Parametric Ship Design in MATLAB

David Timmer Endal
Adam Sven Åkervall

Bachelor's thesis in Marine technology
Bergen, Norway 2024

Høgskulen på Vestlandet

Western Norway
University of
Applied Sciences

Parametric Ship Design in MATLAB

David Timmer Endal
Adam Sven Åkervall

Department of Mechanical- and Marine Engineering
Western Norway University of Applied Sciences
NO-5063 Bergen, Norge

IMM 2024-M40

Høgskulen på Vestlandet
Institutt for Maskin- og Marinfag
Inndalsveien 28,
NO-5063 Bergen, Norge

Cover and backside images © Norbert Lümmen

Norsk tittel:	Parametrisk Skipsdesign i MATLAB
Author(s), student number:	David Timmer Endal 602435 Adam Sven Åkervall 599065
Study program:	Marine technology
Date:	May 2024
Report number:	IMM 2024-M 40
Supervisor at HVL:	Thomas Henrik Hertzfelder Hansen
Assigned by:	Breeze ship design
Contact person:	Solveig Aasheim
Antall filer levert digitalt:	2

Preface

In this project conducted at the Western Norway University of Applied Sciences, Department of Mechanical and Marine Engineering, we aimed to develop software capable of designing ship hulls based on a set of parameters and exporting these designs to 3D modeling and simulation programs. This thesis is part of the Marine Technology study program and the project was supported by Breeze Ship Design.

We extend our gratitude to Dr. Thomas Henrik Hertzfelder Hansen, our supervisor from the university, for his guidance and rapid response to our questions throughout the project, which facilitated our progress.

We also extend our gratitude to Breeze ship design, especially Solveig Aasheim Johansen, Head of Hydrodynamics at the company.

We also acknowledge the Western Norway University of Applied Sciences for providing the resources and support to complete this thesis. This thesis does not include specific funding or belong to a numbered project.

This project not only made our technical skills better, but it also enhanced our understanding of ship design. We hope this software can be further developed to create even more advanced versions. We are grateful for the opportunities and support we have received.

Figure 1: Logo of Breeze Ship Design.

Abstract

This study explores the need for a ship parametrization program to aid in the design process by quickly generating easily customizable ship hulls. The designs are meant to serve as a base for further work in more sophisticated modeling software, closing the gap between early ship design and analysis. The program incorporates a set of parametric curves called Bézier curves to define the structure of the hull. These curves are employed to generate the hull surface, which is then exported to be used in appropriate modeling software.

The program, named Floke, serves as a foundation for a larger hull optimization project. The method of parametrization leaves room for additional features to simplify the design process and provide calculations for the evaluation of hull performance. This report presents the design philosophy and methodology behind Floke, and includes descriptions of the program's strengths, limitations and future potential.

Sammendrag

Denne studien utforsker behovet for et skipsparametriseringsprogram med det formål å hjelpe til i designprosessen ved å raskt å generere skipsskrog som enkelt kan tilpasses. Designet er ment å fungere som en base for videre arbeid i mer sofistikert modelleringsprogramvare, slik at skillet mellom tidlig skipsdesign og analyse minkes. Programmet inneholder et sett med parametriske kurver kalt Bézier-kurver for å definere skrogets struktur. Disse kurvene brukes til å generere skrogoverflaten, som deretter eksporteres til bruk i passende modelleringsprogramvare.

Programmet, som har fått navnet Floke, fungerer som et grunnlag for et større skrogoptimaliseringsprosjekt. Metoden for parametrisering gir rom for tilleggsfunksjoner som forenkler designprosessen og utføring av beregninger for evaluering av skrogets ytelse. Denne rapporten presenterer designfilosofien og metodikken bak Floke, og inkluderer beskrivelser av programmets styrker, begrensninger og fremtidige potensial.

Contents

Abstract iii

Sammendrag v

1 Introduction 1

2 Coordinate system 2

3 Method 2
3.1 Tools 2
3.2 Parametrization 3
3.3 Bézier curves 3
3.4 Hull sections 4
3.5 Design method 6
3.5.1 Parameters 6
3.5.2 Bézier curves and control points 7
3.5.3 Linearization 8
3.5.4 Translation 9
3.5.5 Midship Coefficient 11
3.5.6 Interpolation 12
3.5.7 Scaling 13
3.5.8 Surface generation 14
3.6 Export of design 15

4 Results 15
4.1 Input parameters 16
4.2 Main curves 16
4.3 Ship models 17
4.4 CFD analysis 17
4.4.1 Mesh and cell count 18
4.4.2 y^+ values 18
4.4.3 Wave pattern and surface elevation 19
4.4.4 Resistance 20

5 Discussion 21
5.1 Review of results 21
5.2 Scope of project 21
5.3 Limitations 22
5.4 Challenges 23
5.4.1 Accuracy 23
5.4.2 Export 24
5.5 Further potential 24
5.5.1 Improved functionality 24
5.5.2 Programming language 24
5.5.3 Conditions 25
5.5.4 Hull optimization 25

6 Conclusion 25

References 26

Appendices 27
Appendix A: Matlab script 27
Appendix B: Software documentation 43

1 Introduction

The marine engineering industry is rapidly expanding [1], suggesting a need for the advancement of technologies that support the development of maritime systems, in order to keep pace with this growth.

Focusing on the ship design branch, the development of software that supports the creation of efficient hull designs becomes essential for a more cohesive design process. The ship design task is a meticulous and resource-demanding process, partly due to the cost and time spent simulating the designs [2]. This process would benefit from a program capable of generating a customizable hull, with built-in tools made to alter the design to satisfy the requirements of the project. By increasing the quality of the design at an early stage, the number of simulations needed would decrease, which would serve to reduce cost and time spent with software analysis.

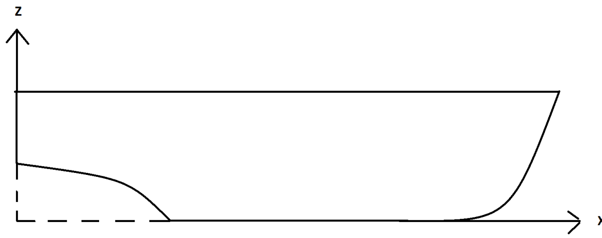
This report presents the development of Floke, a ship parametrization program created to bridge the gap between early ship design and simulation, drawing inspiration from the works of Ingrassia et al. [3]. Floke provides a simple approach to modifying and creating ship hulls based on a set of input parameters and lets the developer visualize the design and export it to more sophisticated modeling and simulation programs. Floke employs a set of Bézier curves to define the frame of the hull, based on a set of control points and the input parameters. The program also includes a feature for the adjustment of the hull shape in terms of the midship coefficient.

Although similar programs already exist [4], they are often expensive and require the developer to have a deep understanding of the software. Floke's purpose is to be a private program, serving as the foundation for a larger and more sophisticated hull optimization tool, and limiting the need for expensive software and third-party services.

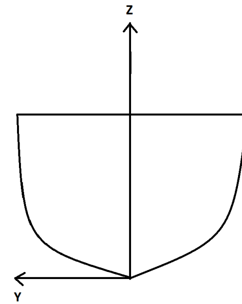
This report begins by establishing the tools used to develop Floke, and continues with describing the methodology behind the program. Second, the results of the simulation for two different hull designs are presented. Finally, the report provides a discussion of the results and the program's strengths, limitations, and further potential.

2 Coordinate system

Floke uses a 3-dimensional Cartesian coordinate system with the origin at the stern. The X-direction defines the hull's longitudinal axis, the Y-direction defines the lateral axis and the Z-direction defines the vertical axis. It is important to note that the longitudinal axis defines the waterline, meaning the submerged positions of the hull lie on the negative Z-axis.



(a) XZ-plane as seen from starboard side



(b) YZ-plane as seen from the stern

Figure 2: Coordinate system for waterline at Z = 0

3 Method

In this section, each part of the ship design process is explained. It starts with an introduction of the software tools used, followed by descriptions of parametrization and Bézier curves as fundamental concepts. Finally, the design method describes the process in which Floke generates a ship hull based on these concepts. These chapters only describe the method in terms of design philosophy and mathematics. They do not contain descriptions or explanations for how the program code works, which is reserved for the software documentation. The program code and software documentation is included in **APPENDIX A** and **APPENDIX B** respectively.

3.1 Tools

Floke is developed using MATLAB version R2023b [5]. MATLAB is a programming software with an extensive environment for algorithm development and numerical analysis. The capabilities of the program in terms of parametric modeling, data visualization, graphical representation, and iterative design are important to achieve the desired results.

The second program used throughout the development is Creo Parametric [6]. Creo is a powerful three-dimensional CAD (Computer-Aided Design) program that provides a method for analysis, representation, and simulation with a user-friendly interface. The program is used to analyze and verify the MATLAB results from various development stages, to ensure that Floke performs correctly.

Another tool used in this project is Simcenter Star-CCM+ [7], a Navier-Stokes solving program. This program is used to simulate the hull designs by solving Unsteady Reynolds-Averaged Navier-Stokes (URANS) equations.

This report is written using LaTeX through the cloud-based LaTeX editor Overleaf [8]. LaTeX is a software system for typesetting documents, as opposed to formatted text processors like Microsoft Word. LaTeX is widely used in the communication and publication of scientific documents.

This project incorporates the use of AI technology. ChatGPT from OpenAI [9] is primarily used as a troubleshooting tool, due to its ease of use and effective solution generation. It is important to stress, that for this project, ChatGPT is only employed to generate debugging solutions and give suggestions for code changes where more efficient alternatives are present. ChatGPT is not used in any decision-making processes or to generate any code present in Floke. For this written document, ChatGPT is only used to quickly generate command templates in LaTeX for insertion of equations, code snippets, tables, and figures. ChatGPT is not used to generate any text present in this report.

3.2 Parametrization

The central focus of this project is the parametric modeling of ship hulls. The goal is to create a design based on a set of parameters that define the characteristics of the hull. This approach allows developers to obtain a wide range of configurations just by altering the parameters instead of redesigning the hull from the ground up.

The use of parametrization in ship design is a valuable method for several reasons. The efficiency increases, and it reduces the time required to produce several designs of the hull just by changing the value of the parameters. It also ensures standardization in how changes are implemented, as the impact of modifying any parameter is predictable and is automatically updated in the model. Another reason is to allow better integration with other computational tools, such as simulation software, to validate designs under various conditions.

3.3 Bézier curves

The ship design process utilizes Bézier curves to shape the hull structure. These are parametric curves that rely on control points to determine their shape [10]. The flexibility and precision of Bézier curves make them particularly useful in applications where complex shapes need to be designed with high accuracy and smoothness. An example of a Bézier curve can be seen in Figure 3.

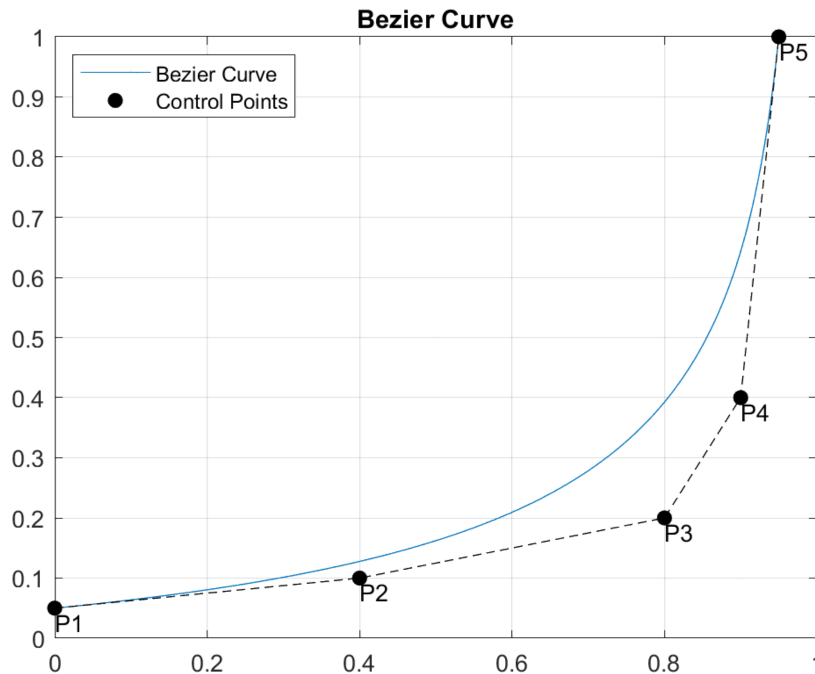


Figure 3: An image of a Bézier curve and its control points.

There are different variations of Bézier curves, the two primary ones are the classical (non-rational) Bézier curves and rational Bézier curves.

The classical Bézier curves are formulated using a polynomial expression where each point's influence on the curve's shape is determined by Bernstein polynomials. These curves can be easily manipulated by adjusting a set of control points, resulting in a smoothly shaped curve. A classical Bézier curve of degree n is expressed as Equation 1:

$$B(t) = \sum_{i=0}^{n} P_i \binom{n}{i} t^i (1-t)^{n-i} \tag{1}$$

Where P_i are the control points and t is a parameter that varies from 0 to 1.

The rational Bézier curves are an extended version of the classical ones. These curves incorporate a set of weights for each control point, which makes the process of creating a curve more flexible allowing for the representation of more complex shapes. A rational Bézier curve of degree n is expressed as Equation 2:

$$R(t) = \frac{\sum_{i=0}^{n} w_i P_i \binom{n}{i} t^i (1-t)^{n-i}}{\sum_{i=0}^{n} w_i \binom{n}{i} t^i (1-t)^{n-i}} \tag{2}$$

Where P_i are the control points, w_i are the weights associated with these control points, and t is a parameter that varies from 0 to 1.

During the program development, both classical and rational Bézier curves are considered for their respective strengths in curve manipulation and accuracy. While rational Bézier curves offer the capability to precisely model complex shapes, they also introduce additional complexity due to the management of weights for each control point. Given the scope of the project, classical Bézier curves are selected as they provide sufficient accuracy for most ship hull designs and are simpler to implement and manipulate.

3.4 Hull sections

The hull is divided into three sections: the foreship, midship, and aft, where each section is handled independently, meaning the shape of one section can be altered without making adjustments to the others. The only exception is the midship, which defines a few of the most important parameters, such as maximum breadth and draught. Furthermore, the main curves defining the shape of the hull are generated for the port side. The curves are then mirrored to generate the starboard side.

The design begins with the definition of the midship section. It is here the parameters of maximum breadth and draught are implemented. This section is made up of four curves: the midship section curve is translated to a forward and aft position, and the shape is enclosed by a keel line and a deck line. It is important to note that the section curves are identical. This is a conscious design choice, as the front and aft sections have greater influence over the hull's hydrodynamic performance. As the midship section is the largest, it has the greatest influence over the hull's hydrostatic performance.

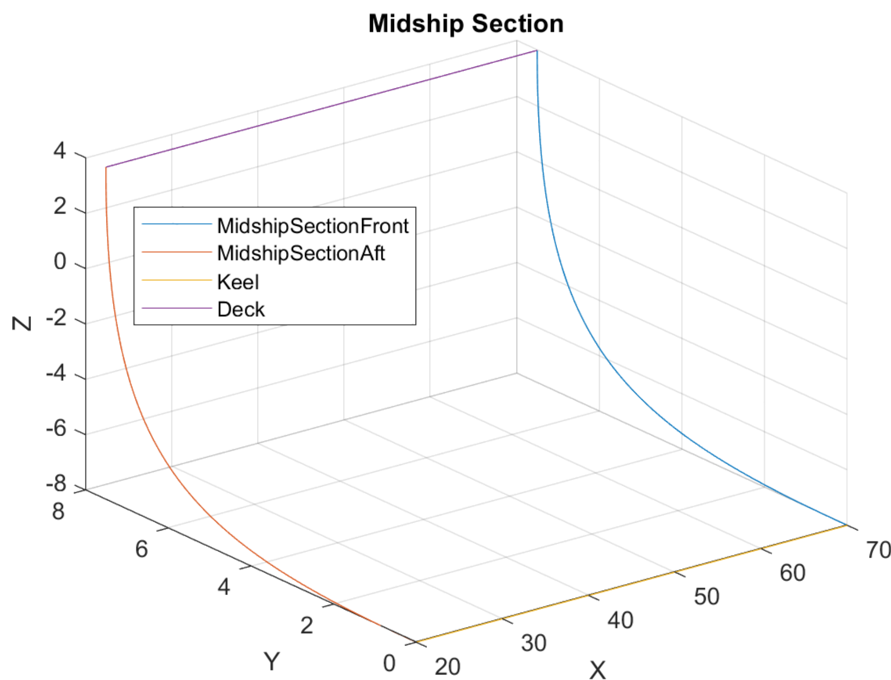


Figure 4: An image of the midship curves

The foreship is defined by three curves: the forward midship section, the bow, and the foredeck curves. At the intersection between the foredeck and bow, there is a fourth, yet insignificant curve dubbed the bow tip curve. This curve is very small and has no influence over the shape of the hull. Its only purpose is to aid in surface generation by allowing the surface to be drawn from four edges instead of three. A more detailed description of the tip curve is provided in Chapter (3.5.6).

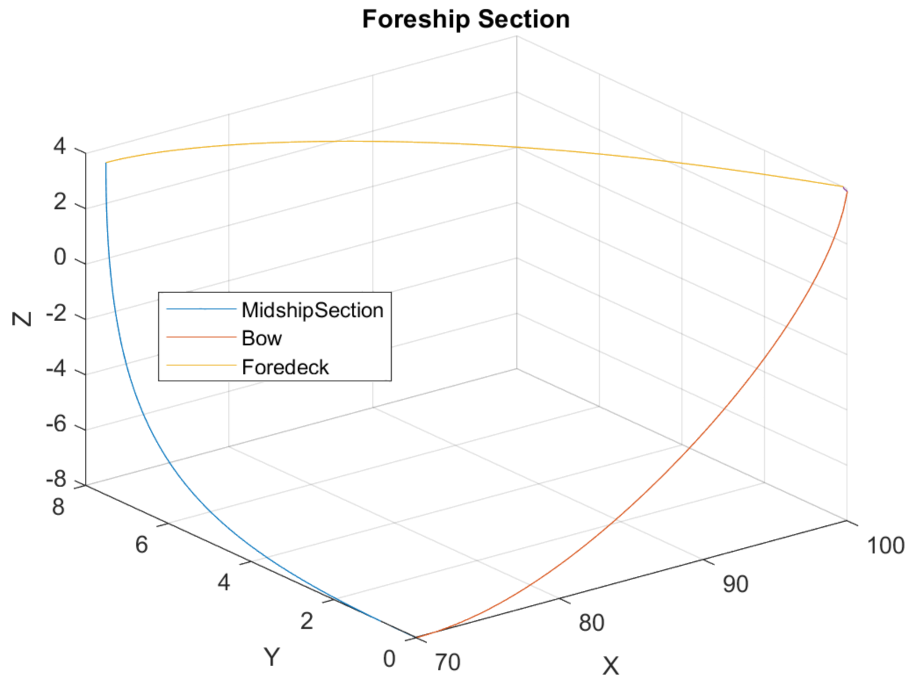


Figure 5: An image of the bow curves

The shape of the aft section is defined by four curves: the stern section, aft midship section, aft deck, and aft keel curves. The fifth curve, the stern deck curve as seen in Figure 6, serves only to provide an enclosed stern surface and is drawn from the endpoint of the stern section curve to the center of the hull. The most defining characteristic of the aft section's appearance is the shape of the aft keel curve. This curve in junction with the **aftHeight** parameter can be manipulated to create space for ship elements such as axles, propellers, and rudders.

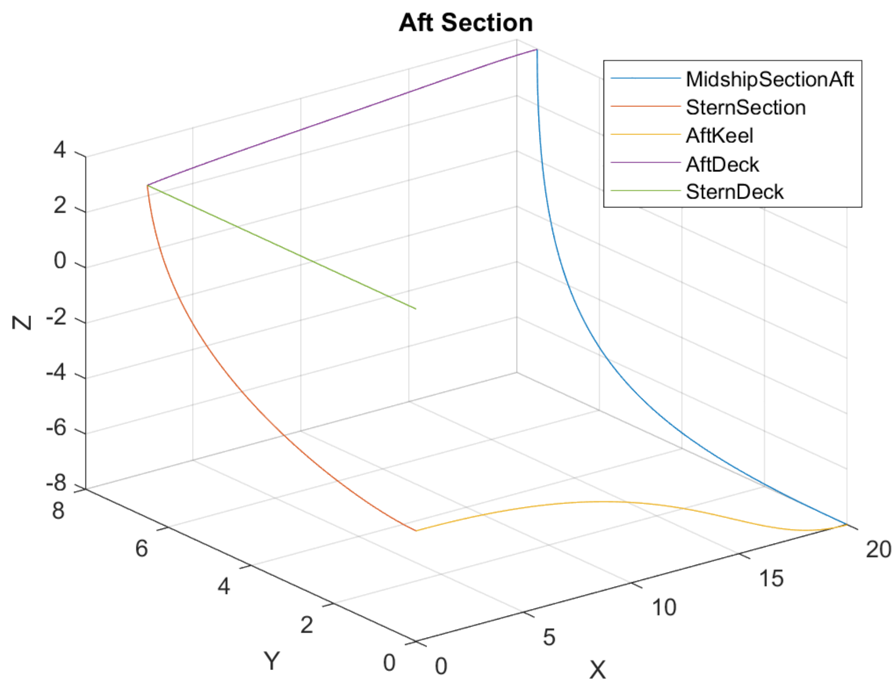


Figure 6: An image of the aft curves

After each section is properly defined, the shape is mirrored across the Y-plane to generate the starboard side. The surface is generated after the overall shape of the hull is finished.

3.5 Design method

This section describes the order of operations when generating a hull design in Floke, beginning with the input parameters. Each subsection provides detailed descriptions of the methods used.

The design method is divided into the following sections:

1. **Parameters**: Introduces the hull's main parameters and defines their purposes.
2. **Control points and Bézier curves**: Description of how the control points are implemented, and the generation of the hull's main curves including figures.
3. **Linearization**: Operation for generating linearly spaced curve points. Simplifies the interpolation and surface stages.
4. **Translation**: Description of how the main curves are translated to three dimensions and made to fit the hull parameters. Includes a figure of the ship skeleton in three dimensions with all the main curves and their mirrors.
5. **Midship coefficient adjustment**: Adjustment of the hull to meet the conditional midship coefficient parameter.
6. **Interpolation**: Operation for generating interpolated curves to define the surface of the hull.
7. **Scaling**: Scaling of the interpolated curves to coincide with the hull's longitudinal curves.
8. **Surface figure**: Figure of the ship with its surface in three dimensions.

The design method ends with a description of how the hull can be exported as a three-dimensional model.

3.5.1 Parameters

The central component of the program is its parameters, which are established at the very beginning of the code. These parameters define all the main dimensions of the ship, including its height, lengths, breadths, waterline level, and midship coefficient. This section of the code incorporates four distinct types of parameters, starting with the midship parameters. These parameters are crucial as they define the characteristics of the midship section and extend to other areas of the ship. Specifically, the **midshipHeight** and **midshipBreadth** parameters set the maximum height and breadth of the ship. These parameters also affect the shapes and sizes of the foreship and aft sections.

The next parameter sections are dedicated to defining values for the bow and stern sections of the ship. These sections specify values exclusively for their respective areas and do not influence other parts of the ship. It is important that these parameters are handled carefully and aligned cohesively with the overall ship design to ensure that the final product accurately reflects the intended design.

The last parameter section defines more specific parameters, the most important of which are **numPoints** and **DWL**. The **numPoints** parameter determines the accuracy of the hull design by defining the number of points generated for each curve. This number is also used to determine how many surface curves are generated for each section. The runtime of the program is heavily dependent on this parameter. Higher accuracy improves the results of numerical calculations, but increases the number of calculations needed. The **DWL** (Design Water Line) parameter designates the draught of the ship, and determines where the longitudinal axis is positioned. This means the waterline level is at Z = 0 in the three-dimensional figures.

It is important to note that any specific adjustments to the ship's structure, specifically its shape, should be made through the control points. This process is more detailed in Chapter (3.5.2).

The following table shows the different parameters and their intended use:

Table 1: Input paramaters

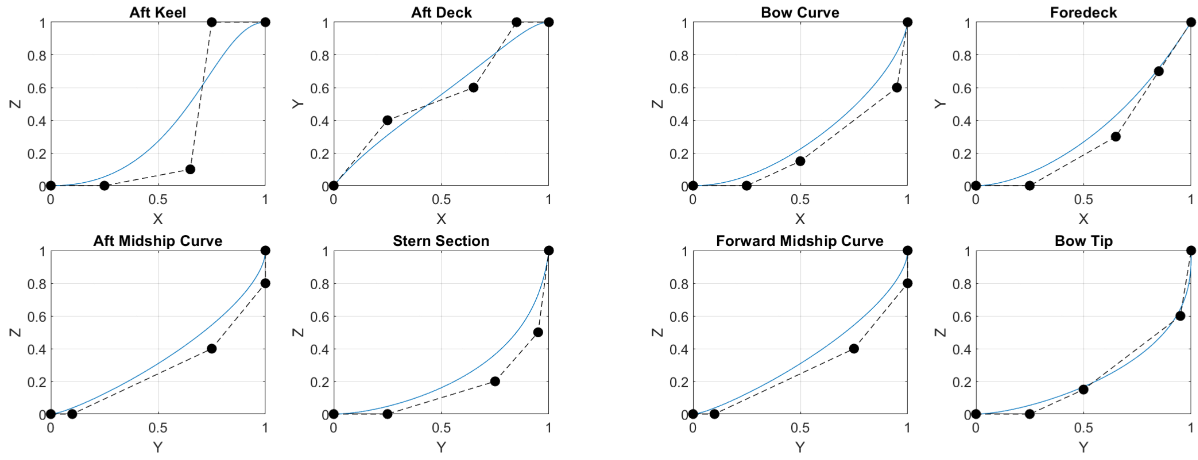
midshipBreadth	Breadth of the midship at maximum height
midshipLength	Total length of midship in the X-direction
midshipHeight	Total height of midship, measured from keel
foreshipLength	Total length of foreship in the X-direction
tipHeight	Height of the bow tip surface in the YZ-plane
tipBottom	Calculated bottom position of the bow tip surface in the YZ-plane
tipBreadth	Breadth of the bow tip surface in the YZ-plane
aftLength	Total length of the aft section in the X-direction
sternBreadth	Breadth of the aft-most part of the hull, measured from deck height
aftHeight	Height of the stern as measured from the lowest point to deck height
midAftHeightDiff	Calculated height difference between the bottom of the stern and the midship keel line
numPoints	Defines the accuracy and the number of points per curve
DWL	Design waterline
shipLength	Calculated total length of the ship
Cm	Desired midship coefficient
adjust	Decides whether to adjust for the midship coefficient, Yes (1) or No (0)

3.5.2 Bézier curves and control points

As mentioned in (3.3), the structure and shape of the ship are defined by a set of Bézier curves. These curves are controlled by a set of control points that can be adjusted to change the form of the hull. The Bézier curves are two-dimensional and dimensionless, meaning their lengths lie in the [0 1] domain. The Bézier curves provided in the program contain five control points each, but more may be added to increase the accuracy of adjustments. These points can be adjusted between the values of 0 and 1 in both the horizontal and vertical directions.

The process starts with defining a set of five control points, these are assigned as XY or YZ coordinates, depending on the plane in which the curve lies. When this is done, the parameter **numPoints** determines the number of curve points to generate and the program creates the desired Bézier curve. All the control points are adjusted manually, unless the midship coefficient feature is used. If so, the control points for the midship section are automatically updated to satisfy the **Cm** requirement.

The following figures demonstrate the different Bézier curves for the foreship and aft sections. For the remainder of the report, these curves will be referenced as the main curves of the hull. Two variants of the midship section curve are included as part of the front and aft sections.



(a) Aft Bézier curves.

(b) Foreship Bézier curves.

Figure 7: Bézier curves for the foreship and aft sections.

3.5.3 Linearization

To prepare the main curves for surface generation in a later stage, the longitudinal curves defining the foreship and aft sections are linearized in the X-direction. This ensures that each point along the longitudinal curves has corresponding X-values. It is done by generating a set of linearly spaced X-values and performing linear interpolation on the Y or Z-values. Whether it is Y or Z-values that are interpolated is determined by the plane in which the curve lies. As the lengths of the curves are dimensionless at this stage, the X-values are spaced in the [0 1] domain. The number of X-values, and the number of points per curve, is determined by the **numPoints** parameter. MATLAB handles linear interpolation with a built-in function, which employs the following formula:

$$y = y_1 + \frac{(x - x_1) \cdot (y_2 - y_1)}{(x_2 - x_1)} \tag{3}$$

Where:

- x_1, y_1: Coordinates of the first known data point.
- x_2, y_2: Coordinates of the second known data point.
- x: Query point to interpolate the value of y. The number of query points is determined by **numPoints**.

After a new set of linearized curve points are created, they are drawn over the old curve to verify that the shapes are identical. Using the bow curve as an example, this can be seen in Figure 8:

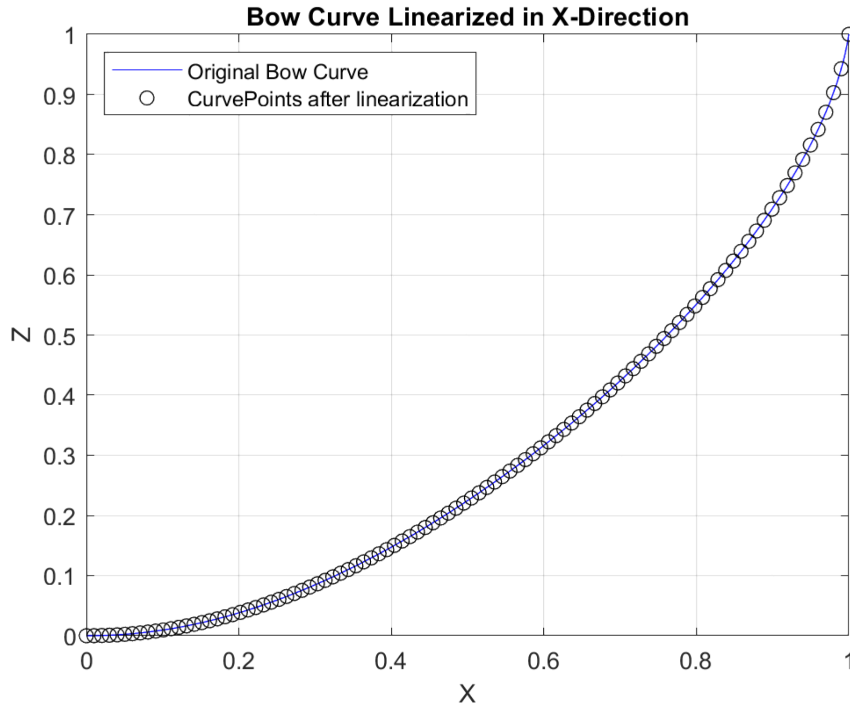


Figure 8: Bow curve after linearization.

3.5.4 Translation

In the previous stages, the main curves are two-dimensional and dimensionless. Therefore, they need to be translated and their curve points converted to three dimensions by employing the ship dimension parameters. For each curve, the curve points are manipulated to fit the curve's plane. The curves are then translated in a third direction to their positions in three-dimensional space. For example, the midship section curve is manipulated in the YZ-plane to fit with the **midshipBreadth** and **midshipHeight** parameters. The curve is then translated to its position along the longitudinal axis by adding the appropriate X-positions to the curve's coordinates.

Foreship curves
The bow curve is fit to the XZ-plane by translating the curve points in the X and Z-directions. The curve is then translated to three dimensions by concatenating the Y-positions (which are all zeros) into the curve points matrix:

$$\text{translatedBowX} = \mathbf{foreshipLength} \times \text{bowX} + \mathbf{midshipLength} + \mathbf{aftLength}$$

$$\text{translatedBowZ} = \text{bowZ} \times \mathbf{tipBottom} - \mathbf{DWL}$$

$$\text{translatedBow} = [\text{translatedBowX}, \text{bowYposition}, \text{translatedBowZ}]$$

The foredeck curve is translated into three dimensions by fitting it to the XY-plane, and translating it in the Z-direction:

$$\text{translatedForedeckX} = \mathbf{foreshipLength} \times \text{foredeckX} + \mathbf{midshipLength} + \mathbf{aftLength}$$

$$\text{translatedForedeckY} = \left(-\frac{\mathbf{midshipBreadth}}{2} + \frac{\mathbf{tipBreadth}}{2}\right) \times \text{foredeckY} + \frac{\mathbf{midshipBreadth}}{2}$$

$$\text{translatedForedeck} = [\text{translatedForedeckX}, \text{translatedForedeckY}, \mathbf{midshipHeight}]$$

The bow tip curve is made to fit the YZ-plane, and translating it in the X-direction:

$$\text{translatedTipY} = \frac{\textbf{tipBreadth}}{2} \times \text{tipY}$$

$$\text{translatedTipZ} = \textbf{tipHeight} \times \text{tipZ} - \textbf{DWL} + \textbf{tipBottom}$$

$$\text{translatedTip} = [\textbf{shipLength}, \text{translatedTipY}, \text{translatedTipZ}]$$

Aft curves
The aft keel line is fit to the XZ-plane, then converted to three dimensions with the Y-positions (which are zeros):

$$\text{translatedKeelX} = \textbf{aftLength} \times \text{aftKeelX}$$

$$\text{translatedKeelZ} = -\textbf{midAftHeightDiff} \times \text{aftKeelZ} + \textbf{midAftHeightDiff} - \textbf{DWL}$$

$$\text{translatedKeel} = [\text{translatedKeelX}, \text{keelYposition}, \text{translatedKeelZ}]$$

The aft deck curve is fit to the XY-plane, then converted to three dimensions by translating it in the Z-direction:

$$\text{translatedDeckX} = \textbf{aftLength} \times \text{aftDeckX}$$

$$\text{translatedDeckY} = \left(\frac{\textbf{midshipBreadth}}{2} - \frac{\textbf{sternBreadth}}{2}\right) \times \text{aftDeckY} + \frac{\textbf{sternBreadth}}{2}$$

$$\text{translatedDeck} = [\text{translatedDeckX}, \text{translatedDeckY}, \textbf{midshipHeight}]$$

The stern section curve is fit to the YZ-plane, then converted to three dimensions with the X-Positions (which are zeros):

$$\text{translatedSternY} = \frac{\textbf{sternBreadth}}{2} \times \text{sternY}$$

$$\text{translatedSternZ} = \textbf{aftHeight} \times \text{sternZ} + \textbf{midAftHeightDiff} - \textbf{DWL}$$

$$\text{translatedStern} = [\text{sternXposition}, \text{translatedSternY}, \text{translatedSternZ}]$$

Midship curves
The midship section curve is fit to the YZ-plane, then converted to three dimensions by translating it in the X-direction. The curve is translated twice, once for the forward midship and once for the aft midship:

$$\text{translatedMidSectionY} = \frac{\textbf{midshipBreadth}}{2} \times \text{midSectionY}$$

$$\text{translatedMidSectionZ} = \text{midshipHeight} \times \text{midSectionZ} - \text{DWL}$$

$$\text{aftMidLength} = \textbf{aftLength} + \textbf{midshipLength}$$

$$\text{translatedMidSectionFront} = [\text{aftMidLength}, \text{translatedMidSectionY}, \text{translatedMidSectionZ}]$$

$$\text{translatedMidSectionAft} = [\textbf{aftLength}, \text{translatedMidSectionY}, \text{translatedMidSectionZ}]$$

Once all the main curves are translated, a three-dimensional figure of the hull skeleton is generated:

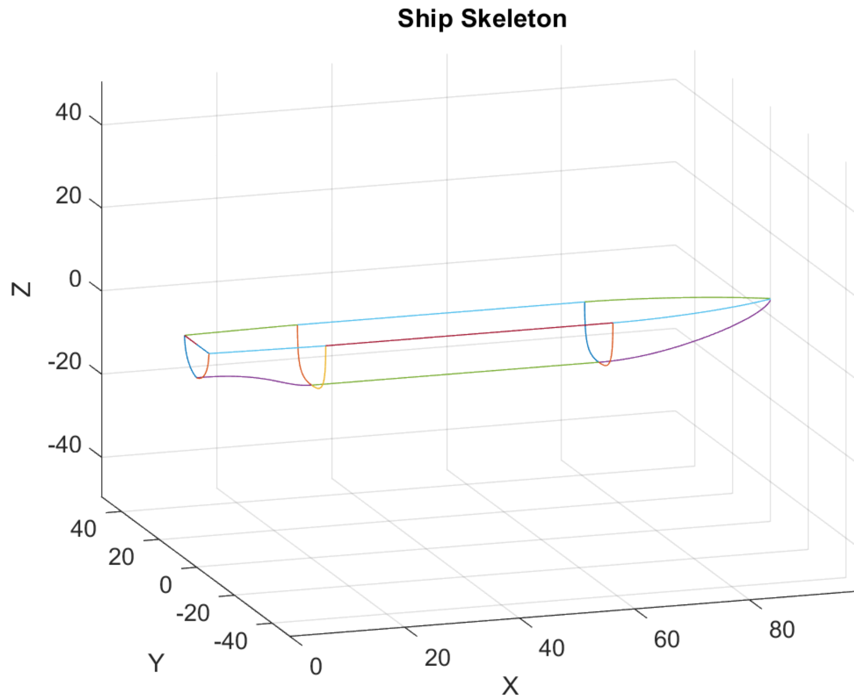


Figure 9: 3D-figure with all translated main curves defining the ship skeleton.

3.5.5 Midship Coefficient

If the **adjust** parameter is set to 1, Floke will adjust the midship section curve until the midship coefficient is equal to the desired value, as defined in the **Cm** parameter. The midship coefficient is defined as the ratio between the submerged area of the midship section and the area of a rectangle of the same depth and width [11]:

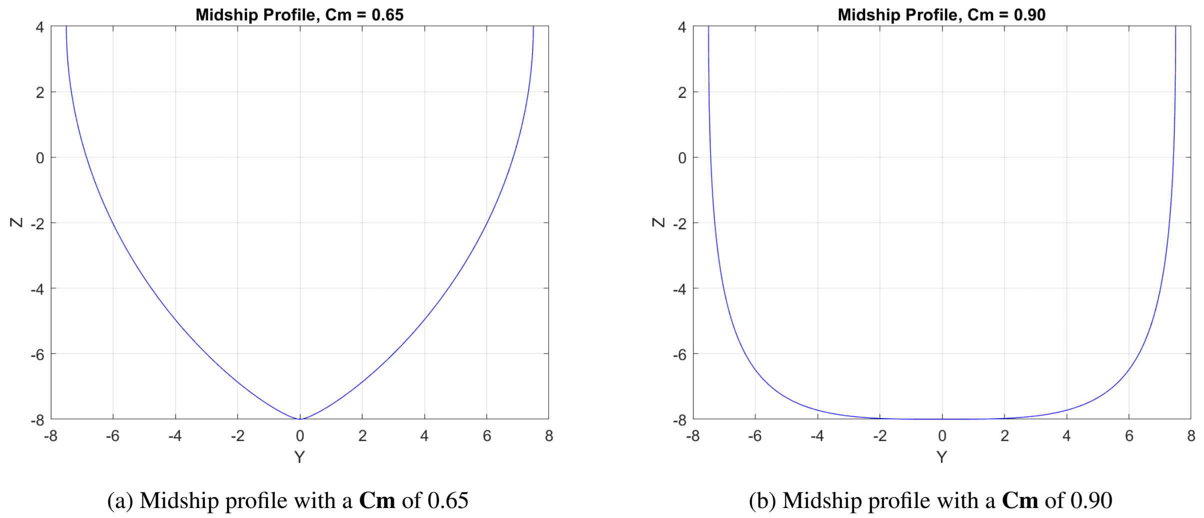
$$C_m = \frac{A_m}{B \times T} \tag{4}$$

where:

- A_m is the submerged cross-sectional area of the midship section.
- B is the breadth of the midship at the waterline.
- T is the draught of the midship.

The adjustment is performed by moving the midship section control points in small increments, generating a new midship section curve, translating it, and checking whether the updated **Cm**-value is correct. If not, the process starts over. Floke does this by determining whether the desired **Cm** is larger or smaller than the calculated value and adjusting the control points accordingly. When the calculated **Cm**-value is equal to the desired value (within a tolerance of 0.01), the process stops. The process will also stop in the case where the control points reach the limits of their domain. The control points are adjusted in the vertical direction first. Once the points reach their limits in the vertical direction, they are adjusted in the horizontal direction. To ensure that the start and end points of the midship section curve remain in the same places, and to enforce tangency at the curve intersections, not all of the curve's control points are adjusted. For a more detailed description of the function code, see **APPENDIX B**.

The following Figures, 10a and 10b, illustrate the difference in the shape of the midship section curve with two different **Cm**-values. An increase in **Cm** will expand the hull, while a decrease contracts the hull, resulting in a slimmer profile.



(a) Midship profile with a **Cm** of 0.65

(b) Midship profile with a **Cm** of 0.90

Figure 10: Midship profile for different midship coefficient values

3.5.6 Interpolation

The surface of the hull is generated by interpolating surface curves between the lateral main curves using linear interpolation. Since the hull is divided into three sections, three interpolation operations are performed. The foreship surface curves are generated first, followed by the midship and aft sections. Floke handles the interpolation in the positive X-direction. This means the curves are generated from the rear to the forward reference curves. The foreship curves are interpolated from the midship section front to the tip curves. For the midship, the surface curves are interpolated from the aft to front midship curves. The aft surface curves are interpolated from the stern section to the aft midship section. Interpolation for this stage employs the same function and equation as mentioned in (3.5.3), Equation 3. The number of query points used for interpolation, and therefore also the number of curves generated, is dependent on the accuracy parameter, **numPoints**. An example of the interpolation operation can be observed for the foreship section in Figure 11.

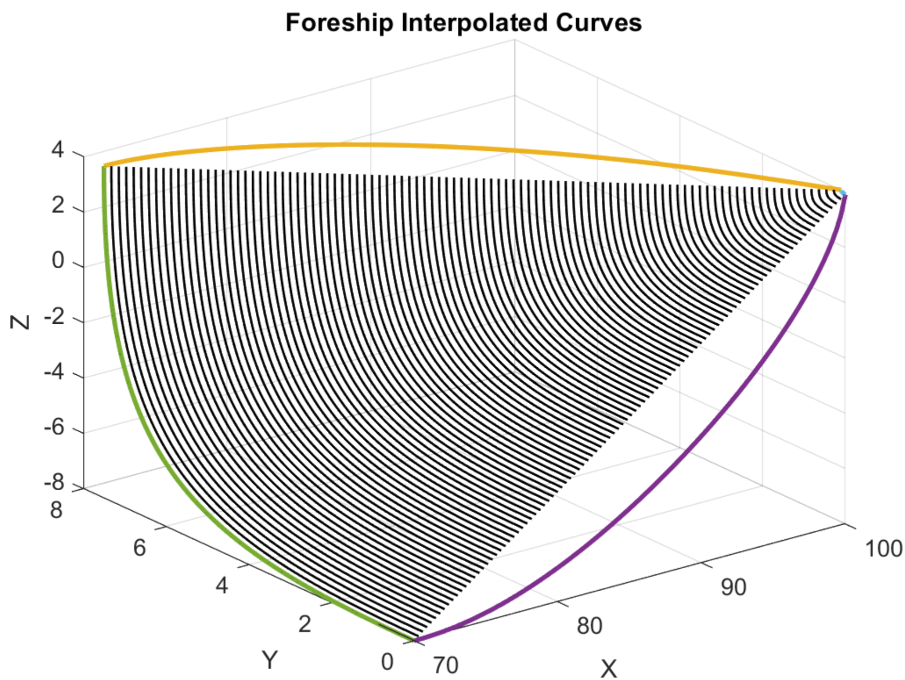


Figure 11: Interpolated surface curves for the foreship section.

3.5.7 Scaling

As the interpolation stage does not account for the longitudinal curves, a scaling operation is needed. This ensures that the start and end points of the interpolated curves coincide with the curves defining the bow, foredeck, aft deck, and aft keel. The foreship is handled first, followed by the aft section. As the midship is defined by two identical curves, one front and one aft, the surface curves already coincide with the keel and deck lines. Therefore, there is no need to perform the scaling operation on the midship surface curves.

The scaling operation is performed in two stages. First, the Z-values of the surface curves are scaled in the Z-direction to coincide with the bow and aft keel curves. Second, the Y-values are scaled in the Y-direction to coincide with the foredeck and aft deck curves. The scaling operation is performed by first defining a new start point for each curve. Using the foreship as an example, the new start points in the Z-direction lie along the bow curve. In the Y-direction, the new start points lie along the foredeck curve. For each surface curve, a scaling factor is calculated using the following formula:

$$\text{scalingFactor} = \frac{\text{endValue} - \text{newStart}}{\text{endValue} - \text{originalStart}} \tag{5}$$

Each point along the surface curve is then scaled using this formula:

$$\text{scaledValue} = \text{currentValue} \times \text{scalingFactor} + (1 - \text{scalingFactor}) \times \text{endValue} \tag{6}$$

The scaling operation is performed for the foreship first, beginning with the Z-direction, as seen in Figure 12:

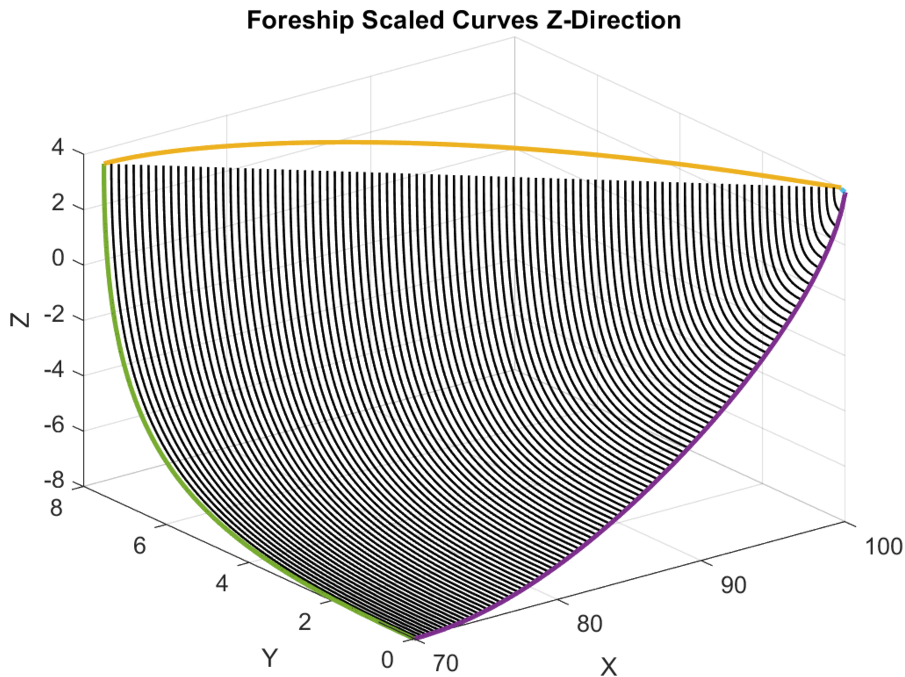
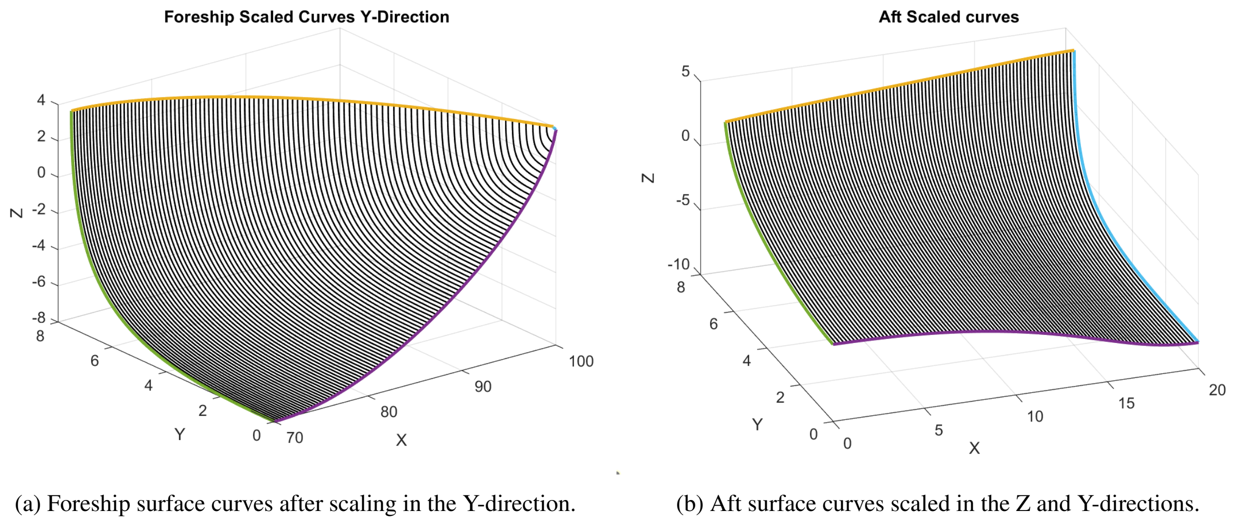


Figure 12: Foreship surface curves scaled in the Z-direction.

The curves are then scaled in the Y-direction, as seen in Figure 16a. The surface curves for the aft section are scaled using the same operations, as seen in Figure 16b.



(a) Foreship surface curves after scaling in the Y-direction.

(b) Aft surface curves scaled in the Z and Y-directions.

Figure 13: Scaled curves for the foreship and aft sections.

3.5.8 Surface generation

In this stage, a three-dimensional surface model of the ship is generated using the surface curves and their mirrors. A figure is created, with different colors above and below the waterline. The purpose of this model is to inspect and verify that the surface of the hull is generated correctly before exporting the design.

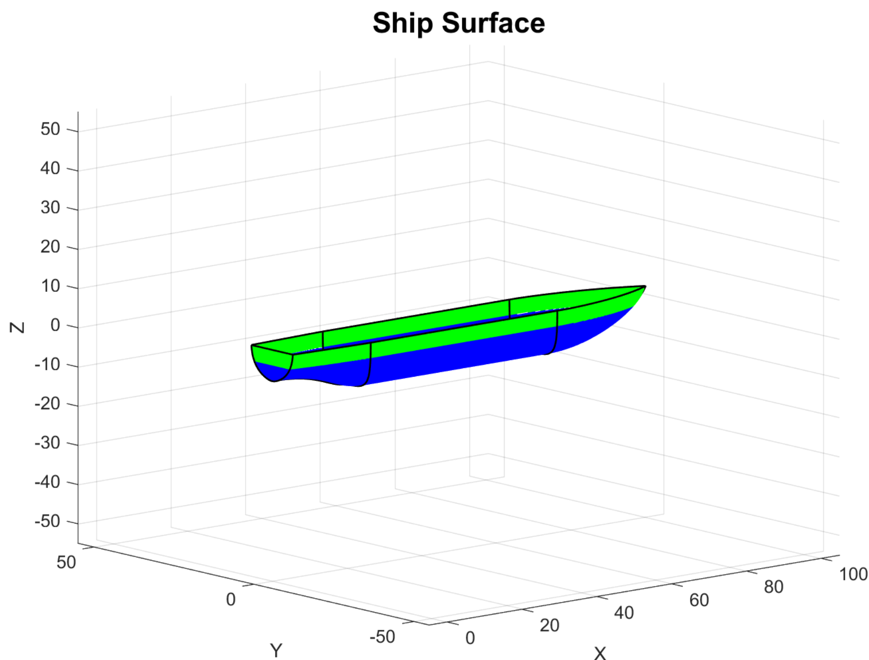


Figure 14: Three dimensional model of the hull surface.

3.6 Export of design

The hull surface can be exported through two different methods, which can be used within three-dimensional modeling software. The first method involves the export of the surface, while the other exports the main curves themselves.

Once the surface of the hull is clearly defined, it can be exported as a stereolithographic file (.stl). An STL-file stores the hull surface as a set of triangular polygons called facets, which are linked together and oriented to define the boundary of the hull. The accuracy of the exported surface is heavily dependent on the number of facets generated in the export process, which is determined by the **numPoints** parameter. Higher accuracy leads to a smoother surface as the facet sizes get smaller, but can increase the runtime of the program significantly. For the purposes of this program, the greatest characteristic of an STL-file is that it can be directly imported into CAD-software. Floke stores the hull into eight different files to eliminate the risk of unexpected interactions between surfaces at the intersections:

- Foreship surfaces:
  - foreSurface
  - foreDeckSurface
  - tipSurface
- Midship surfaces:
  - midSurface
  - midDeckSurface
- Aft surfaces:
  - aftSurface
  - aftDeckSurface
  - sternSurface

Should there be a need to contain the entire hull surface as a single file, the recommended procedure is to import each surface individually, then create a copy of the entire hull and save it as a separate file.

A different method involves exporting each curve as individual .txt files. These files can subsequently be renamed with a .pts extension and imported into Creo Parametric or similar modeling software. Using this approach, only the curves are exported, not the ship's surface. Therefore, once all the curves are imported into the software, the surface must be constructed within the modeling program.

4 Results

This section demonstrates the capabilities of Floke by performing a step-by-step generation of two different hull designs. In addition, the hulls designed are also simulated in Star CCM+ to get an overview of the difference between hull shapes in real-life conditions. This demonstration includes all the parameters defined at the beginning of the code, including the midship coefficient.

For this demonstration, two hulls are designed. The reason is to exhibit the ability of the program to create different types of ship hulls and therefore, different results when simulated. The ships are created using different parameters, but have the same original control points.

The results are divided into the following sections:

1. **Input parameters**: Introduces the parameters for each of the ships.
2. **Main curves**: Representation of the Bézier curves that define the hull structures.
3. **Ship models**: Representation of the surfaces of both ships.
4. **CFD analysis**: Displayed results from the CFD simulation.

4.1 Input parameters

The parameters for both hulls are displayed in Table 2. The parameters chosen for both ships represent two separate hulls to demonstrate Floke's capability of generating different hull designs.

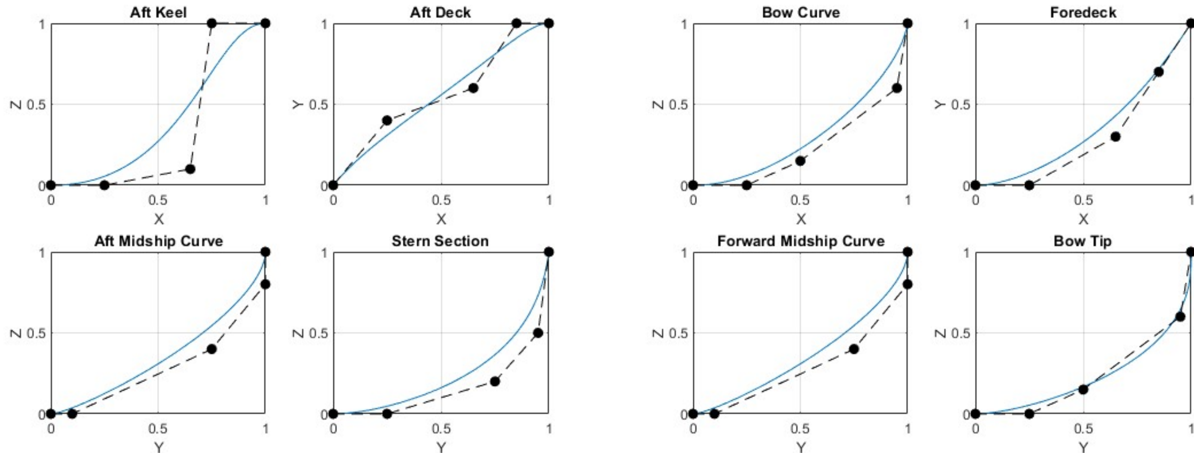
Table 2: Comparison of parameters (values in meters).

Parameter	First ship [m]	Second ship [m]
MidshipBreadth	15	30
MidshipLength	50	50
MidshipHeight	12	20
ForeshipLength	30	30
AftLength	20	20
SternBreadth	13	30
AftHeight	8	18
Total length	100	100
DWL	6	8
Cm	0.59	0.90

4.2 Main curves

The main curves for both hulls are generated from the same set of control points, meaning the differences in shape are only due to changes made in the parameter section.

Represented in Figure 15 are the results of the main curves for both ships:

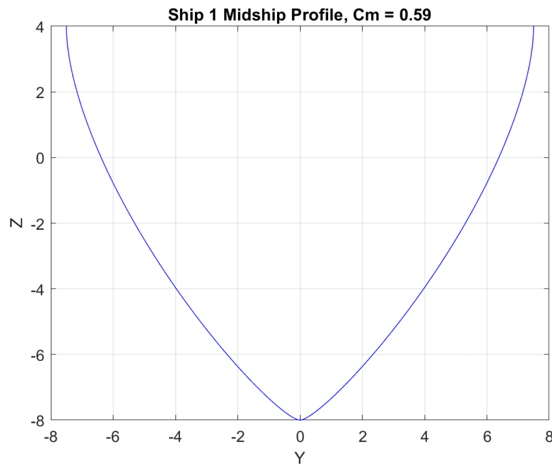


(a) Aft curves for both ships.

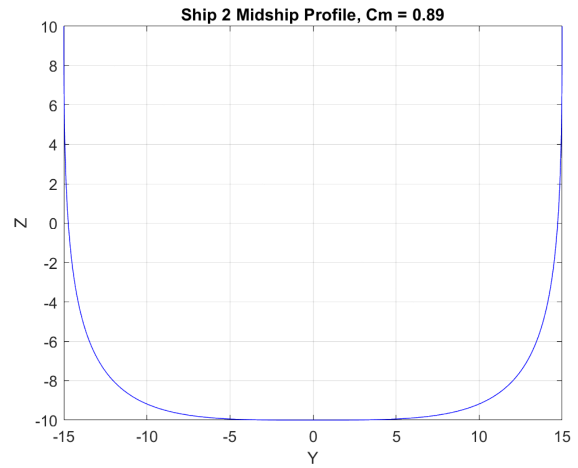
(b) Foreship curves for both ships.

Figure 15: Main curves for both ships.

As the second ship adjusts for a different midship coefficient, the midship curves are altered compared to the first. These changes are reflected in Figure 16. Though the input parameter is set to 0.90, the generated curve results in a midship coefficient of 0.89 due to tolerances built into the program code.



(a) Midship section curve for the first ship.

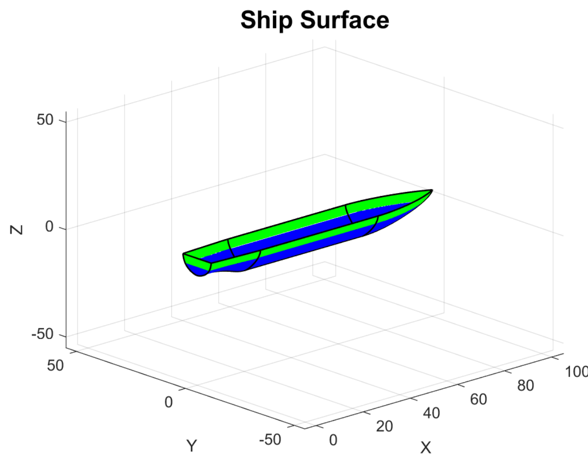


(b) Midship section curve for the second ship.

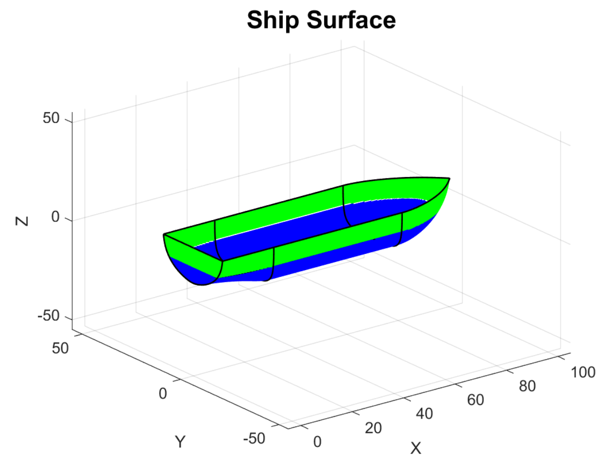
Figure 16: Midship section curves for different values of **Cm**.

4.3 Ship models

The final result of the surface generated for the ships is displayed in Figure 17a and 17b.



(a) Surface for the first ship.



(b) Surface for the second ship.

Figure 17: Surface figures for both ships.

4.4 CFD analysis

A comparison of simulation results between the two ship hulls is displayed in this section. Both hulls are simulated in calm water at a speed of 11 knots, with two degrees of freedom:

- **Heave**: Ship motion along the vertical axis, Z.
- **Pitch**: Ship rotation about the lateral axis, Y.

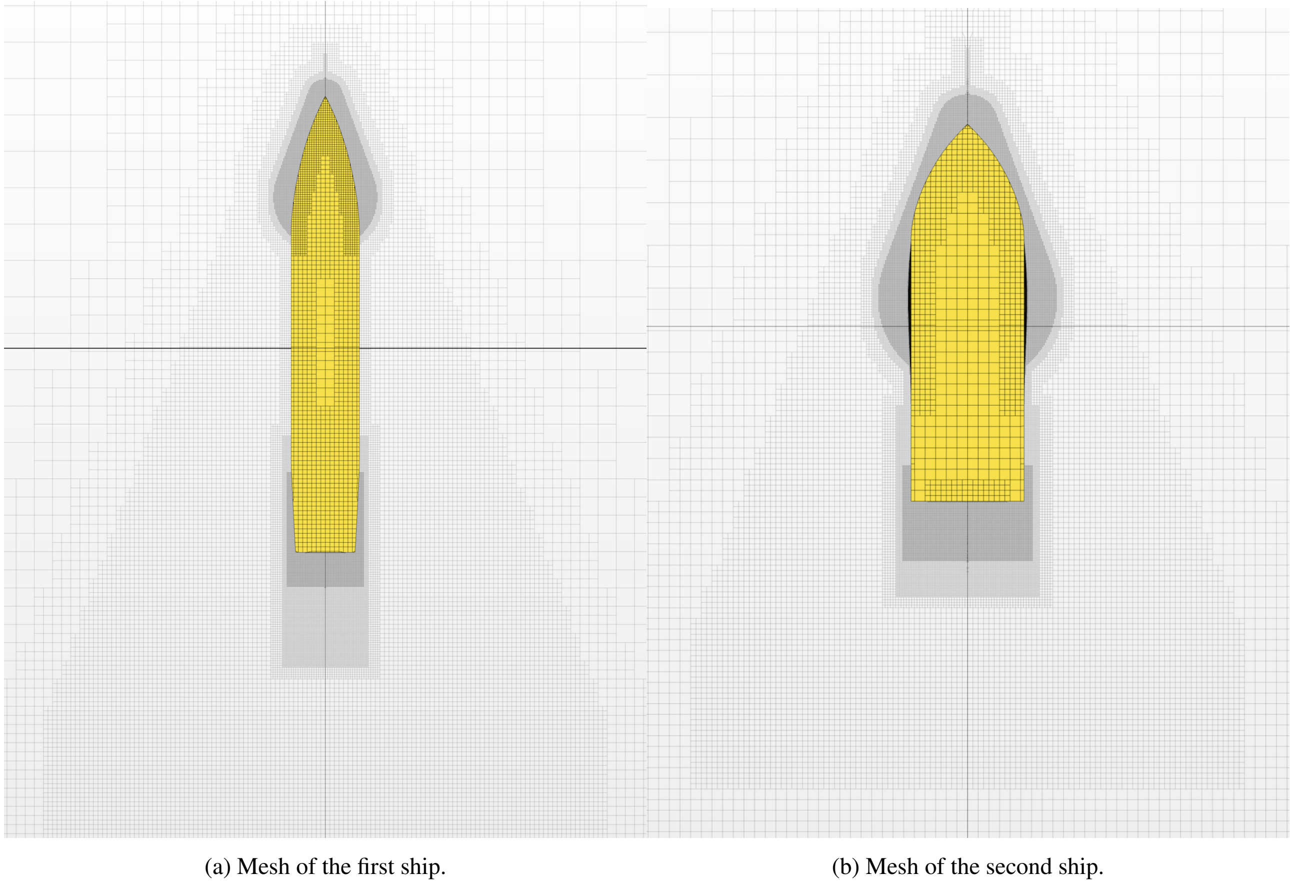
The results will be shown in four different parts: Mesh and cell count, y^+ values, wave pattern, surface elevation, and their resistance.

4.4.1 Mesh and cell count

The result from the mesh and cell count is displayed in Figure 18a and 18b. The mesh breaks down the area being studied into a grid for analysis, and cells are the individual blocks within this grid. It is important to emphasize that a finer mesh with more cells can give more detailed and accurate results, but it also requires more computational power.

Table 3: Table of cell counts

Cell Count 1	Cell Count 2
3.24073e+6	3.07927e+06



(a) Mesh of the first ship. (b) Mesh of the second ship.

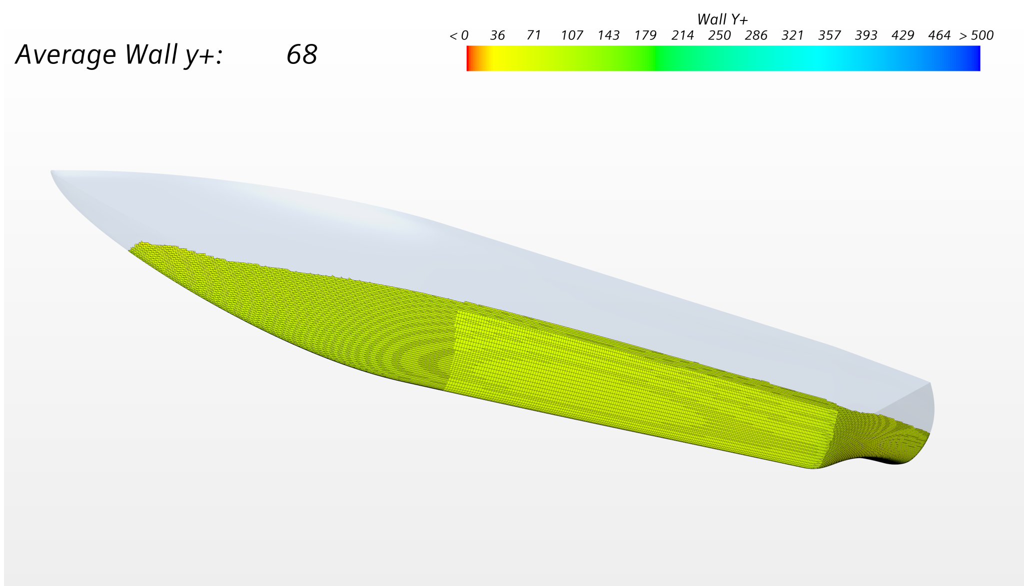
Figure 18: Top view of the mesh of both ships.

4.4.2 y^+ values

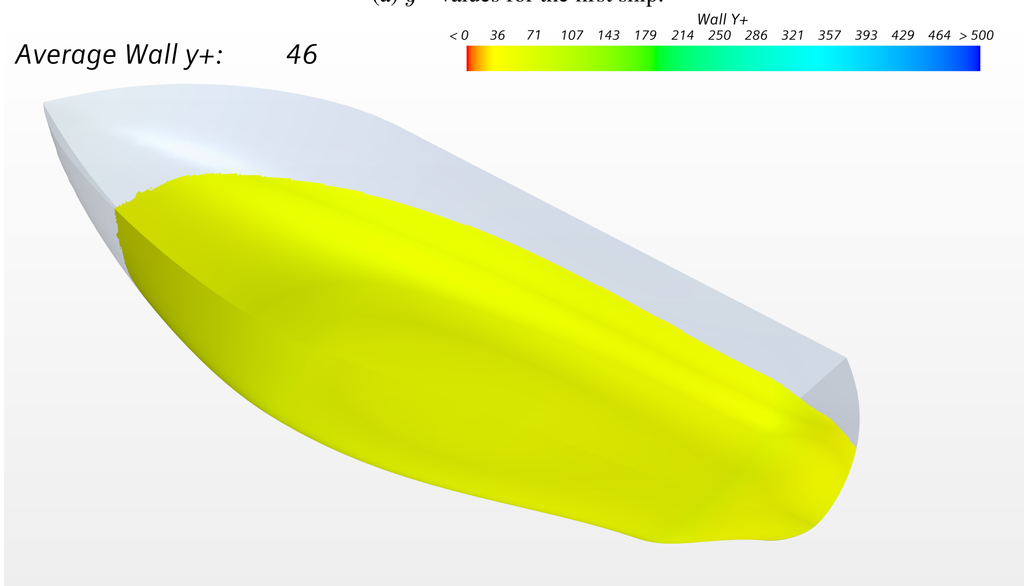
The y^+ values represent a non-dimensional distance from the wall to the first cell center in the mesh [12]. Values between 30 and 300 are effective for capturing the majority of the turbulent boundary layer using standard wall functions. These functions help predict how the water flows around the hull without needing an extremely detailed mesh. The color gradient displayed on the top-right side of the figures displays the y^+ values. The y^+ given by the CFD analysis for both ships are displayed in Figure 19.

Table 4: Table of y^+ values for both ships.

y^+ ship 1	y^+ ship 2
68	46



(a) y^+ values for the first ship.



(b) y^+ values for the second ship.

Figure 19: y^+ values for both ships.

4.4.3 Wave pattern and surface elevation

The surface elevation, as shown in Figure 20, illustrates how the water flows and reacts around the ship hull. This is key to understanding how hydrodynamic forces affect the ship's stability and efficiency. The gradient colors displayed on the right side of the figures explain the elevation of the water around the hull.

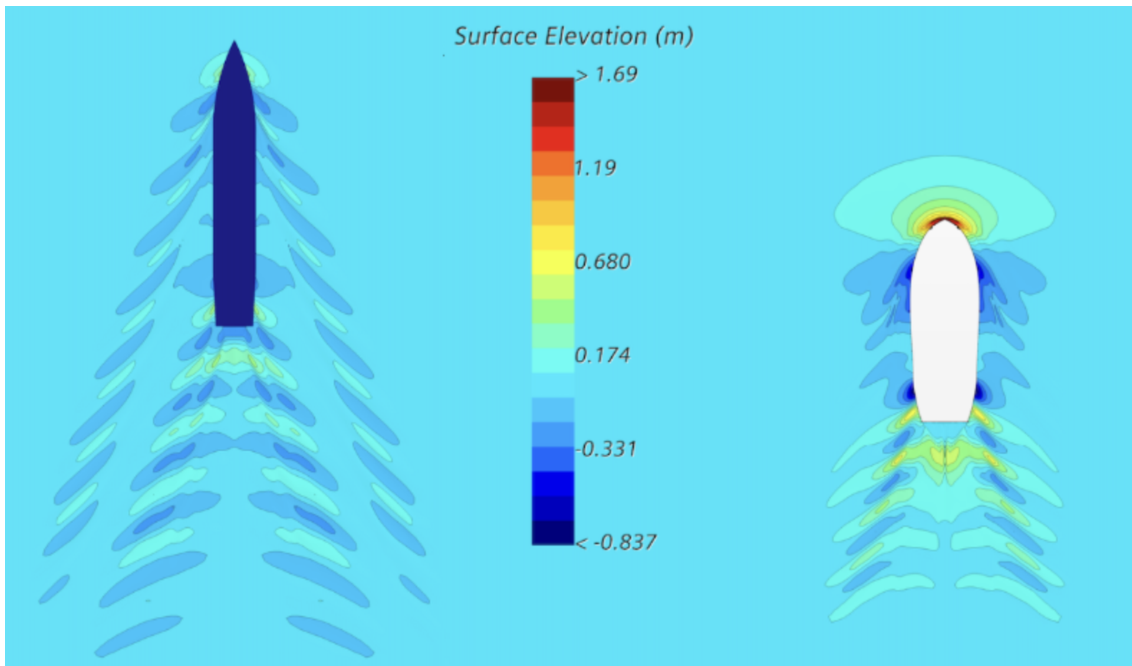
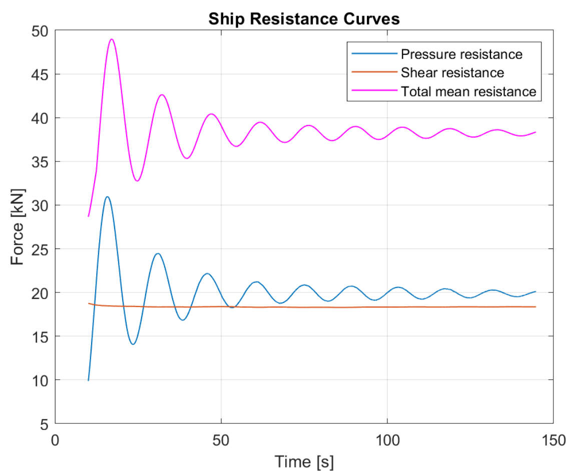


Figure 20: Top view of the wave pattern and surface elevation for both ships.

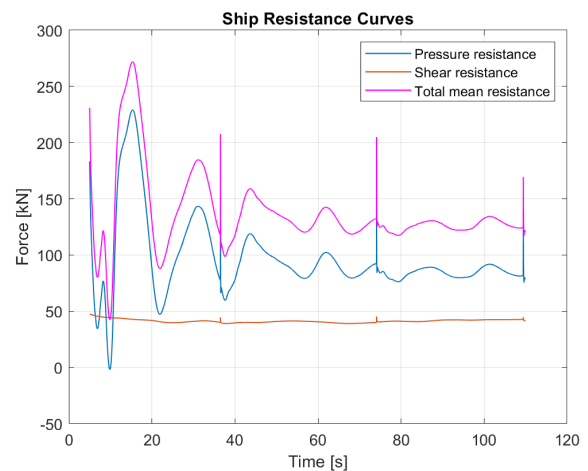
4.4.4 Resistance

The following figures illustrate the simulated resistance forces [13] applied to the ship hulls. These curves consist of:

- **Pressure resistance**: As the ship moves through the water, pressure resistance is generated as a result of the displaced fluid. This displacement creates waves, which require energy to form. The pressure resistance arises from the pressure gradients associated with these waves.
- **Shear resistance**: Also known as viscous resistance. These forces are generated as a result of the interaction between the hull and water surfaces. The resistance arises due to the viscosity of water and the frictional forces acting on the hull surface.
- **Total mean resistance**: The sum of the pressure and shear resistances. Represents the overall resistances experienced by the hull as it travels through water.



(a) Resistance curves for the first ship.



(b) Resistance curves for the second ship.

Figure 21: Resistance curves for both ships.

5 Discussion

This chapter details the results of the simulations performed in Chapter (4), and continues with a discussion of the project scope and the adjustments made to the original. Finally, the program itself is discussed, beginning with Floke's limitations, development challenges, and further potential.

5.1 Review of results

As demonstrated in Chapter (4), Floke is able to quickly generate different hull designs by simply changing the input parameters, and the design may then be used for further analysis with CFD software.

The designs can be altered further by adjusting the control points of the main curves, resulting in significant differences in the hull shape. By selecting the option to adjust for the midship coefficient, the shape is allowed to be adjusted without having to manually change the control points, resulting in an easier and faster user experience. It is important to note that the adjustment of the midship curves will result in a midship coefficient unequal to the desired value as defined in the parameters section. This is due to tolerances built into Floke's code. This tolerance may be lowered to produce more accurate results, but increases the risk of creating an endless loop. See **APPENDIX B** for more detailed descriptions.

The values obtained from the CFD analysis in Chapter (4) are not the main areas of interest, rather the possibility to import the ship designs and simulate them. As shown in the results, Star CCM+ is able to simulate the imported ships under real-life conditions. However, because of the limitations of the testing hardware, some simplifications are made. Only two degrees of freedom are included in the analysis: heave and pitch. These changes are made to decrease the time needed to run the simulations, as the analysis is only meant to serve as a demonstration of Floke's capability of exporting the designs.

The mesh and cell counts are different for both ships. The first ship, characterized by a thinner hull, has a cell count of approximately 3.08 million cells. The second ship, which has a wider hull, has a slightly higher cell count of approximately 3.24 million cells. Despite having the same length, the difference in cell counts is because of the varying hull geometries, which affect the complexity of the mesh. However, the difference in the geometry of both ships does not affect the cell count to a large extent.

The y^+ values show differences in the boundary layer for the two ships. The first ship has an average wall y^+ of 68, showing a less detailed mesh near the hull surface. The second ship has a lower average wall y^+ of 46, indicating a more detailed mesh near the hull surface. These differences reflect how the hull shapes affect the mesh and simulation detail.

The wave pattern and surface elevation results demonstrate differences between the two ship designs. The first hull creates spread-out waves with a maximum surface elevation of about 0.68 meters at the wake. However, the second ship generates a higher and slimmer wave pattern, with surface elevations reaching higher than 1.69 meters. As expected, the figure indicates that the wider hull displaces more water, generating higher waves. The figure also illustrates a large displacement of water at the bow, meaning the second ship pushes water forward instead of to the sides due to its significantly wider shape.

The resistance curves for the two ship designs show clear differences. The first ship has a total mean resistance of approximately 38 kN, with pressure resistance being the dominant contributor. The second ship has a much higher total mean resistance of about 125 kN, with pressure resistance also being dominant. This demonstrates that the wider ship experiences much higher resistance, mainly due to pressure resistance, which is expected.

5.2 Scope of project

The fundamental purpose of the project is to bridge the gap between the creation of a ship design and simulation. The use of simulation software is an expensive and tedious task, and the design may not be changed during the operation. Floke is designed to deliver a ship parametrization algorithm as a base for a larger hull optimization program. Once the optimization process is developed, Floke will be able to provide a better representation of a base hull design. These designs may be altered and optimized for a wide range of configurations, to serve as foundations for further simulation. Improved hull designs through optimization will demand fewer simulation runs, which is beneficial for reducing cost and time consumption.

The initial idea behind the creation of Floke is to create a functional program with the ability to generate hull designs, as well as supporting the simulation process through export of compatible formats. The hull consists of three independent

parts, the foreship, midship, and aft sections. In each section, the shape is determined by Bézier curves through the manipulation of each curve's control points. Furthermore, Floke produces files representing the hull surfaces, which can be imported into Creo Parametric or similar modeling software. Floke can also generate files for the curves themselves, which can then be imported into appropriate modeling software and employed to draw the hull surface.

Adjustments to the scope of the project are mostly made due to limitations of the programming software and time restrictions. However, some of these changes are also made to discard unsuitable functionality in favor of better solutions, as will be discussed further.

- **Floke is restricted in its usability.** By producing control points based on a set of input parameters such as length, width, draught, free board and dead-rise angle, and employing these to determine the shape of the ship, the ease-of-use would improve significantly. If the shape is not satisfactory, the curves could then be changed by manually manipulating the control points. These ideas are replaced in favor of the adjustment of the ship's form coefficients instead. At present, Floke is designed for a more manual approach, employing tasks for adjustment of the midship coefficient to improve usability. Further improvement could be made through the implementation of adjustments for other ship form coefficients such as the prismatic and block coefficients.
- **Floke produces eight different files** instead of one file containing the entire hull surface. The files produced in the final product contain the individual section surfaces to limit the risk of generating discontinuous or overlapping surface meshes in the models. The curves themselves may also be imported individually. Though this is a more tedious approach, the models generated through this method are likely easier to reformat.
- **The program does not contain hydrostatic calculations.** The program can be further improved by incorporating functionality in terms of hydrostatic performance. Characteristics such as waterline area, submerged volume, center of buoyancy, and center of mass can be implemented to determine the stability of the given design.
- **The final product does not contain complex structures,** such as a bulbous bow. However, the Bézier curve defining the aft keel is able to hollow out the hedge, but it cannot account for the shafts, propellers, and rudders.
- **The program is not able to automatically update changes** during the simulation. Despite this, it enables the developer to make fast design changes to the hull within Floke.

5.3 Limitations

Currently, Floke is a program best managed by engineers and naval architects who possess a deep understanding of the ship design process. If not handled with care, the program might produce unpredictable results in terms of the hull design, meaning sufficient programming ability and a high degree of expertise is needed when working with Floke.

For the present version, the program has several limitations:

- **Floke is limited in its ability to generate highly detailed designs.** This limitation stems from the ship's hull being constructed from a series of Bézier curves, as discussed in Chapter (3.4). These curves define the ship's form, but their limited number restricts the variety of design options available.
- **The program is not bug-free.** Most software packages are tested rigorously to ensure the programs function as intended. This process is not within the scope of the project, and Floke is not tested in the same manner. This means there may be undetected faults within the program itself, which may cause unpredictable issues. However, Floke is not a highly complex program, which means the program can be debugged and improved without extensive programming skills. For software documentation, see **APPENDIX B**.
- **Floke is limited to the functionality built into MATLAB.** The functionality potential of Floke is limited by the restrictions imposed by the use of MATLAB. Unlike open-source programming languages like Python, the functionality within MATLAB is limited to the tools provided by MathWorks.

5.4 Challenges

This subchapter discusses the challenges related to the development of Floke, which consist of issues related to the accuracy parameter and the export process.

5.4.1 Accuracy

Some of the issues during the development of the program are related to accuracy. The accuracy of the program is defined by the **numPoints** parameter, which directly affects the density of points on each curve. It also determines the number of surface curves generated for each section of the hull. Increasing the number of points results in a finer resolution of the curve. However, it decreases the efficiency of the program, leading to longer running time and heavier files.

The accuracy of calculations for different sections of the ship can be significantly impacted by the number of points used to define each curve. If the resolution of these curves is insufficient, it can lead to substantial errors, causing the results to deviate from the intended design outcome. This issue is especially evident in the functions used to determine the midship coefficient, which are heavily dependent on the waterline level. At low resolutions, there may be a significant difference between the calculated and actual waterline breadth, due to the method used to find the waterline height. This issue affects the calculation of the submerged midship section area, which carries into the calculation of the midship coefficient. These issues are negated by increasing the accuracy parameter.

There is an evident issue with the visual representation within MATLAB. This can be seen in the dividing section between the waterline and the section above water. The method used to draw the waterline around the ship is done by determining all the Z-values that are closest to zero. The problem encountered is that if the number of points is too low, the Z-values will be further away from zero, leading to a noticeable gap between the waterline and the rest of the hull above water. This issue can be seen in Figure 22.

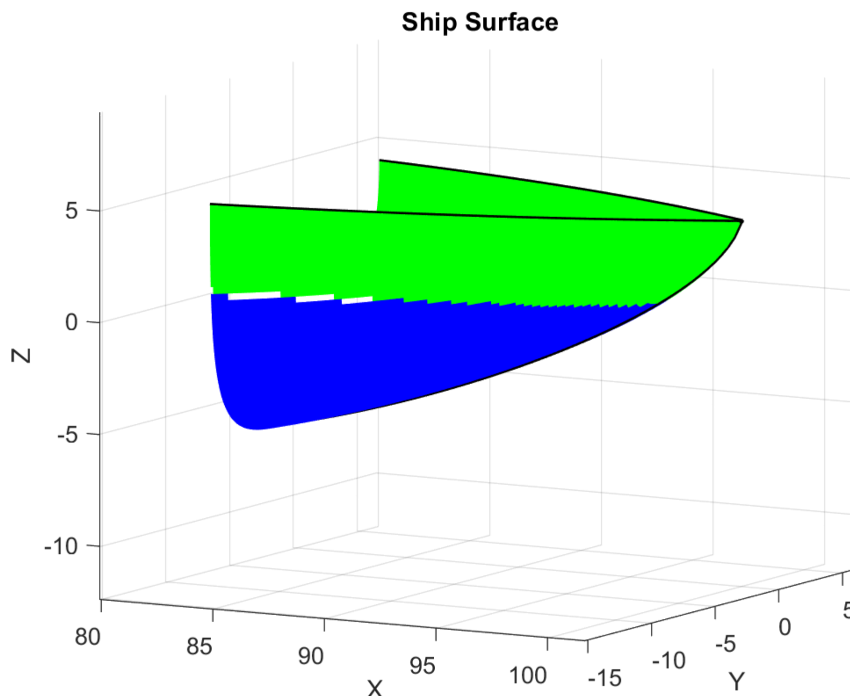


Figure 22: A zoomed in image of the bow.

The visual discrepancies observed within MATLAB are primarily due to the function used to generate the surface figure, which may struggle with a low number of points. These limitations are shown as visible gaps in the waterline area when viewed in MATLAB. However, it is important to note that these visual issues do not persist when the ship design is exported to CAD or simulation software. In these external applications, the design is rendered accurately, ensuring that the visual representation aligns with the intended design specifications.

5.4.2 Export

Regarding the export of the ship design, the first method detailed in Chapter (3.6) presents several challenges specifically related to exporting the STL file and the conversion to a solid body. An STL file must meet several conditions to properly represent a solid shape [14]:

- The mesh must be fully closed without any gaps
- Triangle normals must be consistently oriented outward
- The geometry must maintain integrity with accurately placed vertices

The requirements ensure the model is watertight, which is essential for applications like simulation. These conditions are not integrated within the program. As a result, the surface representation of the ship may become corrupted, preventing the CAD software from converting the file into a solid body. Currently, the only method capable of converting it into a solid shape is the second one described in Chapter (3.6). However, this method is more time-consuming.

5.5 Further potential

The scope of the project, as outlined in (5.2), presents opportunities for further enhancement. With this in mind, the following section will explore various upgrade suggestions, with the goal of creating a more refined version of the ship design program.

5.5.1 Improved functionality

- **The addition of more coefficients** can further optimize the ship design by enhancing accuracy, enabling customization, and facilitating more realistic designs. Some coefficients that can be integrated are the block and prismatic coefficients.
- **Integrating regulations and real-world data** can significantly enhance the optimization of ship design. Firstly, the program ensures that the hull design always complies with the specific regulations applicable to the type of ship being designed. Additionally, by leveraging real-world data, the program can automatically base its designs on proven models, ensuring that the final design is both practical and effective.
- **Real-time adjustments** can significantly improve the visualization of changes made to the hull. Instead of having to rerun the entire code after each modification, the program would be capable of instantly updating and reflecting changes. This enhances efficiency and allows for immediate feedback on design alterations.
- **The addition of a user-friendly interface** can significantly improve the ship design process. Rather than requiring users to manage the entire code and understand its complexities, the interface could present only the adjustable elements that directly contribute to the design of the ship. This approach not only enhances efficiency but also makes the design process more accessible to users without deep technical expertise.
- **Increasing the number of Bézier curves to define the ship** can enhance the variety of designs the program can create. Each Bézier curve is defined by its control points. Therefore, incorporating more curves increases the program's flexibility to alter and refine the ship's design.
- **The implementation of hydro-static computations** can further enhance the functionality of the program. The addition of this calculation could help the user find critical errors in the design before importing it into CAD or simulation software.
- **A wider range of design application** could be incorporated into Floke. The way the program works makes it easy to make new designs such as ferries, just by replacing the stern of the ship with another bow.

5.5.2 Programming language

The use of other programming languages can further improve the functionality of the program. While MATLAB is a great choice when it comes to mathematical computations due to its powerful built-in functionality, there are other programming languages that may offer significant advantages. Python offers a wide selection of packages for numerical calculation and optimization algorithms. Python also has a large and active community, which makes it easy to find support and resources. As computational efficiency is important when designing a program like Floke, a lower-level language such as C++ would offer a higher degree of customization in terms of memory management and performance. However, developing a program using C++ or other low-level languages is generally more difficult.

5.5.3 Conditions

At present, the code is best managed by skilled ship developers. However, integrating conditions into the ship design code could make the program accessible to users without specialized knowledge. By doing so, the program would impose constraints that shape the final design, ensuring it is realistic and viable.

Currently, the program may produce designs that are impractical for real-world application. Therefore, this implementation is essential to connect the gap between expert knowledge and user-friendly design capabilities, ensuring that all generated designs are both functional and compliant with real-world conditions.

5.5.4 Hull optimization

A hull optimization process would improve the hull design to comply with a set of desired characteristics such as speed, maneuverability, stability, and fuel consumption. Hull optimization is widely used within ship development to improve hull design [15]. This is performed using numerical algorithms that manipulate the hull shape and calculate the characteristics of the current iteration. The algorithm is run multiple times with different sets of parameters until the optimal combination is found for the desired characteristic.

Within Floke, the optimization process could search for the best solution for a desired characteristic by altering the control points of the main curves. These are the degrees of freedom (DOF) that the algorithm alters to obtain the best solution. If constraints are included, such as stability constraints, the optimizer will search for the best possible solution within the constraint boundaries. However, this process would benefit from the inclusion of more easily tuned parameters. Currently, Floke includes a process to adjust for the midship coefficient, but could easily be improved by implementing more form parameters such as the prismatic and block coefficients. The program would also need functionality to calculate the results of each adjustment, to verify whether the current iteration is improved with respect to the desired characteristic. Therefore, it is recommended that Floke is upgraded with the suggested features as mentioned in this subchapter, especially in regards to the functionality for determining hydrostatic performance.

6 Conclusion

Floke is a ship parametrization program developed in MATLAB. The program is capable of quickly generating customizable ship hulls using a set of input parameters in junction with Bézier curves defining the hull's main curves. This report serves to present the current state of the program and highlight its functionality and further potential.

The main features of Floke are the ability to adjust the size and shape of the hull by adjusting the input parameters and the control points for the main curves, and the ability to export the design to a format compatible with CAD software. The program also includes a feature to adjust the shape of the midship section to comply with a desired midship coefficient. These capabilities are demonstrated by running simulations through CFD software for two different hull designs and presenting the results.

Floke's strengths lie in its simplicity. Each process is clearly defined and documented, as is the potential for further improvement. The program code is structured in the order the operations are processed, with each section performing a single task. Improvements can easily be made by adding new tasks at appropriate stages, and updating the main curves accordingly.

This report provides suggestions for further functionality, especially the implementation and adjustment of additional form coefficients, and calculation of hydrostatic performance. These improvements would allow for the implementation of optimization algorithms to shape the design in terms of a desired characteristic. In addition, the program would likely benefit from a transfer to a different programming language like Python.

Although Floke does not incorporate all the initial ideas, it still delivers a product that may serve as a solid foundation for a larger hull optimization project. It is essential to highlight that the final result is meant to provide a base for further work, and is still capable of including more functionality within the code, leaving room for a more advanced and sophisticated tool for the ship design process.

References

[1] K. A. Hossain and N. M. G. Zakaria, "A Study on Global Shipbuilding Growth, Trend and Future Forecast," in Procedia Engineering, vol. 194, pp. 247-253, 2017. doi: 10.1016/j.proeng.2017.08.142.

[2] Actiflow. "CFD: The Truth and the Tales," [Online]. Available: https://actiflow.com/cfd-the-truth-and-the-tales-2/

[3] T. Ingrassia, A. Mancuso, V. Nigrelli, A. Saporito, and D. Tumino, "Parametric Hull Design with Rational Bézier Curves and Estimation of Performances," J. Mar. Sci. Eng., vol. 9, p. 360, 2021. doi: 10.3390/jmse9040360.

[4] CAESES, (n.d.). "Design and Optimization of Marine Systems." [Online]. Available: https://www.caeses.com/applications/marine/

[5] MathWorks, "MathWorks - MATLAB and Simulink," 2024. [Online]. Available: https://se.mathworks.com/products/matlab.html

[6] PTC, "Creo Parametric - 3D CAD Software — PTC," 2024. [Online]. Available: https://www.ptc.com/en/products/creo/parametric

[7] Siemens Digital Industries Software, "STAR-CCM+ — Simcenter for Fluids and Thermal Simulation," 2024. [Online]. Available: https://plm.sw.siemens.com/en-US/simcenter/fluids-thermal-simulation/star-ccm/

[8] Overleaf, "Overleaf, Online LaTeX Editor," 2024. [Online]. Available: https://www.overleaf.com

[9] OpenAI, "ChatGPT: Advanced AI System," 2024. [Online]. Available: https://chatgpt.com

[10] T. W. Sederberg, "Computer Aided Geometric Design," Computer Aided Geometric Design Course Notes, January 10, 2012. [Online]. Available: https://scholarsarchive.byu.edu/facpub/1/

[11] Wärtsilä. "Coefficients of Form." [Online]. Available: https://www.wartsila.com/encyclopedia/term/coefficients-of-form

[12] Cadence CFD Solutions, "Y+ Boundary Layer Thickness." [Online]. Available: https://resources.system-analysis.cadence.com/blog/msa2023-y-boundary-layer-thickness

[13] A. F. Molland, S. R. Turnock, and D. A. Hudson, Ship Resistance and Propulsion: Practical Estimation of Ship Propulsive Power, 2nd ed. Cambridge, United Kingdom: Cambridge University Press, 2017.

[14] 3D Print UK, "The Basic Rules for STL Files," 2023. [Online]. Available: https://www.3dprint-uk.co.uk/the-basic-rules-for-stl-files/

[15] M. Tadros, M. Ventura, and C. Guedes Soares, "Review of the Decision Support Methods Used in Optimizing Ship Hulls towards Improving Energy Efficiency," Journal of Marine Science and Engineering, vol. 11, no. 4, p. 835, 2023. doi: 10.3390/jmse11040835.

Appendices

Appendix A: Matlab script

Listing 1: Main Script

```
1 clc
2 clear
3 close all
4
5 %% Parameters
6 global midshipBreadth midshipLength midshipHeight ...
7 foreshipLength tipHeight tipBottom tipBreadth ...
8 aftLength sternBreadth aftHeight midAftHeightDiff ...
9 numPoints DWL shipLength zeroColumn heightColumn
10
11 % Midship parameters [m]
12 midshipBreadth = 15;
13 midshipLength = 50;
14 midshipHeight = 12;
15
16 % Foreship parameters [m]
17 foreshipLength = 30;
18 tipHeight = 0.1;
19 tipBottom = midshipHeight - tipHeight;
20 tipBreadth = 0.2;
21
22 % Aft parameters [m]
23 aftLength = 20;
24 sternBreadth = 13;
25 aftHeight = 8;
26 midAftHeightDiff = midshipHeight - aftHeight;
27
28 % General
29 numPoints = 100;        % Number of points per curve. Used to change accuracy
30 DWL = 8;                % Design Water Line
31 shipLength = aftLength + midshipLength + foreshipLength;
32 Cm = 0.9;               % Midship coefficient
33
34 % Would you like to adjust for midship coefficient?
35 % Yes(1) or no(0)
36 adjust = 1;
37
38 % Columns
39 zeroColumn = zeros(numPoints,1);
40 heightColumn = midshipHeight * ones(numPoints,1) - DWL;
41
42 %% Bezier Curves, default values
43
44 % Foreship curves:
45 % Bow tip
46 tipA = [0, 0];
47 tipAB = [0.25, 0];
48 tipB = [1, 0.30];
49 tipBC = [1, 0.70];
50 tipC = [1, 1];
51 bowTipControlPoints = [tipA; tipAB; tipB; tipBC; tipC];
52 bowTipBezierPoints = CreateBezierCurve(bowTipControlPoints, numPoints);
53
54 % Bow profile
55 bowA = [0, 0];
56 bowAB = [0.25, 0];
57 bowB = [0.50, 0.15];
58 bowBC = [0.95, 0.60];
59 bowC = [1, 1];
60 bowControlPoints = [bowA; bowAB; bowB; bowBC; bowC];
61 bowBezierPoints = CreateBezierCurve(bowControlPoints, numPoints);
62
63 % Foredeck
64 fdA = [0, 0];
65 fdAB = [0.25, 0];
```

```
66 fdB = [0.65, 0.30];
67 fdBC = [0.85, 0.70];
68 fdC = [1, 1];
69 foredeckControlPoints = [fdA; fdAB; fdB; fdBC; fdC];
70 foredeckBezierPoints = CreateBezierCurve(foredeckControlPoints, numPoints);
71 %-----------------------------------------------------------------
72
73 % Midship curves
74 % Default values
75 midA = [0, 0];
76 midAB = [0.10, 0];
77 midB = [0.75, 0.40];
78 midBC = [1, 0.80];
79 midC = [1, 1];
80 midshipSectionControlPoints = [midA; midAB; midB; midBC; midC];
81 midshipSectionBezierPoints = CreateBezierCurve(midshipSectionControlPoints, numPoints)
   ;
82 %-----------------------------------------------------------------
83
84 % Aft curves
85 % Stern section
86 sternA = [0, 0];
87 sternAB = [0.25, 0];
88 sternB = [0.75, 0.20];
89 sternBC = [0.95, 0.50];
90 sternC = [1, 1];
91 sternSectionControlPoints = [sternA; sternAB; sternB; sternBC; sternC];
92 sternSectionBezierPoints = CreateBezierCurve(sternSectionControlPoints, numPoints);
93
94 % Aft deck
95 adA = [0, 0];
96 adAB = [0.25, 0.40];
97 adB = [0.65, 0.60];
98 adBC = [0.85, 1];
99 adC = [1, 1];
100 aftDeckControlPoints = [adA; adAB; adB; adBC; adC];
101 aftDeckBezierPoints = CreateBezierCurve(aftDeckControlPoints, numPoints);
102
103 % Aft keel
104 aftKeelA = [0, 0];
105 aftKeelAB = [0.25, 0];
106 aftKeelB = [0.65, 0.10];
107 aftKeelBC = [0.75, 1];
108 aftKeelC = [1, 1];
109 aftKeelControlPoints = [aftKeelA; aftKeelAB; aftKeelB; aftKeelBC; aftKeelC];
110 aftKeelBezierPoints = CreateBezierCurve(aftKeelControlPoints, numPoints);
111 %-----------------------------------------------------------------
112
113 %% Plot curves and control points
114
115 % Foreship curves
116 figure('Name','Foreship␣Curves')
117 subplot(2,2,1)
118 PlotBezier(bowBezierPoints, bowControlPoints, 'Bow␣Curve', 'xz')
119 subplot(2,2,2)
120 PlotBezier(foredeckBezierPoints, foredeckControlPoints, 'Foredeck', 'xy')
121 subplot(2,2,3)
122 PlotBezier(midshipSectionBezierPoints, midshipSectionControlPoints, 'Forward␣Midship␣
   Curve', 'yz')
123 subplot(2,2,4)
124 PlotBezier(bowTipBezierPoints, bowControlPoints, 'Bow␣Tip', 'yz')
125
126 % Aft curves
127 figure('Name', 'Aft␣Curves')
128 subplot(2,2,1)
129 PlotBezier(aftKeelBezierPoints, aftKeelControlPoints, 'Aft␣Keel', 'xz')
```

```
130 subplot(2,2,2)
131 PlotBezier(aftDeckBezierPoints, aftDeckControlPoints, 'Aft␣Deck', 'xy')
132 subplot(2,2,3)
133 PlotBezier(midshipSectionBezierPoints, midshipSectionControlPoints, 'Aft␣Midship␣Curve
    ', 'yz')
134 subplot(2,2,4)
135 PlotBezier(sternSectionBezierPoints, sternSectionControlPoints, 'Stern␣Section', 'yz')
136
137
138 %% Generate linear x-values, interpolate and verify shape
139
140 [bowBezierPoints,foredeckBezierPoints, aftKeelBezierPoints, ...
141     aftDeckBezierPoints] = Linear(bowBezierPoints,foredeckBezierPoints, ...
142     aftKeelBezierPoints, aftDeckBezierPoints);
143
144 %% Translate foreship curves to fit ship dimentions
145
146 [bowTipBezierPoints, bowBezierPoints, foredeckBezierPoints, ...
147     bowTipTop] = TranslateForeship(bowTipBezierPoints, ...
148     bowBezierPoints, foredeckBezierPoints);
149
150 %% Translate aft curves to fit ship dimentions
151
152 [sternSectionBezierPoints,aftKeelBezierPoints, aftDeckBezierPoints, ...
153     sternDeckPoints] = TranslateAft(sternSectionBezierPoints,aftKeelBezierPoints,
        aftDeckBezierPoints);
154
155 %% Translate midship curves to fit ship dimentions
156
157 [midshipSectionBezierPointsAft, midshipSectionBezierPointsFront, midshipKeelPoints,
    ...
158     midshipDeckPoints] = TranslateMidship(midshipSectionBezierPoints);
159
160 %% Midship Coefficient check
161
162 if adjust == 1
163
164     % Check current Cm value
165     currentCm = FindCm(midshipSectionBezierPointsAft, DWL);
166
167     while currentCm < (Cm - 0.01) || currentCm > (Cm + 0.01)
168
169         % Adjust Control Points Z direction
170         if currentCm < Cm
171             direction = 0;
172             newControlPoints = AdjustMidshipControlPointsZ(midshipSectionControlPoints
                , direction);
173
174         elseif currentCm > Cm
175             direction = 1;
176             newControlPoints = AdjustMidshipControlPointsZ(midshipSectionControlPoints
                , direction);
177         end
178
179         % Adjust Control Points Y direction
180         if newControlPoints == midshipSectionControlPoints
181             if currentCm < Cm
182                 direction = 1;
183                 newControlPoints = AdjustMidshipControlPointsY(
                    midshipSectionControlPoints, direction);
184
185             elseif currentCm > Cm
186                 direction = 0;
187                 newControlPoints = AdjustMidshipControlPointsY(
                    midshipSectionControlPoints, direction);
188             end
```

```
189         end
190 
191         if newControlPoints == midshipSectionControlPoints
192             break;
193         end
194 
195         % New midship curve
196         midshipSectionControlPoints = newControlPoints;
197         midshipSectionBezierPoints = CreateBezierCurve(midshipSectionControlPoints,
                numPoints);
198 
199         % Translate new midship curve
200         [midshipSectionBezierPointsAft, midshipSectionBezierPointsFront,
                midshipKeelPoints, ...
201         midshipDeckPoints] = TranslateMidship(midshipSectionBezierPoints);
202 
203         % Update Cm
204         currentCm = FindCm(midshipSectionBezierPointsAft, DWL);
205 
206     end
207 
208     disp(['The␣closest␣Cm-value␣generated␣is:␣', num2str(currentCm, 4)])
209 
210 end
211 
212 %% Midship curve figure
213 % Uncomment to generate new midship section figure after Cm-adjustment
214 
215 % figure
216 % plot(midshipSectionBezierPointsFront(:,2), midshipSectionBezierPointsFront(:,3), 'b
        -'), hold on
217 % plot(-midshipSectionBezierPointsFront(:,2), midshipSectionBezierPointsFront(:,3), 'b
        -')
218 % title('Midship Profile')
219 % xlabel('Y')
220 % ylabel('Z')
221 % grid on
222 
223 %% 3D-plot of main curves
224 
225 figure
226 PlotThis(bowBezierPoints)
227 PlotThis(foredeckBezierPoints)
228 PlotThis(bowTipBezierPoints)
229 PlotThis(midshipSectionBezierPointsAft)
230 PlotThis(midshipSectionBezierPointsFront)
231 PlotThis(midshipKeelPoints)
232 PlotThis(midshipDeckPoints)
233 PlotThis(sternSectionBezierPoints)
234 PlotThis(aftKeelBezierPoints)
235 PlotThis(aftDeckBezierPoints)
236 PlotThis(sternDeckPoints)
237 PlotThis(bowTipTop)
238 
239 title('Ship␣Skeleton')
240 grid on
241 xlabel('X')
242 ylabel('Y')
243 zlabel('Z')
244 
245 xlim([-5 (shipLength + 5)])
246 ylim([-(shipLength/2 + 5) (shipLength/2 + 5)])
247 zlim([-(shipLength/2 + 5) (shipLength/2 + 5)])
248 
249 %% Export of main curves
250 % Uncomment to generate
```

```
251
252 % save('bowBezierPoints.pts', 'bowBezierPoints', '-ascii')
253 % save('foredeckBezierPoints.pts', 'foredeckBezierPoints', '-ascii')
254 % save('bowTipBezierPoints.pts', 'bowTipBezierPoints', '-ascii')
255 % save('midshipSectionBezierPointsAft.pts', 'midshipSectionBezierPointsAft', '-ascii')
256 % save('midshipSectionBezierPointsFront.pts', 'midshipSectionBezierPointsFront', '-
      ascii')
257 % save('midshipKeelPoints.pts', 'midshipKeelPoints', '-ascii')
258 % save('midshipDeckPoints.pts', 'midshipDeckPoints', '-ascii')
259 % save('sternSectionBezierPoints.pts', 'sternSectionBezierPoints', '-ascii')
260 % save('aftKeelBezierPoints.pts', 'aftKeelBezierPoints', '-ascii')
261 % save('aftDeckBezierPoints.pts', 'aftDeckBezierPoints', '-ascii')
262 % save('sternDeckPoints.pts', 'sternDeckPoints', '-ascii')
263 % save('bowTipTop.pts', 'bowTipTop', '-ascii')
264
265 %% Interpolate and scale curves
266 % Uncomment plots to generate figures to visualize the interpolation
267 % and scaling stages
268
269 % Foreship
270 % Interpolate foreship x-direction
271 foreshipInterpolatedCurves = Interpolate(midshipSectionBezierPointsFront,
      bowTipBezierPoints);
272 % figure
273 % PlotCurves(foreshipInterpolatedCurves, 'Foreship Interpolated Curves')
274 % PlotThis(foredeckBezierPoints)
275 % PlotThis(bowBezierPoints)
276 % PlotThis(midshipSectionBezierPointsFront)
277 % PlotThis(bowTipBezierPoints)
278
279 % Scale to bow
280 foreshipScaledCurves = ScaleZ(foreshipInterpolatedCurves, bowBezierPoints);
281 % figure
282 % PlotCurves(foreshipScaledCurves, 'Foreship Scaled Curves Z-Direction')
283 % PlotThis(foredeckBezierPoints)
284 % PlotThis(bowBezierPoints)
285 % PlotThis(midshipSectionBezierPointsFront)
286 % PlotThis(bowTipBezierPoints)
287
288 % Scale to deck
289 foreshipScaledCurves = ScaleY(foreshipScaledCurves, foredeckBezierPoints);
290 % figure
291 % PlotCurves(foreshipScaledCurves, 'Foreship Scaled Curves Y-Direction')
292 % PlotThis(foredeckBezierPoints)
293 % PlotThis(bowBezierPoints)
294 % PlotThis(midshipSectionBezierPointsFront)
295 % PlotThis(bowTipBezierPoints)
296
297 % Interpolate tip
298 tipInterpolatedCurves = Interpolate(bowTipBezierPoints, bowTipTop);
299 %---------------------------------------------------------------
300
301 % Midship
302 % Interpolate midship x-direction
303 midshipInterpolatedCurves = Interpolate(midshipSectionBezierPointsAft,
      midshipSectionBezierPointsFront);
304 %---------------------------------------------------------------
305
306 % Aft
307 % Interpolate aft x-direction
308 aftInterpolatedCurves = Interpolate(sternSectionBezierPoints,
      midshipSectionBezierPointsAft);
309
310 % Scale to aft keel
311 aftScaledCurves = ScaleZ(aftInterpolatedCurves, aftKeelBezierPoints);
312
```

```
313 % Scale to aft deck
314 aftScaledCurves = ScaleY(aftScaledCurves, aftDeckBezierPoints);
315 % figure
316 % PlotInterpCurves(aftScaledCurves, 'Aft Scaled curves')
317 % PlotThis(aftDeckBezierPoints)
318 % PlotThis(aftKeelBezierPoints)
319 % PlotThis(sternSectionBezierPoints)
320 % PlotThis(midshipSectionBezierPointsAft)
321
322 % Interpolate stern
323 sternInterpolatedCurves = Interpolate(sternSectionBezierPoints, sternDeckPoints);
324
325 %% Surface
326
327 % Generate surface figure
328 figure
329 CreateSurface(sternInterpolatedCurves);
330 CreateSurface(aftScaledCurves);
331 CreateSurface(midshipInterpolatedCurves);
332 CreateSurface(foreshipScaledCurves);
333 CreateSurface(tipInterpolatedCurves);
334 % Include curves
335 PlotThis(midshipSectionBezierPointsFront)
336 PlotThis(bowBezierPoints)
337 PlotThis(foredeckBezierPoints)
338 PlotThis(bowTipBezierPoints)
339 PlotThis(midshipSectionBezierPointsAft)
340 PlotThis(midshipKeelPoints)
341 PlotThis(midshipDeckPoints)
342 PlotThis(sternSectionBezierPoints)
343 PlotThis(aftKeelBezierPoints)
344 PlotThis(aftDeckBezierPoints)
345 PlotThis(sternDeckPoints)
346 PlotThis(bowTipTop)
347
348
349 title('Ship␣Surface', 'FontSize', 16)
350 xlabel('X')
351 ylabel('Y')
352 zlabel('Z')
353 xlim([-5 (shipLength + 5)])
354 ylim([-(shipLength/2 + 5) (shipLength/2 + 5)])
355 zlim([-(shipLength/2 + 5) (shipLength/2 + 5)])
356
357 %% Generate stl
358
359 % Main surfaces
360 [sternSurfX, sternSurfY, sternSurfZ] = SurfVec(sternInterpolatedCurves);
361 [aftSurfX, aftSurfY, aftSurfZ] = SurfVec(aftScaledCurves);
362 [midSurfX, midSurfY, midSurfZ] = SurfVec(midshipInterpolatedCurves);
363 [foreSurfX, foreSurfY, foreSurfZ] = SurfVec(foreshipScaledCurves);
364 [tipSurfX, tipSurfY, tipSurfZ] = SurfVec(tipInterpolatedCurves);
365
366 % Deck surfaces
367 [aftDeckSurfX, aftDeckSurfY, aftDeckSurfZ] = CreateDeckSurface(aftDeckBezierPoints);
368 [midDeckSurfX, midDeckSurfY, midDeckSurfZ] = CreateDeckSurface(midshipDeckPoints);
369 [foreDeckSurfX, foreDeckSurfY, foreDeckSurfZ] = CreateDeckSurface(foredeckBezierPoints
    );
370
371 surf2stl('sternSurface.stl', sternSurfX, sternSurfY, sternSurfZ)
372 surf2stl('aftSurface.stl', aftSurfX, aftSurfY, aftSurfZ)
373 surf2stl('midSurface.stl', midSurfX, midSurfY, midSurfZ)
374 surf2stl('foreSurface.stl', foreSurfX, foreSurfY, foreSurfZ)
375 surf2stl('tipSurface.stl', tipSurfX, tipSurfY, tipSurfZ)
376 surf2stl('aftDeckSurface.stl', aftDeckSurfX, aftDeckSurfY, aftDeckSurfZ)
377 surf2stl('midDeckSurface.stl', midDeckSurfX, midDeckSurfY, midDeckSurfZ)
```

```
378 surf2stl('foreDeckSurface.stl', foreDeckSurfX, foreDeckSurfY, foreDeckSurfZ)
379
380
381 %% Curve functions
382
383 % Interpolation
384 function[interpolatedCurves] = Interpolate(curve1, curve2)
385
386 % Define the number of interpolation points (including reference points)
387 numPoints = size(curve1,1);
388
389 % Interpolate between the two curves
390 interpolatedCurves = zeros(size(curve1, 1), 3, numPoints);
391
392 % Loop through the number of interpolations
393 for j = 1:numPoints
394     % Loop through X,Y,Z
395     for i = 1:size(curve1, 2)
396         % Loop through the rows of each curve
397         for k = 1:size(curve1, 1)
398
399             % Start and end values to evaluate
400             startStop = [curve1(k,i), curve2(k,i)];
401             % Set of query points between start and end positions
402             xq = linspace(1, 2, numPoints);
403             % Linear interpolation
404             vq = interp1(startStop, xq);
405
406             % Store the interpolated values
407             interpolatedCurves(k, i, :) = vq;
408         end
409     end
410 end
411
412 end
413 %----------------------------------------------------------
414
415 % Scale in Z-direction
416 function[scaledCurves] = ScaleZ(interpolatedCurves, curve)
417
418 % Initialize the scaled curves
419 scaledCurves = interpolatedCurves;
420     % Loop through all curves
421     for i = 1:size(curve,1)
422
423         % Calculate the scaling factor based on the desired start Z-value
424         startZ = interpolatedCurves(1, 3, i);
425         endZ = interpolatedCurves(size(curve,1), 3, i);
426         newStart = curve(i,3);
427         zScalingFactor = (endZ - newStart) / (endZ - startZ);
428
429         % Store the scaled values
430         scaledCurves(:, 3, i) = interpolatedCurves(:, 3, i) .* zScalingFactor + (1 -
                zScalingFactor) .* endZ;
431     end
432 end
433 %----------------------------------------------------------
434
435 % Scale in y-direction
436 function[scaledCurves] = ScaleY(interpolatedCurves, curve)
437
438 % Initialize the scaled curves
439 scaledCurves = interpolatedCurves;
440     % Loop through all curves
441     for i = 1:size(curve,1)
442
```

```
443         % Calculate the scaling factor based on the desired Y-value
444         startY = interpolatedCurves(size(curve,1), 2, i);
445         endY = interpolatedCurves(1, 2, i);
446         newStart = curve(i,2);
447         yScalingFactor = (endY - newStart) / (endY - startY);
448 
449         % Store the scaled values
450         scaledCurves(:, 2, i) = interpolatedCurves(:, 2, i) .* yScalingFactor + (1 -
                yScalingFactor) .* endY;
451     end
452 end
453 
454 %% Plot functions
455 
456 % Plot 2D curves
457 function[] = PlotBezier(curve, controlPoints, Title, plane)
458 
459 % Plot of main curve and its control points
460 plot(curve(:,1), curve(:,2)), hold on
461 plot(controlPoints(:, 1), controlPoints(:, 2), 'ko', 'MarkerFaceColor', 'k')
462 plot(controlPoints(:, 1), controlPoints(:, 2), 'k--'), hold off
463 title(Title)
464 
465 % Determine the axis labels
466 if plane == 'xz' %#ok<BDSCA>
467     xlabel('X')
468     ylabel('Z')
469 elseif plane == 'yz' %#ok<BDSCA>
470     xlabel('Y')
471     ylabel('Z')
472 elseif plane == 'xy' %#ok<BDSCA>
473     xlabel('X')
474     ylabel('Y')
475 end
476 
477 grid on
478 
479 end
480 %-----------------------------------------------------------
481 
482 % Plot interpolated curves
483 function[] = PlotInterpCurves(scaledCurves, Title)
484 
485 % Loop through the number of curves to be plotted
486 for i = 1:size(scaledCurves,1)
487     plot3(scaledCurves(:,1,i), scaledCurves(:,2,i), scaledCurves(:,3,i), 'k', '
            LineWidth',1)
488     hold on
489 end
490 
491 xlabel('X');
492 ylabel('Y');
493 zlabel('Z');
494 title(Title);
495 grid on;
496 end
497 %-----------------------------------------------------------
498 
499 % Plot main curves + mirror
500 function[] = PlotThis(curve)
501 
502 plot3(curve(:,1), curve(:,2), curve(:,3), 'k', 'LineWidth', 1), hold on
503 plot3(curve(:,1), -curve(:,2), curve(:,3), 'k', 'LineWidth', 1)
504 
505 end
506 
```

```
507 %% Surface functions
508
509 % Surface plot
510 function[] = CreateSurface(surfaceCurves)
511
512 % Extract X, Y and Z values
513 varX = squeeze(surfaceCurves(:,1,:));
514 varY = squeeze(surfaceCurves(:,2,:));
515 varZ = squeeze(surfaceCurves(:,3,:));
516
517 % Create copies of the Z matrix
518 zAbove = varZ;
519 zBelow = varZ;
520 % Separate the Z values above and below the waterline
521 zAbove(zAbove <= -0.05) = NaN;
522 zBelow(zBelow > 0.05) = NaN;
523
524 % Generate surface
525 surf(varX, varY, zAbove, 'FaceColor', 'g', 'EdgeColor','none'), hold on
526 surf(varX, varY, zBelow, 'FaceColor', 'b', 'EdgeColor','none')
527 % Mirror
528 surf(varX, -varY, zAbove, 'FaceColor', 'g', 'EdgeColor','none')
529 surf(varX, -varY, zBelow, 'FaceColor', 'b', 'EdgeColor','none')
530
531 end
532 %------------------------------------------------------------
533
534 % Create XYZ for stl
535 function[surfX, surfY, surfZ] = SurfVec(surfaceCurves)
536
537 % Extract X, Y and Z values
538 surfX = squeeze(surfaceCurves(:,1,:));
539 surfY = squeeze(surfaceCurves(:,2,:));
540 surfZ = squeeze(surfaceCurves(:,3,:));
541
542 % Create mirror matrices by flipping them
543 mirroredX = flip(surfX);
544 mirroredY = flip(surfY);
545 mirroredZ = flip(surfZ);
546 % Store the coordinates in new matrices
547 surfX = [mirroredX; surfX];
548 surfY = [-mirroredY; surfY];
549 surfZ = [mirroredZ; surfZ];
550
551 end
552 %------------------------------------------------------------
553
554 % Create deck surface
555 function[surfX, surfY, surfZ] = CreateDeckSurface(deckCurve)
556
557 % Extract X, Y and Z values
558 surfX = deckCurve(:,1);
559 surfY = deckCurve(:,2);
560 surfZ = deckCurve(:,3);
561
562 % Generate deck surface vector with mirror
563 surfX = [surfX; surfX];
564 surfY = [surfY; -surfY];
565 surfZ = [surfZ; surfZ];
566
567 % Reshape vector into n x 2 matrix
568 surfX = reshape(surfX, [], 2);
569 surfY = reshape(surfY, [], 2);
570 surfZ = reshape(surfZ, [], 2);
571
572 end
```

```
573
574 %% Midship Coefficient
575
576 function[currentCm] = FindCm(midshipPoints, DWL)
577
578 % Define tolerance and extract Z and Y-values
579 tolerance = 0.2;
580 zValues = abs(midshipPoints(:,3));
581 yValues = abs(midshipPoints(:,2));
582
583 % Find the indices where Z is close to zero
584 zZeroIndices = find(zValues <= tolerance);
585
586 % Generate array of the closest Z-values
587 closestValue =[];
588     for i = 1:size(zZeroIndices)
589
590         closestValue(i) = zValues(zZeroIndices(i));
591
592     end
593
594 % Determine the Z-value and find the corresponding index
595 closestValue = min(closestValue);
596 zZeroIndex = find(zValues == closestValue);
597
598 % Determine waterline breadth
599 waterlineBreadth = abs(midshipPoints(zZeroIndex,2)) * 2;
600
601 % Determine the curve points below the waterline and calculate submerged
602 % area
603 zValuesBelow = zValues(1:zZeroIndex);
604 yValuesBelow = yValues(1:zZeroIndex);
605 currentArea = trapz(yValuesBelow, zValuesBelow) * 2;
606
607 % Calculate Cm
608 currentCm = currentArea/(waterlineBreadth * DWL);
609
610 end
611 %
    ---------------------------------------------------------------------------------------
612
613 function[adjustedControlPoints] = AdjustMidshipControlPointsY(oldControlPoints,
    direction)
614
615 % Determine the number of control points and define limits
616 numControlPoints = length(oldControlPoints);
617 limit = [0 1];
618 % Define a tolerance and initialize the adjusted control points
619 tolerance = 1e-10;
620 adjustedControlPoints = oldControlPoints;
621
622     % Determine direction of adjustment
623     if direction == 1
624
625         % Adjust control points while ignoring the first and two last
626         for i = 2:(numControlPoints - 2)
627             if oldControlPoints(i,1) < limit(2)
628                 oldControlPoints(i,1) = oldControlPoints(i,1) + 0.01;
629             end
630             % Store the updated values
631             adjustedControlPoints(i,1) = oldControlPoints(i,1);
632         end
633
634     elseif direction == 0
635
```

```
636         for i = 2:(numControlPoints - 2)
637             if oldControlPoints(i,1) > (limit(1) + tolerance)
638                 oldControlPoints(i,1) = oldControlPoints(i,1) - 0.01;
639             end
640             adjustedControlPoints(i,1) = oldControlPoints(i,1);
641         end
642 
643     end
644 
645 end
646 %
    ------------------------------------------------------------------------------------------
647 
648 function [adjustedControlPoints] = AdjustMidshipControlPointsZ(oldControlPoints,
    direction)
649 
650 % Determine the number of control points and define limits
651 numControlPoints = length(oldControlPoints);
652 limit = [0 1];
653 % Define a tolerance and initialize the adjusted control points
654 tolerance = 1e-10;
655 adjustedControlPoints = oldControlPoints;
656 
657     % Determine direction of adjustment
658     if direction == 1
659 
660         % Adjust control points while ignoring the last and two first
661         for i = 3:(numControlPoints - 1)
662             if oldControlPoints(i,2) < limit(2)
663                 oldControlPoints(i,2) = oldControlPoints(i,2) + 0.01;
664             end
665             % Store the updated values
666             adjustedControlPoints(i,2) = oldControlPoints(i,2);
667         end
668 
669     elseif direction == 0
670 
671         for i = 3:(numControlPoints - 1)
672             if oldControlPoints(i,2) > (limit(1) + tolerance)
673                 oldControlPoints(i,2) = oldControlPoints(i,2) - 0.01;
674             end
675             adjustedControlPoints(i,2) = oldControlPoints(i,2);
676         end
677 
678     end
679 
680 end
681 
682 
683 %% Bezier Curve generator
684 function[bezierPoints] = CreateBezierCurve(controlPoints, numPoints)
685 
686 % Number of control points
687 n = size(controlPoints, 1);
688 
689 % Set of points to evaluate the curve at
690 t = linspace(0, 1, numPoints);
691 
692 % Preallocate array to store curve points
693 bezierPoints = zeros(length(t), 2);
694 
695 % Calculate curve points for each parameter value
696 for j = 1:length(t)
697 
698     % Initialize the curve point
```

```
699     point = [0, 0];
700
701     % Calculate the curve point based on the control points
702     for i = 1:n
703         binomialCoeff = nchoosek(n - 1, i - 1);
704         basis = binomialCoeff * (1 - t(j))^(n - i) * t(j)^(i - 1);
705         point = point + basis * controlPoints(i, :);
706     end
707
708     % Store the calculated point in the curvePoints array
709     bezierPoints(j, :) = point;
710 end
711
712 end
```

Listing 2: Linear MATLAB function

```
1 function [bowBezierPoints,foredeckBezierPoints, aftKeelBezierPoints, ...
2     aftDeckBezierPoints] = Linear(bowBezierPoints,foredeckBezierPoints, ...
3     aftKeelBezierPoints, aftDeckBezierPoints)
4
5 % Import appropriate variables
6 global numPoints
7
8 % New set of X-values
9 xPositions = linspace(0, 1, numPoints);
10
11 % Interpolate Z positions bow
12 zPositionsBow = interp1(bowBezierPoints(:,1), bowBezierPoints(:,2), xPositions);
13 % Interpolate Y positions foredeck
14 yPositionsForedeck = interp1(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2),
     xPositions);
15
16 % Interpolate Z positions aft keel
17 zPositionsAftKeel = interp1(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2),
     xPositions);
18 % Interpolate Y positions aft deck
19 yPositionsAftDeck = interp1(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2),
     xPositions);
20
21 %figure
22 %plot(bowBezierPoints(:,1), bowBezierPoints(:,2), 'b-'), hold on
23 % plot(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2), 'r-')
24
25 % Create new Bezier curves from the updated positions
26 bowBezierPoints = [xPositions', zPositionsBow'];
27 foredeckBezierPoints = [xPositions', yPositionsForedeck'];
28
29 %plot(bowBezierPoints(:,1), bowBezierPoints(:,2), 'ko')
30 %legend('Original Bow Curve', 'CurvePoints after linearization', 'Location','northwest
     ')
31 %title('Bow Curve Linearized in X-Direction')
32 %xlabel('X')
33 %ylabel('Z')
34 %grid on
35 %plot(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2), 'ko'), hold off
36
37 % figure
38 % plot(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2), 'b-'), hold on
39 % plot(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2), 'r-')
40
41 % Create new Bezier curvs from the updated positions
42 aftKeelBezierPoints = [xPositions', zPositionsAftKeel'];
43 aftDeckBezierPoints = [xPositions', yPositionsAftDeck'];
44
45 % plot(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2), 'ko')
```

```
46 % plot(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2), 'ko'), hold off
47
48 end
```

Listing 3: TranslateForeship MATLAB function

```
1 function [bowTipBezierPoints, bowBezierPoints, foredeckBezierPoints, ...
2     bowTipTop] = TranslateForeship(bowTipBezierPoints, ...
3     bowBezierPoints, foredeckBezierPoints)
4
5 global midshipBreadth midshipLength midshipHeight ...
6 foreshipLength tipHeight tipBottom tipBreadth ...
7 aftLength ...
8 numPoints DWL shipLength zeroColumn heightColumn
9
10
11 % Bow tip
12 % Fit to yz-plane
13 bowTipBezierPoints(:,1) = tipBreadth/2 * bowTipBezierPoints(:,1);
14 bowTipBezierPoints(:,2) = tipHeight * bowTipBezierPoints(:,2) - DWL + tipBottom;
15
16 % Translation in x-direction, generate 3D-coordinates
17 lengthColumn = shipLength * ones(numPoints,1);
18 bowTipBezierPoints = [lengthColumn, bowTipBezierPoints];
19 %-----------------------------------------------------------
20
21 % Bow tip top
22 bowTopX = bowTipBezierPoints(:,1);
23 bowTopY = bowTipBezierPoints(:,2);
24 heightColumn = midshipHeight * ones(numPoints,1) - DWL;
25 bowTipTop = [bowTopX, bowTopY, heightColumn];
26 %-----------------------------------------------------------
27
28 % Bow
29 % Fit to xz-plane
30 bowBezierPoints(:,1) = foreshipLength * bowBezierPoints(:, 1) + midshipLength +
    aftLength;
31 bowBezierPoints(:,2) = bowBezierPoints(:,2) * tipBottom - DWL;
32
33 % Generate 3D-coordinates
34 zeroColumn = zeros(numPoints,1);
35 bowBezierPoints = [bowBezierPoints(:,1), zeroColumn, bowBezierPoints(:,2)];
36 %-----------------------------------------------------------
37
38 % Foredeck
39 % Fit to xy-plane
40 foredeckBezierPoints(:,1) = foreshipLength * foredeckBezierPoints(:, 1) +
    midshipLength + aftLength;
41 foredeckBezierPoints(:,2) = (-(midshipBreadth)/2  + tipBreadth/2) *
    foredeckBezierPoints(:, 2) + midshipBreadth/2;
42
43 % Generate 3D-coordinates
44 foredeckBezierPoints = [foredeckBezierPoints, heightColumn];
45
46 end
```

Listing 4: TranslateAft MATLAB function

```
1 function [sternSectionBezierPoints,aftKeelBezierPoints, aftDeckBezierPoints, ...
2     sternDeckPoints] = TranslateAft(sternSectionBezierPoints,aftKeelBezierPoints,
        aftDeckBezierPoints)
3
4 global midshipBreadth  midshipHeight ...
5 aftLength sternBreadth aftHeight midAftHeightDiff ...
6 numPoints DWL zeroColumn heightColumn
7
8 % Stern section
9 % Fit to yz-plane
10 sternSectionBezierPoints(:,1) = sternBreadth/2 * sternSectionBezierPoints(:,1);
11 sternSectionBezierPoints(:, 2) = aftHeight * sternSectionBezierPoints(:, 2) +
    midAftHeightDiff - DWL;
12
13 % Translation in x-direction, create 3D-coordinates
14 sternSectionBezierPoints = [zeroColumn, sternSectionBezierPoints];
15 %-----------------------------------------------------------
16
17 % Aft keel
18 % Fit to xz-plane
19 aftKeelBezierPoints(:,1) = aftLength * aftKeelBezierPoints(:, 1);
20 aftKeelBezierPoints(:, 2) = -midAftHeightDiff * aftKeelBezierPoints(:, 2) +
    midAftHeightDiff - DWL;
21
22 % Create 3D-coordinates
23 aftKeelBezierPoints = [aftKeelBezierPoints(:,1), zeroColumn, aftKeelBezierPoints(:, 2)
    ];
24 %-----------------------------------------------------------
25
26 % Aft deck
27 % Fit to xy-plane
28 aftDeckBezierPoints(:,1) = aftLength * aftDeckBezierPoints(:,1);
29 aftDeckBezierPoints(:,2) = (midshipBreadth/2 - sternBreadth/2) * aftDeckBezierPoints
    (:,2) + sternBreadth/2;
30
31 % Translation in z-direction, create 3D-coordinates
32 aftDeckBezierPoints = [aftDeckBezierPoints, heightColumn];
33 %-----------------------------------------------------------
34
35 % Stern deck
36 sternDeckY = linspace(0,sternBreadth/2,numPoints)';
37 sternDeckPoints = [zeroColumn, sternDeckY, heightColumn];
38
39 end
```

Listing 5: TranslateMidship MATLAB function

```
1 function [midshipSectionBezierPointsAft, midshipSectionBezierPointsFront,
    midshipKeelPoints, ...
2     midshipDeckPoints] = TranslateMidship(midshipSectionBezierPoints)
3
4 global midshipBreadth midshipLength midshipHeight ...
5 aftLength  ...
6 numPoints DWL
7
8 % % Midship section
9 % % Fit to yz-plane
10 midshipSectionBezierPoints(:,1) = midshipBreadth/2 * midshipSectionBezierPoints(:,1);
11 midshipSectionBezierPoints(:,2) = (midshipHeight) * midshipSectionBezierPoints(:,2) -
    DWL;
12
13 % Midship section aft
14 % Translation in x-direction, generate 3D-coordinates
15 lengthColumn = (aftLength) * ones(numPoints,1);
16 midshipSectionBezierPointsAft = [lengthColumn, midshipSectionBezierPoints];
```

```
17 %-----------------------------------------------------------
18
19 % Midship section front
20 % Translation in x-direction, generate 3D-coordinates
21 lengthColumn = (aftLength + midshipLength) * ones(numPoints,1);
22 midshipSectionBezierPointsFront = [lengthColumn, midshipSectionBezierPoints];
23 %-----------------------------------------------------------
24
25 % Midship keel line
26 keelX = linspace(0,midshipLength,numPoints)' + aftLength;
27 midshipDraught = -DWL*ones(numPoints,1);
28 midshipKeelPoints = [keelX, zeros(numPoints,1), midshipDraught];
29 %-----------------------------------------------------------
30
31 % Midship deck line
32 midshipDeckPoints = midshipKeelPoints;
33 midshipDeckPoints(:,2) = ones(numPoints,1) * midshipBreadth/2;
34 midshipDeckPoints(:,3) = ones(numPoints,1) * midshipHeight - DWL;
35
36 end
```

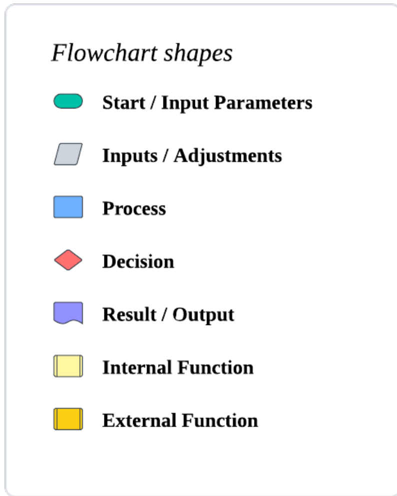
Appendix B: Software documentation

Software Documentation
Floke

David Timmer Endal
Adam Sven Åkervall

May 2024

Floke



Input Parameters

Adjust for Cm Y/N

Define/Adjust Control Points

Bezier Curve Generation

CreateBezierCurve

Figures of Main Curves

PlotBezier

Linearize X-Values Fore and Aft

Linear

Translation of Foreship Curves

TranslateForeship

Translation of Aft Curves

TranslateAft

AdjustMidshipControlPointsY

AdjustMidshipControlPointsZ

CreateBezierCurve

Adjust Midship Control Points

Translation of Midship Curves

TranslateMidship

No

FindCm

Check currentCm = inputCm

Yes

Adjust for Cm

No

Yes

3D Figure Ship Skeleton

PlotThis

Interpolate

Interpolate Main Curves

Scale Main Curves

ScaleZ

ScaleY

3D Figure Ship Surface

CreateSurface

PlotThis

SurfVec

CreateDeckSurface

surf2stl

STL Generation

Contents

1 Introduction **4**

2 Coordinate system **4**

3 Method Description **4**

3.1 Input parameters . . . 5
3.2 Control Points . . . 5
3.3 Bezier Curve Generation . . . 5
3.4 Figures of main curves . . . 6
3.5 Linearization . . . 6
3.6 Translation . . . 6
3.7 Midship Coefficient Adjustment . . . 6
3.8 Interpolation . . . 7
3.9 Scaling . . . 7
3.10 Surface generation and export . . . 7

4 Functions **8**

4.1 Internal functions . . . 8
4.1.1 CreateBezierCurve . . . 8
4.1.2 FindCm . . . 9
4.1.3 AdjustMidshipControlPoints functions . . . 11
4.1.4 Interpolate . . . 13
4.1.5 Scaling functions . . . 14
4.1.6 PlotBezier . . . 15
4.1.7 PlotThis . . . 16
4.1.8 PlotInterpCurves . . . 17
4.1.9 CreateSurface . . . 18
4.1.10 SurfVec . . . 19
4.1.11 CreateDeckSurface . . . 20
4.2 External functions . . . 21
4.2.1 Linear . . . 21
4.2.2 TranslateForeship . . . 23
4.2.3 TranslateAft . . . 25
4.2.4 TranslateMidship . . . 27
4.2.5 surf2stl . . . 29

5 References **31**

1 Introduction

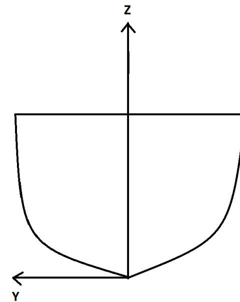
Floke is a program designed to generate a ship model based on a set of Bezier curves defining the skeleton of the hull. These curves are created from a set of manually defined control points, and translated to fit the hull's dimensions. The program creates a surface from the curves and exports it to an STL (Stereolithography) file. The purpose of Floke is to aid in the design of a ship by quickly generating a hull from a set of input parameters.

Floke is developed using MATLAB version R2023b [1].

2 Coordinate system

Floke uses a 3-dimensional Cartesian coordinate system with the origin at the stern. The X-direction defines the hull's longitudinal axis, the Y-direction defines the lateral axis and the Z-direction defines the vertical axis.

(a) XZ-plane as seen from starboard side



(b) YZ-plane as seen from the stern

Figure 1: Coordinate system

3 Method Description

Floke divides the hull into three sections: foreship, midship and aft. As the program runs, these sections are handled independently in all stages. The program generates a hull through this order of operations:

1. Definition of input parameters.
2. Definition of control points for the main curves.
3. Bezier curve generation.
4. Figures of the main curves.
5. Linearization of foreship and aft curves.
6. Translation of main curves to fit ship dimensions.
7. Midship coefficient adjustment.
8. Figure of ship skeleton.
9. Interpolation of main curves.
10. Scaling the interpolated curves.
11. Figure of ship surface.
12. Surface generation and export.

3.1 Input parameters

This section handles the main inputs for the program to run. When generating a hull using Floke, this is where to start. These parameters consist of the main ship dimensions, and some other parameters in regards to accuracy and other conditions. The main dimensions of the hull consist of:

Table 1: Ship dimensions

midshipBreadth	Breadth of the midship at maximum height
midshipLength	Total length of midship in the X-direction
midshipHeight	Total height of midship, measured from keel
foreshipLength	Total length of foreship in the X-direction
tipHeight	Height of the bow tip surface in the YZ-plane
tipBottom	Calculated bottom position of the bow tip surface in the YZ-plane
tipBreadth	Breadth of the bow tip surface in the YZ-plane
aftLength	Total length of the aft section in the X-direction
sternBreadth	Breadth of the aft-most part of the hull, measured from deck height
aftHeight	Height of the stern as measured from the lowest point to deck height
midAftHeightDiff	Calculated height difference between the bottom of the stern and the midship keel line

The following parameters are also included in this section:

Table 2: Other parameters

numPoints	Designates the accuracy. Determines the number of points per curve.
DWL	Design water line
Cm	Desired midship coefficient
adjust	Determines whether the program adjusts for the desired midship coefficient. 1 for yes, 0 for no.

Most of these parameters are used as public variables by using the global keyword. This ensures that the external functions have access to information about the ship's main dimensions.

3.2 Control Points

The main curves of the hull are defined by sets of two-dimensional control points, which may be manipulated to change the shape of the curves. Each curve has five control points by default, but more can be added if needed. This section is responsible for defining the shape of the hull once the main parameters are declared. The coordinate values of the control points are within the range [0 1]. Although possible, it is not advised to define control points outside this range, as this may cause adverse effects on the processes following this section.

3.3 Bezier Curve Generation

To prepare the control points for Bezier curve generation, they are collected in a column array. Each curve is generated by calling the CreateBezierCurve (4.1.1) function and passing in the control points array and the accuracy parameter **numPoints**. The **numPoints** paramater will hereby be referred to as **n**. The curves generated in this stage are stored in ($\mathbf{n} \times 2$) matrices, and make up the main curves that define the shape of the hull.

3.4 Figures of main curves

The main curves are illustrated in two different figures, foreship and aft. Each figure is divided into four subplots, and the curves are plotted using the PlotBezier (4.1.6) function. For the foreship, the figure contains plots of the bow, foredeck, front midship and tip curves. The aft section figure contains plots of the keel, aft deck, aft midship and stern section curves.

3.5 Linearization

The longitudinal curves defining the foreship and aft sections are linearized in the X-direction using the external function Linear (4.2.1). This function generates linearly spaced X-values for the foredeck, bow, aft deck and aft keel curves. It then performs linear interpolation on the curve's Y or Z-values, depending on orientation. This ensures that the points on each curve has corresponding X-values to simplify the surface generation in the interpolation and scaling stages.

3.6 Translation

This section employs the ship dimension parameters to translate the main curves to their appropriate positions in three dimensions. This is done using the external TranslateForeship (4.2.2), TranslateAft (4.2.3) and TranslateMidship (4.2.4) functions. These functions import the global variables from the parameters in 3.1 to use as references when translating. Once the operation is finished, the new and translated curves are stored in ($\mathbf{n} \times 3$) matrices, where the columns designate the X, Y and Z-values respectively.

3.7 Midship Coefficient Adjustment

If the **adjust** parameter is set to 1, this section will first call the FindCm (4.1.2) function to calculate the midship coefficient of the current design, **currentCm**. Then, the control points of the midship section are adjusted. This process alters the shape of the midship by moving the control points until **currentCm** equals the desired **Cm**, as defined in the parameters section. This is done using a while loop for as long as the following condition is true:

while **currentCm** <(**Cm** - 0.01) || **currentCm** >(**Cm** + 0.01)

After each adjustment, the CreateBezierCurve, TranslateMidship and FindCm functions are called again to update the variables. The tolerances are included to ensure that the loop breaks when **currentCm** is approximately equal to **Cm**. These tolerances may be reduced to increase accuracy, but will also increase the risk of creating an endless loop. The loop will also break if the control points reach the limits of their domain [0 1]. The directions in which the control points need to be changed are decided using if-statements inside the loop. Adjustment in the negative direction is designated as 0 and adjustment in the positive direction is designated as 1. The vertical direction is evaluated first:

```
if currentCm <Cm
        direction = 0;
elseif currentCm >Cm
        direction = 1;
```

The AdjustMidshipControlPointsZ (4.1.3) function is then called to adjust the midship section control points in the vertical direction. This means the midship section will only be adjusted in the vertical direction until the condition of the while loop is met. It is only when the adjusted control points reach their limits that the loop moves on to evaluate the lateral direction by calling the AdjustMidshipControlPointsY (4.1.3) function. This is done by assessing whether there has been a change in the control points after being evaluated for the vertical direction:

if **newControlPoints == midshipSectionControlPoints**

The loop breaks once currentCm is equal to the desired Cm or when the control points reach their limits, whichever comes first.

The control points are adjusted in the vertical direction first, due to the characteristics of the midship coefficient. To increase the transverse submerged area while maintaining the original draught and total breadth, there are two possibilities. The first is to increase the control points horizontally, generating a wider midship section curve. The second is to decrease

the control points vertically, generating a deeper curve. As an increase in the horizontal direction also increases the waterline breadth, there is an increase in both the submerged area and the rectangular area. There is therefore a greater change in the midship coefficient by changing the control points in the vertical direction first.

3.8 Interpolation

Using the internal function Interpolate (4.1.4) and passing in two lateral curves generates linearly interpolated curves between the two. The number of intermediate curves is decided by the accuracy. Floke handles the interpolation in the positive X-direction. The foreship curves are generated first, between the front midship section and the bow tip curve. The midship curves are second, generated between the aft midship section and the front midship section. Then, the aft curves are generated between the stern section and aft midship section. Lastly, the vertical bow tip and stern surfaces are generated using the same function. The surfaces are stored in ($\mathbf{n} \times 3 \times \mathbf{n}$) matrices for **n** number of rows, three columns representing X, Y and Z, and **n** number of surface curves.

3.9 Scaling

As the interpolation stage does not account for the longitudinal curves, a scaling operation is needed. This ensures that the start and end points of the interpolated curves coincide with the curves defining the bow, foredeck, aft deck and aft keel. The foreship is handled first by calling the ScaleZ (4.1.5) and ScaleY (4.1.5) functions. The aft section is handled second. As the midship is defined by two identical curves, the front and aft midship section curves, there is no need to perform the scaling operation on them.

3.10 Surface generation and export

The interpolated surface points are separated into ($2\mathbf{n} \times \mathbf{n}$) vectors containing X, Y and Z-values with the SurfVec (4.1.10) function. The number of rows are doubled, due to the mirroring of the surface curves. This is to prepare the surfaces for the export function surf2stl. The deck surfaces are generated and stored in the same way using the CreateDeckSurface (4.1.11) function and passing in the appropriate deck curve. The export of the model is handled by the surf2stl function by passing in the X,Y and Z-values from each surface. The function is called once for each surface, which creates several STL-files. This is to make sure there will not be any additional surfaces drawn between the start and end points of two different surfaces. Should there be a desire to generate the surface from a single file, the recommended procedure is to import each surface into a 3D-modelling program and saving the entire hull as a single file. Otherwise, there may be parts of the hull surfaces that overlap or are discontinuous.

4 Functions

This section provides detailed descriptions of the functions used in Floke. There are two different types of functions: internal and external. The internal functions are integrated in the program itself, while the external functions are called from outside the script.

4.1 Internal functions

4.1.1 CreateBezierCurve

Input parameters:

- **controlPoints**: A matrix containing the control points of the Bezier curve, where the columns represent the X and Y-values.
- **numPoints**: The number of points to evaluate the curve at.

Output:

- **bezierPoints**: A matrix containing the points lying on the Bezier curve, where the columns represent the X and Y-values.

This function generates the main curves of the hull through the following algorithm:

1. Determine the number of control points.
2. Generate a set of evenly spaced parameters ranging from 0 to 1 based on the specified number of points.
3. Preallocate an array to store the curve points.
4. For each parameter value, calculate the corresponding point on the Bezier curve using De Casteljau's algorithm:
   (a) Initialize the point to [0 0].
   (b) For each control point, calculate the Bernstein basis polynomial and multiply it with the control point. Add the result to the current point on the Bezier curve.
   (c) Store the point in the BezierPoints array.

Listing 1: CreateBezierCurve MATLAB function

```
function [bezierPoints] = CreateBezierCurve(controlPoints, numPoints)

    % Number of control points
    n = size(controlPoints, 1);

    % Set of points to evaluate the curve at
    t = linspace(0, 1, numPoints);

    % Preallocate array to store curve points
    bezierPoints = zeros(length(t), 2);

    % Calculate curve points for each parameter value
    for j = 1:length(t)

        % Initialize the curve point
        point = [0, 0];

        % Calculate the curve point based on the control points
        for i = 1:n
            binomialCoeff = nchoosek(n - 1, i - 1);
            basis = binomialCoeff * (1 - t(j))^(n - i) * t(j)^(i - 1);
            point = point + basis * controlPoints(i, :);
        end

        % Store the calculated point in the curvePoints array
        bezierPoints(j, :) = point;
    end
end
```

4.1.2 FindCm

Input parameters:

- **midshipPoints**: The matrix containing the Bezier points for the midship section curve.
- **DWL**: Design water line

Output:

- **currentCm**: Midship coefficient for the current configuration of main curves.

This function generates the value of the midship coefficient through the following algorithm:

1. Find breadth at waterline.
   (a) Define a tolerance at waterline level.
   (b) Extract absolute Z and Y-values from the midship section curve.
   (c) Find all Z-values close to zero, within the tolerance.
   (d) Iterate through the chosen Z-values to find the closest.
   (e) Determine the index of the closest value.
   (f) Extract waterline breadth as the Y-value at the determined index.
2. Generate a set of Z and Y-values below the waterline.
3. Determine the transverse submerged area by applying the trapezoidal rule using the trapz function with the previous Z and Y-values as arguments.
4. Determine the midship coefficient by dividing the submerged area by the rectangle defined by the waterline breadth and draught.

Please note that the waterline breadth and submerged area are multiplied by two. This is because the method determines the distance from the centerline to the side of the hull, not the entire breadth.

Listing 2: FindCm MATLAB function

```
function[currentCm] = FindCm(midshipPoints, DWL)

% Define tolerance and extract Z and Y-values
tolerance = 0.2;
zValues = abs(midshipPoints(:,3));
yValues = abs(midshipPoints(:,2));

% Find the indices where Z is close to zero
zZeroIndices = find(zValues <= tolerance);

% Generate array of the closest Z-values
closestValue =[];
    for i = 1:size(zZeroIndices)

        closestValue(i) = zValues(zZeroIndices(i));

    end

% Determine the Z-value and find the corresponding index
closestValue = min(closestValue);
zZeroIndex = find(zValues == closestValue);

% Determine waterline breadth
waterlineBreadth = abs(midshipPoints(zZeroIndex,2)) * 2;

% Determine the curve points below the waterline and calculate submerged
% area
zValuesBelow = zValues(1:zZeroIndex);
yValuesBelow = yValues(1:zZeroIndex);
currentArea = trapz(yValuesBelow, zValuesBelow) * 2;
```

```
31
32 % Calculate Cm
33 currentCm = currentArea/(waterlineBreadth * DWL);
34
35 end
```

4.1.3 AdjustMidshipControlPoints functions

There are two nearly identical functions designed to adjust the control points of the midship section curve. The first adjusts the control points in the Y-direction only by calling the AdjustMidshipControlPointsY version. The other adjusts the control points in the Z-direction only by calling the AdjustMidshipControlPointsZ version. To ensure that the curve remains tangent at the start and end points, both functions will ignore a selection of three control points. The reason for the distinction between the two versions of the function is because this selection is determined differently for each one. In the Y-direction, the first (at the keel) and the two last (at the deck) control points are ignored. In the Z-direction, the last and the two first control points are ignored.

Input parameters:

- **oldControlPoints**: The original control points to be changed.
- **direction**: The direction of the change in Y or Z, 1 for the positive and 0 for the negative.

Output:

- **adjustedControlPoints**: New control points, adjusted in the appropriate direction.

These functions generate adjusted control points through the following algorithm:

1. Determine the number of control points.
2. Define the lower and upper limits of the control points as [0 1].
3. Define a tolerance close to zero.
4. Initialize the new control points to be equal to the old.
5. Determine direction of adjustment.
6. Adjust the control points.
   (a) For each applicable control point, determine whether it has reached its limits.
   (b) If not, increase or decrease the Y or Z-value by 0.01, dependent on the direction and the version of the function used.
   (c) Update the adjustedControlPoints array.

Please note that the function includes a tolerance when checking whether the control points have reached the lower limit, i.e. zero. This is to compensate for precision errors in the way MATLAB handles floating point numbers. Because of rounding errors, MATLAB may represent zero as a very low number instead. However, the limit check in the function determines whether the value is exactly zero. This causes the control points to pass the check even though the limit has been reached, and run another adjustment. The inclusion of the tolerance negates this issue.

Listing 3: AdjustMidshipControlPointsY MATLAB function

```
1 function[adjustedControlPoints] = AdjustMidshipControlPointsY(oldControlPoints,
      direction)
2
3 % Determine the number of control points and define limits
4 numControlPoints = length(oldControlPoints);
5 limit = [0 1];
6 % Define a tolerance and initialize the adjusted control points
7 tolerance = 1e-10;
8 adjustedControlPoints = oldControlPoints;
9
10    % Determine direction of adjustment
11    if direction == 1
12
13        % Adjust control points while ignoring the first and two last
14        for i = 2:(numControlPoints - 2)
15            if oldControlPoints(i,1) < limit(2)
16                oldControlPoints(i,1) = oldControlPoints(i,1) + 0.01;
17            end
18            % Store the updated values
19            adjustedControlPoints(i,1) = oldControlPoints(i,1);
```

```
20          end
21
22      elseif direction == 0
23
24          for i = 2:(numControlPoints - 2)
25              if oldControlPoints(i,1) > (limit(1) + tolerance)
26                  oldControlPoints(i,1) = oldControlPoints(i,1) - 0.01;
27              end
28              adjustedControlPoints(i,1) = oldControlPoints(i,1);
29          end
30      end
31  end
```

Listing 4: AdjustMidshipControlPointsZ MATLAB function

```
1  function [adjustedControlPoints] = AdjustMidshipControlPointsZ(oldControlPoints,
       direction)
2
3  % Determine the number of control points and define limits
4  numControlPoints = length(oldControlPoints);
5  limit = [0 1];
6  % Define a tolerance and initialize the adjusted control points
7  tolerance = 1e-10;
8  adjustedControlPoints = oldControlPoints;
9
10     % Determine direction of adjustment
11     if direction == 1
12
13         % Adjust control points while ignoring the last and two first
14         for i = 3:(numControlPoints - 1)
15             if oldControlPoints(i,2) < limit(2)
16                 oldControlPoints(i,2) = oldControlPoints(i,2) + 0.01;
17             end
18             % Store the updated values
19             adjustedControlPoints(i,2) = oldControlPoints(i,2);
20         end
21
22     elseif direction == 0
23
24         for i = 3:(numControlPoints - 1)
25             if oldControlPoints(i,2) > (limit(1) + tolerance)
26                 oldControlPoints(i,2) = oldControlPoints(i,2) - 0.01;
27             end
28             adjustedControlPoints(i,2) = oldControlPoints(i,2);
29         end
30     end
31 end
```

4.1.4 Interpolate

Input parameters:

- **curve1**: The start curve for the interpolation.
- **curve2**: The end curve for the interpolation

Output:

- **interpolatedCurves**: Set of interpolated curves between the start and end-curves.

This function generates a set of interpolated curves through the following algorithm.

1. Define the number of interpolated curves, including the start and end curves.
2. Initialize a three-dimensional array of zeros to store the interpolated curves.
3. Loop through the number of interpolated curves.
4. Loop through X,Y and Z.
5. Loop through the rows of each curve.
   (a) Define a vector with the start and end values to interpolate between.
   (b) Define a vector of query points between the start and end values, with the same length as the number of curves to evaluate.
   (c) Interpolate between the start and end values using the interp1 function with the previous vectors as arguments.
   (d) Store the interpolated values at the appropriate positions in the interpolatedCurves array.

Listing 5: Interpolate MATLAB function

```
function[interpolatedCurves] = Interpolate(curve1, curve2)

% Define the number of interpolation points (including reference points)
numPoints = size(curve1,1);

% Interpolate between the two curves
interpolatedCurves = zeros(size(curve1, 1), 3, numPoints);

% Loop through the number of interpolations
for j = 1:numPoints
    % Loop through X,Y,Z
    for i = 1:size(curve1, 2)
        % Loop through the rows of each curve
        for k = 1:size(curve1, 1)

            % Start and end values to evaluate
            startStop = [curve1(k,i), curve2(k,i)];
            % Set of query points between start and end positions
            xq = linspace(1, 2, numPoints);
            % Linear interpolation
            vq = interp1(startStop, xq);

            % Store the interpolated values
            interpolatedCurves(k, i, :) = vq;
        end
    end
end
end
```

4.1.5 Scaling functions

There are two nearly identical functions designed to scale the interpolated curves, ScaleZ for scaling in the Z-direction and ScaleY for the Y-direction.

Input parameters:

- **interpolatedCurves**: The set of interpolated curves to be scaled to a reference curve.
- **curve**: The reference curve.

Output:

- **scaledCurves**: A set of adjusted curves.

These functions generate sets of scaled curves through the following algorithm:

1. Initialize the scaled curves.
2. Loop through each curve.
   (a) Define the start and end value of Y or Z in the original curve, dependent on the version of the function used.
   (b) Define the desired start value from the reference curve.
   (c) Calculate the scaling factor using linear interpolation based on the desired start value and the original start and end values.
   (d) Update the scaledCurves array by applying the scaling factor.

Listing 6: ScaleZ MATLAB function

```
1  function[scaledCurves] = ScaleZ(interpolatedCurves, curve)
2
3  % Initialize the scaled curves
4  scaledCurves = interpolatedCurves;
5      % Loop through all curves
6      for i = 1:size(curve,1)
7
8          % Calculate the scaling factor based on the desired start Z-value
9          startZ = interpolatedCurves(1, 3, i);
10         endZ = interpolatedCurves(size(curve,1), 3, i);
11         newStart = curve(i,3);
12         zScalingFactor = (endZ - newStart) / (endZ - startZ);
13
14         % Store the scaled values
15         scaledCurves(:, 3, i) = interpolatedCurves(:, 3, i) .* zScalingFactor + (1 -
               zScalingFactor) .* endZ;
16     end
17 end
```

Listing 7: ScaleY MATLAB function

```
1  function[scaledCurves] = ScaleY(interpolatedCurves, curve)
2
3  % Initialize the scaled curves
4  scaledCurves = interpolatedCurves;
5      % Loop through all curves
6      for i = 1:size(curve,1)
7
8          % Calculate the scaling factor based on the desired Y-value
9          startY = interpolatedCurves(size(curve,1), 2, i);
10         endY = interpolatedCurves(1, 2, i);
11         newStart = curve(i,2);
12         yScalingFactor = (endY - newStart) / (endY - startY);
13
14         % Store the scaled values
15         scaledCurves(:, 2, i) = interpolatedCurves(:, 2, i) .* yScalingFactor + (1 -
               yScalingFactor) .* endY;
16     end
17 end
```

4.1.6 PlotBezier

Input parameters:

- **curve**: The Bezier curve to be drawn.
- **controlPoints**: The curve's control points.
- **Title**: The title of the figure.
- **plane**: The two-dimensional plane in which the curve lies.

Output:

- A figure containing the Bezier curve and its control points, with named axes and a title.

The figure is generated through the following process:

1. Plot the main curve using the plot function.
2. Plot the control points as black circles using the optional 'ko' argument, where 'k' and 'o' designate the color and style respectively. Use the additional arguments 'MarkerFaceColor' and 'k' to fill in the circles.
3. Plot dashed lines through the control points using the optional argument 'k–'.
4. Insert the figure title.
5. Insert the axis labels determined from the plane parameter.
6. Turn the grid on.

Listing 8: PlotBezier MATLAB function

```
function[] = PlotBezier(curve, controlPoints, Title, plane)

% Plot of main curve and its control points
plot(curve(:,1), curve(:,2)), hold on
plot(controlPoints(:, 1), controlPoints(:, 2), 'ko', 'MarkerFaceColor', 'k')
plot(controlPoints(:, 1), controlPoints(:, 2), 'k--'), hold off
title(Title)

% Determine the axis labels
if plane == 'xz' %#ok<BDSCA>
    xlabel('X')
    ylabel('Z')
elseif plane == 'yz' %#ok<BDSCA>
    xlabel('Y')
    ylabel('Z')
elseif plane == 'xy' %#ok<BDSCA>
    xlabel('X')
    ylabel('Y')
end

grid on

end
```

4.1.7 PlotThis

Input parameters:

- **curve**: The curve to be drawn.

Output:

- A three-dimensional figure containing the main curves of the hull and their mirrors.

This function generates a three-dimensional figure of the main curves through the following process:

1. Plot the curve in 3D using the plot3 function. Include the optional 'k', 'LineWidth' and '1' arguments to generate black lines with a line width of size 1.
2. Mirror the curve by changing the sign of the Y-values.

Listing 9: PlotThis MATLAB function

```
function[] = PlotThis(curve)

plot3(curve(:,1), curve(:,2), curve(:,3), 'k', 'LineWidth', 1), hold on
plot3(curve(:,1), -curve(:,2), curve(:,3), 'k', 'LineWidth', 1)

end
```

4.1.8 PlotInterpCurves

Input parameters:

- **scaledCurves**: The scaled curves to be drawn.
- **Title**: The title of the figure.

Output:

- A three-dimensional figure containing the interpolated and scaled curves along the hull's surface.

This function generates a three-dimensional figure of the scaled and interpolated curves through the following process:

1. Loop through the number of curves to be plotted.
   (a) Plot the curves using the plot3 function. Include the optional 'k', 'LineWidth' and '1' arguments to generate black lines with a line width of size 1.
2. Label the axes and insert the title.
3. Turn the grid on.

Listing 10: PlotInterpCurves MATLAB function

```
1 function[] = PlotInterpCurves(scaledCurves, Title)
2
3 % Loop through the number of curves to be plotted
4 for i = 1:size(scaledCurves,1)
5     plot3(scaledCurves(:,1,i), scaledCurves(:,2,i), scaledCurves(:,3,i), 'k', '
          LineWidth',1)
6     hold on
7 end
8
9 xlabel('X');
10 ylabel('Y');
11 zlabel('Z');
12 title(Title);
13 grid on;
14 end
```

4.1.9 CreateSurface

Input parameters:

- **surfaceCurves**: The curves defining the surface of the hull.

Output:

- A figure containing a three-dimensional visual representation of the hull surface with different colors above and below the waterline.

This function generates a three-dimensional figure of the hull surface through the following process:

1. Define the X, Y and Z-values as individual matrices using the squeeze function to remove singleton dimensions.
2. Create two copies of the Z matrix to define values above and below the waterline.
3. Separate the two copies by removing values above and below the waterline.
4. Generate surfaces above and below the waterline using the surf function. Include the optional 'EdgeColor' and 'none' arguments to generate a smooth surface without the mesh grid. Separate the surface above and below the waterline with different colors using the 'FaceColor' argument. The 'g' and 'b' arguments designate the colors above and below the waterline respectively.
5. Mirror the surface by changing the sign of the Y-values.

Listing 11: CreateSurface MATLAB function

```
function[] = CreateSurface(surfaceCurves)

% Extract X, Y and Z values
varX = squeeze(surfaceCurves(:,1,:));
varY = squeeze(surfaceCurves(:,2,:));
varZ = squeeze(surfaceCurves(:,3,:));

% Create copies of the Z matrix
zAbove = varZ;
zBelow = varZ;
% Separate the Z values above and below the waterline
zAbove(zAbove <= -0.05) = NaN;
zBelow(zBelow > 0.05) = NaN;

% Generate surface
surf(varX, varY, zAbove, 'FaceColor', 'g', 'EdgeColor','none'), hold on
surf(varX, varY, zBelow, 'FaceColor', 'b', 'EdgeColor','none')
% Mirror
surf(varX, -varY, zAbove, 'FaceColor', 'g', 'EdgeColor','none')
surf(varX, -varY, zBelow, 'FaceColor', 'b', 'EdgeColor','none')

end
```

4.1.10 SurfVec

Input Parameters:

- **surfaceCurves**: The curves defining the surface of the hull.

Output:

- **[surfX, surfY, surfZ]**: Matrices containing the X, Y and Z-values of the surface respectively.

This function generates matrices for the X, Y and Z-values of the surface through the following algorithm:

1. Define the X, Y and Z-values as individual matrices using the squeeze function to remove singleton dimensions.
2. Create mirrors of the matrices using the flip function.
3. Store the mirrored and original values in individual matrices, and change the sign of the mirrored Y-values.

The flip function is applied in the process because of the order of coordinates in the surfaceCurves, where each curve is generated from the keel to the deck. To avoid drawing an additional surface between the deck line and the keel, the mirrored surface is flipped. The mirrored valued are then stored in the output matrices first, with the original values second. This means the surface generation will be performed from the deck line on the mirrored side first, continuing downwards to the keel, after which the original surface will be generated as normal.

Listing 12: SurfVec MATLAB function

```
function[surfX, surfY, surfZ] = SurfVec(surfaceCurves)

% Extract X, Y and Z values
surfX = squeeze(surfaceCurves(:,1,:));
surfY = squeeze(surfaceCurves(:,2,:));
surfZ = squeeze(surfaceCurves(:,3,:));

% Create mirror matrices by flipping them
mirroredX = flip(surfX);
mirroredY = flip(surfY);
mirroredZ = flip(surfZ);
% Store the coordinates in new matrices
surfX = [mirroredX; surfX];
surfY = [-mirroredY; surfY];
surfZ = [mirroredZ; surfZ];

end
```

4.1.11 CreateDeckSurface

Input parameters:

- **deckCurve**: Deck reference curve. The surface is drawn between the original curve and its mirror.

Output:

- **[surfX, surfY, surfZ]**: Matrices containing the X, Y and Z-values of the surface respectively.

This function generates matrices for the X, Y and Z-values of the deck surface through the following algorithm:

1. Extract X, Y and Z-values into individual matrices.
2. Generate surface vectors by concatenating the original values with their mirrors by changing the sign of the Y-values.
3. Reshape the vectors into (n x 2) matrices using the reshape function.

The reshape function is applied in the process to prepare the surface for the use of the surf2stl function, as it will not accept column vectors.

Listing 13: CreateDeckSurface MATLAB function

```
function[surfX, surfY, surfZ] = CreateDeckSurface(deckCurve)

% Extract X, Y and Z values
surfX = deckCurve(:,1);
surfY = deckCurve(:,2);
surfZ = deckCurve(:,3);

% Generate deck surface vector with mirror
surfX = [surfX; surfX];
surfY = [surfY; -surfY];
surfZ = [surfZ; surfZ];

% Reshape vector into n x 2 matrix
surfX = reshape(surfX, [], 2);
surfY = reshape(surfY, [], 2);
surfZ = reshape(surfZ, [], 2);

end
```

4.2 External functions

4.2.1 Linear

This function updates the longitudinal curves defining the foreship and aft sections with linearly spaced X-values. This ensures that the curves have corresponding X-values in preparation of interpolation and scaling. There are commands within the function which can be uncommented to generate figures containing the original and updated curves to verify their shapes.

Input parameters:

- **bowBezierPoints**: The curve defining the bow.
- **foredeckBezierPoints**: The curve defining the foredeck.
- **aftKeelBezierPoints**: The curve defining the keel of the aft section.
- **aftDeckBezierPoints**: The curve defining the aft deck.

Output:

- **bowBezierPoints**: Updated bow curve points.
- **foredeckBezierPoints**: Updated foredeck curve points.
- **aftKeelBezierPoints**: Updated aft keel curve points.
- **aftDeckBezierPoints**: Updated aft deck curve points.

This function generates updated curve points through the following algorithm:

1. Import appropriate variables using the global keyword.
2. Create a new set of linearly spaced X-values using the linspace function.
3. Interpolate new Z-positions for the bow using the interp1 function with the bow Y and Z-values, and the new X-values as arguments.
4. Interpolate new Y-positions for the foredeck using the foredeck X and Y-values, and the new X-values as arguments.
5. Interpolate new Z-positions for the aft keel using the aft keel Y and Z-values, and the new X-values as arguments.
6. Interpolate new Y-positions for the aft deck using the aft deck X and Y-values, and the new X-values as arguments.
7. (Optional) Plot the original bow curves using the plot function.
8. Update the bow and foredeck curves with their new positions.
9. (Optional) Plot the updated bow points to verify the shape of the new curves.
10. (Optional) Plot the original aft curves.
11. Update the aft deck and keel with their new positions.
12. (Optional) Plot the updated aft points to verify the shape of the new curves.

Please note that each curve is two-dimensional at this stage. This means that for the bow and aft keel the indices for the Y and Z values are (:,1) and (:,2) respectively, even though 1 and 2 are more commonly used for X and Y. This gets fixed in a later stage, where the curves are translated to three dimensions.

Listing 14: Linear MATLAB function

```
1 function [bowBezierPoints,foredeckBezierPoints, aftKeelBezierPoints, ...
2     aftDeckBezierPoints] = Linear(bowBezierPoints,foredeckBezierPoints, ...
3     aftKeelBezierPoints, aftDeckBezierPoints)
4
5 % Import appropriate variables
6 global numPoints
7
8 % New set of X-values
```

```
9 xPositions = linspace(0, 1, numPoints);
10
11 % Interpolate Z positions bow
12 zPositionsBow = interp1(bowBezierPoints(:,1), bowBezierPoints(:,2), xPositions);
13 % Interpolate Y positions foredeck
14 yPositionsForedeck = interp1(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2),
     xPositions);
15
16 % Interpolate Z positions aft keel
17 zPositionsAftKeel = interp1(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2),
     xPositions);
18 % Interpolate Y positions aft deck
19 yPositionsAftDeck = interp1(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2),
     xPositions);
20
21 % figure
22 % plot(bowBezierPoints(:,1), bowBezierPoints(:,2), 'b-'), hold on
23 % plot(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2), 'r-')
24
25 % Create new Bezier curves from the updated positions
26 bowBezierPoints = [xPositions', zPositionsBow'];
27 foredeckBezierPoints = [xPositions', yPositionsForedeck'];
28
29 % plot(bowBezierPoints(:,1), bowBezierPoints(:,2), 'ko')
30 % legend('Original Bow Curve', 'CurvePoints after linearization', 'Location','
     northwest')
31 % title('Bow Curve Linearized in X-Direction')
32 % plot(foredeckBezierPoints(:,1), foredeckBezierPoints(:,2), 'ko'), hold off
33
34 % figure
35 % plot(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2), 'b-'), hold on
36 % plot(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2), 'r-')
37
38 % Create new Bezier curvs from the updated positions
39 aftKeelBezierPoints = [xPositions', zPositionsAftKeel'];
40 aftDeckBezierPoints = [xPositions', yPositionsAftDeck'];
41
42 % plot(aftKeelBezierPoints(:,1), aftKeelBezierPoints(:,2), 'ko')
43 % plot(aftDeckBezierPoints(:,1), aftDeckBezierPoints(:,2), 'ko'), hold off
44
45 end
```

4.2.2 TranslateForeship

Input parameters:

- **bowTipBezierPoints**: The curve defining the shape of the tip of the bow.
- **bowBezierPoints**: The curve defining the shape of the bow.
- **foredeckBezierPoints**: The curve defining the shape of the foredeck.

Output:

- **bowTipBezierPoints**: Updated bow tip curve.
- **bowBezierPoints**: Updated bow curve.
- **foredeckBezierPoints**: Updated foredeck curve.
- **bowTipTop**: New curve defining the upper bound of the bow tip.

This function translates the foreship curves to fit the ship's dimensions defined in the parameters section:

1. Import appropriate variables using the global keyword.
2. Translate bow tip.
   (a) Scale in the Y-direction.
   (b) Scale in the Z-direction.
   (c) Create a ship length vector.
   (d) Translate the tip curve to three dimensions by concatenating the ship length vector with the scaled Y and Z-values.
3. Generate bow tip top curve.
   (a) Define the curve's X-values.
   (b) Define the curve's Y-values.
   (c) Create a ship height vector.
   (d) Translate the curve to three dimensions by concatenating the scaled X and Y-values with the ship height vector.
4. Translate bow.
   (a) Scale in the X-direction.
   (b) Scale in the Z-direction.
   (c) Create a zero vector representing the center line of the ship.
   (d) Translate the curve to three dimensions by concatenating the scaled X-values, zero vector and the scaled Y-values.
5. Translate foredeck.
   (a) Scale in the X-direction.
   (b) Scale in the Y-direction.
   (c) Translate the curve to three dimension by concatenating the scaled X and Y-values with the ship height vector.

Listing 15: TranslateForeship MATLAB function

```
1 function [bowTipBezierPoints, bowBezierPoints, foredeckBezierPoints, ...
2     bowTipTop] = TranslateForeship(bowTipBezierPoints, ...
3     bowBezierPoints, foredeckBezierPoints)
4
5 global midshipBreadth midshipLength midshipHeight ...
6 foreshipLength tipHeight tipBottom tipBreadth ...
7 aftLength ...
8 numPoints DWL shipLength zeroColumn heightColumn
9
10
```

```
11 % Bow tip
12 % Fit to yz-plane
13 bowTipBezierPoints(:,1) = tipBreadth/2 * bowTipBezierPoints(:,1);
14 bowTipBezierPoints(:,2) = tipHeight * bowTipBezierPoints(:,2) - DWL + tipBottom;
15
16 % Translation in x-direction, generate 3D-coordinates
17 lengthColumn = shipLength * ones(numPoints,1);
18 bowTipBezierPoints = [lengthColumn, bowTipBezierPoints];
19 %----------------------------------------------------------
20
21 % Bow tip top
22 bowTopX = bowTipBezierPoints(:,1);
23 bowTopY = bowTipBezierPoints(:,2);
24 heightColumn = midshipHeight * ones(numPoints,1) - DWL;
25 bowTipTop = [bowTopX, bowTopY, heightColumn];
26 %----------------------------------------------------------
27
28 % Bow
29 % Fit to xz-plane
30 bowBezierPoints(:,1) = foreshipLength * bowBezierPoints(:, 1) + midshipLength +
       aftLength;
31 bowBezierPoints(:,2) = bowBezierPoints(:,2) * tipBottom - DWL;
32
33 % Generate 3D-coordinates
34 zeroColumn = zeros(numPoints,1);
35 bowBezierPoints = [bowBezierPoints(:,1), zeroColumn, bowBezierPoints(:,2)];
36 %----------------------------------------------------------
37
38 % Foredeck
39 % Fit to xy-plane
40 foredeckBezierPoints(:,1) = foreshipLength * foredeckBezierPoints(:, 1) +
       midshipLength + aftLength;
41 foredeckBezierPoints(:,2) = (-(midshipBreadth)/2  + tipBreadth/2) *
       foredeckBezierPoints(:, 2) + midshipBreadth/2;
42
43 % Generate 3D-coordinates
44 foredeckBezierPoints = [foredeckBezierPoints, heightColumn];
45
46 end
```

4.2.3 TranslateAft

Input parameters:

- **sternSectionBezierPoints**: The curve defining the shape of the stern.
- **aftKeelBezierPoints**: The curve defining the shape of the aft keel.
- **aftDeckBezierPoints**: The curve defining the shape of the aft deck.

Output:

- **sternSectionBezierPoints**: Updated stern section curve.
- **aftKeelBezierPoints**: Updated aft keel curve.
- **aftDeckBezierPoints**: Updated aft deck curve.
- **sternDeckPoints**: New curve defining the upper bound of the stern.

This function translates the aft curves to fit the ship's dimensions defined in the parameters section:

1. Import appropriate variables using the global keyword.
2. Translate stern section curve.
   (a) Scale in the Y-direction.
   (b) Scale in the Z-direction.
   (c) Translate the curve to three dimensions by concatenating a zero vector with the scaled Y and Z-values.
3. Translate aft keel.
   (a) Scale in the X-direction.
   (b) Scale in the Z-direction.
   (c) Translate the curve to three dimensions by concatenating the scaled X-values, a zero vector and the scaled Z-values.
4. Translate aft deck.
   (a) Scale in the X-direction.
   (b) Scale in the Y-direction.
   (c) Translate the curve to three dimensions by concatenating the scaled X and Y-values with a height vector.
5. Generate stern deck curve.
   (a) Define the curve's Y-values.
   (b) Translate the curve to three dimensions by concatenating a zero vector, the Y-values and a height vector.

Listing 16: TranslateAft MATLAB function

```
1 function [sternSectionBezierPoints,aftKeelBezierPoints, aftDeckBezierPoints, ...
2     sternDeckPoints] = TranslateAft(sternSectionBezierPoints,aftKeelBezierPoints,
        aftDeckBezierPoints)
3
4 global midshipBreadth  midshipHeight ...
5 aftLength sternBreadth aftHeight midAftHeightDiff ...
6 numPoints DWL zeroColumn heightColumn
7
8 % Stern section
9 % Fit to yz-plane
10 sternSectionBezierPoints(:,1) = sternBreadth/2 * sternSectionBezierPoints(:,1);
11 sternSectionBezierPoints(:, 2) = aftHeight * sternSectionBezierPoints(:, 2) +
     midAftHeightDiff - DWL;
12
13 % Translation in x-direction, create 3D-coordinates
14 sternSectionBezierPoints = [zeroColumn, sternSectionBezierPoints];
```

```
15 %------------------------------------------------------------
16
17 % Aft keel
18 % Fit to xz-plane
19 aftKeelBezierPoints(:,1) = aftLength * aftKeelBezierPoints(:, 1);
20 aftKeelBezierPoints(:, 2) = -midAftHeightDiff * aftKeelBezierPoints(:, 2) +
       midAftHeightDiff - DWL;
21
22 % Create 3D-coordinates
23 aftKeelBezierPoints = [aftKeelBezierPoints(:,1), zeroColumn, aftKeelBezierPoints(:, 2)
       ];
24 %------------------------------------------------------------
25
26 % Aft deck
27 % Fit to xy-plane
28 aftDeckBezierPoints(:,1) = aftLength * aftDeckBezierPoints(:,1);
29 aftDeckBezierPoints(:,2) = (midshipBreadth/2 - sternBreadth/2) * aftDeckBezierPoints
       (:,2) + sternBreadth/2;
30
31 % Translation in z-direction, create 3D-coordinates
32 aftDeckBezierPoints = [aftDeckBezierPoints, heightColumn];
33 %------------------------------------------------------------
34
35 % Stern deck
36 sternDeckY = linspace(0,sternBreadth/2,numPoints)';
37 sternDeckPoints = [zeroColumn, sternDeckY, heightColumn];
38
39 end
```

4.2.4 TranslateMidship

Input parameters:

- **midshipSectionBezierPoints**: The curve defining the shape of the midship section.

Output:

- **midshipSectionBezierPointsAft**: Updated midship section curve, translated to the rear of the midship.
- **midshipSectionBezierPointsFront**: Updated midship section curve, translated to the front of the midship.
- **midshipKeelPoints**: New curve defining the keel of the midship.
- **midshipDeckPoints**: New curve defining the deck of the midship.

This function translates the midhsip curves to fit the ship's dimensions defined in the parameters section:

1. Import appropriate variables using the global keyword.
2. Translate the midship section curve.
   (a) Scale the midship curve in the Y-direction.
   (b) Scale the midship curve in the Z-direction.
   (c) Create a length vector containing the aft length of the ship.
   (d) Create the aft curve by concatenating the length vector with the scaled Y and Z-values.
   (e) Create a new length vector containing the length of both the aft section and the midship.
   (f) Create the front curve by concatenating the length vector with the scaled Y and Z-values.
3. Generate the midship keel line.
   (a) Create a vector containing the X-values of the midship.
   (b) Create a midship draught vector.
   (c) Create the midship keel line by concatenating the X-values, a zero vector and the draught vector.
4. Generate the midship deck line.
   (a) Initialize the midship deck line to be equal to the midship keel line.
   (b) Adjust the Y-values to fit the breadth.
   (c) Adjust the Z-values to fit the height.

Listing 17: surf2stl MATLAB function

```
1 function [midshipSectionBezierPointsAft, midshipSectionBezierPointsFront,
    midshipKeelPoints, ...
2     midshipDeckPoints] = TranslateMidship(midshipSectionBezierPoints)
3
4 global midshipBreadth midshipLength midshipHeight ...
5 aftLength  ...
6 numPoints DWL
7
8 % % Midship section
9 % % Fit to yz-plane
10 midshipSectionBezierPoints(:,1) = midshipBreadth/2 * midshipSectionBezierPoints(:,1);
11 midshipSectionBezierPoints(:,2) = (midshipHeight) * midshipSectionBezierPoints(:,2) -
    DWL;
12
13 % Midship section aft
14 % Translation in x-direction, generate 3D-coordinates
15 lengthColumn = (aftLength) * ones(numPoints,1);
16 midshipSectionBezierPointsAft = [lengthColumn, midshipSectionBezierPoints];
17 %-----------------------------------------------------------
18
19 % Midship section front
20 % Translation in x-direction, generate 3D-coordinates
```

```
21 lengthColumn = (aftLength + midshipLength) * ones(numPoints,1);
22 midshipSectionBezierPointsFront = [lengthColumn, midshipSectionBezierPoints];
23 %-------------------------------------------------------------
24
25 % Midship keel line
26 keelX = linspace(0,midshipLength,numPoints)' + aftLength;
27 midshipDraught = -DWL*ones(numPoints,1);
28 midshipKeelPoints = [keelX, zeros(numPoints,1), midshipDraught];
29 %-------------------------------------------------------------
30
31 % Midship deck line
32 midshipDeckPoints = midshipKeelPoints;
33 midshipDeckPoints(:,2) = ones(numPoints,1) * midshipBreadth/2;
34 midshipDeckPoints(:,3) = ones(numPoints,1) * midshipHeight - DWL;
35
36 end
```

4.2.5 surf2stl

It is referred to [2] for documentation.

Listing 18: TranslateMidship MATLAB function

```
function surf2stl(filename,x,y,z,mode)
%SURF2STL   Write STL file from surface data.
%   SURF2STL('filename',X,Y,Z) writes a stereolithography (STL) file
%   for a surface with geometry defined by three matrix arguments, X, Y
%   and Z.  X, Y and Z must be two-dimensional arrays with the same size.
%
%   SURF2STL('filename',x,y,Z), uses two vector arguments replacing
%   the first two matrix arguments, which must have length(x) = n and
%   length(y) = m where [m,n] = size(Z).  Note that x corresponds to
%   the columns of Z and y corresponds to the rows.
%
%   SURF2STL('filename',dx,dy,Z) uses scalar values of dx and dy to
%   specify the x and y spacing between grid points.
%
%   SURF2STL(...,'mode') may be used to specify the output format.
%
%     'binary' - writes in STL binary format (default)
%     'ascii'  - writes in STL ASCII format
%
%   Example:
%
%     surf2stl('test.stl',1,1,peaks);
%
%   See also SURF.
%
%   Author: Bill McDonald, 02-20-04

error(nargchk(4,5,nargin));

if (ischar(filename)==0)
    error( 'Invalid␣filename');
end

if (nargin < 5)
    mode = 'binary';
elseif (strcmp(mode,'ascii')==0)
    mode = 'binary';
end

if (ndims(z) ~= 2)
    error( 'Variable␣z␣must␣be␣a␣2-dimensional␣array' );
end

if any( (size(x)~=size(z)) | (size(y)~=size(z)) )

    % size of x or y does not match size of z

    if ( (length(x)==1) & (length(y)==1) )
        % Must be specifying dx and dy, so make vectors
        dx = x;
        dy = y;
        x = ((1:size(z,2))-1)*dx;
        y = ((1:size(z,1))-1)*dy;
    end

    if ( (length(x)==size(z,2)) & (length(y)==size(z,1)) )
        % Must be specifying vectors
        xvec=x;
        yvec=y;
        [x,y]=meshgrid(xvec,yvec);
    else
```

```
62          error('Unable␣to␣resolve␣x␣and␣y␣variables');
63      end
64
65  end
66
67  if strcmp(mode,'ascii')
68      % Open for writing in ascii mode
69      fid = fopen(filename,'w');
70  else
71      % Open for writing in binary mode
72      fid = fopen(filename,'wb+');
73  end
74
75  if (fid == -1)
76      error( sprintf('Unable␣to␣write␣to␣%s',filename) );
77  end
78
79  title_str = sprintf('Created␣by␣surf2stl.m␣%s',datestr(now));
80
81  if strcmp(mode,'ascii')
82      fprintf(fid,'solid␣%s\r\n',title_str);
83  else
84      str = sprintf('%-80s',title_str);
85      fwrite(fid,str,'uchar');         % Title
86      fwrite(fid,0,'int32');           % Number of facets, zero for now
87  end
88
89  nfacets = 0;
90
91  for i=1:(size(z,1)-1)
92      for j=1:(size(z,2)-1)
93
94          p1 = [x(i,j)     y(i,j)     z(i,j)];
95          p2 = [x(i,j+1)   y(i,j+1)   z(i,j+1)];
96          p3 = [x(i+1,j+1) y(i+1,j+1) z(i+1,j+1)];
97          val = local_write_facet(fid,p1,p2,p3,mode);
98          nfacets = nfacets + val;
99
100         p1 = [x(i+1,j+1) y(i+1,j+1) z(i+1,j+1)];
101         p2 = [x(i+1,j)   y(i+1,j)   z(i+1,j)];
102         p3 = [x(i,j)     y(i,j)     z(i,j)];
103         val = local_write_facet(fid,p1,p2,p3,mode);
104         nfacets = nfacets + val;
105
106     end
107 end
108
109 if strcmp(mode,'ascii')
110     fprintf(fid,'endsolid␣%s\r\n',title_str);
111 else
112     fseek(fid,0,'bof');
113     fseek(fid,80,'bof');
114     fwrite(fid,nfacets,'int32');
115 end
116
117 fclose(fid);
118
119 disp( sprintf('Wrote␣%d␣facets',nfacets) );
120
121
122
123 % Local subfunctions
124
125 function num = local_write_facet(fid,p1,p2,p3,mode)
126
127 if any( isnan(p1) | isnan(p2) | isnan(p3) )
```

```
    num = 0;
    return;
else
    num = 1;
    n = local_find_normal(p1,p2,p3);

    if strcmp(mode,'ascii')

        fprintf(fid,'facet normal %.7E %.7E %.7E\r\n', n(1),n(2),n(3) );
        fprintf(fid,'outer loop\r\n');
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p1);
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p2);
        fprintf(fid,'vertex %.7E %.7E %.7E\r\n', p3);
        fprintf(fid,'endloop\r\n');
        fprintf(fid,'endfacet\r\n');

    else

        fwrite(fid,n,'float32');
        fwrite(fid,p1,'float32');
        fwrite(fid,p2,'float32');
        fwrite(fid,p3,'float32');
        fwrite(fid,0,'int16');  % unused

    end

end


function n = local_find_normal(p1,p2,p3)

v1 = p2-p1;
v2 = p3-p1;
v3 = cross(v1,v2);
n = v3 ./ sqrt(sum(v3.*v3));
```

5 References

1. MATLAB Documentation, https://se.mathworks.com/help/matlab/. [Accessed: January 9, 2024]
2. Bill McDonald, "surf2stl," MATLAB Central File Exchange, 2024. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/4512-surf2stl. [Accessed: March 20, 2024].