poluga je isti kao i tok krivulje momenata. One se razlikuju samo u mjerilu, pa se mogu predočiti istom krivuljom sa dvije skale. No stabilitet broda je s fizičkog stanovišta karakteriziran jedino momentom statičkog stabiliteta, jer je jedino taj moment mjerilo sposobnosti broda da se odupire djelovanju prekretnih momenata, dok su poluge statičkog stabiliteta samo zgodno sredstvo za predočivanje i uspoređivanje. [7]



Slika 13. Prikaz poluge statičkog stabiliteta kod nagnutog broda [8]



Slika 14. Slika 3 Krivulja momenta statičkog stabiliteta [8]

5.1. Python modul

Cilj ove metode je dobivanje krivulje ovisnosti poluge statičkog stabiliteta o kutu nagiba broda za proizvoljno stanje krcanja.
U funkciji kojoj su ulazni podaci:

- maksimalni kut nagiba
- korak kuta nagiba

U funkciji je proveden sljedeći postupak:

- stvara se BoundingBox broda kako bi se izračunale njegove dimenzije i pozicija
- nakon što je postavljena pozicija vodne linije, poziva se funkcija koja određuje kut trima za trenutno stanje
- poziva se funkcija koja računa pomak i položaj težišta istisnine za trenutno stanje
- računa se poluga stabiliteta GZ za trenutni kut nagiba kao paralelna udaljenost između vektora uzgona i vektora težine broda
- ako se dogodi greška pri izračunu bilo kojeg podatka za određeni kut nagiba, taj podatak se ne dodaje u listu, već se ispisuje pogreška
- nakon što su za sve kutove nagiba broda izračunate vrijednosti GZ, funkcija vraća niz vrijednosti heel i GZ koja se također vizualizira

5.1.1. Stanje krcanja 1

Prilikom pokretanja proračuna odabirom u izborniku *Ship stability* → *Curve of static stability.*

Za određene ulazne parametre težine i težišta broda:

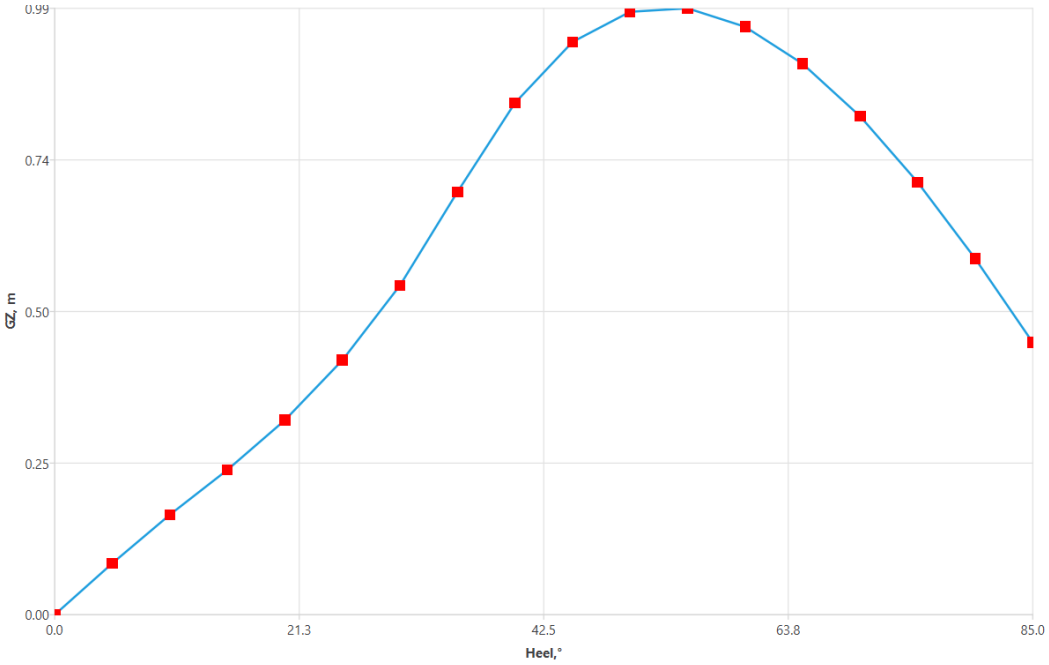
sscalc.set_ship_G_and_CG(3160.0, 50.0, 0.0, 5.5)

- težina broda: 3160 t
- položaj težišta broda: (50, 0, 5.5)

Dobivamo sljedeći ispis tablice i krivulje statičkog stabiliteta prikazane u nastavku.

	Heel,°	GZ, m	Trim,°	)isplacement,	Drought, m	Calc Heel,°
1	0.0	0.000	0.233	3160.0	5.106	0.000
2	5.0	0.084	0.243	3160.0	5.085	5.000
3	10.0	0.163	0.279	3160.0	5.054	10.000
4	15.0	0.237	0.350	3160.0	5.104	15.000
5	20.0	0.318	0.441	3160.0	5.147	20.000
6	25.0	0.416	0.539	3160.0	5.173	25.000
7	30.0	0.538	0.643	3160.0	5.175	30.000
8	35.0	0.691	0.748	3160.0	5.144	35.000
9	40.0	0.837	0.867	3160.0	5.092	40.000
10	45.0	0.937	0.991	3160.0	5.056	45.000
11	50.0	0.986	1.103	3160.0	5.021	50.000
12	55.0	0.992	1.203	3160.0	5.003	55.000
13	60.0	0.961	1.292	3160.0	4.993	60.000
14	65.0	0.901	1.373	3160.0	5.008	65.000
15	70.0	0.815	1.449	3160.0	5.064	70.000
16	75.0	0.707	1.522	3160.0	5.196	75.000
17	80.0	0.582	1.594	3160.0	5.512	80.000
18	85.0	0.445	1.662	3160.0	5.964	85.000

Slika 15. Tablica statičkog stabiliteta u Pythonu za stanje krcanja 1



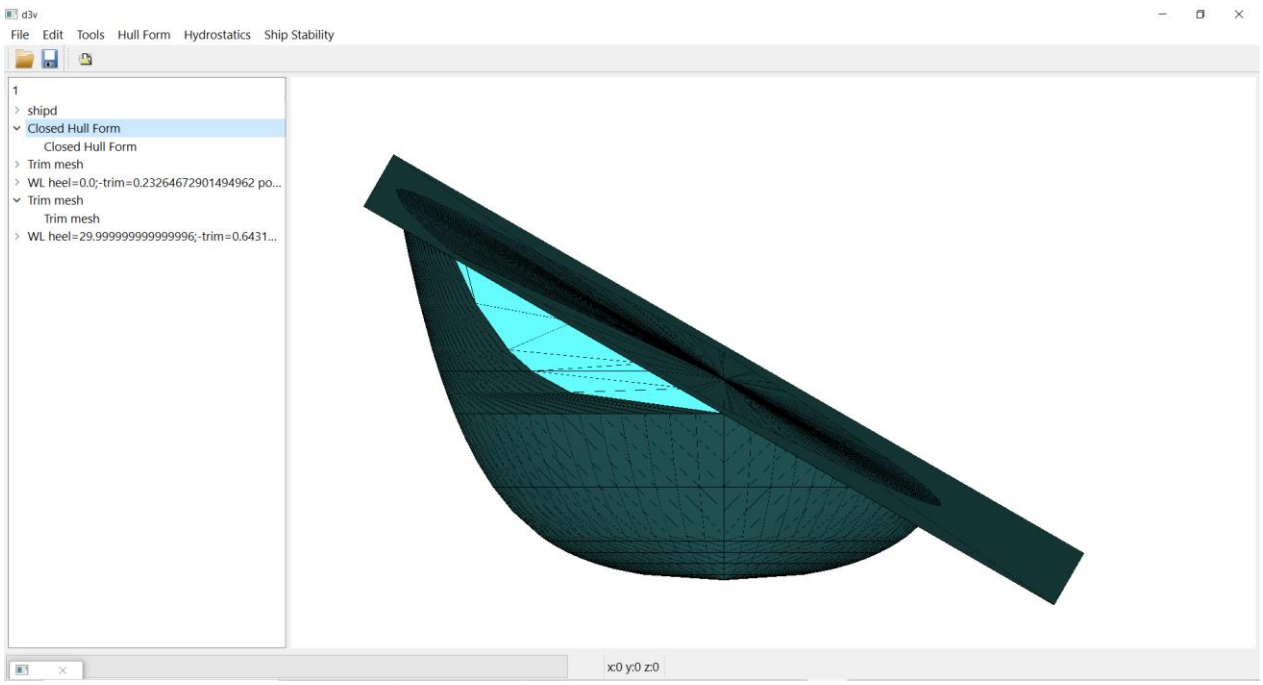
Slika 16. Krivulja poluge statičkog stabiliteta u Pythonu za stanje krcanja 1

Ispisuje se tablica koja opisuje stabilitet broda u određenim uvjetima nagiba i trima. Svaki redak u tablici opisuje stabilitet broda u određenom položaju nagiba.

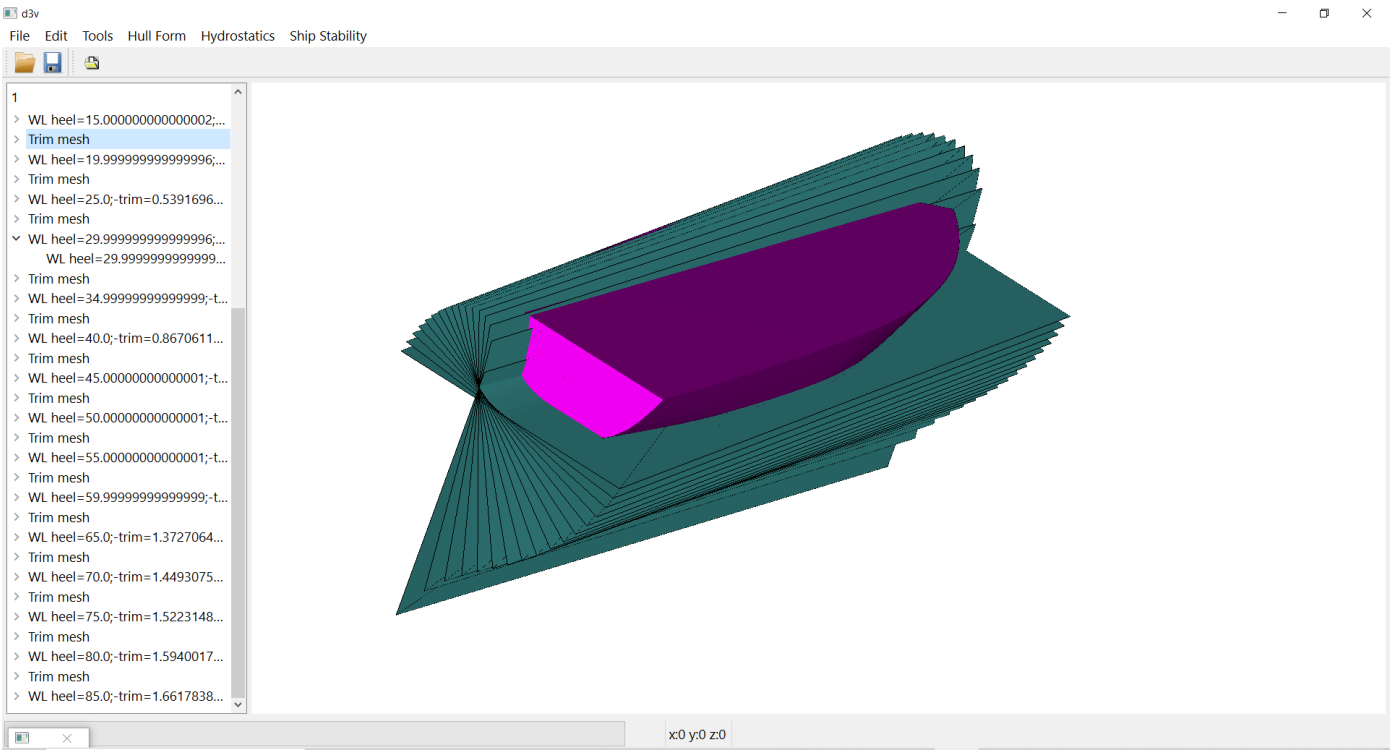
Stupci u tablici predstavljaju sljedeće parametre:

- heel: nagib broda u stupnjevima
- GZ: poluga statičkog stabiliteta u metrima što je mjera stabiliteta broda
- calc heel: vrijednost nagiba koja se koristi za izračunavanje GZ, jer se tijekom mjerenja može dogoditi da brod nije u potpunosti stabilan pa se koristi manji nagib za mjerenje
- trim: kuta uranjanja pramca ili krme broda, odnosno uzdužni kut nagiba horizontalne ravnine koja prolazi kroz težište broda
- T: gaz broda definiran kao udaljenost ravnine i neke točke na formi broda u metrima, što je također važan parametar za stabiliteta broda

Tablica se sastoji od 18 redaka, pri čemu se prva tri retka odnose na male nagibe i manje vrijednosti trima, a ostalih 15 redaka opisuje stabiliteta broda pri većim nagibima i većim vrijednostima trima.



Slika 17. Prikaz istisnine broda pri nagibu od 30° za stanje krcanja 1



Slika 18. Prikaz ravnina vodne linije pri određenim nagibima broda

5.1.2. Stanje krcanja 2

Za određene ulazne parametre težine i težišta broda:

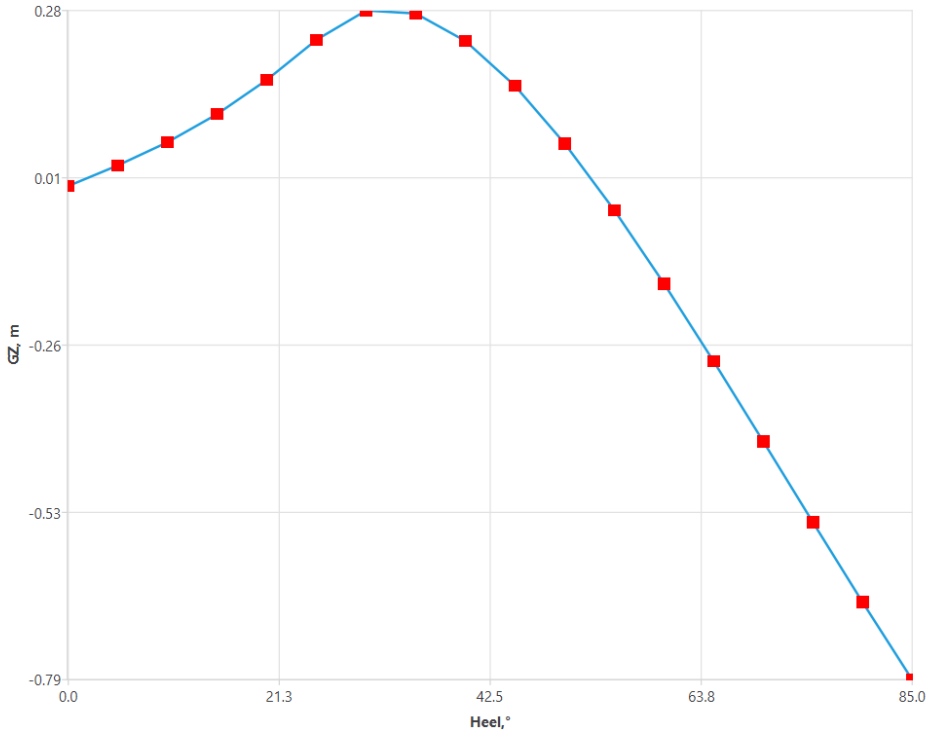
sscalc.set_ship_G_and_CG(4500.0, 48.0, 0.0, 6.5)

- težina broda: 4500 t
- položaj težišta broda: (58, 0, 6.5)

Dobivamo sljedeći ispis tablice i krivulje statičkog stabiliteta prikazane u nastavku.

	Heel,°	GZ, m	Trim,°	)isplacement,	Drought, m	Calc Heel,°
1	0.0	0.000	0.135	4500.0	6.277	0.000
2	5.0	0.033	0.140	4500.0	6.250	5.000
3	10.0	0.071	0.153	4500.0	6.262	10.000
4	15.0	0.116	0.176	4500.0	6.308	15.000
5	20.0	0.171	0.212	4500.0	6.336	20.000
6	25.0	0.235	0.271	4500.0	6.338	25.000
7	30.0	0.282	0.352	4500.0	6.323	30.000
8	35.0	0.278	0.426	4500.0	6.316	35.000
9	40.0	0.233	0.489	4500.0	6.311	40.000
10	45.0	0.161	0.543	4500.0	6.325	45.000
11	50.0	0.068	0.590	4500.0	6.335	50.000
12	55.0	-0.039	0.630	4500.0	6.359	55.000
13	60.0	-0.157	0.664	4500.0	6.387	60.000
14	65.0	-0.282	0.692	4500.0	6.433	65.000
15	70.0	-0.411	0.714	4500.0	6.516	70.000
16	75.0	-0.541	0.732	4500.0	6.667	75.000
17	80.0	-0.670	0.747	4500.0	6.980	80.000
18	85.0	-0.794	0.761	4500.0	7.445	85.000

Slika 19. Tablica statičkog stabiliteta u Pythonu za stanje krcanja 2



Slika 20. Krivulja poluge statičkog stabiliteta u Pythonu za stanje krcanja 2

5.2. Orca3D

5.2.1. Osnove o programu Orca3d

Orca 3D (Rhinov dodatak) je softverski alat za modeliranje brodova i nautičkih konstrukcija u 3D formatu. Program je specijaliziran za modeliranje brodova i nautičkih konstrukcija, dok Rhino nudi širi spektar alata za 3D modeliranje u različitim industrijskim granama. Ovaj softver omogućuje inženjerima, projektantima i dizajnerima brodova da stvore detaljne virtualne modele brodova, što omogućuje detaljnije analize performansi, kao i bolje razumijevanje izgleda i dizajna broda. Orca 3D također pruža alate za optimizaciju performansi broda, što može dovesti do poboljšanja brzine, potrošnje goriva i upravljivosti. Softver nudi alate za proračun i analizu poluge statičkog stabiliteta na brodu za što će prvenstveno biti korišten u ovom radu. Alati za analizu poluge statičkog stabilizatora omogućuju korisnicima da izmijene konfiguraciju broda kako bi se poboljšala njegov stabilitet, a što bi u konačnici moglo dovesti do bolje sigurnosti i učinkovitosti broda.

5.2.2. Stanje krcanja 1

Na početku je bilo potrebno uvesti triangluliziranu formu u program te zadati određeno stanje krcanja kako bi se proračun mogao provesti. U ovom dijelu ćemo analizirati stabilitet broda pri određenim kutovima nagiba pomoću krivulje poluge statičkog stabiliteta. Usporedbu rezultata napravit ćemo za dva stanja krcanja.

Overall Dimensions

Length Overall, LOA	100,000 m	Loa / Boa	6,667
Beam Overall, Boa	15,000 m	Boa / D	1,596
Depth Overall, D	9,400 m		

Slika 21. Geometrijski karakteristike za korištenu formu broda

Condition Summary

Load Condition Parameters

Condition	Weight / Sinkage	LCG / Trim	TCG / Heel	VCG (m)
Condition 1	3160,000 tonne-f	0,000 deg	0,000 m	5,5

Resulting Model Attitude and Hydrostatic Properties

Condition	Sinkage (m)	Trim(deg)	Heel(deg)	Ax(m^2)
Condition 1	4,920	0,000	0,000	49,80

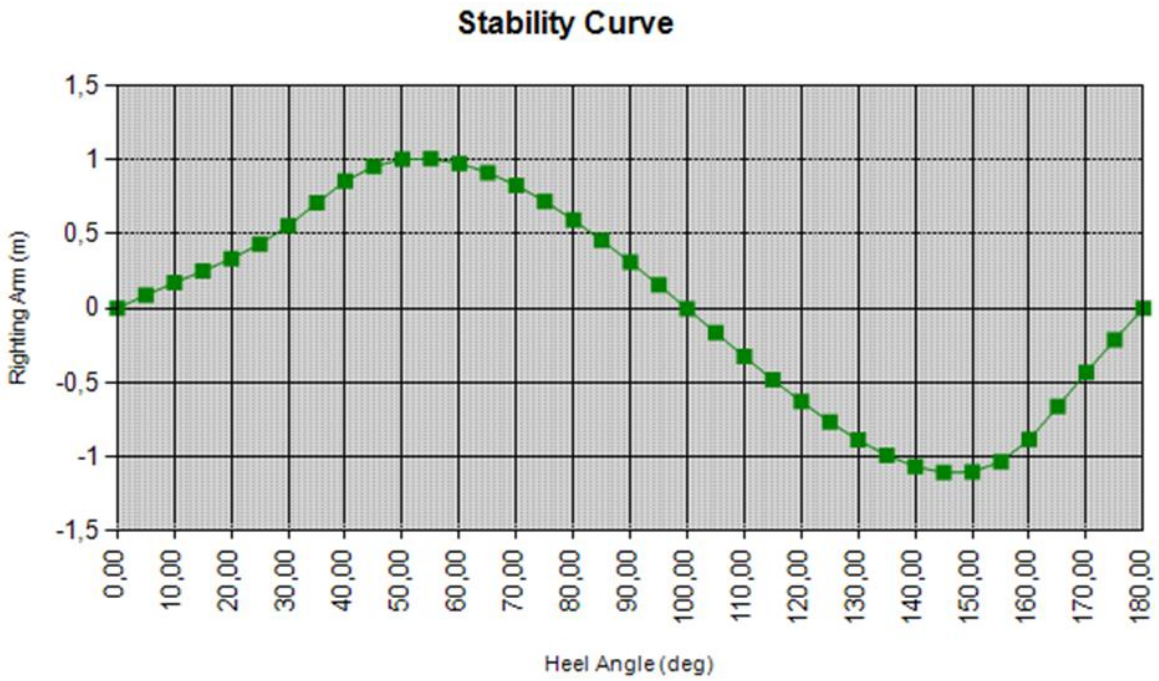
Condition	Displacement	LCB(m)	TCB(m)	VCB(m)	Wet Area (m^2)
Condition 1	3160,004	49,238	0,000	2,962	1431,129

Condition	Awp(m^2)	LCF(m)	TCF(m)	VCF(m)
Condition 1	997,436	44,074	0,000	4,920

Condition	BMt(m)	BMl(m)	GMt(m)	GMl(m)
Condition 1	3,534	190,373	0,996	187,835

Condition	Cb	Cp	Cwp	Cx	Cws	Cvp
Condition 1	0,499	0,640	0,795	0,779	2,623	0,628

Slika 22. Hidrostatičke karakteristike dobivene u Orca3D za stanje krcanja 1



Slika 23. Krivulja poluge statičkog stabiliteta u Orca3D za stanje krcanja 1

U sljedećoj tablici je prikazano kako se brod ponaša za poprečno nagibanje s korakom od 5°.

Za svaki kut je moguće očitati:

- polugu statičkog stabiliteta
- moment kojeg dobivamo množenjem s istisninom broda
- trim izražen u stupnjevima (negativni stupnjevi označavaju veći gaz na krmi tj. zategu)
- gaz na pramcu i krmi

Heel(deg)	Trim(deg)	Righting Arm (m)	Righting Moment	Point Name and Distance	
0,000	0,000	0,000	0,00	Gaz krma	-4,920
				Gaz pramac	-4,920
5,000	0,010	0,087	275,23	Gaz krma	-4,884
				Gaz pramac	-4,901
10,000	0,042	0,171	541,54	Gaz krma	-4,772
				Gaz pramac	-4,845
15,000	0,109	0,249	788,00	Gaz krma	-4,568
				Gaz pramac	-4,759
20,000	0,201	0,333	1051,95	Gaz krma	-4,287
				Gaz pramac	-4,637
25,000	0,305	0,432	1365,78	Gaz krma	-3,941
				Gaz pramac	-4,473
30,000	0,416	0,555	1753,64	Gaz krma	-3,535
				Gaz pramac	-4,261
35,000	0,530	0,709	2239,34	Gaz krma	-3,072
				Gaz pramac	-3,998
40,000	0,653	0,856	2705,42	Gaz krma	-2,567
				Gaz pramac	-3,707
45,000	0,774	0,955	3016,90	Gaz krma	-2,050
				Gaz pramac	-3,401
50,000	0,881	1,002	3166,22	Gaz krma	-1,537
				Gaz pramac	-3,075
55,000	0,976	1,006	3179,10	Gaz krma	-1,027
				Gaz pramac	-2,730
60,000	1,061	0,974	3079,28	Gaz krma	-0,518
				Gaz pramac	-2,370
65,000	1,139	0,913	2884,94	Gaz krma	-0,010
				Gaz pramac	-1,997
70,000	1,211	0,827	2611,81	Gaz krma	0,499
				Gaz pramac	-1,614
75,000	1,282	0,719	2270,72	Gaz krma	1,012
				Gaz pramac	-1,226
80,000	1,353	0,593	1875,32	Gaz krma	1,526
				Gaz pramac	-0,836
85,000	1,422	0,456	1440,54	Gaz krma	2,037
				Gaz pramac	-0,444
90,000	1,486	0,310	978,62	Gaz krma	2,541
				Gaz pramac	-0,051

Slika 24. Tablica statičkog stabiliteta dobivena u Orca3D za stanje krcanja 1

5.2.3. Stanje krcanja 2

Condition Summary

Load Condition Parameters

Condition	Weight / Sinkage	LCG / Trim	TCG / Heel	VCG (m)
Condition 1	4500,000 tonne-f	48,000 m	0,000 m	6,5

Resulting Model Attitude and Hydrostatic Properties

Condition	Sinkage (m)	Trim(deg)	Heel(deg)	Ax(m^2)
Condition 1	6,065	0,135	0,000	66,66

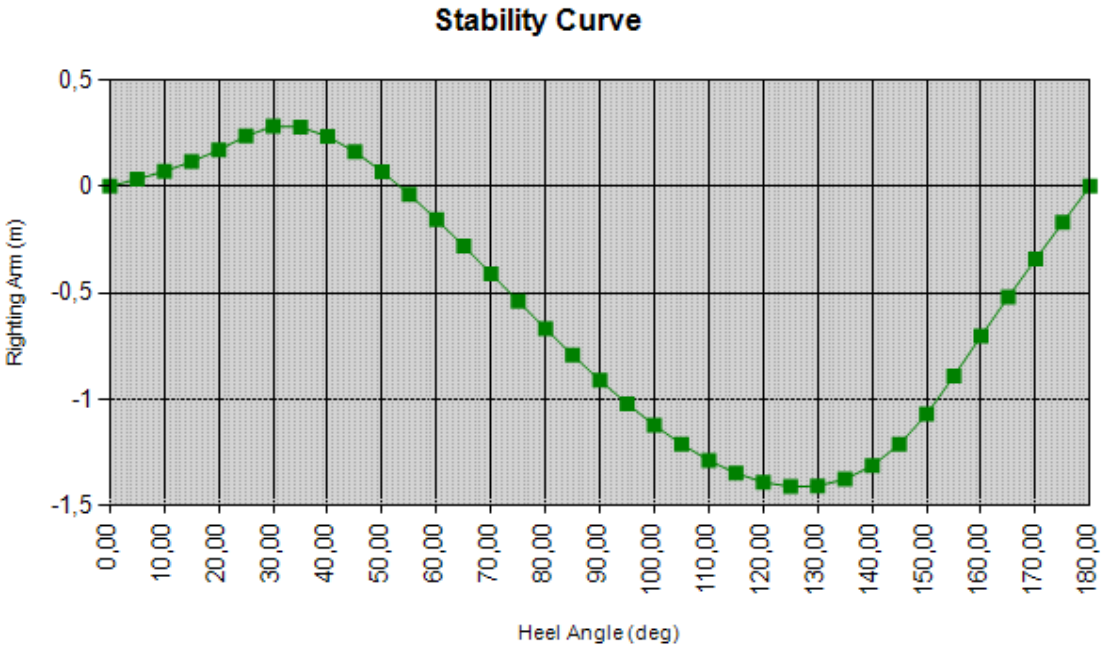
Condition	Displacement	LCB(m)	TCB(m)	VCB(m)	Wet Area (m^2)
Condition 1	4500,004	48,007	0,000	3,734	1708,683

Condition	Awp(m^2)	LCF(m)	TCF(m)	VCF(m)
Condition 1	1092,127	43,947	0,000	6,168

Condition	BMt(m)	BMl(m)	GMt(m)	GMl(m)
Condition 1	3,143	153,274	0,377	150,508

Condition	Cb	Cp	Cwp	Cx	Cws	Cvp
Condition 1	0,522	0,673	0,816	0,776	2,609	0,640

Slika 25. Hidrostatičke karakteristike dobivene u Orca3D za stanje krcanja 2



Slika 26. Krivulja poluge statičkog stabiliteta u Orca3D za stanje krcanja 2

Heel(deg)	Trim(deg)	Righting Arm (m)	Righting Moment	Point Name and Distance	
0,000	0,135	0,000	0,00	Gaz krma	-6,065
				Gaz pramac	-6,301
5,000	0,140	0,033	149,49	Gaz krma	-6,030
				Gaz pramac	-6,274
10,000	0,153	0,071	317,36	Gaz krma	-5,926
				Gaz pramac	-6,193
15,000	0,176	0,116	520,72	Gaz krma	-5,753
				Gaz pramac	-6,059
20,000	0,212	0,171	768,53	Gaz krma	-5,505
				Gaz pramac	-5,874
25,000	0,269	0,235	1058,80	Gaz krma	-5,173
				Gaz pramac	-5,643
30,000	0,350	0,283	1271,38	Gaz krma	-4,777
				Gaz pramac	-5,388
35,000	0,424	0,278	1250,17	Gaz krma	-4,375
				Gaz pramac	-5,115
40,000	0,488	0,234	1051,39	Gaz krma	-3,971
				Gaz pramac	-4,823
45,000	0,543	0,161	726,55	Gaz krma	-3,564
				Gaz pramac	-4,511
50,000	0,590	0,069	309,79	Gaz krma	-3,151
				Gaz pramac	-4,181
55,000	0,630	-0,038	(173,04)	Gaz krma	-2,732
				Gaz pramac	-3,832
60,000	0,665	-0,156	(703,68)	Gaz krma	-2,306
				Gaz pramac	-3,466
65,000	0,693	-0,281	(1266,32)	Gaz krma	-1,874
				Gaz pramac	-3,083
70,000	0,715	-0,410	(1846,73)	Gaz krma	-1,436
				Gaz pramac	-2,685
75,000	0,733	-0,540	(2431,66)	Gaz krma	-0,995
				Gaz pramac	-2,275
80,000	0,748	-0,669	(3010,21)	Gaz krma	-0,551
				Gaz pramac	-1,857
85,000	0,762	-0,794	(3571,20)	Gaz krma	-0,105
				Gaz pramac	-1,435
90,000	0,776	-0,911	(4101,43)	Gaz krma	0,341
				Gaz pramac	-1,013

Slika 27. Tablica statičkog stabiliteta dobivena u Orca3D za stanje krcanja 2

5.3. Usporedba rezultata

Cilj usporedbe rezultata je provjeriti i usporediti više različitih aspekata između različitih alata, kako bi se dobila cjelokupna slika i utvrdile eventualne greške.

Kada se radi o uspoređivanju različitih softverskih alata, razlike u rezultatima mogu biti očekivane jer različiti programi koriste različite algoritme za izračun. Usporedba krivulja poluge statičkog stabiliteta u programu Python i programu Orca3D je važan korak u verifikaciji i validaciji numeričkih simulacija. Ova usporedba će nam omogućiti da vidimo koliko su rezultati dobiveni u Pythonu usporedivi s onima dobivenima u Orca3D-u, koji se smatra standardom u pomorskom inženjerstvu.

Kako bi usporedba bila točna i pouzdana, potrebno je da su uvjeti krcanja broda isti u oba programa, kao i da je model broda identičan. Također, potrebno je osigurati da su parametri korišteni u oba programa jednaki.

Uzroci malih razlika u rezultatima mogu uključivati:

- različite metode aproksimacije istisnine broda
- različite metode za izračun brodskih sila i momenta
- zbog različitih numeričkih metoda koje se koriste u programima
- različite tolerancije u metodama za trimanje

5.3.1. Stanje krcanja 1

Tablica 1. Usporedba rezultata za stanje krcanja 1

	Python		Orca3D		Relativna pogreška[%]	
heel[°]	GZ[m]	trim[°]	GZ[m]	trim[°]	GZ	trim
0.0	0.000	0.000	0.000	0.000	0.000	0.000
5.0	0.087	0.010	0.087	0.010	0.000	0.000
10.0	0.171	0.042	0.0171	0.042	0.000	0.000
15.0	0.249	0.111	0.249	0.109	0.000	1.835
20.0	0.333	0.202	0.333	0.201	0.000	0.495
25.0	0.432	0.308	0.432	0.305	0.000	0.498
30.0	0.555	0.420	0.555	0.416	0.000	0.961
35.0	0.708	0.535	0.709	0.530	0.000	0.943
40.0	0.856	0.659	0.856	0.653	0.000	0.919
45.0	0.954	0.780	0.955	0.774	0.000	0.775
50.0	1.002	0.887	1.002	0.881	0.000	0.681
55.0	1.006	0.981	1.006	0.976	0.000	0.512
60.0	0.974	1.066	0.974	1.061	0.000	0.471

5.3.3. Stanje krcanja 2

Tablica 2. Usporedba rezultata za stanje krcanja 2

	Python		Orca3D		Relativna pogreška[%]	
heel[°]	GZ[m]	trim[°]	GZ[m]	trim[°]	GZ	trim
0.0	0	0.135	0	0.135	0.000	0.000
5.0	0.033	0.140	0.033	0.140	0.000	0.000
10.0	0.071	0.153	0.071	0.153	0.000	0.000
15.0	0.116	0.176	0.116	0.176	0.000	0.000
20.0	0.171	0.212	0.171	0.212	0.000	0.000
25.0	0.235	0.271	0.235	0.269	0.000	0.743
30.0	0.282	0.352	0.283	0.350	-0.353	0.571
35.0	0.278	0.426	0.278	0.424	0.000	0.472
40.0	0.233	0.489	0.234	0.488	-0.427	-0.427
45.0	0.161	0.543	0.161	0.543	0.000	0.000
50.0	0.068	0.590	0.069	0.590	-1.449	0.000
55.0	-0.039	0.630	-0.038	0.63	2.630	0.000
60.0	-0.157	0.664	-0.156	0.665	0.640	-0.150

S obzirom na to da je u ovom slučaju riječ o razlici u rezultatima vidljivima tek na trećoj decimali, može se smatrati da su rezultati prilično slični. Ovime potvrđujemo da su naše simulacije valjane i pouzdane, što nam omogućuje da ih koristimo u daljnjem razvoju i projektiranju brodova.

6. ZAKLJUČAK

U ovom radu izrađen je modul za proračun stabiliteta triangulirane forme broda u okviru programa otvorenog koda d3v-gsd. Modul se sastoji od funkcija koje se koriste za izračun istisnine broda i njenog težišta za određena stanja krcanja. Također u radu je napravljen algoritam za izračun gaza za ravnu vodnu liniju, odnosno trima za nagnuto stanje pri ulaznim podacima težine broda i njenog težišta. Kao glavna stavka u proračunu poprečnog stabiliteta izrađen je Python modul za generiranje krivulje poluge statičkog stabiliteta.

Napravljen je usporedni test između modela kreiranog u Pythonu i komercijalnom programu Orca3D, gdje su dobiveni praktično identični rezultati. Rezultati pokazuju da je modul koji je razvijen u ovom radu uspješno izračunavao polugu statičkog stabiliteta za različite vrijednosti kuta nagiba broda. Ovaj rad može biti koristan za inženjere i arhitekte u brodogradnji te za sve koji rade na projektiranju brodova i analizi njihovog stabiliteta.

LITERATURA

[1] *Python*, https://www.python.org/ (pristup 21.11.2022.).

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

[3] Itković H., *Analiza proračuna poprečne i uzdužne stabilnosti broda*, Sveučilište u Rijeci, Pomorski fakultet, Diplomski rad, 2014.

[4] Boban T., *Standardi za trim i stabilnost broda*, Pomorski fakultet u Splitu, Završni rad 2016.

[5] Uršić J., *Plovnost broda*, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, 1966.

[6] Erhardt L., J., *Vizualizacija i izračun hidrostatičkih karakteristika triangularizirane forme broda primjenom programa otvorenog koda linaetal-fsb/d3v*, Fakultet strojarstva i brodogradnje, Završni rad, Zagreb, 2020.

[7] https://tehnika.lzmk.hr/tehnickaenciklopedija/brod_3_stabilitet_broda.pdf

[8] Wilson P., A., *Basic Naval Architecture Ship Stability*, Springer International Publishing, 2018.

PRILOZI

I. Python programski kod za računanje karakteristika stabiliteta broda

```python
from hullformdir.hullform import *
from scipy import optimize
import numpy as np
from scipy.spatial.transform import Rotation
import math
import time
import numpy.linalg as la
from  typing import List

def angle_between_vectors_deg(v1, v2):
    """ Returns the angle in degrees between vectors 'v1' and 'v2'
    Return angle range is between 0 and 180° """

    cosang = np.dot(v1, v2)
    sinang = la.norm(np.cross(v1, v2))
    ang_rad = np.arctan2(np.clip(sinang,-1.0,1.0), np.clip(cosang,-1.0,1.0))
    return np.rad2deg(ang_rad)

def deg_to_vec(deg):
    return np.cos(np.deg2rad(deg))

class LoadCondition():
    def __init__(self,weight:float,CG:np.ndarray):
        self._ship_weight: float = 0.0
        self._ship_CG: np.ndarray = np.zeros(3)

class Waterline():
    @classmethod
    def ref_plane_normal(cls):
        return np.array([0.0, 0.0, -1])

    @classmethod
    def get_point_on_plane_close_to_input_point(cls,plane_point:np.ndarray,plane_normal:np.ndarray,
in_point:np.ndarray):
        w = np.array([-1.0, 0.0, 0.0])
        v = np.cross(w, plane_normal)
        u = np.cross(plane_normal, v)
        p1 = v*(in_point[1]-plane_point[1])
        p2 = u*(in_point[0]-plane_point[0])
        cp = plane_point+p1+p2
        return cp

    def __init__(self, pp_z:float, pp_x:float=0.0):
        self._plane_point = np.array([pp_x, 0.0, pp_z])
        self._plane_normal =  Waterline.ref_plane_normal()

    def set_plane_normal_component(self,value:float,index:int):
        self._plane_normal[index]= value

    def set_plane_normal_x(self,value:float):
        self.set_plane_normal_component(value,0)

    def set_plane_normal_y(self,value:float):
        self.set_plane_normal_component(value,1)

    def set_plane_normal_z(self,value:float):
        self.set_plane_normal_component(value,2)

    def set_plane_point_component(self, value: float, index: int):
        self._plane_point[index] = value

    def set_plane_point_x(self, value: float):
        self.set_plane_point_component(value,0)

    def set_plane_point_y(self, value: float):
        self.set_plane_point_component(value,1)

```

```
    def set_plane_point_z(self, value: float):
        self.set_plane_point_component(value,2)

    def reset_to_heel_angle_deg(self,heel:float):
        self._plane_normal = Waterline.ref_plane_normal()
        # heel is rotation over x axix (y-z plane)
        r = Rotation.from_euler('x', heel, degrees=True)
        self._plane_normal = r.apply(self._plane_normal)

    def get_angle_between_plane_normal_and_ref_plane_normal(self):
        return angle_between_vectors_deg(Waterline.ref_plane_normal(),self._plane_normal)

    def get_trim_angle(self):
        trim_vec = Waterline.ref_plane_normal()
        trim_vec[0]= self.normal[0]
        return angle_between_vectors_deg(Waterline.ref_plane_normal(),trim_vec)

    def set_trim_angle(self,trim):
        self._plane_normal = Waterline.ref_plane_normal()
        r = Rotation.from_euler('y', trim, degrees=True)
        self._plane_normal = r.apply(self._plane_normal)

    def get_heel_angle(self):
        heel_vec= self.normal.copy()
        heel_vec[0] = 0.0
        return angle_between_vectors_deg(Waterline.ref_plane_normal(),heel_vec)


    def get_max_distance_to_mesh(self, fvs, pts):
        max_dist = -np.inf
        most_dist_point = None
        for triangle in fvs:
            for i_p in triangle:
                p = np.array(pts[i_p])
                v = p - self.point  # vektor koji spaja točku na ravnini i točku u trokutu
                distance = np.dot(v, self.normal)  # Udaljenost točke od ravnine
                if distance > max_dist:
                    max_dist = distance
                    most_dist_point = p

        return max_dist, most_dist_point

    def create_and_emmit_geometry(self,name,length:float, width:float):
        w = np.array([-1.0, 0.0, 0.0])
        v = np.cross(w, self.normal)
        u = np.cross(self.normal, v)
        cp = self.point
        points = []
        dl = length*0.6
        dw = width*0.6
        points.append(cp + u * dl + v * dw)
        points.append(cp + u * dl - v * dw)
        points.append(cp - u * dl + v * dw)
        points.append(cp - u * dl - v * dw)
        fvs = []
        fvs.append([0, 1, 3])
        fvs.append([0, 3, 2])
        name+=' point {0}, normal {1}'.format(self.point, self.normal)
        g = GeometryExtension(name)
        g.mesh = om.TriMesh(points,fvs)
        g.emit_geometry_built()



    @property
    def point(self):
        return self._plane_point
```

```
134
135     @property
136     def normal(self):
137         return self._plane_normal
138
139
140
141 class ShipStability():
142     def __init__(self, hull_form:HullForm,main_deck_z,sea_density = 1025.9):
143         self._hf = hull_form
144         self._xmf=self._hf.get_x_main_frame_from_mesh()
145         self._sea_density = sea_density
146         self._ship_weight:float =0.0
147         self._ship_CG:np.ndarray = np.zeros(3)
148         self._wl:Waterline = Waterline(0.0,self.xmf)
149         self._main_deck_z = main_deck_z
150         self._mesh = self.get_mesh_form_closed_with_main_deck()
151
152     def get_mesh_form_closed_with_main_deck(self):
153         fvs = self._hf.mesh.fv_indices().tolist()
154         points = self._hf.mesh.points().tolist()
155         plane_point = np.array([self._xmf,0.0,self._main_deck_z])
156         plane_normal = Waterline.ref_plane_normal()
157         new_fvs, new_pts = self.get_mesh_below_inclined_waterline(fvs, points, plane_point, plane_normal)
158         mesh =  om.TriMesh(new_pts,new_fvs)
159         testgeo = GeometryExtension('Closed Hull Form')
160         testgeo.mesh = om.TriMesh(new_pts, new_fvs)
161         testgeo.emit_geometry_built()
162         return mesh
163     @property
164     def wl(self)->Waterline:
165         return self._wl
166
167     def get_drought_and_most_dist_point(self):
168         fvs = self._hf.mesh.fv_indices().tolist()
169         pts = self._hf.mesh.points().tolist()
170         T,p = self._wl.get_max_distance_to_mesh(fvs,pts)
171         return T,p
172
173
174     def calculate_displacement_and_displacementCG_example(self):
175
176         displacement, displacementCG, new_fvs, new_pts = self.calculate_displacement_and_displacementCG()
177         print('displacement, m3', displacement)
178         print('displacement, t', displacement*self._sea_density/1000.0)
179         print('displacement CG', displacementCG)
180         testgeo = GeometryExtension('Displ_Calc_Mesh')
181         testgeo.mesh = om.TriMesh(new_pts, new_fvs)
182         testgeo.emit_geometry_built()
183
184
185     def calculate_displacement_and_displacementCG(self):
186         fvs = self._mesh.fv_indices().tolist()
187         points = self._mesh.points().tolist()
188         plane_point = self._wl.point
189         plane_normal= self._wl.normal
190         new_fvs, new_pts = self.get_mesh_below_inclined_waterline(fvs,points,plane_point,plane_normal)
191         displacement, displacementCG = self.calculate_displacement_and_centroid(new_fvs, new_pts)
192         return displacement, displacementCG, new_fvs, new_pts
193
194
195     def determine_horizontal_waterline_for_current_weight(self):
196         z_mid,z_max = self._hf.get_z_mid_z_max_from_mesh()
197         self._wl = Waterline(z_mid,self.xmf)
198         res = optimize.root(self.fun_opt_root_drought_calculation, z_mid)
199         print(res)
200         x=res['x']
```

```
201         pp_z=x[0]
202         self._wl.set_plane_point_z(pp_z)
203         print('pp_z = ',pp_z)
204         fvs = self._mesh.fv_indices().tolist()
205         points = self._mesh.points().tolist()
206         print('T = ',self.wl.get_max_distance_to_mesh(fvs, points)[0])
207 
208 
209     def determine_trim_for_current_load_condition(self):
210         #self.wl.reset_to_heel_angle_deg(20)
211         x0 = np.array([self.wl.point[2],self.wl.normal[0]])
212         res = optimize.root(self.fun_opt_root_trim_calculation, x0)
213         x=res['x']
214         pp_z = x[0]
215         pl_nv_x = x[1]
216         self.wl.set_plane_point_z(pp_z)
217         self.wl.set_plane_normal_x(pl_nv_x)
218         fvs = self._mesh.fv_indices().tolist()
219         points = self._mesh.points().tolist()
220         displacement, displacementCG, new_fvs, new_pts = self.calculate_displacement_and_displacementCG()
221         displacement =  displacement * self._sea_density / 1000.0
222         print('trim, deg = ', self.wl.get_trim_angle())
223         print('displacement, t = ', displacement)
224         if False: # Print additional data
225             print('T = ', self.wl.get_max_distance_to_mesh(fvs, points))
226             print(self.wl.point)
227             print(self.wl.normal)
228         if True: # Create geometries for visualization
229             testgeo = GeometryExtension('Trim mesh')
230             testgeo.mesh = om.TriMesh(new_pts, new_fvs)
231             testgeo.emit_geometry_built()
232             bb = self._hf.bbox
233             length = bb.maxCoord[0] - bb.minCoord[0]
234             width = bb.maxCoord[1] - bb.minCoord[1]
235             self.wl.create_and_emmit_geometry('WL heel={0};-trim={1}'.format(self.wl.get_heel_angle(),
    self.wl.get_trim_angle()),length,width)
236 
237     def fun_opt_root_drought_calculation(self, pp_z: float):
238         diff = self._ship_weight - self.calculate_displacement_horizontal_waterline(pp_z)*self.
    _sea_density/1000.0
239         return diff
240 
241     def fun_opt_root_trim_calculation(self, x):
242         pp_z = x[0]
243         pl_nv_x = x[1]
244         self._wl.set_plane_point_z(pp_z)
245         self._wl.set_plane_normal_x(pl_nv_x)
246         displacement, displacementCG, new_fvs, new_pts = self.calculate_displacement_and_displacementCG()
247         diff1 = self._ship_weight - displacement*self._sea_density/1000.0
248         # Racunanje projekcija tezista istisnine i tezista broda na ravninu vodne linije
249         displacementCG_proj = self.project_point_on_plane(displacementCG)
250         shipCG_proj = self.project_point_on_plane(self._ship_CG)
251         # Racunanje razlike izmedu projekcija tezista istisnine i tezista broda
252         diff2 = np.linalg.norm(displacementCG_proj - shipCG_proj)
253         return diff1,diff2
254 
255     def project_point_on_plane(self, point):
256         plane_point = self._wl.point
257         plane_normal = self._wl.normal
258         d = -plane_normal.dot(plane_point)
259         t = -(plane_normal.dot(point) + d) / (plane_normal.dot(plane_normal))
260         projection = point + t * plane_normal
261         return projection
262 
263     def calculate_displacement_horizontal_waterline(self,pp_z :float):
264         fvs = self._mesh.fv_indices().tolist()
265         points = self._mesh.points().tolist()
```

```
266         plane_point = self._wl.point
267         plane_normal = self._wl.normal
268         plane_point[2] = pp_z
269         new_fvs, new_pts = self.get_mesh_below_inclined_waterline(fvs, points, plane_point, plane_normal)
270         displacement, centroid = self.calculate_displacement_and_centroid(new_fvs, new_pts)
271         return displacement
272 
273     def set_ship_G_and_CG(self,G:float,xCG,yCG,zCG):
274         self._ship_weight: float = G
275         self._ship_CG[0]= xCG
276         self._ship_CG[1] = yCG
277         self._ship_CG[2] = zCG
278 
279     def generate_static_stability_curve(self, max_heel_angle:float, step_heel_angle:float) -> (np.ndarray
    ,np.ndarray):
280         heeling_angles = np.arange(0.0, max_heel_angle+step_heel_angle, step_heel_angle).tolist()
281         fvs = self._hf.mesh.fv_indices().tolist()
282         pts = self._hf.mesh.points().tolist()
283         bb= self._hf.bbox
284         length = bb.maxCoord[0]-bb.minCoord[0]
285         width = bb.maxCoord[1] - bb.minCoord[1]
286         z_mid, z_max = self._hf.get_z_mid_z_max_from_mesh()
287         stability_curve_data=[]
288         i=0
289         for heel in heeling_angles:
290             heeling_angle = math.radians(heel)
291             self._wl.reset_to_heel_angle_deg(heel)
292             #if self.wl.point[2] < z_mid/5:
293             #    self.wl.set_plane_point_z(z_mid/5)
294             self.wl.set_plane_point_z(z_mid)
295             try:
296                 self.determine_trim_for_current_load_condition()
297                 displacement, CB,new_fvs, new_pts = self.calculate_displacement_and_displacementCG()
298                 GZ = 0
299                 if i > 0:
300                     CB_proj = self.project_point_on_plane(CB)
301                     ship_cg_proj = self.project_point_on_plane(self._ship_CG)
302                     vd = np.cross(self.wl.normal,(CB-self._ship_CG))
303                     GZ=np.linalg.norm(vd)
304                     if vd[1] > 0:
305                         GZ =-GZ
306                 T = self._wl.get_max_distance_to_mesh(new_fvs, new_pts)[0]
307                 #self.wl.create_and_emmit_geometry('Point {0}: heel {1:3.0f},°; GZ {2:6.3f}'.format(i + 1
    , heel, GZ),length,width)
308                 # if T > z_max*1.1:
309                 #     print(
310                 #         'Point {0}: heel {1:3.0f},°; GZ {2:6.3f} ; out heel {3:6.2f},°; trim {4:6.2f
    },°; T {5:6.2f}, m'.format(
311                 #             i + 1, heel, GZ, self._wl.get_heel_angle(), self._wl.get_trim_angle(), T))
312                 #     print('Further calculation of stability curve stopped, Drought to high!')
313                 #     continue
314                 displacement =displacement*self._sea_density/1000.0
315                 stability_curve_data.append([heel, GZ, self._wl.get_trim_angle(),displacement,T, self._wl
    .get_heel_angle()])
316                 print(
317                     'Point {0}: heel {1:3.0f},°; GZ {2:6.3f} ; out heel {3:6.2f},°; trim {4:6.2f},°; T {5
    :6.2f}, m'.format(
318                         i + 1, heel, GZ, self._wl.get_heel_angle(), self._wl.get_trim_angle(),T))
319             except BaseException as error:
320                 print('Point {0}: heel {1:3.0f},°; ERROR: {2}'.format(i + 1, heel,error))
321                 print('Further calculation of stability curve stopped!')
322                 continue
323             except:
324                 print('Unknown exception occurred during right arm calculation')
325             i+=1
326         return stability_curve_data
327 
```

```
328
329     def point_distance(self, a, b):
330         return math.sqrt((b[0] - a[0]) ** 2 + (b[1] - a[1]) ** 2 + (b[2] - a[2]) ** 2)
331
332     def get_mesh_below_inclined_waterline\
333                     (self, fvs:List[np.ndarray],points:List[np.ndarray],plane_point:np.ndarray,
    plane_normal:np.ndarray):
334         # Add central point for hull_cover mesh
335         cp = np.zeros(3)
336         icp = len(points)
337         points.append(cp)
338
339         d = plane_point.dot(plane_normal)
340         trianagles_bwl = []
341         i_wl_0 = len(points)
342         for fh in fvs:  # for TROKUT in TROKUTI:
343             points_bwl = []
344             points_owl = []
345             bwl_order = []
346             owl_order = []
347             i = 0
348             for vh in fh:  # for TOCKA in TROKUT
349                 p_np = np.array(points[vh])
350                 distance_to_plane = p_np.dot(plane_normal) - d
351                 if distance_to_plane < 0:
352                     points_owl.append(points[vh])
353                     owl_order.append(i)
354                 else:
355                     points_bwl.append(points[vh])
356                     bwl_order.append(i)
357                 i += 1
358             if len(points_bwl) == 3:
359                 trianagles_bwl.append(fh)
360                 continue
361             if len(points_owl) == 3:
362                 continue
363             if len(points_bwl) == 1:
364                 a = np.array(points_bwl[0])  # a je jedina ispod vodne linije
365                 b = np.array(points_owl[0])
366                 c = np.array(points_owl[1])
367
368                 t1 = (d - plane_normal.dot(a)) / plane_normal.dot(b - a)
369                 contact1 = a + (b - a) * t1
370
371                 t2 = (d - plane_normal.dot(a)) / plane_normal.dot(c - a)
372                 contact2 = a + (c - a) * t2
373
374                 n = len(points)
375                 points.append(contact1)
376                 points.append(contact2)
377
378                 # add hull triangle
379                 if bwl_order[0] == 1:
380                     fh_new = np.array([fh[bwl_order[0]], n + 1, n])
381                     trianagles_bwl.append(fh_new)
382                     # add hull_cover triangle
383                     fh_new = np.array([n, n + 1, icp])
384                     trianagles_bwl.append(fh_new)
385                 else:
386                     fh_new = np.array([fh[bwl_order[0]], n, n + 1])
387                     trianagles_bwl.append(fh_new)
388                     # add hull_cover triangle
389                     fh_new = np.array([n+1, n, icp])
390                     trianagles_bwl.append(fh_new)
391
392             elif len(points_bwl) == 2:
393                 a = np.array(points_owl[0])  # a je jedina iznad vodne linije
```

```
                b = np.array(points_bwl[0])
                c = np.array(points_bwl[1])

                t1 = (d - plane_normal.dot(a)) / plane_normal.dot(b - a)
                contact1 = a + (b - a) * t1

                t2 = (d - plane_normal.dot(a)) / plane_normal.dot(c - a)
                contact2 = a + (c - a) * t2

                n = len(points)
                points.append(contact1)
                points.append(contact2)
                # add hull triangles
                if owl_order[0] == 1:
                    fh_new = np.array([n, n + 1, fh[bwl_order[1]]])
                    trianagles_bwl.append(fh_new)
                    fh_new = np.array([n, fh[bwl_order[1]], fh[bwl_order[0]]])
                    trianagles_bwl.append(fh_new)
                    # add hull_cover triangle
                    fh_new = np.array([n+1, n, icp])
                    trianagles_bwl.append(fh_new)
                else:
                    fh_new = np.array([n, fh[bwl_order[1]], n + 1])
                    trianagles_bwl.append(fh_new)
                    fh_new = np.array([n, fh[bwl_order[0]], fh[bwl_order[1]]])
                    trianagles_bwl.append(fh_new)
                    # add hull_cover triangle
                    fh_new = np.array([n, n + 1, icp])
                    trianagles_bwl.append(fh_new)

        deck_points = np.array(points[i_wl_0:]).tolist()
        xmin = deck_points[0][0]
        xmax = deck_points[0][0]
        ixmax = 0
        ixmin = 0
        for i in range(len(deck_points)):
            if deck_points[i][0] > xmax:
                xmax = deck_points[i][0]
                ixmax = i
            if deck_points[i][0] < xmin:
                xmin = deck_points[i][0]
                ixmin = i
        xcp = (xmax + xmin) / 2.0
        zcp = (deck_points[ixmax][2] + deck_points[ixmin][2]) / 2.0
        cp[0]=xcp
        cp[2]=zcp
        new_cp= Waterline.get_point_on_plane_close_to_input_point(plane_point,plane_normal,cp)
        cp[:] = new_cp[:]
        return trianagles_bwl, points

    def generate_hull_cover_triangles(self,points:list,i_wl_0,fvi):
        #Method is not used
        deck_points_ixs=np.arange(i_wl_0,len(points)).tolist()
        deck_points=np.array(points[i_wl_0:]).tolist()
        xmin=deck_points[0][0]
        xmax = deck_points[0][0]
        ixmax=0
        ixmin=0
        for i in range(len(deck_points)):
            if deck_points[i][0]>xmax:
                xmax = deck_points[i][0]
                ixmax =i
            if deck_points[i][0] < xmin:
                xmin = deck_points[i][0]
                ixmin = i
        for i in range(len(deck_points)):
            deck_points[i][0]=deck_points[i][0]-(xmax - xmin)/2
```

```
461         #circular sort
462         deck_points_ixs_s =  [x for _, x in sorted(zip(deck_points, deck_points_ixs), key=lambda c:np.
    arctan2(c[0][0], c[0][1]))]
463         xcp=(xmax+xmin)/2.0
464         ycp=(deck_points[ixmax][1]+deck_points[ixmin][1])/2.0
465         zcp = (deck_points[ixmax][2] + deck_points[ixmin][2]) / 2.0
466         cp=np.array([xcp,ycp,zcp])
467         icp=len(points)
468         points.append(cp)
469         for i in range(len(deck_points_ixs_s)-1):
470             fvi.append(np.array([deck_points_ixs_s[i], deck_points_ixs_s[i+1],icp]))
471         fvi.append(np.array([deck_points_ixs_s[-1], deck_points_ixs_s[0], icp]))
472 
473     def calculate_displacement_and_centroid(self, fvs, points):
474         displacement = 0
475         centroid = 0
476         for fh in fvs:
477             v, t = self.calculate_tethraedar_displacement_and_centroid(fh, points)
478             displacement += v
479             centroid += v * t
480         return displacement, centroid / displacement
481 
482     def calculate_tethraedar_displacement_and_centroid(self, fh, points):
483         try:
484             p1 = np.array(points[fh[0]])
485             p2 = np.array(points[fh[1]])
486             p3 = np.array(points[fh[2]])
487 
488             v321 = p3[0] * p2[1] * p1[2]
489             v231 = p2[0] * p3[1] * p1[2]
490             v312 = p3[0] * p1[1] * p2[2]
491             v132 = p1[0] * p3[1] * p2[2]
492             v213 = p2[0] * p1[1] * p3[2]
493             v123 = p1[0] * p2[1] * p3[2]
494             displacement_of_tetrahedron = (1.0 / 6.0) * (-v321 + v231 + v312 - v132 - v213 + v123)
495             centroid_of_tetrahedron = (p1 + p2 + p3) / 4
496         except BaseException as error:
497             print('An exception occurred: {}'.format(error))
498         except:
499             print('Unknown exception occurred during signals connection')
500 
501         return displacement_of_tetrahedron, centroid_of_tetrahedron
502 
503     @property
504     def xmf(self):
505         return  self._xmf
506 
507 def test_wl_1():
508     wl1 = Waterline(5)
509     wl1.reset_to_heel_angle_deg(181)
510     print(wl1._plane_normal)
511     angle = wl1.get_angle_between_plane_normal_and_ref_plane_normal()
512     print(angle)
513 
514 def test_wl_2():
515     wl = Waterline(5)
516     wl.reset_to_heel_angle_deg(20)
517     wl.normal[0]=0.05
518     print(wl.normal)
519     w = np.array([-1.0,0.0,0.0])
520     u= np.cross(w,wl.normal)
521     v = np.cross(wl.normal,u)
522     print(v)
523 
524 if __name__ == "__main__":
525     # Test
526     test_wl_2()
```

File - C:\Users\lovro\Desktop\d3vps6\d3v-gsd\commands\ship_stability_gui.py

```python
from PySide6.QtWidgets import QApplication, QMenu, QFormLayout,QWidget, QLineEdit
from PySide6.QtWidgets import QPushButton, QVBoxLayout,QHBoxLayout
from PySide6.QtCore import Qt
from PySide6.QtWidgets import QInputDialog
from interactive_chart_widget import InteractiveTableChartWidget
#d3v imports
#d3v-gsd
from hullformdir.hullform import HullForm
from hullformdir.shipstability import ShipStability
#from hullformdir.resistance_h&m import Holtrop_and_Mennen_resistance_prediction_ver2

class ShipStabilityGUI():
    def __init__(self,hfc):
        self._hf_command = hfc #HullFormCommand
        self._app = QApplication.instance()

        self.menuMain = QMenu("Ship &Stability")
        mb = self.mainwin.menuBar()
        mb.addMenu(self.menuMain)

        menu_result = self.menuMain.addAction("Calculate Displacement")
        menu_result.triggered.connect(self.on_calculate_displacements)
        menu_result = self.menuMain.addAction("Calculate Drought")
        menu_result.triggered.connect(self.on_calculate_drought)
        menu_result = self.menuMain.addAction("Calculate Trim")
        menu_result.triggered.connect(self.on_calculate_trim)
        menu_result = self.menuMain.addAction("Curve of Static Stability")
        menu_result.triggered.connect(self.on_calculate_css)

    @property
    def selected_hull_form(self):
        return self._hf_command.selected_hull_form

    @property
    def active_hull_form(self):
        return self._hf_command.active_hull_form

    @property
    def hull_forms(self):
        return self._hf_command.hull_forms


    def on_calculate_displacements(self):
        if isinstance(self.active_hull_form, HullForm):
            main_deck_z =self.active_hull_form.bbox.maxCoord[2]-0.01
            text, ok = QInputDialog.getText(self.mainwin, 'Input Dialog',
                                                'Input drought for calculation:')
            if ok:
                try:
                    wl_z = float(text)
                    if wl_z <= main_deck_z:
                        sscalc = ShipStability(self.active_hull_form,main_deck_z)
                        sscalc.wl.set_plane_point_z(wl_z)
                        sscalc.calculate_displacement_and_displacementCG_example()
                    else:
                        print('ERROR: Inputed drought greater than ship height. Calculation ommitted!')
                except:
                    print('ERROR: Inputed drought is not a number. Calculation ommitted!')



    def on_calculate_drought(self):
        if isinstance(self.active_hull_form, HullForm):
            main_deck_z = self.active_hull_form.bbox.maxCoord[2] - 0.01
            sscalc = ShipStability(self.active_hull_form, main_deck_z)
            z_mid, z_max = self.active_hull_form.get_z_mid_z_max_from_mesh()
            # sscalc.wl.set_plane_point_x(self.active_hull_form.xmf)
```

```
            sscalc.wl.set_plane_point_x(50.0)
            sscalc.wl.set_plane_point_z(z_mid)
            sscalc.set_ship_G_and_CG(3160.0, 50.0, 0.0, 5.5)
            sscalc.determine_horizontal_waterline_for_current_weight()

    def on_calculate_trim(self):
        if isinstance(self.active_hull_form, HullForm):
            main_deck_z = self.active_hull_form.bbox.maxCoord[2]-0.01
            sscalc = ShipStability(self.active_hull_form, main_deck_z)
            z_mid, z_max = self.active_hull_form.get_z_mid_z_max_from_mesh()
            #sscalc.wl.set_plane_point_x(self.active_hull_form.xmf)
            sscalc.wl.set_plane_point_x(50.0)
            sscalc.wl.set_plane_point_z(z_mid)
            sscalc.set_ship_G_and_CG(3160.0, 50.0, 0.0, 5.5)
            sscalc.determine_trim_for_current_load_condition()

    def on_calculate_css(self):
        if isinstance(self.active_hull_form, HullForm):
            main_deck_z = self.active_hull_form.bbox.maxCoord[2]-0.001
            sscalc = ShipStability(self.active_hull_form, main_deck_z)
            z_mid, z_max = self.active_hull_form.get_z_mid_z_max_from_mesh()
            #sscalc.set_ship_G_and_CG(3160.0,50.0,0.0,5.5)
            #sscalc.set_ship_G_and_CG(3160.0, 49.238, 0.0, 5.5)
            sscalc.set_ship_G_and_CG(4500.0, 48.0, 0.0, 6.5)
            widget_css = DialogStaticStabilityDiagram(self.mainwin,sscalc)
            widget_css.setWindowFlag(Qt.WindowType.Window)
            widget_css.show()


    @property
    def app(self):
        return self._hf_command.app

    @property
    def mainwin(self):
        return self._hf_command.mainwin

    @property
    def glwin(self):
        return self._hf_command.glwin

class DialogStaticStabilityDiagram(QWidget):
    def __init__(self, parent,sscalc):
        super().__init__(parent)
        self.setWindowFlag(Qt.WindowType.Window)
        self.sscalc = sscalc
        sizetxt=25
        self.mainwin = parent
        #Interactive Chart
        self.chart = InteractiveTableChartWidget()
        #self.chart.setTitle('Static stability curve plot')

        self.setWindowTitle("Static stability curve")
        self.btnGenerate = self.createButton("&Generate", self.refreshResults)

        self.txtMaxHeelAngle = QLineEdit()
        self.txtMaxHeelAngle.setFixedHeight(sizetxt)
        self.txtMaxHeelAngle.setText('90')
        self.txtMaxHeelAngle.setAlignment(Qt.AlignRight)
        self.txtHeelAngleStep = QLineEdit()
        self.txtHeelAngleStep.setFixedHeight(sizetxt)
        self.txtHeelAngleStep.setText('5')
        self.txtHeelAngleStep.setAlignment(Qt.AlignRight)


        mainLayout = QVBoxLayout()
        mainLayout.setStretch(0,1)
```

```
        mainLayout.setStretch(1, 0)

        controlLayout = QHBoxLayout()
        controlWidget = QWidget()
        controlWidget.setFixedHeight(sizetxt*3)
        controlWidget.setLayout(controlLayout)

        inputLayout = QFormLayout()
        controlLayout.addLayout(inputLayout)
        controlLayout.addWidget(self.btnGenerate)

        inputLayout.addRow("&Max. heeling angle:", self.txtMaxHeelAngle)
        inputLayout.addRow("&Heel angle step:", self.txtHeelAngleStep)

        mainLayout.addWidget(self.chart)
        mainLayout.addLayout(controlLayout)
        mainLayout.addWidget(controlWidget)

        self.setLayout(mainLayout)



    def createButton(self, text, member):
        button = QPushButton(text)
        button.clicked.connect(member)
        return button

    def refreshResults(self):
        max_heel_angle= float(self.txtMaxHeelAngle.text())
        step_heel_angle = float(self.txtHeelAngleStep.text())
        data =  self.sscalc.generate_static_stability_curve(max_heel_angle,step_heel_angle)
        n_rows = len(data)
        # chart
        n_cols=len(data[0])
        data_names = ['Heel,°', 'GZ, m','Trim,°', 'Displacement, t','Drought, m','Calc Heel,°']
        data_formats = ['{:.1f}', '{:.3f}', '{:.3f}', '{:.1f}', '{:.3f}', '{:.3f}']
        chart_data_pairs = [0, 1]
        self.chart.set_data(n_cols, n_rows, data_names, data_formats, data, chart_data_pairs, 'line')
```