5-2021

Design of an Ice-Class Propeller for the MV Yahtse, an Icebreaking, Car and Cargo, RoRo Ferry

Mara Kramer
University of New Orleans

Design of an Ice-Class Propeller for the MV Yahtse, an Icebreaking, Car and Cargo, RoRo Ferry

An Honors Thesis

Presented to

the School of Naval Architecture and Marine Engineering

of the University of New Orleans

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Science, with University High Honors

and Honors in Naval Architecture and Marine Engineering

by

Mara Kramer

May 2021

Table of Contents

List of Tables

List of Figures

Abstract

During early-stage ship design, a propulsion system must be matched with data from a resistance and propulsion analysis to determine the propulsion power required for the vessel to run at its design speed. Typically, this process is completed within NavCAD; however, NavCAD does not have a method to calculate icebreaking resistance or design a propeller to meet the ice-class criteria stipulated by the International Association of Classification Societies (IACS). This paper displays and discusses Python scripts written to complete the resistance and propulsion analysis, propeller optimization, and propeller structural design meeting IACS criteria for an icebreaking, RoRo car and cargo ferry, the MV Yahtse. This code was designed to complete propeller design for the preliminary design stage of the vessel; however, the code can be modified for any stage of design as well as for use with any icebreaking vessel with principal characteristics that fall within the parameters required for the use of Holtrop and Mennen's resistance and propulsion analysis method. The Python scripts were proven to be able to generate resistance and propulsion analysis results comparable to the results found from NavCAD as well as design two propellers suitable for the MV Yahtse that pass the criteria imposed by the IACS ice-class regulations.

Keywords: resistance, icebreaking, propeller, Python, Wageningen B-Series

1. Introduction

Typically, during the ship design process, engineers will match a propeller with their ship based on results obtained through a resistance and propulsion analysis completed either with a simulation tool (such as NavCad) or model testing. This step is critical for both early-stage design and subsequent iterations as it determines the amount of propulsive power the vessel requires to operate at its design speed, and from this information the engines can be sized. However, for ships with a unique design, a propeller must be designed that considers any special operating conditions or missions that the vessel is designed to handle without compromising performance. The MV Yahtse is one of these outlying cases. It is an overnight, car and cargo, roll on-roll off (RoRo), twin-screw (two propellers) ferry designed to service the Alaskan coast from the southwest Aleutian Islands to the north slope town of Utqiaġvik (formerly known as Barrow) which lies within the Arctic Circle. Due to the geographical range over which the MV Yahtse will provide service, the hull will be Ice-Class 3 according to the International Association of Classification Societies' (IACS) "Requirements Concerning Polar Class" [1] and any additional American Bureau of Shipping (ABS) guidelines in "Guidance Notes on Ice Class" [2]. This ice class operational requirement imposes special design considerations upon the propulsion system. To aid in the design of the propeller regarding these requirements, a propeller design tool for the preliminary vessel design stage has been written in Python. This paper aims to breakdown the operation of the Python script as well as the theory and methodology for the propeller design methods used within.

To write the propeller design tool, five major "steps" had to be coded. The first step completed a resistance and propulsion analysis using Holtrop and Mennen's method. The second step used the Wageningen B systematic propeller series developed by the Netherlands Ship Model

Basin (MARIN) and the results from the resistance and propulsion analysis of the MV Yahtse to optimize a propeller for the vessel. Next, the geometry of the propeller was calculated. Then, using the propeller geometry, the maximum stresses acting on the blade were calculated. Finally, using the IACS requirements, the required thickness of the blade edges as well as an evaluation on whether the blade would meet the stress requirements was determined.

The code developed for this project is intended to be run in an iterative manner, assuming the propeller would not meet the structural criteria immediately. The propeller was designed with the ship running at service speed without any icebreaking resistance. This decision was made since the ship will only be using its icebreaking capabilities in a few specific scenarios (i.e., winter cargo deliveries to Utqiaġvik). This set of input values also served as the basis for a comparison to NavCAD, as NavCAD does not have any icebreaking resistance calculation tools. A second set of inputs was used to design and test the structure of the propeller. This set of inputs used the propeller parameters optimized from the first run and resistance and propulsion results considering the ship running at design speed while icebreaking. This is a very unrealistic scenario, as the power required to run at 15 knots through a meter of ice is unreasonable for a vessel the size of the MV Yahtse; however, due to the absurdity, this condition will be sure to blanket any other operating condition that could possibly require the highest propulsion power. If the designed propeller can withstand the forces imposed by this extreme, then it stands to reason that there should be no concerns about the propeller's structural integrity for all normal operation conditions.

Utilizing Holtrop and Mennen's method for the resistance and propulsion analysis as well as Wageningen B-Series propeller data, two propellers were able to be designed to work optimally on the MV Yahtse and further geometry was developed such that these propellers met the ice-class criteria. However, the developed propellers are meant solely for a preliminary design and there are plenty of areas in which the processes and code can be improved for later design stages.

2. Resistance and Propulsion Analysis

A resistance and propulsion analysis is an essential part of the ship design process. There have been many methods developed to complete this process, optimized for a wide variety of ships. However, they all fundamentally complete the same process. Principal characteristics of a vessel's hull form (and superstructure as necessary) act as input values and resistance estimates are made using developed formulas suitable for that vessel type. Then, using established theory, the thrust values required to propel the vessel over a range of speeds (including the design speed) are calculated. This data, in combination with propeller's geometric data and characteristics, will provide data points for required power and the efficiency of the propulsion system, completing the resistance and propulsion analysis.

There is no complete resistance estimation method dedicated to ice-breaking vessels, so the resistance and propulsion analysis was completed using Holtrop and Mennen's method [3,4] with the addition of the Jeong formulas for icebreaking resistance. Holtrop and Mennen's method was chosen as it is a complete resistance and propulsion method developed using statistical regression on data of both full-scale ships and model tests completed at MARIN. Due to the vast range of data used to develop the method, Holtrop and Mennen's method provides accurate results for a wide range of ships. In general, this method will work for monohull vessels that fall approximately into the following range of values for Froude number, prismatic coefficient, and length-to-beam ratio [5]:

$$Fr \leq 0.45$$

$$0.55 \leq C_P \leq 0.85 \qquad \text{(Eq. 1)}$$

$$3.9 \leq \frac{L}{B} \leq 9.5$$

The MV Yahtse meets these three criteria, so Holtrop and Mennen's method was used as the basis for the resistance and propulsion analysis.

The selection of the icebreaking resistance formula followed much more simple reasoning. All the older icebreaking formulas were developed using detailed hull parameters (stem angle, flare angle, buttock angle, etc.) as variables, whereas Jeong et. al, in "Ice Resistance Prediction for Standard Icebreaker Model Ship" [6], proposed a resistance estimation method that did not require the same level of hull detail. This was critical for the stage of design in which the resistance and propulsion analysis was completed, as the hull form was not yet designed with certainty for such items as the stem angle, flare angle, etc. Additionally, as the icebreaking mission of the standard icebreaker model used in Jeong et. al's study is the same as the MV Yahtse's (breaking first-year ice), the Jeong formulas were determined to be an adequate fit for an icebreaking resistance estimate. The Jeong formulas are as follows:

$$R_I = 13.14V^2 + C_B \Delta\rho g h_i B T + C_C {F_h}^{-\alpha} \rho_i B h_i V^2 + C_{BR} {S_N}^{-\beta} \rho_i B h_i V^2 \quad \text{(Eq. 2)}$$

$$F_h = \frac{V}{\sqrt{g h_i}} \quad \text{(Eq. 3)}$$

$$S_N = \frac{V}{\sqrt{\frac{\sigma_f h_i}{\rho_i B}}} \quad \text{(Eq. 4)}$$

where C_B=0.5 is the coefficient of ice buoyancy resistance, C_C=1.11 is the coefficient of ice clearing resistance, and C_{BR}=2.73 is the coefficient of the ice breaking resistance; F_h is the Froude number of the ice thickness, h_i, and S_N is the strength number. Finally, $\alpha = 1.157$, $\beta = 1.54$, ρ_i is the ice density and $\Delta\rho$ is the difference between the ice and water densities, V is the ship speed, and σ_f is the flexural strength of the ice.

The full resistance and propulsion Python script can be seen in Appendix A, but next few paragraphs aim to summarize the general structure of the script and the outcome of each portion.

The first portion of the code is dedicated to defining and calculating the ship hull characteristics necessary to complete Holtrop and Mennen's method. This involves correcting several values to the same frame of reference used by Holtrop and Mennen as well as estimating the remaining required values as necessary, which depends current stage of design for which the resistance and propulsion estimate is being completed. For a preliminary estimate, a good portion of the ship particulars will likely still be estimated using regression formulas or formulas outlined in Holtrop and Mennen's method. For a later stage resistance and propulsion analysis, it is expected that all the necessary input values are measured straight from a completed hull model.

The next portion of the code completed the resistance estimate. This is simply a long string of equations for resistance components or coefficients that ends in the total resistance being calculated with the following formula:

$$R_T = (1 + k)R_F + R_{APP} + R_A + R_W + R_{TR} + R_{AA} + R_I \qquad \text{(Eq. 5)}$$

where k is the ITTC form factor, R_F is the frictional resistance, R_{APP} is the total appendage resistance, R_A is the correlation allowance resistance, R_W is the wave resistance, R_{TR} is the transom resistance, R_{AA} is the air resistance, and R_I is the icebreaking resistance found from the Jeong formulas.

The final part of Holtrop and Mennen's method completes a powering estimate by estimating the wake fraction, thrust deduction fraction, advance speed, and required thrust for the vessel with regression formulas developed by Holtrop and Mennen. Much like the resistance components, these values would normally be found during a model test and then scaled up for the full-size ship. However, due to the cost, ship models are not developed for feasibility studies and the preliminary design of a vessel, so these estimation methods serve to help engineers complete a vital part of the ship design process with high accuracy (typically within 10% error) at a fraction of the time and cost.

Holtrop and Mennen do provide a method for estimating the relative rotative efficiency and open water efficiency of a Wageningen B-Series propeller in their paper; however, they simply provide some small corrections for a full-scale propeller built upon the work completed by Oosterveld and Van Oossanen in “Further Computer-Analyzed Data of the Wageningen B-Screw Series” [7]. Additionally, Holtrop and Mennen’s work is only suitable for a situation in which the propeller characteristics are already known. Therefore, the powering estimate of the vessel will be completed as part of the propeller optimization using Oosterveld and Van Oossanen’s work.

Before completing the propeller optimization and structural design, it is critical to determine that (a) the code developed thus far produces accurate results for Holtrop and Mennen’s method and (b) Holtrop and Mennen’s method serves as a good resistance and propulsion analysis method for the MV Yahtse. The first concern serves to simply check the correctness of the results generated by the Python script; however, the second concern exists on a much more theoretical plane. While Holtrop and Mennen’s method was developed from regression analyses of a wide variety of ships, giving the method its broad range of applicability, this general applicability can sometimes cause Holtrop and Mennen’s method to be rather inaccurate for unique vessel designs that do not follow the general trends found in the relations used by Holtrop and Mennen during their regression analyses. The MV Yahtse is a unique vessel design, as very few passenger vessels operate within the arctic circle, much less RoRo car ferries. Therefore, before proceeding, the resistance results from Python were compared to several resistance estimation methods in NavCAD, a resistance, propulsion, and propeller-selection software. Four methods were compared in NavCAD – Holtrop and Mennen, Andersen, Fung Transom-Stern (CRTS), and Fung High-

Speed Transom-Stern (HSTS). Andersen's method is named after the work published by Andersen and Guldhammer developing a numerical method for Guldhammer's earlier graphical procedure, "A Computer-Oriented Power Prediction Procedure" [8]. The last two methods were developed by Fung for early-stage resistance prediction of general transom-stern hulls and high-speed transom stern hulls in "Resistance and Powering Prediction for Transom-Stern Hull Forms During Early-Stage Ship Design" [9] and "Revised Speed-Dependent Powering Predictions for High-Speed Transom-Stern Hull Forms" [10] respectively. The MV Yahtse was evaluated by NavCAD to fulfill the parameters for each of these methods and each method was manually reviewed to ensure that the MV Yahtse fell within the intended vessel type(s) for each method. Figure 1 shows the resistance components and total resistance as calculated with the Python script and Figure 2 shows the comparison of all four methods in NavCAD. As NavCAD does not have any method to calculate icebreaking resistance, for the purpose of comparison, the icebreaking resistance does not factor into the total resistance calculated for Figure 1 even though it is displayed on the graph as a component.

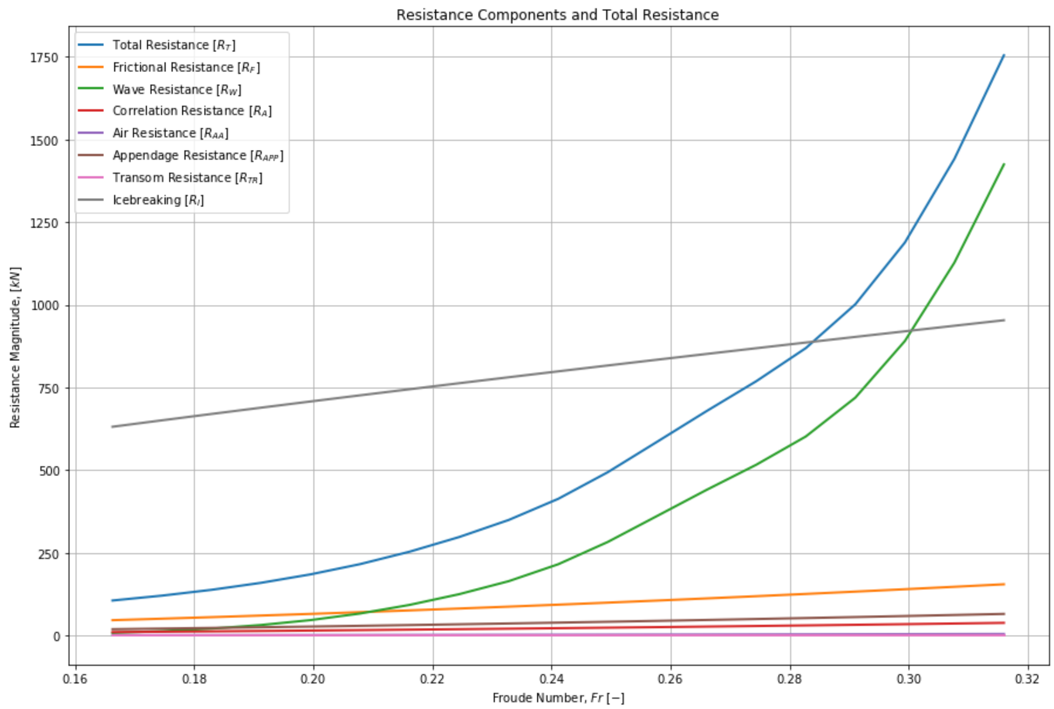

Figure 1: Resistance components and total resistance (without icebreaking) from Python.

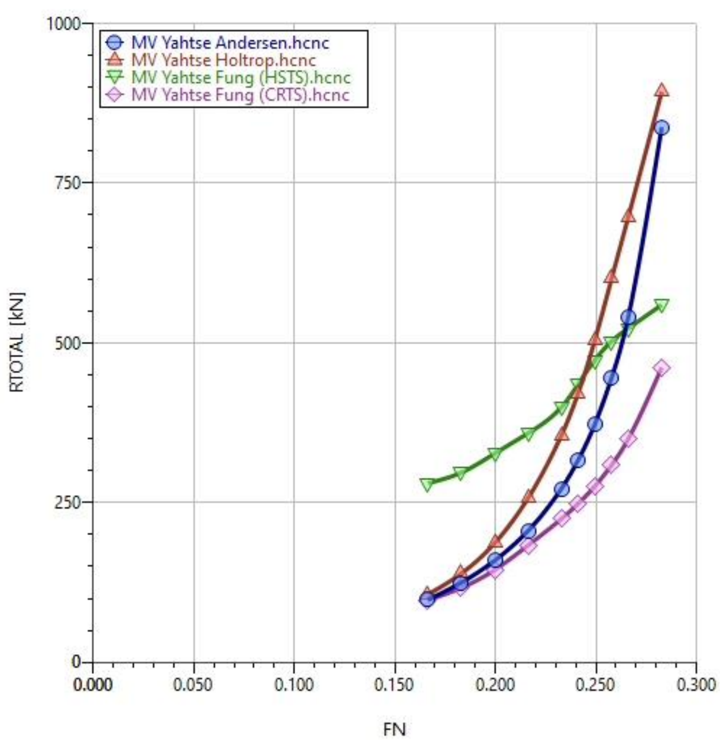

Figure 2: Total resistance for each method calculated using NavCAD.

Table 1 presents a comparison of the total resistance values calculated by Holtrop and Mennen's method in Python and in NavCAD. For a complete comparison of the resistance components and total resistance calculated via Python and for all methods run in NavCad, please see Appendices D-H.

Table 1: Comparison of Holtrop and Mennen total resistance values.

Speed (kt)	Python R_T with icebreaking (kN)	Python R_T w/o icebreaking (kN)	NavCAD R_T (kN)	% Error
10.00	737.756	106.236	106.12	0.109
11.00	808.749	138.548	138.87	0.232
12.00	893.236	185.271	186.56	0.691
13.00	998.766	253.848	256.96	1.211
14.00	1130.821	349.677	355.67	1.685
14.50	1212.564	413.558	421.46	1.875
15.00	1310.723	494.009	504.64	2.107
15.50	1420.611	586.339	600.62	2.378
16.00	1530.893	679.205	697.34	2.601
17.00	1755.402	869.285	893.46	2.706

Table 1 proves that the Python code meets the first condition required before proceeding on to the propeller development – the Holtrop and Mennen resistance estimate compares well to that calculated through NavCAD. As the vessel speed increases, the percent error between the two sets of results does increase, but this is expected as any small differences between the two methods become magnified; however, even at the largest speed of 17 knots, the error is still well under 5%. For the preliminary stage of vessel design, a 5% error is very much acceptable, so this proves that Holtrop and Mennen's method as coded in Python is working properly. Secondly, this data establishes that Holtrop and Mennen's method is a good resistance and propulsion analysis method for the MV Yahtse. Typically for a preliminary resistance estimate, a conservative estimate is best as it is unwise to risk under-designing the vessel for the missions and specifications it is to meet. In this regard, looking at Figure 1, there are two conservative options to choose from. Fung (HSTS) is greatly conservative for lower Froude numbers (vessel speeds) and at the higher speeds, Holtrop and Mennen’s method outstrips it. However, Fung’s (HSTS) method can be discarded in favor of Holtrop and Mennen’s method as, although NavCAD suggests that Fung’s (HSTS) method is a good fit, reading the original literature, this method is clearly meant for high-speed (large Froude number) vessels and the MV Yahtse does not meet this criterion. Therefore, when comparing Holtrop and Mennen’s method to several other prediction methods, Holtrop and Mennen’s method is still the best choice for preliminary vessel design and the results from this method can be used for propeller optimization.

3. Propeller Optimization

The propeller optimization is completed within the same script used for Holtrop and Mennen's method as it is necessary to obtain and use propeller characteristics to complete the powering estimate (as noted in the section above). Due to the simplicity in design and abundance of research done on them, Wageningen B-Series propellers were chosen for the MV Yahtse. The geometry of this series is very well documented so that optimizing a propeller of this series for any type of ship is possible. Additionally, to aid in maneuvering into all manner of ports, many of them simplistic or practically non-existent, it was determined that the propellers would have to be controllable pitch propellers (CPP) which sets an additional criterion to have an expanded area ratio no greater than 0.75. This criterion ensures that each blade can rotate a compete 180^{o} without contacting another blade, which would prevent the propeller from providing fully reversible thrust.

To implement propeller optimization code into the resistance and propulsion estimate, two supplementary scripts were written. The first script (shown in Appendix I) uses the open water thrust (K_T) and torque (K_Q) curve polynomials defined by Oosterveld and Van Oossanen to create functions for the open water efficiency and the self-propulsion point (the operating point for a propeller at an advance speed). Finally, the first script contains a function to calculate the minimum area ratio required by Burrill's criteria for cavitation [11]. For merchant vessels, Burrill's 5% back cavitation limit curve was chosen meaning that up to 5% of the back of the blade can be covered with cavitation. This limit is expressed with the following regression curve and equations:

$$\tau_c = 0.715{\sigma_b}^{0.814} - 0.437 \quad \text{(Eq. 6)}$$

$$\sigma_b = \frac{p_0 - p_v}{0.5\rho {v_1}^2} \quad \text{(Eq. 7)}$$

$$p_0 = p_A + \rho g e \quad \text{(Eq. 8)}$$

$$v_1 = \sqrt{{v_A}^2 + (0.7\pi n D)^2} \quad \text{(Eq. 9)}$$

In Equations 6-9, p_v is the vapor pressure of water, p_A is the atmospheric pressure, and e is the propeller shaft submergence depth. Using these supplemental equations, the minimum required area ratio to meet the set cavitation criteria as defined by Burrill is

$$\left(\frac{A_E}{A_0}\right)_{req} = \frac{T}{0.5\rho v_1{}^2\tau_c\left(1.067-\frac{0.229P}{D}\right)\frac{\pi D^2}{4}} \quad \text{(Eq. 10)}$$

The second script (shown in Appendix J) defines a function to iterate and converge upon an optimal propeller considering the self-propulsion point and the minimum area ratio. The propeller optimization is done using what is known as “design task 4” which uses the inputs of propeller blade number (Z), propeller diameter (D), required thrust (T), and speed of advance (v_A) to optimize the pitch-diameter ratio and expanded area ratio of the propeller [12]. This design task was chosen since it is the most logical task for preliminary ship design. Compared to the other characteristics, the number of blades is slightly more arbitrary and for the MV Yahtse, the number of blades was chosen by looking at the propeller characteristics of vessels within the Alaskan Marine Highway System (AMHS). The diameter of the propeller was chosen as the maximum propeller diameter that would work for the hull form of the MV Yahtse to maximize efficiency. The optimization functions defined in the two supplementary scripts were imported for use in the Holtrop and Mennen script.

Within the resistance and propulsion estimation code, the optimization functions were imported and run with the appropriate input values from Holtrop and Mennen’s method. To use the optimization function, initial guesses for the pitch-diameter ratio and expanded area ratio had to be calculated to give the algorithm a starting point. The initial pitch-diameter ratio was simply given a common value, but the initial expanded area ratio was calculated using Keller’s formula which was developed to calculate an initial expanded area ratio that would avoid cavitation [13].

$$\left(\frac{A_E}{A_0}\right)_{req} = \frac{(1.3+0.3Z)T}{(p_0-p_v)D^2} + K \qquad \text{(Eq. 11)}$$

After running the optimization function, the self-propulsion points of the propellers were found as defined in the Wageningen B-Series polynomial functions.

The optimal propeller characteristics were calculated without icebreaking resistance as the propeller of a vessel should always be designed to operate optimally in the normal service condition and the MV Yahtse is only expected to operate as an icebreaking ship for a small portion of the year for a few route locations. The propeller characteristics for both propellers as well as the optimum efficiency at the design speed are shown below in Table 2.

Table 2: Optimum propeller characteristics for the MV Yahtse.

Optimum Propeller Characteristics	Value
Number of Blades (Z)	5
Diameter (D)	3.048 m
Pitch-Diameter Ratio (PD)	0.7568
Expanded Area Ratio (ar)	0.7520
Open Water Efficiency at Design Speed (η_{OS})	0.4542
RPM at Design Speed (n)	338.3 rpm

The open water chart with self-propulsion points marking the K_T, K_Q, and open water efficiency (η_O) values for each speed is shown below in Figure 3.

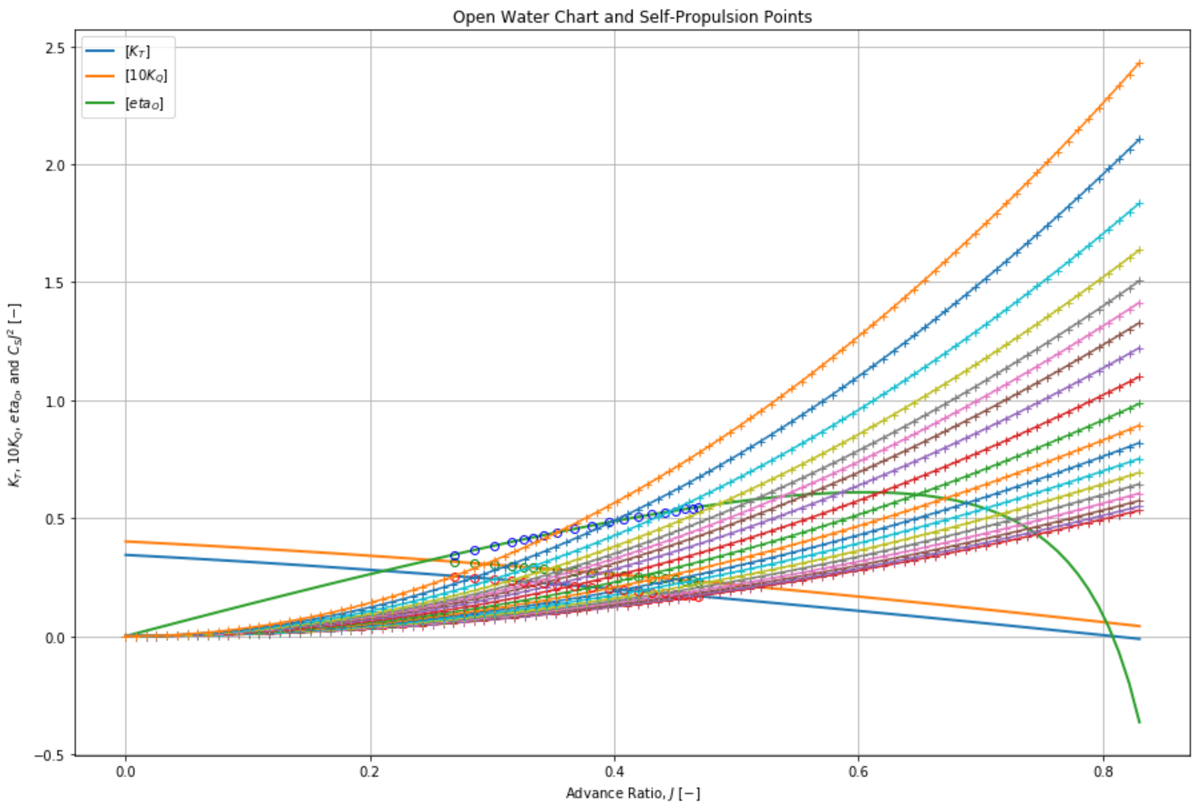

Figure 3: Open water chart with self-propulsion points.

An open water chart plots the thrust, torque, and open water efficiency curves over a range of advance ratios, J.

$$J = \frac{v_A}{nD} \tag{Eq. 12}$$

The design constant curves for each advance speed are then plotted on the graph. For design task 4, the design constant is defined as

$$\left[\frac{K_T}{J^2}\right] = \frac{T}{\rho D^2 {v_A}^2} \tag{Eq. 13}$$

For plotting, the design constant is multiplied by the denominator on the left-hand side of the equation which, for this design task, is J^2. The intersection between each of these design curves is marked where it intersects with the K_T polynomial curve and these points of intersection are extrapolated vertically to the K_Q and efficiency curves. This series of self-propulsion points can be seen marked on Figure 3 with open circles. The optimum propeller efficiency is then chosen from this data by finding the efficiency self-propulsion point at the design speed.

4. Propeller Structural Analysis and Design

While these propeller characteristics work in theory, IACS ice class rules impose additional structural requirements upon the propeller to ensure that both the material used for the propeller and the blades themselves are strong enough to withstand the forces and stresses imposed upon them during icebreaking. However, to complete the structural analysis of the propeller, the maximum stress acting on the blade must be calculated. The stress on the blade will be greatest at the root of the blade, so the IACS criteria can be completed by evaluating the blade using this maximum stress. Normally, the maximum stress is found using finite element analysis. However, for early-stage design without a propeller model, this is not possible. Therefore, an alternative method for finding the maximum stress had to be used. The blade stress was calculated using Tables 2 and 3 from Section 3: Propeller Blade Stress from “Marine Engineering Vol. 1” [14]. The bending moment and blade stress calculation tables are shown below in Figure 4.

TABLE 2.—BENDING MOMENT CALCULATION

Item	Symbol	Formula	Units	Fig. 1	Fig. 6	Fig. 2
Diameter over blade tips	D	....	in.	210	135	236
Diameter at root section	d	....	in.	35.5	24	50
Thrust moment arm factor	K_T	$0.66D-d$	in.	103	65	106
Shaft horsepower per screw	P	....	hp	4000	25,000	4000
Propulsive efficiency	e	....	...	0.77	0.57	0.77
Speed of ship	v	....	knots	14.5	37.8	13.6
Number of blades	n	....	...	4	3	4
Thrust deduction factor	$1-t$	....	...	0.80	0.99	0.81
Moment due to thrust	M_T	$163PeK_T/vn(1-t)$	in-lb	1,120,000	1,350,000	1,210,000
Shaft revolutions per minute	N	....	rpm	87	390	79
Developed area of propeller	A_d	....	sq ft	110	73.6	121.6
Maximum thickness at root	t_r	....	in.	7.12	6.88	8.40
Centrifugal force	F	$DN^2A_dt_r/7450n$	lb	41,500	462,000	50,000
Arm due to rake	r	....	in.	4	2	6
Moment due to rake	M_R	rF	in-lb	166,000	924,000	300,000
Total axial moment	M_A	$M_T + M_R$	in-lb	1,290,000	2,270,000	1,510,000
Torque moment arm ratio	K_Q	$1-1.67d/D$		0.72	0.70	0.65
Moment due to torque	M_Q	$63{,}000PK_Q/nN$	in-lb	522,000	942,000	518,000
Arm due to skewback	b	....	in.	−1	0	0
Moment due to skewback	M_S	bF	in-lb	−41,500	0	0
Total circumferential moment	M_C	$M_Q - M_S$	in-lb	560,000	942,000	518,000
Pitch at root section	p	....	in.	167	148.5	182
Tangent of pitch angle	x	$p/\pi d$	...	1.50	1.97	1.16
Secant of pitch angle	y	$\sqrt{1 + x^2}$	...	1.80	2.21	1.53
Moment normal to root	M_N	$M_A/y + xM_C/y$	in-lb	1,180,000	1,850,000	1,380,000
Moment parallel to root	M_P	$xM_A/y - M_C/y$	in-lb	760,000	1,610,000	800,000

TABLE 3.—BLADE STRESS CALCULATION

Item	Symbol	Formula	Units	Fig. 1	Fig. 6	Fig. 2
Length of root section	l	....	in.	38.3	55.0	43.7
Moment of inertia of section, normal	I_N	$K_N l t_r^3$	(in.)^4	672	810	1160
Distance from N-A to point t, normal	y_t	....	in.	+3.03	+3.19	+3.75
Stress at t due to M_N	s_1	$M_N y_t/I_N$	psi	+5320	+7280	+4450
Moment of inertia of section, parallel	I_P	$K_P l^3 t_r$	(in.)^4	16,300	42,000	26,700
Distance from N-A to point t, parallel	x_t	....	in.	+13.7	+8.0	+5.8
Stress at t due to M_P	s_2	$M_P x_t/I_P$	psi	+640	+310	170
Area of section	A_r	$K_A l t_r$	sq in	203	266	258
Stress due to F	s_F	F/A_r	psi	+200	+1740	+190
Total stress at t	...	$s_1 + s_2 + s_F$	psi	+6160	+9330	+4810
Distance from N-A to point c, normal	y_c	....	in.	−4.09	−3.69	−4.65
Stress at c due to M_N	s_3	$M_N y_c/I_N$	psi	−7180	−8420	−5520
Distance from N-A to point c, parallel	x_c	....	in.	−5.4	0	−8.6
Stress at c due to M_P	s_4	$M_P x_c/I_P$	psi	−250	0	−260
Total stress at c	...	$s_3 + s_4 + s_F$	psi	−7230	−6680	−5590

Note: Use signs for x_c, x_t, y_c, y_t in accordance with Figs. 11 and 12.

Figure 4: Blade moment and stress calculation tables from Marine Engineering Vol. 1 [14].

To complete the stress calculations, a significant amount of information about the geometry of the propeller had to be found, including the rake arm, skewback arm, and several distances (y_t,

x_t, y_c, and x_c) which are found according to Figure 11 in the original document or Figure 5 shown below.

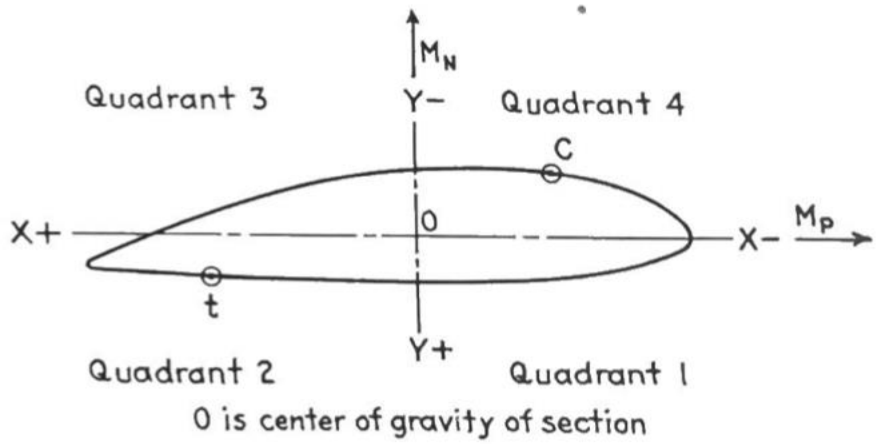

FIG. 11.—COMBINATION OF STRESSES ON AEROFOIL BLADE SECTION

Figure 5: Blade cross-section geometry as defined by Marine Engineering Vol. 1 [14].

All these distances require information about centers of gravity, whether this be the individual center of gravity of each radial cross-section of the blade (as seen in Figure 5) or the center of gravity of the entire blade, which is necessary to determine the rake and skewback arms. Therefore, an additional Python script was written dedicated to the integration of the blade cross-sections (Appendix B). Firstly, the coordinate points outlining each radial cross-section were calculated using the following equations from Oosterveld and Van Oossanen's paper

$$y_{face} = \begin{cases} V_1(t_{max} - t_{te}) \ \ for\ P \leq 0 \\ V_1(t_{max} - t_{le}) \ \ for\ P > 0 \end{cases} \quad \text{(Eq. 14)}$$

$$y_{back} = \begin{cases} (V_1 + V_2)(t_{max} - t_{te}) + t_{te} \ \ for\ P \leq 0 \\ (V_1 + V_2)(t_{max} - t_{le}) + t_{le} \ \ for\ P > 0 \end{cases} \quad \text{(Eq. 15)}$$

where y_{face} and y_{back} are the ordinate points on the face and back of the blade cross-section, respectively, for the corresponding coordinate P that varies from -1 to 1 from the trailing edge (TE) to the leading edge (LE) as seen below in Figure 6 [7].

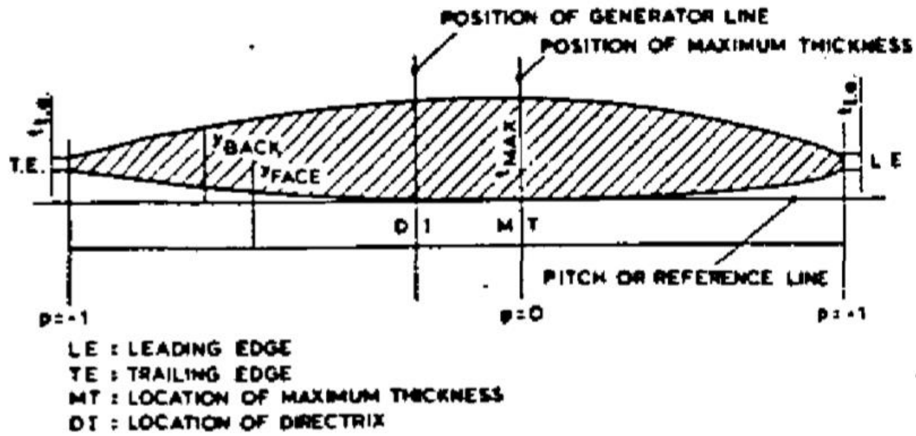

Figure 6: Blade cross-section geometry as defined by Oosterveld and Van Oossanen [7].

t_{max}, t_{te}, and t_{le} are the blade thicknesses at the position of maximum thickness (P=0), trailing edge, and leading edge respectively. The maximum blade section thickness can be found from geometry tables found in "Further Computer-Analyzed Data of the Wageningen B-Screw Series" [7], but the trailing edge thickness and leading edge thickness for each blade section were found from Carlton's "Marine Propellers and Propulsion" as seen below in Figure 7 [15].

r/R	Edge thickness ratios $\frac{t(x_c/x=0 \text{ or } 1.0)}{t_{max}}$	
	Leading edge	Trailing edge
0.9	0.245	0.245
0.8	0.170	0.152
0.7	0.143	0.120
0.6	0.134	0.100
0.5	0.130	0.085
0.4	0.127	0.075
0.3	0.124	0.068
0.2	0.120	0.057

Figure 7: Edge thickness ratios for conventional, low-skew propellers [15].

With the coordinate point series for each radial cross section of the blade having been defined, the integrations and calculations of the centers of gravity could begin. The first set of integrations integrated over the coordinate points to find the area and center of gravity of each cross-sectional slice. This was done using equations for the integration of a closed loop. However, due to the small thickness at the trailing edge, the first and last point of each array did not completely close, and each section is not truly a closed loop. But the offset between these points is minimal compared to the overall propeller, the error was determined to be insignificant.

$$A = \frac{1}{2}\sum_{i=1}^{n}[(y_{i+1} - y_i)(x_i + x_{i+1})] \quad \text{(Eq. 16)}$$

$$M_x = -\frac{1}{6}\sum_{i=1}^{n}[(x_{i+1} - x_i)({y_i}^2 + y_i y_{i+1} + {y_{i+1}}^2)] \quad \text{(Eq. 17)}$$

$$M_Y = \frac{1}{6}\sum_{i=1}^{n}[(y_{i+1} - y_i)({x_i}^2 + x_i x_{i+1} + {x_{i+1}}^2)] \quad \text{(Eq. 18)}$$

$$CG_X = \frac{M_Y}{A} \quad \text{(Eq. 19)}$$

$$CG_Y = \frac{M_X}{A} \quad \text{(Eq. 20)}$$

After integrating to find the area and the area moments, the centers of gravity in both the x and y-directions were calculated using Equations 19 and 20.

The overall center of gravity with respect to the x, y, and r dimensions was found by integrating all the cross-sections vertically with respect to r. Before doing this, the radial slice centers of gravity had to be adjusted with respect to the generator line. This is because propeller surfaces are curved, so the individual radial slices have a constantly changing pitch angle making the centers of gravity have additional offsets from each other in addition to the offsets caused by the changing cross-sectional areas.

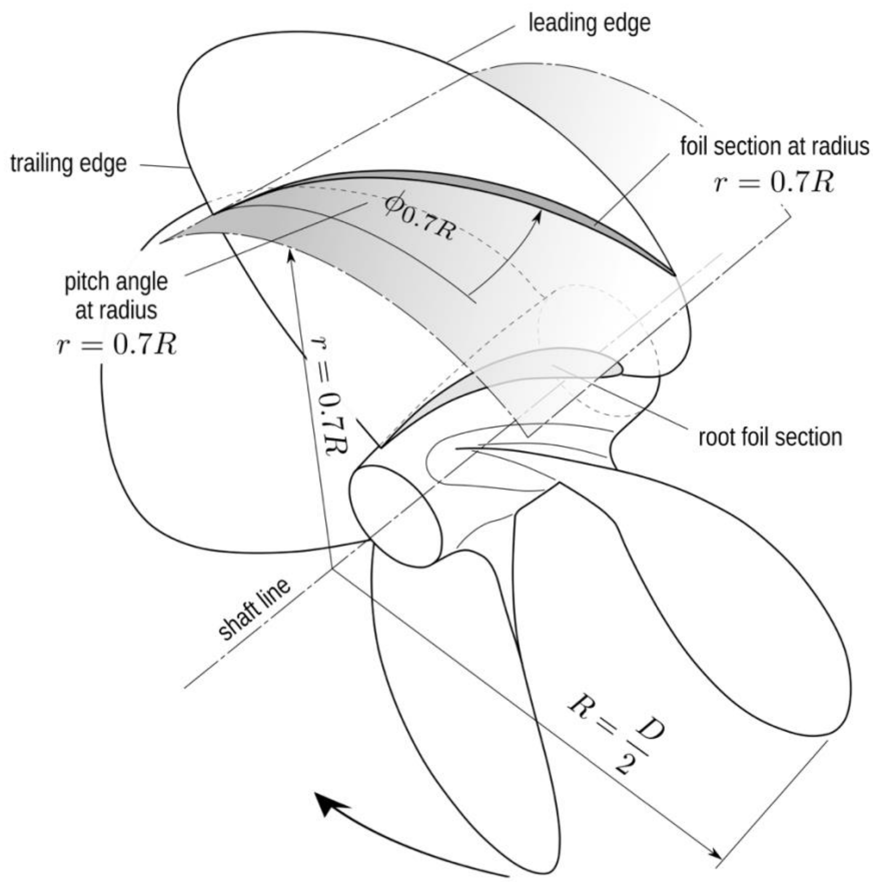

Figure 7: Hydrofoil cross-section located on a propeller blade [16].

After this correction was performed, the volumes were calculated using trapezoidal rule and the centers of gravity were found analogously to the calculation performed for the cross-sectional areas.

$$V = \frac{1}{2}\sum_{i=1}^{n}[(A_{i+1} + A_i)(r_i - r_{i+1})] \quad \text{(Eq. 21)}$$

$$CG_r = \frac{M_{Vr}}{V} \quad \text{(Eq. 22)}$$

With the centers of gravity, the equations in Figure 4 for the blade bending moment and blade stress calculations could be completed (Appendix C). However, two inputs required for the calculation of the bending moment equations, skewback arm and rake arm, had to be estimated. As of right now, no good method to calculate the rake and skewback arm of the propeller without a fully defined model. This may have contributed towards the obsolescence of the method presented in Marine Engineering Vol. 1 as finite element analysis requires a full propeller model [14]. However, finite element analysis would produce more accurate results with the same set of input data.

The final step of the propeller design process was to compare the results from the stress calculation and to the requirements set by IACS [1]. The result of the IACS requirements was the maximum allowable propeller stress and a set of minimum blade edge thicknesses. Both requirements ensure that the propeller can repeatedly withstand the forces imposed upon it as the vessel is icebreaking as well as any occasional ice collisions into the propeller itself. The material used for the calculations, 316/316L stainless steel, was chosen according to the ABS Guidance Notes section on material requirements for ice-class propellers [2]. The allowable stress for the propellers was calculated as follows:

$$\sigma_{ref} = \begin{cases} 0.7\sigma_u \\ 0.6\sigma_y + 0.4\sigma_u \end{cases} \text{whichever is less} \quad \text{(Eq. 23)}$$

$$\sigma_{all} = \frac{\sigma_{ref}}{S}\text{, S=1.5} \quad \text{(Eq. 24)}$$

where σ_u is the ultimate strength of 316/316L stainless steel and σ_y is the yield strength. S is a safety factor to ensure that the calculated stress does not come close to reaching the limit strengths of the material. The stress values and comparison are presented in Table 3 below.

Table 3: Propeller steel properties, calculated stress, and allowable stress.

316/316 stainless steel ultimate strength [17]	627 MPa
316/316 stainless steel yield strength [17]	290 MPa
Calculated maximum stress	60.328 MPa
Allowable stress	283.200 MPa

Finally, the minimum blade edge thicknesses were calculated along the radius of the blade. The edge thickness calculation has three components: leading edge thickness, trailing edge thickness, and tip thickness (which is the thickness for any edge above a radius ratio of 0.975). Figure 8 presents the minimum thickness distribution along the radius for each edge.

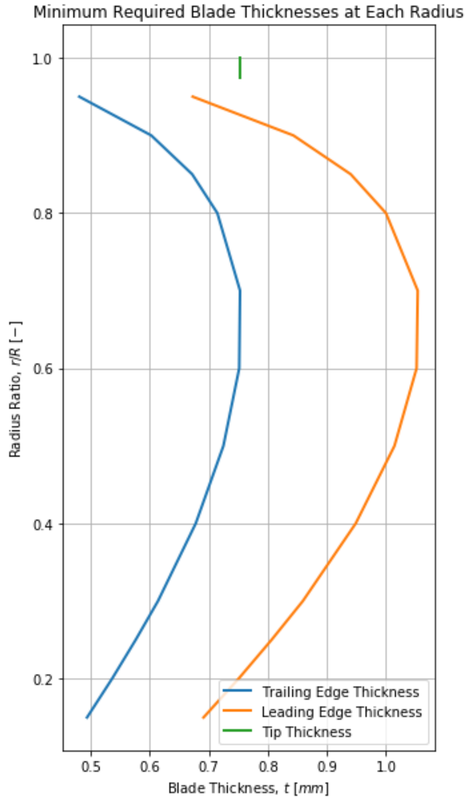

Figure 8: Minimum required blade edge thickness distribution.

The thickness calculation is dependent on the chord length at each radial section and the distribution reflects that. However, this type of distribution is impractical to implement, as a fluctuating thickness distribution would only make some areas of the blade edge more prone to failure as additional stresses are imposed on them. Figure 8 seems to suggest that near the tip of the blade, the edge should suddenly increase after a steady decrease of both the leading and trailing edge minimums. However, having a thick tip at the end of a thinner blade would only make that tip prone to snapping off due to torque or ice impact. Therefore, Figure 8 does not represent what a real thickness distribution on a propeller would look like. More preferably, the blade edge would have a near uniform thickness for the midsection, decreasing thickness at the tip where the blade section itself becomes as thin as the edge itself, and increasing thickness at the root to provide the propeller with more strength at the base. Table 4 below shows the maximum thickness value calculated from each category to provide a more realistic picture of what the propeller edges are likely to look like.

Table 4: Maximum minimum blade edge thicknesses.

Trailing edge thickness (t_{te})	0.7528 mm
Leading edge thickness (t_{le})	1.0539 mm
Tip thickness (t_{tip})	0.7518 mm

Modeling the propeller to meet the minimum blade edge thickness requirements as well as the principal characteristics developed using the optimization algorithm, the blades of the propeller can be visualized. The propeller seen in Figure 8 is for visualization purposes only and does not represent a realistically constructed propeller, since the hub diameter used in this model is far too small for a controllable-pitch propeller. This model was created using the free browser tool "B-Series Propeller Generator" by Friendship Services AG which is still in its beta phase of development and does not yet allow for precise hub design [18].

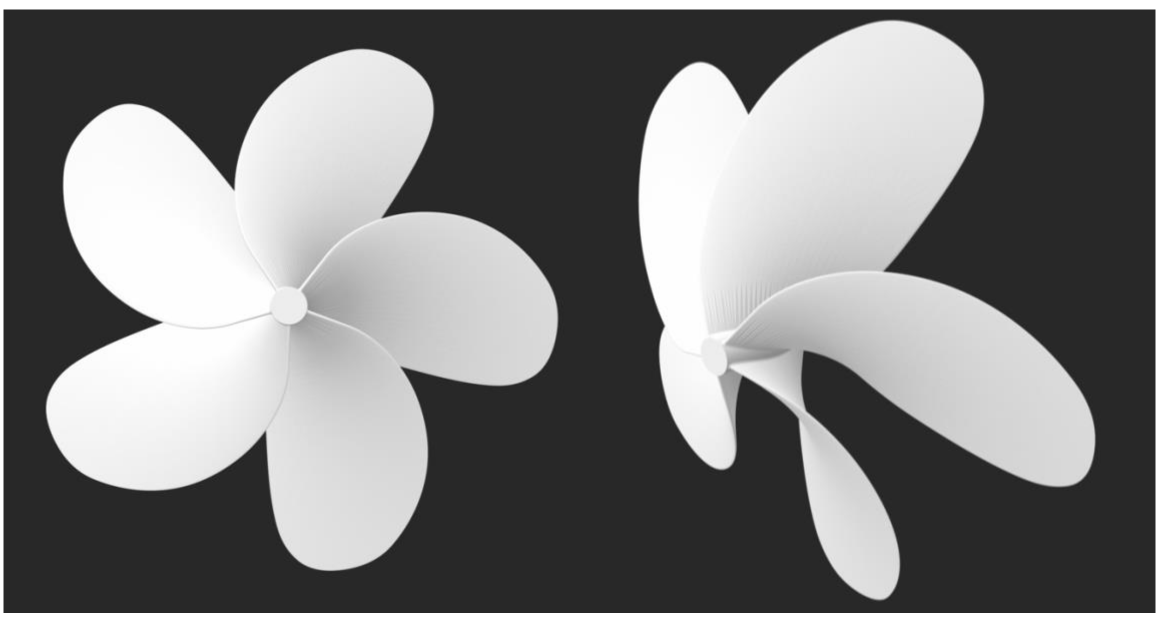

Figure 8: Propeller model for blade visualization from B-Series Propeller Generator [18].

5. Conclusion

Combining the results of the resistance and propulsion analysis, propeller optimization algorithm, and IACS structural requirements, the code developed for the purpose of designing ice-class Wageningen B-Series propellers did find success creating propellers for the MV Yahtse. The propeller optimization criteria were able to converge to a solution for optimal propeller characteristics and the IACS requirements validated the propeller for ice-breaking applications. However, these scripts have only been proven to work for the MV Yahtse and are only applicable for a preliminary design. Due to the lack of some information and the inability to choose more accurate methods for the icebreaking resistance estimate and the stress calculation, the code has room for improvements, and under those new conditions, the current propeller design may fail. Additionally, it has not been investigated how suitable Holtrop and Mennen's method is for this type of vessel, only that it produces the most conservative result from a small selection of applicable methods. Holtrop and Mennen's method was developed using regression analyses with data from existing ships, but since the MV Yahtse attempts to combine aspects of typically disparate types of ships (icebreaking bow with a wide and shallow RoRo midbody), Holtrop and Mennen's method has a significant chance for inaccuracy. Ideally, model tests would be used for a vessel like the MV Yahtse; however, as stated above that is not feasible for early-stage design. There is also a similar issue in the decision to use Wageningen B-Series propellers for this vessel. Wageningen B-Series propellers were designed as fixed-pitch propellers and can not operate optimally with the area ratio restriction required for fully reversible controllable-pitch propellers. For iterations of this code intended to complete post-preliminary design of ice-class controllable-pitch propellers, the following improvements should be considered:

1. A model test (for post-preliminary work) or CFD analysis (future preliminary work) should be completed to obtain the most accurate resistance and propulsion data possible for the propeller optimization.
2. If the model test or CFD analysis is completed without ice conditions, calculate and compare several icebreaking resistance estimate methods and choose the most applicable and conservative among them.
3. Alternative systematic propeller series should be investigated. For controllable-pitch propellers, the recently developed Wageningen C systematic series for open, CPPs is likely most suitable [19].
4. More accurate stress calculations should be performed for the completion of the IACS ice-class requirements calculations. This should be done preferably with finite element analysis; however, other preliminary methods should be researched.

References

[1] IACS. (2019). Requirements concerning polar class.

[2] ABS. (2014). Guidance notes on ice class.

[3] Holtrop J. and Mennen G. J. J. (1982.) An approximate power prediction method. *International Shipbuilding Progress*, 29(335):166-170.

[4] Holtrop, J. (1984). A statistical re-analysis of resistance and propulsion data. *International Shipbuilding Progress*, 31(363):3-5.

[5] Birk, L. (2019). NAME 3150 Lecture 19: Holtop and Mennen's Method.

[6] Jeong et. al (2010). Ice resistance prediction for standard icebreaker model ship. *20th International Offshore and Polar Engineering Conference*, 1300-1304.

[7] Oosterveld, M. W. C. and Van Oossanen, P. (1975). Further computer-analyzed data of the Wageningen B-screw series. *International Shipbuilding Progress*, 22(251):3-14.

[8] Andersen, P. and Guldhammer, H. E. (1986). A computer-oriented power prediction procedure. *International Conference on Computer Aided Design, Manufacture and Operation in the Marine and Offshore Industries*.

[9] Fung, S. C. (1992). Resistance and powering prediction for transom-stern hull forms during early-stage ship design. *SNAME Transactions*, 99:29-74.

[10] Fung, S. C. and Leibman, L. (1995). Revised speed-dependent powering predictions for high-speed transom-stern hull forms. *3rd International Conference on Fast Sea Transportation (FAST)*, 151-165.

[11] Burrill, L. and Emerson, A. (1963). Propeller cavitation: further tests on 16in propeller models in the King's College cavitation tunnel. *Transactions of the North East Coast Institution of Engineers and Shipbuilders (NECIES)*, 79:295-320.

[12] Birk, L. (2019). NAME 3150 Lecture 25: Propeller Series Data and Propeller Selection.

[13] auf'm Keller J (1966). Enige aspecten bij het antwerpen van scheepsschroeven. *Schpen en Werf*, 33(24): 658-663.

[14] Tingey, Richard H. (1942). Propeller Blade Stress, Propellers and Shafting, Section 3, SNAME Marine Engineering Vol 1. *SNAME*, 281-291.

[15] Carlton, J. S. (2007). Marine Propellers and Propulsion. *Elsevier Butterworth-Heinemann*, pg. 46.

[16] Birk, L. (2021). NAME 4160 Lecture 2: Foil and Propeller Geometry.

[17] AK Steel (2016). 316/316L stainless steel product data bulletin. *AK Steel*.

[18] Friendship Services AG (2021). B-Series Propeller Generator. https://www.wageningen-b-series-propeller.com/.

[19] Boom, H. J. J. et al. (2013). The Wageningen C- and D-series propellers. *12th International Conference on Fast Sea Transportation (FAST)*.

Appendix A: R&P Code – NAME3150RPHoltrop.py

```
# NAME 3150 RP Analysis Holtrop
# Date Last Modified: 04/28/2021

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve,minimize

from WBPolynomials import K_Tfunc,K_Qfunc,eta_Ofunc
from WBPolynomials import findJTS2,openwaterchart
from WBOpt import optimumprop

## Steps for Holtrop and Mennen's Method:
## 1. Input Data (check by using paper's provided example
##   to see if calculations and answers are correct)
## 2. Derived Data
## 3. Resistance Estimate
## 4. Powering Estimate + Propeller Optimization
## 5. Save Data to a File
## 6. Generate Plots
##
=======================================================================

## 1. Input Data

g=9.807
rho=1027.8336 #kg/m^3; density at 4ºC
nu=1.6262e-6 #m^2/s; viscosity at 4ºC

L_pp=97.319 #m
L_fore=4.39 #m
L_aft=0.25 #m
B=21.616   #m
T=4.72  #m

T_F=T
T_A=T
L_wl=L_pp+L_aft
```

```
print('L_wl = {:6.4f} m'.format(L_wl))

# lcb is estimated until known
#Fr_d=15.0*1852./3600./np.sqrt(g*L_wl)
#lcbp=-(0.44*Fr_d -0.094)*100. #lcb percentage
#print('lcbp = {:6.4f} '.format(lcbp))

V=7413.03895 #m_3

# S is estimated until a hydrostatic analysis is completed, need C_M
S=1931.231 #m^2
#A_M=341.803 #m^2
# no A_BT for senior design project
A_BT=0. #m^2
A_T=0. #m^2
A_V=108.7 #m^2

# wetted surface of each appendage in this order:
# rudder behind stern, twin screw rudder, shaft brackets, skeg,
# strut bossing, hull bossing, exposed shafts (10º), bilge keels
S_APPi=np.array([16.72,16.72,11.89,87.33,6.69,9.48,14.86,196.03]) #m^2

h_B=0. #m

# rudder behind stern, twin screw rudder, shaft brackets, skeg,
# strut bossing, hull bossing, exposed shafts (10º), bilge keels
k_2i=np.array([0.5,1.5,3.0,1.0,3.0,1.0,1.0,0.4])
C_B=0.7459
C_M=0.9313
C_P=0.8009
C_WP=0.9181 #if using formula, need C_P
C_stern=10.
k_s=150.*10**-6. #from ITTC procedures since no test data


# required propeller input data
Z=5.
D=3.048 #m

v_kn=np.linspace(10.0,19.0,num=19) #kn
```

```
#number within v_kn range that the service speed is
#starts counting at 0 (i.e. 0=10, 1=10.5, etc.)
ssid=10


# air properties
rho_A=1.225
C_DA=0.8

##---------------------------------------------------------------------------
## 2. Derived Data

#lcb=L_aft -lcbp/100.*L_pp +L_pp/2. -L_wl/2. #with respect to aft
lcb=50.32 #m
LCB=(lcb-L_wl/2.)/L_wl*100. #must be a percentage
print('LCB = {:6.4f} % L_wl'.format(LCB))

v_s=v_kn*1852./3600. #m/s
v_ss=v_s[ssid] #m/s
print('service speed = {:6.4f} m/s'.format(v_ss))

Fr=v_s/np.sqrt(g*L_wl)
Re=v_s*L_wl/nu

C_Bwl=C_B*L_pp/L_wl #conversion of C_B based on L_wl
print('C_Bwl = {:6.4f} '.format(C_Bwl))

C_WPwl=C_WP
#C_WPwl=0.763*(C_Pwl+0.34)
print('C_WPwl = {:6.4f} '.format(C_WPwl))

#C_P=V/A_M/L_pp
C_Pwl=C_P*L_pp/L_wl
print('C_Pwl = {:6.4f} '.format(C_Pwl))

# once A_M is known, switch
#C_Mwl=A_M/B/T
#C_Mwl=1./(1.+(1.-C_Bwl)**3.5)
C_Mwl=C_M
print('C_Mwl = {:6.4f} '.format(C_Mwl))
```

```
#waterline entrance angle
L_R=L_wl*(1.-C_Pwl+(0.06*C_Pwl*LCB)/(4.*C_Pwl -1.))

print('L_R = {:6.4f} m'.format(L_R))

a_1=(L_wl/B)**0.80856
a_2=(1.-C_WPwl)**0.30484
a_3=(1.-C_Pwl -0.0225*LCB)**0.6367
a_4=(L_R/B)**0.34574
a_5=((100.*V)/L_wl**3.)**0.16302

a=-(a_1*a_2*a_3*a_4*a_5)

print('a = {:6.4f} '.format(a))

i_E=1.+89.*np.exp(a)
#i_E=55.4 #deg.

print('i_E = {:6.4f} degrees'.format(i_E))

##-----------------------------------------------------------------------------
## 3. Resistance Estimate

C_F=0.075/(np.log10(Re)-2.)**2

R_F=0.5*rho*(v_s**2)*S*C_F

c_14=1.+0.011*C_stern
print('c_14 = {:6.4f} '.format(c_14))


k_a=(B/L_wl)**1.06806
k_b=(T/L_wl)**0.46106
k_c=(L_wl/L_R)**0.121563
k_d=((L_wl**3)/V)**0.36486
k_e=(1.-C_Pwl)**-0.604247

k=-0.07+ 0.487118*c_14*(k_a*k_b*k_c*k_d*k_e)
```

```
print('k = {:6.4f} '.format(k))

for i in range(len(k_2i)):
    print('k_2i = {:6.4f} '.format(k_2i[i]))
for i in range(len(S_APPi)):
    print('S_APPi = {:6.4f} '.format(S_APPi[i]))

S_APP=np.sum(S_APPi)
k_app=np.sum((1.+k_2i)*S_APPi)/np.sum(S_APPi)

print('S_APP = {:6.4f} '.format(S_APP))
print('k_app = {:6.4f} '.format(k_app))



#tunnel thrusters:
d_TH=1.54 #m
n_TH=2.
C_DTH=0.003+ 0.003*(10*d_TH/T -1.)
R_TH=rho*(v_s**2)*np.pi*(d_TH**2)*C_DTH

R_APP=0.5*rho*(v_s**2)*k_app*C_F*S_APP +n_TH*R_TH



#wave resistance coefficients/calculation:
if (B/L_wl) <= 0.11:
    c_7=0.229577*(B/L_wl)**(1/3)
elif (B/L_wl) <= 0.25:
    c_7=B/L_wl
else:
    c_7=0.5- 0.0625*(L_wl/B)
print('c_7 = {:6.6f} '.format(c_7))

c_1=2223105*(c_7**3.78613)*((T/B)**1.07961)*(90.-i_E)**-1.37565
print('c_1 = {:6.6f} '.format(c_1))

c_3a=B*T*(0.31*np.sqrt(A_BT)+T_F-h_B)
print('c_3a = {:6.6f} '.format(c_3a))

c_3=0.56*(A_BT**1.5)/c_3a
print('c_3 = {:6.6f} '.format(c_3))
```

```
c_2=np.exp(-1.89*np.sqrt(c_3))
print('c_2 = {:6.6f} '.format(c_2))

c_5=1.-0.8*A_T/B/T/C_Mwl
print('c_5 = {:6.6f} '.format(c_5))

if (L_wl**3)/V <= 512.:
    c_15=-1.69385
elif (L_wl**3)/V <=1726.91:
    c_15a=L_wl/(V**(1/3))-8.
    c_15=-1.69385+c_15a/2.36
else:
    c_15=0.
print('c_15 = {:6.6f} '.format(c_15))

if C_Pwl <=0.8:
    c_16=8.07981*C_Pwl- 13.8673*(C_Pwl**2)+ 6.984388*(C_Pwl**3)
else:
    c_16=1.73014- 0.7067*C_Pwl
print('c_16 = {:6.6f} '.format(c_16))

d=-0.9

if (L_wl/B) <= 12.:
    lamb=1.446*C_Pwl -0.03*L_wl/B
else:
    lamb=1.446*C_Pwl -0.36
print('lambda = {:6.6f} '.format(lamb))

m_1a=0.0140407*L_wl/T
m_1b=1.75254*(V**(1/3))/L_wl
m_1c=4.79323*B/L_wl

m_1=m_1a-m_1b-m_1c-c_16
print('m_1 = {:6.6f} '.format(m_1))

m_4=0.4*c_15*np.exp(-0.034*(Fr**-3.29))
for i in range(len(v_s)):
    print('m_4 = {:6.6f} '.format(m_4[i]))
```

```
r_1=m_1*(Fr**d)+m_4*np.cos(lamb*Fr**-2)

R_Wa=c_1*c_2*c_5*rho*g*V*np.exp(r_1)


c_17a=C_Mwl**-1.3346
c_17b=(V/(L_wl**3))**2.00977
c_17c=(L_wl/B -2.)**1.40692

c_17=6919.3*c_17a*c_17b*c_17c
print('c_17 = {:6.6f} '.format(c_17))

m_3a=(B/L_wl)**0.326869
m_3b=(T/B)**0.605375

m_3=-7.2035*m_3a*m_3b
print('m_3 = {:6.6f} '.format(m_3))

r_2=m_3*(Fr**d)+m_4*np.cos(lamb*Fr**-2)

R_Wb=c_17*c_2*c_5*rho*g*V*np.exp(r_2)


m_4a=0.4*c_15*np.e**(-0.034*(0.4**-3.29)) #interpolation of R_Wa

r_1a=m_1*(0.4**d)+m_4a*np.cos(lamb*0.4**-2)

R_Waa=c_1*c_2*c_5*rho*g*V*np.exp(r_1a)

m_4b=0.4*c_15*np.e**(-0.034*(0.55**-3.29)) #interpolation of R_Wb

r_2b=m_3*(0.55**d)+m_4b*np.cos(lamb*0.55**-2)

R_Wbb=c_17*c_2*c_5*rho*g*V*np.exp(r_2b)


R_W=np.zeros((len(Fr)),float)
for i in range(len(Fr)):
    if Fr[i] <= 0.4:
```

```
        R_W[i]=R_Wa[i]

    elif Fr[i] > 0.55:
        R_W[i]=R_Wb[i]

    else:
        R_W[i]=R_Waa+(20.*Fr[i]- 8.)/3.*(R_Wbb- R_Waa)


#bulbous bow resistance:
#h_f=C_Pwl*C_Mwl*B*T/L_wl*(136.- 316.3*Fr)*(Fr**3)
#h_F=np.where(-0.01*L_wl <= h_f,h_f,-0.01*L_wl)
#h_w=i_E*(v_s**2)/400./g
#h_W=np.where(-0.01*L_wl <= h_w,h_w,-0.01*L_wl)
#r_i=g*(T_F-h_B- 0.25*np.sqrt(A_BT)+h_F+h_W)
#Fr_i=v_s/np.sqrt(r_i)
#P_B=0.56*np.sqrt(A_BT)/(T_F- 1.5*h_B+h_F)
#R_B=0.11*rho*g*(np.sqrt(A_BT)**3)*(Fr_i**3)/(1+Fr_i**2)*np.exp(-3.*(P_B**-2))


#transom resistance:
r_t=np.sqrt(2.*g*A_T/(B+B*C_WPwl))

if A_T > 0.:
    Fr_T=v_s/r_t
else:
    Fr_T=np.zeros((len(Fr)),float)


c_6=np.where(Fr_T< 5.,0.2*(1- 0.2*Fr_T),0.)

for i in range(len(v_s)):
    print('c_6 = {:6.6f}'.format(c_6[i]))

R_TR=0.5*rho*(v_s**2)*A_T*c_6


#correlation allowance resistance:
if T_F/L_wl <= 0.04:
    c_4=T_F/L_wl
```

```
else:
    c_4=0.04
print('c_4 = {:6.6f}'.format(c_4))

C_Aa=np.sqrt(L_wl/7.5)*(C_Bwl**4)*c_2*(0.04-c_4)

C_A=0.00546*((L_wl+ 100.)**-0.16)-0.002+ 0.003*C_Aa
print('C_A*1000 = {:6.4f}'.format(C_A*1000.))


if k_s > 150.*10.**-6:
    deltaC_A=(0.105*(k_s**(1./3.))-0.005579)/L_wl**(1./3.)

else:
    deltaC_A=0.

print('k_s = {:6.6f}'.format(k_s))
print('deltaC_A = {:6.6f}'.format(deltaC_A))

R_A=0.5*rho*(v_s**2)*(C_A+deltaC_A)*(S+np.sum(S_APPi))


#air resistance

R_AA=0.5*rho_A*(v_s**2)*C_DA*A_V

# icebreaking resistance
# Jeong formulas (2010)

c_B=0.5
c_C=1.11
c_BR=2.73

h_i=1. #m; ice thickness
T_i=-2. #ºC; ice temperature
# Arnol'd - Aliab'ev ice flexural stength formula
sigma_f=4.7- 0.96*T_i -0.31*T_i**2 #kg/cm^2; ice flexural strength
sigma_f=sigma_f*100.**2 #kg/m^2
rho_i=918.9 #kg/m^3; density of ice at -10ºC
```

```
rho_diff=rho-rho_i #kg/m^3

F_h=v_s/np.sqrt(g*h_i)
S_N=v_s/np.sqrt(sigma_f*h_i/rho_i/B)

ai=c_B*rho_diff*g*h_i*B*T
bi=c_C*(F_h**-1.157)*rho_i*B*h_i*v_s**2
ci=c_BR*(S_N**-1.54)*rho_i*B*h_i*v_s**2

R_I=13.14*v_s**2 +ai +bi +ci


#total resistance
R_T=(1.+k)*R_F+R_APP+R_A+R_W+R_TR+R_AA #+R_I #+R_B

"""
NOTE: Icebreaking resistance is only being used to calculate the
extreme operating condition of propeller operation.
The vessel and propeller are not being designed for continuous ice-
breaking.
"""


C_W=R_W/0.5/rho/S/(v_s**2)
C_T=R_T/0.5/rho/S/(v_s**2)

##----------------------------------------------------------------------------
## 4. Powering Estimate

#viscous resistance coefficient
C_Va=(1.+k)*R_F+R_APP+R_A
C_Vb=0.5*rho*(v_s**2)*(S+np.sum(S_APPi))

C_V=C_Va/C_Vb

#wake fraction coefficients
if B/T_A <= 5.:
    c_8=S/L_wl/D*B/T_A
else:
    c_8=S*(7.*B/T_A- 25.)/L_wl/D/(B/T_A- 3.)
```

```
print('c_8 = {:6.6f} '.format(c_8))

if c_8 <= 28.:
    c_9=c_8
else:
    c_9=32.- 16./(c_8- 24.)

print('c_9 = {:6.6f} '.format(c_9))

if T_A/D <= 2.:
    c_11=T_A/D
else:
    c_11=0.0833333*((T_A/D)**3)+ 1.33333

print('c_11 = {:6.6f} '.format(c_11))

if C_Pwl <= 0.7:
    c_19=0.12997/(0.95 -C_Bwl)- 0.11056/(0.95 -C_Pwl)
else:
    c_19=0.18567/(1.3571 -C_Mwl)-0.71276 +0.38648*C_Pwl

print('c_19 = {:6.6f} '.format(c_19))

c_20=1. +0.015*C_stern

print('c_20 = {:6.6f} '.format(c_20))

C_P1=1.45*C_Pwl -0.315 -0.0225*LCB

print('C_P1 = {:6.6f} '.format(C_P1))


#full scale wake fraction (single screw)
#w_sa=c_9*c_20*C_V*L_wl/T_A*(0.050776+ 0.93405*c_11*C_V/(1.-C_P1))
#w_sb=0.27915*c_20*np.sqrt(B/L_wl/(1.-C_P1))+c_19*c_20
#w_s=w_sa+w_sb

#full scale wake fraction (twin screw)
w_s=0.3095*C_Bwl +10.*C_V*C_Bwl -0.23*D/np.sqrt(B*T)
```

```
for i in range(len(v_s)):
   print('w_s = {:6.4f} '.format(w_s[i]))

#thrust deduction fraction (single screw)
#t_a=0.25014*((B/L_wl)**0.28956)*((np.sqrt(B*T)/D)**0.2624)
#t_b=(1.-C_Pwl +0.0225*LCB)**0.01762
#t=t_a/t_b +0.0015*C_stern

#thrust deduction fraction (twin screw)
t=0.325*C_Bwl -0.1885*D/np.sqrt(B*T)

print('t = {:6.6f} '.format(t))


v_as=(1.-w_s)*v_s

for i in range(len(v_s)):
   print('v_a = {:6.4f} m/s'.format(v_as[i]))

T_req=R_T/(1.-t)

for i in range(len(v_s)):
   print('T_req = {:6.4f} kN'.format(T_req[i]/1000.))

C_S=S/2./(D**2)*C_T/(1.-t)/(1.-w_s)**2


##############################################################################
## Propeller Selection Program
# more realistic estimation of e
e=T- 0.5*D- 0.03*D

p_A=101325. #Pa
p_v=1671.  #Pa
p_0=p_A+rho*g*e

# find design constant at design speed (service speed)
v_aserv=v_as[ssid]
T_reqs=T_req[ssid]/2. # half the thrust is taken since 2 props
```

```
dc_4=T_reqs/(rho*(D**2)*(v_aserv**2))

# additional arguments for objective functions
propargs=(dc_4,Z,D,T_reqs,v_aserv,rho,e,'CPP')

#initial values of free variables
PD0=1.0 #initial guess at pitch/diameter ratio

# Keller's formula
K=0.2
ar0=(1.3+ 0.3*Z)*T_reqs/(p_0-p_v)/(D**2) +K
x0=np.array([PD0,ar0])

#use optimization algorithm
res=minimize(optimumprop,x0,args=propargs)

#unpack results
PD=res.x[0] #optimum pitch diameter ratio
ar=res.x[1] #optimum expanded area ratio


print('')
print('')
print('Optimum Propeller Data:')
print(' design constant     dc_4  = {:8.4f} '.format(dc_4))
print(' number of blades    Z     = {:8.4f} '.format(Z))
print(' propeller diameter  D     = {:8.4f} m'.format(D))
print(' pitch-dia. ratio    PD    = {:8.4f} '.format(PD))
print(' area ratio          ar    = {:8.4f} '.format(ar))

print('')
print('For total thrust reversal on a CPP propeller, the expanded')
print('area ratio must have a maximum of 0.75.')
print('')

#relative rotative efficiency (single screw)
#eta_R=0.9922 -0.05908*ar +0.07424*(C_Pwl -0.0225*LCB)

#relative rotative efficiency (twin screw)
eta_R=0.9737 +0.111*(C_Pwl -0.0225*LCB) -0.06325*PD
```

```
print(' r.r. efficiency     eta_R = {:6.6f} '.format(eta_R))

#self-propulsion points

J_TS=np.zeros((len(C_S)),float)
K_TS=np.zeros((len(C_S)),float)
K_QTS=np.zeros((len(C_S)),float)
eta_OS=np.zeros((len(C_S)),float)

for j in range(len(C_S)):

    J_TS[j]=findJTS2(C_S[j],PD,ar,Z)

    K_TS[j]=K_Tfunc(J_TS[j],PD,ar,Z)
    K_QTS[j]=K_Qfunc(J_TS[j],PD,ar,Z)
    eta_OS[j]=eta_Ofunc(J_TS[j],PD,ar,Z)


    #print('J_TS = {:6.4f} '.format(J_TS[j]))
    #print('K_TS = {:6.4f} '.format(K_TS[j]))
    #print('10K_QTS = {:6.4f} '.format(10.*K_QTS[j]))
    #print('eta_OS = {:6.4f} '.format(eta_OS[j]))

eta_OSs=eta_OS[ssid]
print(' opt. efficiency    eta_OS = {:8.4f} '.format(eta_OSs))
print('')

# rate of revolution
n=v_as/(J_TS*D)

for i in range(len(v_s)):
    print('n = {:6.4f} 1/s'.format(n[i]))

K_QB=K_QTS/eta_R #behind condition K_Q

# behind efficiency
eta_B=eta_OS*eta_R

# torque
```

```
Q=rho*(n**2)*(D**5)*K_QB

# delivered power
P_D=2.*np.pi*n*Q

for i in range(len(v_s)):
    print('P_D = {:6.4f} kW'.format(P_D[i]/1000.))

# effective power
P_E=R_T*v_s

# delivered efficiency
eta_D=P_E/P_D

# hull efficiency
eta_H=eta_D/(eta_OS*eta_R)


##########################################################################
## 5. Save Data to a File

# allows files of the same project to be grouped together with the
# same base name
base='Holtrop&MennenResistanceAnalysis'
datafile=base+'.dat'

fp=open(datafile,'w')

fp.write('\n')
fp.write('Coefficients:')
fp.write('\n\n')

fp.write('c1= {:6.6f}'.format(c_1))
fp.write('\n')
fp.write('c2= {:6.6f}'.format(c_2))
fp.write('\n')
fp.write('c3= {:6.6f}'.format(c_3))
fp.write('\n')
fp.write('c4= {:6.6f}'.format(c_4))
fp.write('\n')
```

```
fp.write('c5= {:6.6f}'.format(c_5))
fp.write('\n')
fp.write('c7= {:6.6f}'.format(c_7))
fp.write('\n')
fp.write('c8= {:6.6f}'.format(c_8))
fp.write('\n')
fp.write('c9= {:6.6f}'.format(c_9))
fp.write('\n')
fp.write('c11= {:6.6f}'.format(c_11))
fp.write('\n')
fp.write('c14= {:6.6f}'.format(c_14))
fp.write('\n')
fp.write('c15= {:6.6f}'.format(c_15))
fp.write('\n')
fp.write('c16= {:6.6f}'.format(c_16))
fp.write('\n')
fp.write('c17= {:6.6f}'.format(c_17))
fp.write('\n')
fp.write('c19= {:6.6f}'.format(c_19))
fp.write('\n')
fp.write('c20= {:6.6f}'.format(c_20))
fp.write('\n')
fp.write('d= {:6.6f}'.format(d))
fp.write('\n')
fp.write('lambda= {:6.6f}'.format(lamb))
fp.write('\n')
fp.write('m1= {:6.6f}'.format(m_1))
fp.write('\n')
fp.write('m3= {:6.6f}'.format(c_1))
fp.write('\n')
fp.write('C_P1= {:6.6f}'.format(C_P1))
fp.write('\n\n')

fp.write('\n')
fp.write('Froude Numbers and Misc. Coefficients:')
fp.write('\n\n')

fp.write('v_kn'.center(12))
fp.write('Fr'.center(11))
#fp.write('Fr_i'.center(15))
```

```
fp.write('Fr_T'.center(15))
fp.write('c_6'.center(15))
fp.write('m3(Fr^d)'.center(10))
fp.write('m4'.center(9))
fp.write('m4cos(lambda/Fr^2)'.center(9))
#fp.write('P_B'.center(9))
fp.write('\n')

fp.write('[kn]'.center(12))
fp.write('[-]'.center(17))
#fp.write('[-]'.center(17))
fp.write('[-]'.center(15))
fp.write('[-]'.center(15))
fp.write('[-]'.center(20))
fp.write('[-]'.center(14))
fp.write('[-]'.center(18))
#fp.write('[-]'.center(37))
fp.write('\n')

for i in range(len(v_s)):
    fp.write(' {:6.2f}'.format(v_kn[i]))
    fp.write(' {:10.5f}'.format(Fr[i]))
    #fp.write(' {:10.5f}'.format(Fr_i[i]))
    fp.write(' {:10.5f}'.format(Fr_T[i]))
    fp.write(' {:10.4f}'.format(c_6[i]))
    fp.write(' {:10.5f}'.format(m_3*Fr[i]**d))
    fp.write(' {:10.5f}'.format(m_4[i]))
    fp.write(' {:10.5f}'.format(m_4[i]*np.cos(lamb/Fr[i]**2)))
    #fp.write(' {:20.5f}'.format(P_B[i]))
    fp.write('\n')

fp.write('\n\n')

fp.write('\n')
fp.write('Resistance Components and Total Resistance:')
fp.write('\n\n')

fp.write('v_kn'.center(12))
fp.write('Fr'.center(10))
fp.write('R_F'.center(18))
```

```
fp.write('R_A'.center(10))
fp.write('R_W'.center(13))
#fp.write('R_B'.center(10))
fp.write('R_APP'.center(11))
fp.write('R_AA'.center(7))
fp.write('R_TR'.center(15))
fp.write('R_I'.center(12))
fp.write('R_T'.center(8))
fp.write('\n')

fp.write('[kn]'.center(12))
fp.write('[-]'.center(15))
fp.write('[kN]'.center(15))
fp.write('[kN]'.center(15))
fp.write('[kN]'.center(14))
#fp.write('[kN]'.center(12))
fp.write('[kN]'.center(13))
fp.write('[kN]'.center(14))
fp.write('[kN]'.center(14))
fp.write('[kN]'.center(11))
fp.write('[kN]'.center(12))
fp.write('\n')

for i in range(len(v_s)):
    fp.write(' {:6.2f}'.format(v_kn[i]))
    fp.write(' {:10.5f}'.format(Fr[i]))
    fp.write(' {:10.3f}'.format(R_F[i]/1000))
    fp.write(' {:10.3f}'.format(R_A[i]/1000))
    fp.write(' {:10.3f}'.format(R_W[i]/1000))
    #fp.write(' {:10.3f}'.format(R_B[i]/1000))
    fp.write(' {:10.3f}'.format(R_APP[i]/1000))
    fp.write(' {:10.3f}'.format(R_AA[i]/1000))
    fp.write(' {:10.3f}'.format(R_TR[i]/1000))
    fp.write(' {:10.3f}'.format(R_I[i]/1000))
    fp.write(' {:10.3f}'.format(R_T[i]/1000))
    fp.write('\n')

fp.write('\n\n')

fp.write('\n')
```

```
fp.write('Self-Propulsion Point:')
fp.write('\n\n')

fp.write('v_kn'.center(12))
fp.write('Fr'.center(10))
fp.write('w_s'.center(17))
fp.write('v_a'.center(13))
fp.write('T_req'.center(12))
fp.write('C_S'.center(13))
fp.write('J_TS'.center(12))
fp.write('K_TS'.center(12))
fp.write('10K_QTS'.center(10))
fp.write('\n')

fp.write('[kn]'.center(12))
fp.write('[-]'.center(15))
fp.write('[-]'.center(17))
fp.write('[m/s]'.center(15))
fp.write('[kN]'.center(12))
fp.write('[-]'.center(19))
fp.write('[-]'.center(15))
fp.write('[-]'.center(16))
fp.write('[-]'.center(16))
fp.write('\n')

for i in range(len(v_s)):
    fp.write(' {:6.2f}'.format(v_kn[i]))
    fp.write(' {:10.5f}'.format(Fr[i]))
    fp.write(' {:10.4f}'.format(w_s[i]))
    fp.write(' {:10.4f}'.format(v_as[i]))
    fp.write(' {:10.4f}'.format(T_req[i]/1000.))
    fp.write(' {:10.5f}'.format(C_S[i]))
    fp.write(' {:10.4f}'.format(J_TS[i]))
    fp.write(' {:10.4f}'.format(K_TS[i]))
    fp.write(' {:10.4f}'.format(10.*K_QTS[i]))
    fp.write('\n')

fp.write('\n\n')

fp.write('\n')
```

```
fp.write('Efficiency and Powering:')
fp.write('\n\n')

fp.write('v_kn'.center(12))
fp.write('Fr'.center(10))
fp.write('eta_H'.center(17))
fp.write('eta_O'.center(5))
fp.write('eta_D'.center(18))
fp.write('n'.center(10))
fp.write('n'.center(17))
fp.write('P_D'.center(12))
fp.write('\n')

fp.write('[kn]'.center(12))
fp.write('[-]'.center(17))
fp.write('[-]'.center(15))
fp.write('[-]'.center(16))
fp.write('[-]'.center(15))
fp.write('[1/s]'.center(17))
fp.write('[rpm]'.center(10))
fp.write('[kW]'.center(12))
fp.write('\n')

for i in range(len(v_s)):
    fp.write(' {:6.2f}'.format(v_kn[i]))
    fp.write(' {:10.5f}'.format(Fr[i]))
    fp.write(' {:10.4f}'.format(eta_H[i]))
    fp.write(' {:10.4f}'.format(eta_OS[i]))
    fp.write(' {:10.4f}'.format(eta_D[i]))
    fp.write(' {:10.3f}'.format(n[i]))
    fp.write(' {:10.3f}'.format(60*n[i]))
    fp.write(' {:10.2f}'.format(P_D[i]/1000.))
    fp.write('\n')

fp.close()

##-----------------------------------------------------------------------------
## 6. Generate Plots

fig=plt.figure(figsize=(15,10))
```

```
plt.plot(Fr,C_T*1000,lw=2, label=r"Total Resistance Coefficient $[C_T]$")
plt.plot(Fr,C_F*1000,lw=2, label=r"Coefficient of Friction $[C_F]$")
plt.plot(Fr,C_W*1000,lw=2, label=r"Wave Resistance Coefficient $[C_W]$")
plt.title("Resistance Coefficients vs. Froude Number")
plt.xlabel("Froude Number, $Fr$ $[-]$")
plt.ylabel("Friction Coefficient Magnitude, $[-]$")
plt.legend()
plt.grid()
plt.show()

fig=plt.figure(figsize=(15,10))
plt.plot(Fr,R_T/1000,lw=2, label=r"Total Resistance $[R_T]$")
plt.plot(Fr,R_F/1000,lw=2, label=r"Frictional Resistance $[R_F]$")
plt.plot(Fr,R_W/1000,lw=2, label=r"Wave Resistance $[R_W]$")
plt.plot(Fr,R_A/1000,lw=2, label=r"Correlation Resistance $[R_A]$")
plt.plot(Fr,R_AA/1000,lw=2, label=r"Air Resistance $[R_{AA}]$")
plt.plot(Fr,R_APP/1000,lw=2, label=r"Appendage Resistance $[R_{APP}]$")
#plt.plot(Fr,R_B/1000,lw=2, label=r"Bulbous Bow Resistance $[R_B]$")
plt.plot(Fr,R_TR/1000,lw=2, label=r"Transom Resistance $[R_{TR}]$")
plt.plot(Fr,R_I/1000,lw=2, label=r"Icebreaking $[R_{I}]$")
plt.title("Resistance Components and Total Resistance")
plt.xlabel("Froude Number, $Fr$ $[-]$")
plt.ylabel("Resistance Magnitude, $[kN]$")
plt.legend()
plt.grid()
plt.show()


plt.figure(figsize=(15,10))
J=np.linspace(0.,0.83,num=100)
openwaterchart(J,PD,ar,Z)


for j in range(len(C_S)):
    plt.plot(J_TS[j],K_TS[j],'o',color='r',fillstyle='none')
    plt.plot(J_TS[j],10.*K_QTS[j],'o',color='g',fillstyle='none')
    plt.plot(J_TS[j],eta_OS[j],'o',color='b',fillstyle='none')
    plt.plot(J,C_S[j]*J**2,'+-')

plt.xlabel(r'Advance Ratio, $J$ $[-]$')
```

```
plt.ylabel(r'$K_T$, $10K_Q$, $eta_O$, and $C_SJ^2$ $[-]$')
plt.title("Open Water Chart and Self-Propulsion Points")
plt.legend()
plt.grid()
plt.show()


plt.figure(figsize=(15,10))
plt.plot(v_kn,n,lw=2,label=r"Rate of Revolution, $n$")
plt.title("Rate of Revolution vs. Speed")
plt.xlabel(r'Ship Speed, $v_{kn}$ $[kn]$')
plt.ylabel(r'Rate of Revolution, $n$ $[1/s]$')
plt.grid()
plt.show()


plt.figure(figsize=(15,10))
plt.plot(v_kn,P_D/1000,lw=2,label=r"Delivered Power, $P_D$")
plt.title("Delivered Power vs. Speed")
plt.xlabel(r'Ship Speed, $v_{kn}$ $[kn]$')
plt.ylabel(r'Delivered Power, $P_D$ $[kN]$')
plt.grid()
plt.show()


plt.figure(figsize=(15,10))
plt.plot(n,P_D/1000,lw=2,label=r"Delivered Power, $P_D$")
plt.title("Delivered Power vs. Rate of Revolution")
plt.xlabel(r'Rate of Revolution, $n$ $[1/s]$')
plt.ylabel(r'Delivered Power, $P_D$ $[kN]$')
plt.grid()
plt.show()
```

Appendix B: Propeller Geometry Code – WBSeriesPropGeometry.py

```
# Prop Geometry for W-B Series Propellers
# Date Last Modified: 04/26/2021

#from NAME3150RPHoltrop import Z,D,ar,PD

import numpy as np

Z=5.
D=3.0480 #m
ar=0.7520
PD=0.7568

##############################################################################

# maximum thickness (tmax) calculation
rR=np.array([0.15,0.20,0.25,0.30,0.40,0.50,0.60,0.70,0.80,0.85,0.90,\
        0.95,0.975,1.0])
Ar=np.array([0.0588,0.0526,0.0495,0.0464,0.0402,0.0340,0.0278,0.0216,\
        0.0154,0.0123,0.0092,0.0061,0.00455,0.003])
Br=np.array([0.00425,0.0040,0.00375,0.0035,0.0030,0.0025,0.0020,0.0015,\
        0.0010,0.00075,0.0005,0.00025,0.000125,0.0])

tmax=D*(Ar-Br*Z)

# edge thickness approximation code provided by Dr. Birk

# Typical blade edge thickness ratios edge thickness/tmax
# from Carlton, p.46. this seems to work
# reduced initial values for x=0.15, 200422, lb

# r/R order: [0.15,0.20,0.25,0.3,0.4,0.5,0.6,0.7,0.8,0.85,0.9,...,1.0]
tetfactor = np.array([0.049,0.057,0.063,0.068,0.075,0.085,0.100,0.120,\
              0.152,0.192,0.245,0.245,0.245,0.245])
letfactor = np.array([0.115,0.120,0.1224,0.124,0.127,0.130,0.134,0.143,\
              0.170,0.205,0.245,0.245,0.245,0.245])

#trailing edge thickness
tte = tetfactor*tmax
```

```
#leading edge thickness
tle = letfactor*tmax


# chord length calculation
Cr=np.array([1.473,1.600,1.719,1.832,2.023,2.163,2.243,2.247,2.132,\
        2.005,1.798,1.434,1.122,0.0])

cl=Cr*D/Z*ar

# chord length for calculating tip thicknesses
cltip=1.122*D/Z*ar

# distances for adjusting xc
# assuming linear interpolation
brcr=np.array([0.350,0.350,0.350,0.350,0.351,0.355,0.389,0.443,0.479,\
        (0.479+0.5)/2.,0.500,0.250,0.125,0.0])
br=brcr*cl

arcr=np.array([0.617,0.617,(0.617+0.613)/2.,0.613,0.601,0.586,0.561,\
        0.524,0.463,(0.463+0.351)/2.,0.351,(0.351/2.),\
        (0.351/4.),0.0])
ar=arcr*cl

############################################################################
## center of gravity integration

# split each array into positive and negative according to the P-values
# shown below for each of the different y equations
Parrayn=np.array([-1.0,-0.95,-0.90,-0.80,-0.70,-0.60,-0.50,-0.40,-0.20])
Parrayp=np.array([0.0,0.20,0.40,0.50,0.60,0.70,0.80,0.85,0.90,0.95,1.0])

# V1 arrays

V1_15n=np.array([0.3,0.2824,0.265,0.23,0.195,0.161,0.128,0.0955,\
        0.0365,0.0])

V1_15p=np.array([0.0096,0.0384,0.0615,0.092,0.132,0.187,0.223,0.2642,\
        0.315,0.386])
```

```
V1_20n=np.array([0.2826,0.263,0.24,0.1967,0.157,0.1207,0.088,0.0592,\
        0.0172,0.0])

V1_20p=np.array([0.0049,0.0304,0.052,0.0804,0.118,0.1685,0.2,0.2353,\
        0.2821,0.356])

V1_25n=np.array([0.2598,0.2372,0.2115,0.1651,0.1246,0.0899,0.0579,\
        0.035,0.0084,0.0])

V1_25p=np.array([0.0031,0.0224,0.0417,0.0669,0.1008,0.1465,0.1747,\
        0.2068,0.2513,0.3256])

V1_30n=np.array([0.2306,0.204,0.179,0.1333,0.0943,0.0623,0.0376,0.0202,\
        0.0033,0.0])

V1_30p=np.array([0.0027,0.0148,0.03,0.0503,0.079,0.1191,0.1445,0.176,\
        0.2186,0.2923])

V1_40n=np.array([0.1467,0.12,0.0972,0.063,0.0395,0.0214,0.0116,0.0044,\
        0.0,0.0])

V1_40p=np.array([0.0,0.0033,0.009,0.0189,0.0357,0.0637,0.0833,0.1088,\
        0.1467,0.2181])

V1_50n=np.array([0.0522,0.042,0.033,0.019,0.01,0.004,0.0012,0.0,0.0,0.0])

V1_50p=np.array([0.0,0.0,0.0008,0.0034,0.0085,0.0211,0.0328,0.05,0.0778,\
        0.1278])

V1_60n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_60p=np.array([0.0,0.0,0.0,0.0,0.0,0.0006,0.0022,0.0067,0.0169,\
        0.0382])

V1_70n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_70p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_80n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])
```

```
V1_80p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_85n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_85p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_90n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_90p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_95n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_95p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_975n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_975p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_100n=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

V1_100p=np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

# V2 arrays

V2_15n=np.array([0.0,0.054,0.1325,0.287,0.428,0.5585,0.677,0.7805,\
        0.9360,1.0])

V2_15p=np.array([0.976,0.8825,0.8055,0.7105,0.5995,0.452,0.3665,0.26,\
        0.13,0.0])

V2_20n=np.array([0.0,0.064,0.1455,0.306,0.4535,0.5842,0.6995,0.7984,\
        0.9446,1.0])

V2_20p=np.array([0.975,0.8875,0.817,0.7277,0.619,0.4777,0.3905,0.284,\
        0.156,0.0])

V2_25n=np.array([0.0,0.0725,0.1567,0.3228,0.474,0.605,0.7184,0.8139,\
        0.9519,1.0])
```

```
V2_25p=np.array([0.9751,0.8899,0.8259,0.7415,0.6359,0.4982,0.4108,\
        0.3042,0.1758,0.0])

V2_30n=np.array([0.0,0.8,0.167,0.336,0.4885,0.6195,0.7335,0.8265,\
        0.9583,1.0])

V2_30p=np.array([0.975,0.892,0.8315,0.752,0.6505,0.513,0.4265,0.3197,\
        0.189,0.0])

V2_40n=np.array([0.0,0.0905,0.181,0.035,0.504,0.6353,0.7525,0.8415,\
        0.9645,1.0])

V2_40p=np.array([0.9725,0.8933,0.8345,0.7593,0.659,0.522,0.4335,\
        0.3235,0.1935,0.0])

V2_50n=np.array([0.0,0.095,0.1865,0.3569,0.514,0.6439,0.758,0.8456,\
        0.9639,1.0])

V2_50p=np.array([0.971,0.888,0.8275,0.7478,0.643,0.5039,0.4135,0.3056,\
        0.175,0.0])

V2_60n=np.array([0.0,0.0965,0.1885,0.3585,0.511,0.6415,0.753,0.8426,\
        0.9613,1.0])

V2_60p=np.array([0.969,0.879,0.809,0.72,0.606,0.462,0.3775,0.272,\
        0.1485,0.0])

V2_70n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_70p=np.array([0.9675,0.866,0.785,0.684,0.5615,0.414,0.33,0.2337,\
        0.124,0.0])

V2_80n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_80p=np.array([0.9635,0.852,0.7635,0.6545,0.5265,0.3765,0.2925,\
        0.2028,0.105,0.0])

V2_85n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_85p=np.array([0.9615,0.845,0.755,0.6455,0.516,0.366,0.283,0.195,\
```

```
        0.1,0.0])

V2_90n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_90p=np.array([0.96,0.84,0.75,0.64,0.51,0.36,0.2775,0.19,0.0975,0.0])

V2_95n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_95p=np.array([0.96,0.84,0.75,0.64,0.51,0.36,0.2775,0.19,0.0975,0.0])

V2_975n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_975p=np.array([0.96,0.84,0.75,0.64,0.51,0.36,0.2775,0.19,0.0975,0.0])

V2_100n=np.array([0.0,0.0975,0.19,0.36,0.51,0.64,0.75,0.84,0.96,1.0])

V2_100p=np.array([0.96,0.84,0.75,0.64,0.51,0.36,0.2775,0.19,0.0975,0.0])

## blade outline calculations (y_face and y_back):

# x-coordinates are P-values
# goes from -1 to 1 and then back again
x_coor=np.concatenate((Parrayn,Parrayp))
x_coor=np.concatenate((x_coor,np.flip(x_coor)))

# r/R=0.15
rRid=0
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_15n=V1_15n*(a-b)
yf_15p=V1_15p*(a-c)
yf_15=np.concatenate((yf_15n,yf_15p))

# np.min used becuase max y_face is at a minimum offset
yfmax=np.min(yf_15)

print('')
print('Maximum y_face and y_back values for stress calc.')
```

```
print('r/R=15: max. y_face     = {:8.4f} m'.format(yfmax))
print('r/R=15: corresponding P =   0.0 ')

yb_15n=(V1_15n+V2_15n)*(a-b) + b
yb_15p=(V1_15p+V2_15p)*(a-c) + c
yb_15=np.concatenate((yb_15n,yb_15p))

ybmax=yb_15.max()

print('r/R=15: max. y_back     = {:8.4f} m'.format(ybmax))
print('r/R=15: corresponding P = {:8.4f}'.format(x_coor[np.argmax(yb_15)]))

y_15=np.concatenate((yf_15,np.flip(yb_15)))

A_15=0.5*np.sum((y_15[1:]-y_15[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_15=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_15[:-1]**2\
                +y_15[:-1]*y_15[1:]+y_15[1:]**2))

My_15=1./6.*np.sum((y_15[1:]-y_15[:-1])*(x_coor[:-1]**2\
                +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_15=br[rRid]+My_15/A_15
xc_15=ar[rRid]-xc_15
yc_15=Mx_15/A_15

# r/R=0.20
rRid=1
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_20n=V1_20n*(a-b)
yf_20p=V1_20p*(a-c)
yf_20=np.concatenate((yf_20n,yf_20p))

yb_20n=(V1_20n+V2_20n)*(a-b) + b
yb_20p=(V1_20p+V2_20p)*(a-c) + c
yb_20=np.concatenate((yb_20n,yb_20p))
```

```
y_20=np.concatenate((yf_20,np.flip(yb_20)))

A_20=0.5*np.sum((y_20[1:]-y_20[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_20=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_20[:-1]**2\
                +y_20[:-1]*y_20[1:]+y_20[1:]**2))

My_20=1./6.*np.sum((y_20[1:]-y_20[:-1])*(x_coor[:-1]**2\
                +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_20=br[rRid]+My_20/A_20
xc_20=ar[rRid]-xc_20
yc_20=Mx_20/A_20

# r/R=0.25
rRid=2
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_25n=V1_25n*(a-b)
yf_25p=V1_25p*(a-c)
yf_25=np.concatenate((yf_25n,yf_25p))

yb_25n=(V1_25n+V2_25n)*(a-b) + b
yb_25p=(V1_25p+V2_25p)*(a-c) + c
yb_25=np.concatenate((yb_25n,yb_25p))

y_25=np.concatenate((yf_25,np.flip(yb_25)))

A_25=0.5*np.sum((y_25[1:]-y_25[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_25=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_25[:-1]**2\
                +y_25[:-1]*y_25[1:]+y_25[1:]**2))

My_25=1./6.*np.sum((y_25[1:]-y_25[:-1])*(x_coor[:-1]**2\
                +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_25=br[rRid]+My_25/A_25
xc_25=ar[rRid]-xc_25
```

```
yc_25=Mx_25/A_25

# r/R=0.30
rRid=3
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_30n=V1_30n*(a-b)
yf_30p=V1_30p*(a-c)
yf_30=np.concatenate((yf_30n,yf_30p))

yb_30n=(V1_30n+V2_30n)*(a-b) + b
yb_30p=(V1_30p+V2_30p)*(a-c) + c
yb_30=np.concatenate((yb_30n,yb_30p))

y_30=np.concatenate((yf_30,np.flip(yb_30)))

A_30=0.5*np.sum((y_30[1:]-y_30[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_30=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_30[:-1]**2\
        +y_30[:-1]*y_30[1:]+y_30[1:]**2))

My_30=1./6.*np.sum((y_30[1:]-y_30[:-1])*(x_coor[:-1]**2\
        +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_30=br[rRid]+My_30/A_30
xc_30=ar[rRid]-xc_30
yc_30=Mx_30/A_30

# r/R=0.40
rRid=4
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_40n=V1_40n*(a-b)
yf_40p=V1_40p*(a-c)
yf_40=np.concatenate((yf_40n,yf_40p))
```

```
yb_40n=(V1_40n+V2_40n)*(a-b) + b
yb_40p=(V1_40p+V2_40p)*(a-c) + c
yb_40=np.concatenate((yb_40n,yb_40p))

y_40=np.concatenate((yf_40,np.flip(yb_40)))

A_40=0.5*np.sum((y_40[1:]-y_40[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_40=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_40[:-1]**2\
        +y_40[:-1]*y_40[1:]+y_40[1:]**2))

My_40=1./6.*np.sum((y_40[1:]-y_40[:-1])*(x_coor[:-1]**2\
        +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_40=br[rRid]+My_40/A_40
xc_40=ar[rRid]-xc_40
yc_40=Mx_40/A_40

# r/R=0.50
rRid=5
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_50n=V1_50n*(a-b)
yf_50p=V1_50p*(a-c)
yf_50=np.concatenate((yf_50n,yf_50p))

yb_50n=(V1_50n+V2_50n)*(a-b) + b
yb_50p=(V1_50p+V2_50p)*(a-c) + c
yb_50=np.concatenate((yb_50n,yb_50p))

y_50=np.concatenate((yf_50,np.flip(yb_50)))

A_50=0.5*np.sum((y_50[1:]-y_50[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_50=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_50[:-1]**2\
        +y_50[:-1]*y_50[1:]+y_50[1:]**2))

My_50=1./6.*np.sum((y_50[1:]-y_50[:-1])*(x_coor[:-1]**2\
```

```
                 +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_50=br[rRid]+My_50/A_50
xc_50=ar[rRid]-xc_50
yc_50=Mx_50/A_50

# r/R=0.60
rRid=6
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_60n=V1_60n*(a-b)
yf_60p=V1_60p*(a-c)
yf_60=np.concatenate((yf_60n,yf_60p))

yb_60n=(V1_60n+V2_60n)*(a-b) + b
yb_60p=(V1_60p+V2_60p)*(a-c) + c
yb_60=np.concatenate((yb_60n,yb_60p))

y_60=np.concatenate((yf_60,np.flip(yb_60)))

A_60=0.5*np.sum((y_60[1:]-y_60[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_60=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_60[:-1]**2\
                 +y_60[:-1]*y_60[1:]+y_60[1:]**2))

My_60=1./6.*np.sum((y_60[1:]-y_60[:-1])*(x_coor[:-1]**2\
                 +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_60=br[rRid]+My_60/A_60
xc_60=ar[rRid]-xc_60
yc_60=Mx_60/A_60

# r/R=0.70
rRid=7
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]
```

```
yf_70n=V1_70n*(a-b)
yf_70p=V1_70p*(a-c)
yf_70=np.concatenate((yf_70n,yf_70p))

yb_70n=(V1_70n+V2_70n)*(a-b) + b
yb_70p=(V1_70p+V2_70p)*(a-c) + c
yb_70=np.concatenate((yb_70n,yb_70p))

y_70=np.concatenate((yf_70,np.flip(yb_70)))

A_70=0.5*np.sum((y_70[1:]-y_70[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_70=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_70[:-1]**2\
        +y_70[:-1]*y_70[1:]+y_70[1:]**2))

My_70=1./6.*np.sum((y_70[1:]-y_70[:-1])*(x_coor[:-1]**2\
        +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_70=br[rRid]+My_70/A_70
xc_70=ar[rRid]-xc_70
yc_70=Mx_70/A_70

# r/R=0.80
rRid=8
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_80n=V1_80n*(a-b)
yf_80p=V1_80p*(a-c)
yf_80=np.concatenate((yf_80n,yf_80p))

yb_80n=(V1_80n+V2_80n)*(a-b) + b
yb_80p=(V1_80p+V2_80p)*(a-c) + c
yb_80=np.concatenate((yb_80n,yb_80p))

y_80=np.concatenate((yf_80,np.flip(yb_80)))

A_80=0.5*np.sum((y_80[1:]-y_80[:-1])*(x_coor[:-1]+x_coor[1:]))
```

```
Mx_80=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_80[:-1]**2\
          +y_80[:-1]*y_80[1:]+y_80[1:]**2))

My_80=1./6.*np.sum((y_80[1:]-y_80[:-1])*(x_coor[:-1]**2\
          +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_80=br[rRid]+My_80/A_80
xc_80=ar[rRid]-xc_80
yc_80=Mx_80/A_80

# r/R=0.85
rRid=9
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_85n=V1_85n*(a-b)
yf_85p=V1_85p*(a-c)
yf_85=np.concatenate((yf_85n,yf_85p))

yb_85n=(V1_85n+V2_85n)*(a-b) + b
yb_85p=(V1_85p+V2_85p)*(a-c) + c
yb_85=np.concatenate((yb_85n,yb_85p))

y_85=np.concatenate((yf_85,np.flip(yb_85)))

A_85=0.5*np.sum((y_85[1:]-y_85[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_85=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_85[:-1]**2\
          +y_85[:-1]*y_85[1:]+y_85[1:]**2))

My_85=1./6.*np.sum((y_85[1:]-y_85[:-1])*(x_coor[:-1]**2\
          +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_85=br[rRid]+My_85/A_85
xc_85=ar[rRid]-xc_85
yc_85=Mx_85/A_85

# r/R=0.90
rRid=10
```

```
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_90n=V1_90n*(a-b)
yf_90p=V1_90p*(a-c)
yf_90=np.concatenate((yf_90n,yf_90p))

yb_90n=(V1_90n+V2_90n)*(a-b) + b
yb_90p=(V1_90p+V2_90p)*(a-c) + c
yb_90=np.concatenate((yb_90n,yb_90p))

y_90=np.concatenate((yf_90,np.flip(yb_90)))

A_90=0.5*np.sum((y_90[1:]-y_90[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_90=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_90[:-1]**2\
        +y_90[:-1]*y_90[1:]+y_90[1:]**2))

My_90=1./6.*np.sum((y_90[1:]-y_90[:-1])*(x_coor[:-1]**2\
        +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_90=br[rRid]+My_90/A_90
xc_90=ar[rRid]-xc_90
yc_90=Mx_90/A_90

# r/R=0.95
rRid=11
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_95n=V1_95n*(a-b)
yf_95p=V1_95p*(a-c)
yf_95=np.concatenate((yf_95n,yf_95p))

yb_95n=(V1_95n+V2_95n)*(a-b) + b
yb_95p=(V1_95p+V2_95p)*(a-c) + c
yb_95=np.concatenate((yb_95n,yb_95p))
```

```
y_95=np.concatenate((yf_95,np.flip(yb_95)))

A_95=0.5*np.sum((y_95[1:]-y_95[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_95=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_95[:-1]**2\
                +y_95[:-1]*y_95[1:]+y_95[1:]**2))

My_95=1./6.*np.sum((y_95[1:]-y_95[:-1])*(x_coor[:-1]**2\
                +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_95=br[rRid]+My_95/A_95
xc_95=ar[rRid]-xc_95
yc_95=Mx_95/A_95

# r/R=0.975
rRid=12
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_975n=V1_975n*(a-b)
yf_975p=V1_975p*(a-c)
yf_975=np.concatenate((yf_975n,yf_975p))

yb_975n=(V1_975n+V2_975n)*(a-b) + b
yb_975p=(V1_975p+V2_975p)*(a-c) + c
yb_975=np.concatenate((yb_975n,yb_975p))

y_975=np.concatenate((yf_975,np.flip(yb_975)))

A_975=0.5*np.sum((y_975[1:]-y_975[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_975=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_975[:-1]**2\
                +y_975[:-1]*y_975[1:]+y_975[1:]**2))

My_975=1./6.*np.sum((y_975[1:]-y_975[:-1])*(x_coor[:-1]**2\
                +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_975=br[rRid]+My_975/A_975
xc_975=ar[rRid]-xc_975
```

```
yc_975=Mx_975/A_975

# r/R=0.100
rRid=13
a=tmax[rRid]
b=tte[rRid]
c=tle[rRid]

yf_100n=V1_100n*(a-b)
yf_100p=V1_100p*(a-c)
yf_100=np.concatenate((yf_100n,yf_100p))

yb_100n=(V1_100n+V2_100n)*(a-b) + b
yb_100p=(V1_100p+V2_100p)*(a-c) + c
yb_100=np.concatenate((yb_100n,yb_100p))

y_100=np.concatenate((yf_100,np.flip(yb_100)))

A_100=0.5*np.sum((y_100[1:]-y_100[:-1])*(x_coor[:-1]+x_coor[1:]))

Mx_100=-1./6.*np.sum((x_coor[1:]-x_coor[:-1])*(y_100[:-1]**2\
        +y_100[:-1]*y_100[1:]+y_100[1:]**2))

My_100=1./6.*np.sum((y_100[1:]-y_100[:-1])*(x_coor[:-1]**2\
        +x_coor[:-1]*x_coor[1:]+x_coor[1:]**2))

xc_100=br[rRid]+My_100/A_100
xc_100=ar[rRid]-xc_100
yc_100=Mx_100/A_100


# xc is center of gravity x-offset from P=0
# yc is center of gravity y-offset from pitch (reference) line

print('')
print('Propeller Blade Cross-Sectional Areas:')
print('r/R=15:    A =  {:8.4f} m^2'.format(A_15))
print('r/R=20:    A =  {:8.4f} m^2'.format(A_20))
print('r/R=25:    A =  {:8.4f} m^2'.format(A_25))
print('r/R=30:    A =  {:8.4f} m^2'.format(A_30))
```

```
print('r/R=40:    A =  {:8.4f} m^2'.format(A_40))
print('r/R=50:    A =  {:8.4f} m^2'.format(A_50))
print('r/R=60:    A =  {:8.4f} m^2'.format(A_60))
print('r/R=70:    A =  {:8.4f} m^2'.format(A_70))
print('r/R=80:    A =  {:8.4f} m^2'.format(A_80))
print('r/R=85:    A =  {:8.4f} m^2'.format(A_85))
print('r/R=90:    A =  {:8.4f} m^2'.format(A_90))
print('r/R=95:    A =  {:8.4f} m^2'.format(A_95))
print('r/R=97.5:  A =  {:8.4f} m^2'.format(A_975))
print('r/R=100:   A =  {:8.4f} m^2'.format(A_100))

print('')
print('Propeller Blade X-Moments:')
print('r/R=15:    Mx = {:8.4f} m^3'.format(Mx_15))
print('r/R=20:    Mx = {:8.4f} m^3'.format(Mx_20))
print('r/R=25:    Mx = {:8.4f} m^3'.format(Mx_25))
print('r/R=30:    Mx = {:8.4f} m^3'.format(Mx_30))
print('r/R=40:    Mx = {:8.4f} m^3'.format(Mx_40))
print('r/R=50:    Mx = {:8.4f} m^3'.format(Mx_50))
print('r/R=60:    Mx = {:8.4f} m^3'.format(Mx_60))
print('r/R=70:    Mx = {:8.4f} m^3'.format(Mx_70))
print('r/R=80:    Mx = {:8.4f} m^3'.format(Mx_80))
print('r/R=85:    Mx = {:8.4f} m^3'.format(Mx_85))
print('r/R=90:    Mx = {:8.4f} m^3'.format(Mx_90))
print('r/R=95:    Mx = {:8.4f} m^3'.format(Mx_95))
print('r/R=97.5:  Mx = {:8.4f} m^3'.format(Mx_975))
print('r/R=100:   Mx = {:8.4f} m^3'.format(Mx_100))

print('')
print('Propeller Blade Y-Moments:')
print('r/R=15:    My = {:8.4f} m^3'.format(My_15))
print('r/R=20:    My = {:8.4f} m^3'.format(My_20))
print('r/R=25:    My = {:8.4f} m^3'.format(My_25))
print('r/R=30:    My = {:8.4f} m^3'.format(My_30))
print('r/R=40:    My = {:8.4f} m^3'.format(My_40))
print('r/R=50:    My = {:8.4f} m^3'.format(My_50))
print('r/R=60:    My = {:8.4f} m^3'.format(My_60))
print('r/R=70:    My = {:8.4f} m^3'.format(My_70))
print('r/R=80:    My = {:8.4f} m^3'.format(My_80))
print('r/R=85:    My = {:8.4f} m^3'.format(My_85))
```

```
print('r/R=90:    My = {:8.4f} m^3'.format(My_90))
print('r/R=95:    My = {:8.4f} m^3'.format(My_95))
print('r/R=97.5:  My = {:8.4f} m^3'.format(My_975))
print('r/R=100:   My = {:8.4f} m^3'.format(My_100))

print('')
print('Center of Gravity X-Offset:')
print('r/R=15:    xc = {:8.4f} m'.format(xc_15))
print('r/R=20:    xc = {:8.4f} m'.format(xc_20))
print('r/R=25:    xc = {:8.4f} m'.format(xc_25))
print('r/R=30:    xc = {:8.4f} m'.format(xc_30))
print('r/R=40:    xc = {:8.4f} m'.format(xc_40))
print('r/R=50:    xc = {:8.4f} m'.format(xc_50))
print('r/R=60:    xc = {:8.4f} m'.format(xc_60))
print('r/R=70:    xc = {:8.4f} m'.format(xc_70))
print('r/R=80:    xc = {:8.4f} m'.format(xc_80))
print('r/R=85:    xc = {:8.4f} m'.format(xc_85))
print('r/R=90:    xc = {:8.4f} m'.format(xc_90))
print('r/R=95:    xc = {:8.4f} m'.format(xc_95))
print('r/R=97.5:  xc = {:8.4f} m'.format(xc_975))
print('r/R=100:   xc = {:8.4f} m'.format(xc_100))

print('')
print('Center of Gravity Y-Offset:')
print('r/R=15:    yc = {:8.4f} m'.format(yc_15))
print('r/R=20:    yc = {:8.4f} m'.format(yc_20))
print('r/R=25:    yc = {:8.4f} m'.format(yc_25))
print('r/R=30:    yc = {:8.4f} m'.format(yc_30))
print('r/R=40:    yc = {:8.4f} m'.format(yc_40))
print('r/R=50:    yc = {:8.4f} m'.format(yc_50))
print('r/R=60:    yc = {:8.4f} m'.format(yc_60))
print('r/R=70:    yc = {:8.4f} m'.format(yc_70))
print('r/R=80:    yc = {:8.4f} m'.format(yc_80))
print('r/R=85:    yc = {:8.4f} m'.format(yc_85))
print('r/R=90:    yc = {:8.4f} m'.format(yc_90))
print('r/R=95:    yc = {:8.4f} m'.format(yc_95))
print('r/R=97.5:  yc = {:8.4f} m'.format(yc_975))
print('r/R=100:   yc = {:8.4f} m'.format(yc_100))

## volume integrations:
```

```
r=D/2.*np.array([0.15,0.20,0.25,0.30,0.40,0.50,0.60,0.70,0.80,0.85,\
            0.90,0.95,0.975,1.0]) #m

A=np.array([A_15,A_20,A_25,A_30,A_40,A_50,A_60,A_70,A_80,A_85,A_90,\
         A_95,A_975,A_100])

rarray=np.flip(r)

Aarray=np.flip(A)

# pitch angle
phi=np.arctan(PD*D/(2.*np.pi))

# rake array
rake=np.tan(np.deg2rad(15.))*r #m

# centers
xcarray=np.array([xc_15,xc_20,xc_25,xc_30,xc_40,xc_50,xc_60,xc_70,\
            xc_80,xc_85,xc_90,xc_95,xc_975,xc_100])

ycarray=np.array([yc_15,yc_20,yc_25,yc_30,yc_40,yc_50,yc_60,yc_70,\
            yc_80,yc_85,yc_90,yc_95,yc_975,yc_100])

xc=np.cos(phi)*xcarray -np.sin(phi)*ycarray

yc=np.sin(phi)*xcarray +np.cos(phi)*ycarray
yc=yc-rake


# volumes
# volume found via integration
V=0.5*np.sum((Aarray[1:]+Aarray[:-1])*(rarray[:-1]-rarray[1:]))

# volume found with trapezoidal method
Vtrap=np.trapz(A,r)

# volume of the prism containing blade
Vp=tmax[0]*cl[0]*D/2.

# radial volume center
```

```
M_Vr=np.trapz(r*A,r)

CGr=M_Vr/V #m

M_Vx=np.trapz(xc*A,r)

CGy=M_Vx/V #m

M_Vy=np.trapz(yc*A,r)

CGx=M_Vy/V #m

print('')
print('Volume Integration Results:')
print('volume:          V     = {:8.4f} m^3'.format(V))
print('volume (trap.):  V     = {:8.4f} m^3'.format(Vtrap))
print('volume (prism):  V     = {:8.4f} m^3'.format(Vp))
print('radial v.mom:    Mvr   = {:8.4f} m^4'.format(M_Vr))
print('radial CG:       CGr   = {:8.4f} m'.format(CGr))
print('radial CG (r/R): CGr   = {:8.4f} m'.format(CGr/(D/2.)))
print('x-direction CG:  CGx   = {:8.4f} m'.format(CGx))
print('y-direction CG:  CGy   = {:8.4f} m'.format(CGy))
```

Appendix C: Propeller Structural Code – MVYahtsePropellerDev.py

```
# Honors Program Capstone Project
# Ice Class Propeller Design
# Date Last Modified: 04/30/2021

"""
Design and ice-class propeller for the MV Yahtse - an overnighting,
ice-class, car ferry servicing the Alaskan coast and Bering Sea.

Propeller must meet IACS ice-class requirements

Ice Class - PC 3
Number of Propellers - 2
Type - CPP, open

"""

import numpy as np
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

from NAME3150RPHoltrop import n,T_req,v_kn,eta_H,eta_OS,eta_R,t
#from NAME3150RPHoltrop import Z,D,ar,PD
from WBSeriesPropGeometry import cl,cltip,rR,xc_15,yc_15,yfmax,ybmax


Z=5.
D=3.0480 #m
ar=0.7520
PD=0.7568

##############################################################################

## Variables

# for a worst case scenario, look at service speed + icebreaking
# since H&M code designs the propeller at service speed, this will see
# if the service speed propeller can survive worst conditions
```

```
ssid=10
n_ss=n[ssid] # nominal rotational speed at MCR free-running condition

# from W-B series chord equations
c_7=2.247*D/Z*ar #m; length of the blade chord at 0.7R (radius)
P_7=0.7*PD*D #m; propeller pitch at 0.7R
t_7=D*(0.0216 -0.0015*Z) #m; max thickness at 0.7R

# if bollard thrust (T_n) is known, use instead of T and tab out T_n
# estimation calculation
T=T_req[ssid]/2000. #kN; per propeller thrust at MCR open water cond.

# measurements of cylindrical root section of the blade at the weakest
# section outside root fillet; typically will be at the termination of
# the fillet into the blade profile.

# root section measurements
# assuming root is at 16.5% of the total blade diameter

d_h=0.165*D #m; propeller hub diameter
d_r=d_h #approximately true

# cut off at a x=r/R of 0.7 becuase independant variable must be
# increasing only for CubicSpline to work
x=np.array([0.15,0.20,0.25,0.30,0.40,0.50,0.60,0.70])
Cr=np.array([1.473,1.600,1.719,1.832,2.023,2.163,2.243,2.247])
y=Cr*D/Z*ar
cs=CubicSpline(Cr,y)
Crx=np.interp(0.165,x,Cr)
print('')
print('C_rx = {:6.4f} '.format(Crx))
c_r=cs(Crx) #m; chord length at the root
print('c_r = {:6.4f} m'.format(c_r))

xp=np.array([0.15,0.20,0.25])
Ar=np.array([0.0588,0.0526,0.0495])
Br=np.array([0.00425,0.0040,0.00375])
fp=D*(Ar-Br*Z)
t_r=np.interp(0.165,xp,fp) #m; thickness at the root
print('t_r = {:6.4f} m'.format(t_r))
```

```
p=PD*D #m; pitch at root section, constant pitch
r=d_r/2. #m; radius


# blade material constants
# Blade materials are in accordance with ABS
# Stainless Steel 316/316L
# sigma_y is the 0.2% proof stress conventionally considered as
# yield stress
sigma_y=290.0e3 #kPa
# sigma_u is the ultimate strength
sigma_u=627.0e3 #kPa


##############################################################################

## must be done in Imperial units and converted at end
## from 1942 SNAME Marine Engieering Vol. 1

## Bending Moment Calculation

D_i=D*39.37 #in.
d_ri=d_r*39.37 #in.; diameter at root section
P=3655. #hp; shaft horsepower per screw
v=v_kn[ssid] #knots; ship speed
N=n_ss*60. #rpm; shaft revolutions per minute
A_d=ar*np.pi*(D_i/24.)**2 #ft^2; approximately true A_d=A_e
t_ri=t_r*39.37 #in.; maximum thickness at root

# note on coordinate system being used:
# x - horizontal along the face of the blade
# y - horizontal through blade thickness
# z/r - tangent out from hub/blade root

# assume center of root is half of the root thickness
a_r=c_r*0.617 #distance from LE to generator line at (approx.) the root
CRb=a_r*39.37 #in.; distance of center of root in y-direction
CRr=t_r/2.*39.27 #in.; distance of center of root in z-direction
p=p*39.37 #in.; pitch at root section
```

```
# method needs to be found to determine these from blade geometry
r=5.1 #in.; arm due to rake  [**GUESS VALUE**]
b=5.1 #in.; arm due to skewback  [**GUESS VALUE**]

# thrust moment arm factor
K_T=0.66*D_i -d_ri #in.

# moment due to thrust
eta_H=eta_H[ssid] #hull efficiency
eta_OS=eta_OS[ssid]
e=eta_H*eta_OS*eta_R #propulsive efficiency
M_T=163.*P*e*K_T/(v*Z*(1.-t)) #in.-lb

# centrifugal force
F_c=D_i*N**2*A_d*t_ri/(7450.*Z) #lb

# moment due to rake
M_R=r*F_c #in.-lb

# total axial moment
M_A=M_T + M_R #in.-lb

# torque moment arm ratio
K_Q=1. -1.67*d_ri/D_i

# moment due to torque
M_Q=63000.*P*K_Q/(Z*N) #in.-lb

# moment due to skewback
M_S=b*F_c #in.-lb

# total circumferential moment
M_C=M_Q - M_S #in.-lb

# tangent of pitch angle
x=p/(np.pi*d_ri)

# secant of pitch angle
```

```
y=np.sqrt(1.+x**2)

# moment normal to root
M_N=M_A/y + x*M_C/y #in.-lb

# moment parallel to root
M_P=x*M_A/y - M_C/y #in.-lb


## Blade Stress Calculation
# only check most extreme values (at r/R=0.15 where x&y are largest)

# check if correct
l=c_r*39.37 #in.; length of root section

# moment of inertia of section, normal
K_N=0.046
I_N=K_N*l*t_ri**3 #in.^4

# stress at t due to M_N
# y_t is distance between yc and y_back at trailing edge of r/R=0.15
# distance from NA to point t, normal (fig. 11)
y_t=np.abs(yc_15-ybmax)*39.37 #in.
s1=M_N*y_t/I_N #psi

# moment of inertia of section, parallel
K_P=0.039
I_P=K_P*l**3*t_ri #in.^4

# stress at t due to M_P
# x_t is distance between xc and P=-1 of y_face at r/R=0.15
# distance from NA to point t, parallel (fig.11)
x_t=(cl[0]-0.350*cl[0]+xc_15)*39.37 #in.
s2=M_P*x_t/I_P #psi

# area of section
K_A=0.71
A_r=K_A*l*t_ri #in.^2

# stress due to F
```

```
s_F=F_c/A_r #psi

# total stress at t
s_t=s1 + s2 + s_F #psi

# stress at c due to M_N
# y_c is greatest distance between yc and y_face at r/R=0.15
# distance from NA to point c, normal (fig.11)
y_c=np.abs(yc_15-yfmax)*39.37 #in.
s3=M_N*y_c/I_N #psi

# stress at c due to M_P
# x_c is greatest distance between xc and P=0 of y_face at r/R=0.15
# distance from NA to point c, parallel (fig.11)
x_c=xc_15*39.37 #in.
s4=M_P*x_c/I_P #psi

# total stress at c
s_c=s3 + s4 + s_F #psi


## Calculated Blade Stress

if s_t > s_c:
    s_calc=s_t #psi

else:
    s_calc=s_c #psi

sigma_calc=s_calc*6.895 #kPa

print('calc. stress = {:6.4f} kPa'.format(sigma_calc))

##############################################################################

## IACS Propeller Requirements

"""
I3.4 Ice Interaction Load:
    I3.4.1 Propeller Ice Interaction:
```

```
    The loads given in section I3.4 are total loads (unless
    otherwise stated) during ice interaction and are to be applied
    separately (unless otherwise stated) and are intended for
    component strength calculations only. The different loads given
    here are to be applied separately.
    Fb is a force bending a propeller blade backwards when the
    propeller mills an ice block while rotating ahead. Ff is a force
    bending a propeller blade forwards when a propeller interacts
    with an ice block while rotating ahead.
"""

H_ice=3.0 # m; Ice thickness for machinery strength design
S_ice=1.1 # Ice strength index for blade ice force
S_qice=1.15 # Ice strength index for blade ice torque

"""
I3.4.3 Design Ice Loads for Open Propeller:
  I3.4.3.1 Maximum Backward Blade Force, Fb:
"""

D_limit=0.85*H_ice**1.4 #m

if D < D_limit:
  F_b=-27.*S_ice*(n_ss*D)**0.7*(ar/Z)**0.3*(D)**2 #kN

else:
  F_b=-23.*S_ice*(n_ss*D)**0.7*(ar/Z)**0.3*(H_ice)**1.4*D #kN

print('F_b = {:6.4f} kN'.format(F_b))

"""
Fb is to be applied as a uniform pressure distribution to an area on the
back (suction) side of the blade for the following load cases:
  a) Load case 1: from 0.6R to the tip and from the blade leading edge
    to a value of 0.2 chord length.
  b) Load case 2: a load equal to 50% of the Fb is to be applied on
    the propeller tip area outside of 0.9R.
  c) Load case 5: for reversible propellers a load equal to 60% of
    the Fb is to be applied from 0.6R to the tip and from the blade
    trailing edge to a value of 0.2 chord length.
```

```
I3.4.3.2 Maximum Forward Blade Force, Ff:
"""

D_limit=2./(1.-d_h/D)*H_ice #m

if D < D_limit:
   F_f=250.*ar/Z*D**2 #kN

else:
   F_f=500./(1.-d_h/D)*H_ice*ar/Z*D #kN

print('F_f = {:6.4f} kN'.format(F_f))

"""
Ff is to be applied as a uniform pressure distribution to an area on the
face (pressure) side of the blade for the following loads cases:
   a) Load case 3: from 0.6R to the tip and from the blade leading edge
     to a value of 0.2 chord length.
   b) Load case 4: a load equal to 50% of the Ff is to be applied on
     the propeller tip area outside of 0.9R.
   c) Load case 5: for reversible propellers a load equal to 60% Ff is
     to be applied from 0.6R to the tip and from the blade trailing
     edge to a value of 0.2 chord length.

I3.4.3.3 Maximum Blade Spindle Torque, Qsmax:
   Spindle torque Qsmax around the spindle axis of the blade fitting
   shall be calculated both for the load cases described in I3.4.3.1 &
   I3.4.3.2 for Fb Ff. If these spindle torque values are less than
   the default value given below, the default minimum value shall be
   used.
"""

D_limit:1.81*H_ice #m

# F is either Fb or Ff, whichever has the greater absolute value
if np.abs(F_b) > np.abs(F_f):
   F=F_b #kN

else:
```

```
    F=F_f #kN

Q_smax=0.25*F*c_7 #kNm

print('Q_smax = {:6.4f} kNm'.format(Q_smax))

if D < D_limit:
    Q_max=105.*(1-d_h/D)*S_qice*(P_7/D)**0.16*(t_7/D)**0.6*(n_ss*D)**0.17*D**3

else:
    Q_max=202.*(1-
d_h/D)*S_qice*H_ice**1.1*(P_7/D)**0.16*(t_7/D)**0.6*(n_ss*D)**0.17*D**1.9

if Q_max < Q_smax:
    Q_max=Q_smax #kNm

else:
    Q_max=Q_max #kNm

print('Q_max = {:6.4f} kNm'.format(Q_max))

"""
For CP propellers, propeller pitch, P0.7 shall correspond to MCR in
bollard condition. If not known, P0.7 is to be taken as 0.7·P0.7n ,
where P0.7n is propeller pitch at MCR free running condition.

I3.4.3.5 Maximum Propeller Ice Thrust applied to the shaft:
"""

T_f=1.1*F_f #kN
T_b=1.1*F_b #kN

print('T_f = {:6.4f} kN'.format(T_f))
print('T_b = {:6.4f} kN'.format(T_b))

# Structural Design

"""
I3.4.6.2 Maximum Response Thrust:
    Maximum thrust along the propeller shaft line is to be calculated
```

```
   with the formulae below. The factors 2.2 and 1.5 take into account
   the dynamic magnification due to axial vibration. Alternatively, the
   propeller thrust magnification factor may be calculated by dynamic
   analysis.
"""

T_n=1.25*T #kN

T_for=T_n + 2.2*T_f #kN
T_rev=1.5*T_b #kN

print('T_for = {:6.4f} kN'.format(T_for))
print('T_rev = {:6.4f} kN'.format(T_rev))

"""
I3.4.6.3 Blade Failure Load for both Open and Nozzle Propeller:
   The force is acting at 0.8R in the weakest direction of the blade
   and at a spindle arm of 2/3 of the distance of axis of blade
   rotation of leading and trailing edge which ever is the greatest.
"""

sigma_ref=0.6*sigma_y + 0.4*sigma_u #kPa

F_ex=0.3*c_r*t_r**2*sigma_ref/(0.8*D- 2.*r)*10.**3 #kN

print('F_ex = {:6.4f} kN'.format(F_ex))

"""
I3.5 Design:
   I3.5.1 Design Principle:
      The strength of the propulsion line shall be designed
      a) for maximum loads in I3.4;
      b) such that the plastic bending of a propeller blade shall
        not cause damages in other propulsion line components;
      c) with sufficient fatigue strength.
   I3.5.3 Blade Design:
      I3.5.3.1 Maximum Blade Stresses:
         Blade stresses are to be calculated using the backward and
         forward loads given in section 4.3 & 4.4. The stresses
         shall be calculated with recognised and well documented
```

```
            FE-analysis or other acceptable alternative method. The
            stresses on the blade shall not exceed the allowable
            stresses sigma_all for the blade material given below.
"""

sigma_ref1=0.7*sigma_u
sigma_ref2=0.6*sigma_y + 0.4*sigma_u

if sigma_ref1 < sigma_ref2:
    sigma_ref=sigma_ref1

else:
    sigma_ref=sigma_ref2

S=1.5

sigma_all=sigma_ref/S
print('all. stress = {:6.4f} kPa'.format(sigma_all))


if sigma_calc < sigma_all:
    print("PASS")

else:
    print("FAIL")

"""
I3.5.3.2 Blade Edge Thickness:
    The blade edge thicknesses and tip thickness are to be greater than
    t_edge given by the following formula:
"""

## Trailing Edges:

# distance from the blade edge measured along the cylindrical sections
# from the edge and shall be 2.5% of chord length, however, not to be
# taken greater than 45 mm

# rRid starts at 0 for 0.15
```

```
rRid=13
cl=cl[rRid]
x=0.025*cl*1000. #mm

if x > 45.:
    x=45. #mm
else:
    x=x #mm

S=2.5 #safety factor
# calculate for trailing edge
p_ice=16. #MPa; ice pressure

t_te=x*S*S_ice*np.sqrt(3.*p_ice/sigma_ref)

## Leading Edges:

# distance from the blade edge measured along the cylindrical sections
# from the edge and shall be 2.5% of chord length, however, not to be
# taken greater than 45 mm

if x > 45.:
    x=45. #mm
else:
    x=x #mm

S=3.5 #safety factor
p_ice=16. #MPa; ice pressure

t_le=x*S*S_ice*np.sqrt(3.*p_ice/sigma_ref)

## Blade Tips:

# In the tip area (above 0.975R radius) x shall be taken as 2.5% of
# 0.975R section length and is to be measured perpendicularly to the
# edge, however, not to be taken greater than 45 mm

x=0.025*cltip*1000. #mm

if x > 45.:
```

```
    x=45. #mm
else:
    x=x #mm

S=5. #safety factor
p_ice=16. #MPa; ice pressure

t_tip=x*S*S_ice*np.sqrt(3.*p_ice/sigma_ref)

print('')
print('Blade Thickness Requirements:')
print('current r/R ratio  x={:8.4f} '.format(rR[rRid]))
if rRid < 12:
    print('min. trailing edge t={:8.4f} mm'.format(t_te))
    print('min. leading edge  t={:8.4f} mm'.format(t_le))
else:
    print('min. tip thick.    t={:8.4f} mm'.format(t_tip))

"""
NOTE: If the propeller is not a reversible rotation open propeller, the
trailing edge requirement can be ignored.
"""


rRedge=np.array([0.15,0.2,0.25,0.3,0.4,0.5,0.6,0.7,0.8,0.85,0.9,0.95])
t_te=np.array([0.4935,0.5360,0.5759,0.6137,0.6777,0.7246,0.7514,0.7528,\
               0.7143,0.6717,0.6024,0.4804])
t_le=np.array([0.6909,0.7504,0.8062,0.8592,0.9488,1.0145,1.0520,1.0539,\
               1.0000,0.9404,0.8433,0.6726])

rRtip=np.array([0.975,1.0])
t_tip=np.array([0.7518,0.7518])

plt.figure(figsize=(5,10))
plt.plot(t_te,rRedge,lw=2,label=r"Trailing Edge Thickness")
plt.plot(t_le,rRedge,lw=2,label=r"Leading Edge Thickness")
plt.plot(t_tip,rRtip,lw=2,label=r"Tip Thickness")
plt.xlabel(r'Blade Thickness, $t$ $[mm]$')
plt.ylabel(r'Radius Ratio, $r/R$ $[-]$')
plt.title("Minimum Required Blade Thicknesses at Each Radius")
```

```
plt.legend()
plt.grid()
plt.show()
```

Appendix D: Python Resistance Results

Coefficients:

c1= 10.864806
c2= 1.000000
c3= 0.000000
c4= 0.040000
c5= 1.000000
c7= 0.221546
c8= 29.739985
c9= 29.212536
c11= 1.548556
c14= 1.110000
c15= -1.693850
c16= 1.165591
c17= 1.691058
c19= 0.032029
c20= 1.150000
d= -0.900000
lambda= 1.019722
m1= -2.287501
m3= 10.864806
C_P1= 0.807920

Froude Numbers and Misc. Coefficients:

v_kn	Fr	Fr_T	c_6	m3(Fr^d)	m4	m4cos(lambda/Fr^2)
[kn]	[-]	[-]	[-]	[-]	[-]	[-]
10.00	0.16631	0.00000	0.2000	-8.80483	-0.00000	-0.00000
10.50	0.17462	0.00000	0.2000	-8.42656	-0.00002	0.00001
11.00	0.18294	0.00000	0.2000	-8.08104	-0.00008	-0.00004
11.50	0.19125	0.00000	0.2000	-7.76413	-0.00026	0.00024
12.00	0.19957	0.00000	0.2000	-7.47236	-0.00074	-0.00066
12.50	0.20789	0.00000	0.2000	-7.20281	-0.00173	-0.00006
13.00	0.21620	0.00000	0.2000	-6.95299	-0.00357	0.00352
13.50	0.22452	0.00000	0.2000	-6.72079	-0.00659	-0.00125
14.00	0.23283	0.00000	0.2000	-6.50438	-0.01111	-0.01110
14.50	0.24115	0.00000	0.2000	-6.30216	-0.01739	-0.00442

15.00	0.24946	0.00000	0.2000	-6.11278	-0.02560	0.01994
15.50	0.25778	0.00000	0.2000	-5.93502	-0.03579	0.03347
16.00	0.26609	0.00000	0.2000	-5.76784	-0.04791	0.01252
16.50	0.27441	0.00000	0.2000	-5.61029	-0.06183	-0.03466
17.00	0.28272	0.00000	0.2000	-5.46156	-0.07735	-0.07594
17.50	0.29104	0.00000	0.2000	-5.32092	-0.09423	-0.08141
18.00	0.29936	0.00000	0.2000	-5.18771	-0.11220	-0.04199
18.50	0.30767	0.00000	0.2000	-5.06135	-0.13102	0.02901
19.00	0.31599	0.00000	0.2000	-4.94132	-0.15041	0.10607

Resistance Components and Total Resistance:

v_kn	Fr	R_F	R_A	R_W	R_APP	R_AA	R_TR	R_I	R_T
[kn]	[-]	[kN]	[kN]	[kN]	[kN]	[kN]	[kN]	[kN]	[kN]
10.00	0.16631	46.778	10.705	8.259	19.445	1.410	0.000	631.519	106.236
10.50	0.17462	51.238	11.803	13.534	21.327	1.554	0.000	650.982	120.967
11.00	0.18294	55.887	12.953	21.248	23.292	1.706	0.000	670.201	138.548
11.50	0.19125	60.723	14.158	32.146	25.338	1.864	0.000	689.191	159.724
12.00	0.19957	65.747	15.416	47.009	27.467	2.030	0.000	707.965	185.271
12.50	0.20789	70.956	16.727	66.878	29.676	2.203	0.000	726.537	216.230
13.00	0.21620	76.351	18.092	93.001	31.967	2.382	0.000	744.918	253.848
13.50	0.22452	81.930	19.510	125.338	34.339	2.569	0.000	763.117	298.082
14.00	0.23283	87.692	20.982	164.633	36.791	2.763	0.000	781.144	349.677
14.50	0.24115	93.637	22.508	215.814	39.323	2.964	0.000	799.007	413.558
15.00	0.24946	99.764	24.087	283.167	41.936	3.172	0.000	816.714	494.009
15.50	0.25778	106.072	25.719	362.001	44.628	3.387	0.000	834.272	586.339
16.00	0.26609	112.561	27.406	440.975	47.400	3.609	0.000	851.688	679.205
16.50	0.27441	119.229	29.145	516.719	50.251	3.838	0.000	868.968	769.237
17.00	0.28272	126.077	30.938	602.085	53.181	4.074	0.000	886.117	869.285
17.50	0.29104	133.103	32.785	719.503	56.190	4.317	0.000	903.142	1001.778
18.00	0.29936	140.307	34.685	890.607	59.278	4.567	0.000	920.046	1188.349
18.50	0.30767	147.688	36.639	1127.631	62.444	4.824	0.000	936.834	1441.231
19.00	0.31599	155.247	38.646	1424.617	65.689	5.089	0.000	953.511	1754.464

Self-Propulsion Point:

v_kn	Fr	w_s	v_a	T_req	C_S	J_TS	K_TS	10K_QTS
[kn]	[-]	[-]	[m/s]	[kN]	[-]	[-]	[-]	[-]
10.00	0.16631	0.1839	4.1983	130.3379	0.77441	0.4684	0.1699	0.2315
10.50	0.17462	0.1838	4.4089	148.4104	0.79956	0.4638	0.1720	0.2336
11.00	0.18294	0.1837	4.6195	169.9801	0.83417	0.4577	0.1748	0.2364
11.50	0.19125	0.1836	4.8302	195.9595	0.87961	0.4501	0.1782	0.2399
12.00	0.19957	0.1834	5.0408	227.3025	0.93680	0.4411	0.1823	0.2439
12.50	0.20789	0.1833	5.2515	265.2854	1.00737	0.4308	0.1869	0.2486
13.00	0.21620	0.1832	5.4622	311.4373	1.09314	0.4192	0.1921	0.2537
13.50	0.22452	0.1832	5.6730	365.7072	1.19003	0.4073	0.1974	0.2590
14.00	0.23283	0.1831	5.8837	429.0069	1.29779	0.3952	0.2027	0.2642
14.50	0.24115	0.1830	6.0945	507.3801	1.43055	0.3818	0.2085	0.2700
15.00	0.24946	0.1829	6.3053	606.0826	1.59650	0.3669	0.2149	0.2763
15.50	0.25778	0.1828	6.5161	719.3594	1.77426	0.3528	0.2209	0.2822
16.00	0.26609	0.1827	6.7269	833.2941	1.92847	0.3419	0.2255	0.2867
16.50	0.27441	0.1827	6.9378	943.7509	2.05336	0.3338	0.2288	0.2900
17.00	0.28272	0.1826	7.1486	1066.4964	2.18556	0.3259	0.2321	0.2932
17.50	0.29104	0.1825	7.3595	1229.0482	2.37641	0.3154	0.2364	0.2975
18.00	0.29936	0.1825	7.5704	1457.9452	2.66413	0.3014	0.2421	0.3030
18.50	0.30767	0.1824	7.7813	1768.1979	3.05829	0.2851	0.2486	0.3094
19.00	0.31599	0.1823	7.9922	2152.4933	3.52907	0.2688	0.2550	0.3156

Efficiency and Powering:

v_kn	Fr	eta_H	eta_O	eta_D	n	n	P_D
[kn]	[-]	[-]	[-]	[-]	[1/s]	[rpm]	[kW]
10.00	0.16631	0.9988	0.5471	0.5522	2.941	176.445	989.79
10.50	0.17462	0.9986	0.5435	0.5484	3.119	187.127	1191.41
11.00	0.18294	0.9985	0.5386	0.5434	3.311	198.668	1442.69
11.50	0.19125	0.9983	0.5323	0.5371	3.521	211.232	1759.46
12.00	0.19957	0.9982	0.5247	0.5293	3.749	224.942	2160.87
12.50	0.20789	0.9981	0.5157	0.5201	3.999	239.970	2673.44
13.00	0.21620	0.9980	0.5052	0.5095	4.275	256.489	3331.98
13.50	0.22452	0.9978	0.4941	0.4982	4.570	274.195	4155.04
14.00	0.23283	0.9977	0.4825	0.4865	4.885	293.073	5176.65
14.50	0.24115	0.9976	0.4693	0.4731	5.237	314.237	6520.40
15.00	0.24946	0.9975	0.4542	0.4578	5.638	338.298	8326.11
15.50	0.25778	0.9974	0.4395	0.4430	6.059	363.543	10552.79

16.00	0.26609	0.9973	0.4279	0.4313	6.455	387.279	12961.46
16.50	0.27441	0.9973	0.4192	0.4225	6.818	409.100	15454.73
17.00	0.28272	0.9972	0.4106	0.4137	7.197	431.802	18375.12
17.50	0.29104	0.9971	0.3990	0.4020	7.655	459.307	22432.80
18.00	0.29936	0.9970	0.3833	0.3862	8.239	494.361	28492.37
18.50	0.30767	0.9969	0.3647	0.3674	8.954	537.226	37334.78
19.00	0.31599	0.9969	0.3457	0.3483	9.754	585.232	49239.70

Appendix E: NavCAD Holtrop and Mennen Results

Resistance

26 Apr 2021 12:34 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Holtrop**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Holtrop.hcnc**

Analysis parameters

Vessel drag		ITTC-78 (CT)
Technique:	[Calc]	Prediction
Prediction:		Holtrop
Reference ship:		
Model LWL:		
Expansion:		Custom
Friction line:		ITTC-57
Hull form factor:	[On]	1.421
Speed corr:	[Off]	
Spray drag corr:	[Off]	
Corr allowance:		0.000344
Roughness [mm]:	[On]	0.15

Added drag		
Appendage:	[Calc]	Holtrop (Component)
Wind:	[Off]	
Seas:	[Off]	
Shallow/channel:	[Off]	
Towed:	[Off]	
Margin:	[Off]	

Water properties	
Water type:	Salt
Density:	1026.00 kg/m3
Viscosity:	1.18920e-6 m2/s

Prediction method check [Holtrop]

Parameters	FN [design]	CP	LWL/BWL	BWL/T	Lambda
Value	0.25	0.80	4.51	4.58*	1.02
Range	0.06··0.26	0.55··0.85	3.90··14.90	2.10··4.00	0.01··1.07

Prediction results

	SPEED COEFS		ITTC-78 COEFS						
SPEED [kt]	FN	FV	RN	CF	[CV/CF]	CR	dCF	CA	CT
10.00	0.166	0.372	4.22e8	0.001709	1.421	0.000335	0.000000	0.000344	0.003107
11.00	0.183	0.409	4.64e8	0.001687	1.421	0.000706	0.000000	0.000344	0.003447
12.00	0.200	0.446	5.06e8	0.001668	1.421	0.001306	0.000000	0.000344	0.004021
13.00	0.216	0.484	5.49e8	0.001651	1.421	0.002196	0.000000	0.000344	0.004887
14.00	0.233	0.521	5.91e8	0.001636	1.421	0.003348	0.000000	0.000344	0.006017
14.50	0.241	0.539	6.12e8	0.001628	1.421	0.004087	0.000000	0.000344	0.006745
+ 15.00 +	0.249	0.558	6.33e8	0.001621	1.421	0.005009	0.000000	0.000344	0.007657
15.50	0.258	0.576	6.54e8	0.001614	1.421	0.006002	0.000000	0.000344	0.008641
16.00 !	0.266	0.595	6.75e8	0.001608	1.421	0.006869	0.000000	0.000344	0.009498
17.00 !	0.283	0.632	7.18e8	0.001596	1.421	0.008295	0.000000	0.000344	0.010906

	RESISTANCE							
SPEED [kt]	RBARE [kN]	RAPP [kN]	RWIND [kN]	RSEAS [kN]	RCHAN [kN]	RTOWED [kN]	RMARGIN [kN]	RTOTAL [kN]
10.00	81.47	24.65	0.00	0.00	0.00	0.00	0.00	106.12
11.00	109.37	29.50	0.00	0.00	0.00	0.00	0.00	138.87
12.00	151.81	34.75	0.00	0.00	0.00	0.00	0.00	186.56
13.00	216.55	40.41	0.00	0.00	0.00	0.00	0.00	256.96
14.00	309.20	46.48	0.00	0.00	0.00	0.00	0.00	355.67
14.50	371.81	49.66	0.00	0.00	0.00	0.00	0.00	421.46
+ 15.00 +	451.71	52.94	0.00	0.00	0.00	0.00	0.00	504.64
15.50	544.30	56.31	0.00	0.00	0.00	0.00	0.00	600.62
16.00 !	637.55	59.79	0.00	0.00	0.00	0.00	0.00	697.34
17.00 !	826.42	67.04	0.00	0.00	0.00	0.00	0.00	893.46

	EFFECTIVE POWER		OTHER		
SPEED [kt]	PEBARE [kW]	PETOTAL [kW]	CTLR	CTLT	RBARE/W
10.00	419.1	545.9	0.00425	0.03942	0.00109
11.00	618.9	785.8	0.00895	0.04374	0.00146
12.00	937.2	1151.7	0.01657	0.05102	0.00203
13.00	1448.2	1718.5	0.02787	0.06201	0.00290
14.00	2226.9	2561.6	0.04249	0.07634	0.00414
14.50	2773.5	3143.9	0.05185	0.08558	0.00498
+ 15.00 +	3485.7	3894.2	0.06356	0.09715	0.00605
15.50	4340.2	4789.3	0.07616	0.10964	0.00729
16.00 !	5247.7	5739.9	0.08716	0.12052	0.00853
17.00 !	7227.5	7813.8	0.10525	0.13838	0.01106

Resistance

26 Apr 2021 12:34 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Holtrop**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Holtrop.hcnc**

Hull data

General		
Configuration:		**Monohull**
Chine type:		**Round/multiple**
Length on WL:		**97.569 m**
Max beam on WL:	[LWL/BWL 4.514]	**21.616 m**
Max molded draft:	[BWL/T 4.580]	**4.720 m**
Displacement:	[CB 0.746]	**7618.28 t**
Wetted surface:	[CS 2.269]	**1931.231 m2**
ITTC-78 (CT)		
LCB fwd TR:	[XCB/LWL 0.516]	**50.320 m**
LCF fwd TR:	[XCF/LWL 0.484]	**47.220 m**
Max section area:	[CX 0.931]	**95.036 m2**
Waterplane area:	[CWP 0.918]	**1936.740 m2**
Bulb section area:		**0.000 m2**
Bulb ctr below WL:		**0.000 m**
Bulb nose fwd TR:		**0.000 m**
Imm transom area:	[ATR/AX 0.000]	**0.000 m2**
Transom beam WL:	[BTR/BWL 0.000]	**0.000 m**
Transom immersion:	[TTR/T 0.000]	**0.000 m**
Half entrance angle:		**55.57 deg**
Bow shape factor:	[WL flow]	**1.0**
Stern shape factor:	[WL flow]	**1.0**

Planing		
Proj chine length:		0.000 m
Proj bottom area:		0.000 m2
LCG fwd TR:	[XCG/LP 0.000]	0.000 m
VCG below WL:		0.000 m
Aft station (fwd TR):		0.000 m
Deadrise:		0.00 deg
Chine beam:		0.000 m
Chine ht below WL:		0.000 m
Fwd station (fwd TR):		0.000 m
Deadrise:		0.00 deg
Chine beam:		0.000 m
Chine ht below WL:		0.000 m
Propulsor type:		Propeller
Max prop diameter:		3048.0 mm
Shaft angle to WL:		10.00 deg
Position fwd TR:		0.000 m
Position below WL:		0.000 m
Transom lift device:		Flap
Device count:		0
Span:		0.000 m
Chord length:		0.000 m
Deflection angle:		0.00 deg
Tow point fwd TR:		0.000 m
Tow point below WL:		0.000 m
Foil assist (planing)		
Foil count:		0
Total planform area:		0.000 m2
LCE fwd TR:		0.000 m
VCE below WL:		0.000 m
Lift-drag ratio:		0.0
Lift fraction (design):		0.00
Design speed:		0.00 kt

Report ID20210426-1234

HydroComp NavCad 2020 [Premium] 20.00.0085.0518.U0948

Resistance

26 Apr 2021 12:34 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Holtrop**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Holtrop.hcnc**

Appendage data

General		Skeg/Keel	
Definition:	**Component**	Count:	**1**
Percent of hull drag:	**0.00 %**	Type:	**Skeg**
Planing influence		Mean length:	**0.000 m**
LCE fwd TR:	**0.000 m**	Mean width:	**0.000 m**
VCE below WL:	**0.000 m**	Height aft:	**0.000 m**
Shafting		Height mid:	**0.000 m**
Count:	**2**	Height fwd:	**0.000 m**
Max prop diameter:	**3048.0 mm**	Projected area:	**0.000 m2**
Shaft angle to WL:	**10.00 deg**	Wetted surface:	**87.330 m2**
Exposed shaft length:	**0.000 m**	**Stabilizer**	
Shaft diameter:	**0.000 m**	Count:	**0**
Wetted surface:	**14.860 m2**	Root chord:	**0.000 m**
Strut bossing length:	**0.000 m**	Tip chord:	**0.000 m**
Bossing diameter:	**0.000 m**	Span:	**0.000 m**
Wetted surface:	**6.690 m2**	T/C ratio:	**0.000**
Hull bossing length:	**0.000 m**	LE sweep:	**0.00 deg**
Bossing diameter:	**0.000 m**	Wetted surface:	**0.000 m2**
Wetted surface:	**9.480 m2**	Projected area:	**0.000 m2**
Strut (per shaft line)		Dynamic multiplier:	**1.00**
Count:	**2**	**Bilge keel**	
Root chord:	**0.000 m**	Count:	**2**
Tip chord:	**0.000 mm**	Mean length:	**0.000 m**
Span:	**0.000 m**	Mean base width:	**0.000 m**
T/C ratio:	**0.000**	Mean projection:	**0.000 m**
Projected area:	**0.000 m2**	Wetted surface:	**196.030 m2**
Wetted surface:	**3.723 m2**	**Tunnel thruster**	
Exposed palm depth:	**0.000 m**	Count:	**2**
Exposed palm width:	**0.000 m**	Diameter:	**0.000 m**
Rudder		**Sonar dome**	
Count:	**1**	Count:	**0**
Rudder location:	**Behind propeller**	Wetted surface:	**0.000 m2**
Type:	**Balanced foil**	**Miscellaneous**	
Root chord:	**0.000 m**	Count:	**0**
Tip chord:	**0.000 m**	Drag area:	**0.000 m2**
Span:	**0.000 m**	Drag coef:	**0.00**
T/C ratio:	**0.000**		
LE sweep:	**0.00 deg**		
Projected area:	**0.000 m2**		
Wetted surface:	**16.720 m2**		

Environment data

Wind		Seas	
Wind speed:	**0.00 kt**	Significant wave ht:	**0.000 m**
Angle off bow:	**0.00 deg**	Modal wave period:	**0.0 sec**
Gradient correction:	**Off**	**Shallow/channel**	
Exposed hull		Water depth:	**0.000 m**
Transverse area:	**0.000 m2**	Type:	**Shallow water**
VCE above WL:	**0.000 m**	Channel width:	**0.000 m**
Profile area:	**66.890 m2**	Channel side slope:	**0.00 deg**
Superstructure		Hull girth:	**0.000 m**
Superstructure shape:	**Ferry/Liner**		
Transverse area:	**0.000 m2**		
VCE above WL:	**0.000 m**		
Profile area:	**41.810 m2**		

Resistance

26 Apr 2021 12:34 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Holtrop**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Holtrop.hcnc**

Symbols and values

SPEED = Vessel speed
FN = Froude number [LWL]
FV = Froude number [VOL]

RN = Reynolds number [LWL]
CF = Frictional resistance coefficient
CV/CF = Viscous/frictional resistance coefficient ratio [dynamic form factor]
CR = Residuary resistance coefficient
dCF = Added frictional resistance coefficient for roughness
CA = Correlation allowance [dynamic]
CT = Total bare-hull resistance coefficient

RBARE = Bare-hull resistance
RAPP = Additional appendage resistance
RWIND = Additional wind resistance
RSEAS = Additional sea-state resistance
RCHAN = Additional shallow/channel resistance
RTOWED = Additional towed object resistance
RMARGIN = Resistance margin
RTOTAL = Total vessel resistance

PEBARE = Bare-hull effective power
PETOTAL = Total effective power

CTLR = Telfer residuary resistance coefficient
CTLT = Telfer total bare-hull resistance coefficient
RBARE/W = Bare-hull resistance to weight ratio

+ = Design speed indicator
* = Exceeds parameter limit

Appendix F: NavCAD Andersen Results

Resistance

26 Apr 2021 12:32 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Andersen**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Andersen.hcnc**

Analysis parameters

Vessel drag		ITTC-78 (CT)	Added drag		
Technique:	[Calc]	Prediction	Appendage:	[Calc]	Holtrop (Component)
Prediction:		Andersen	Wind:	[Off]	
Reference ship:			Seas:	[Off]	
Model LWL:			Shallow/channel:	[Off]	
Expansion:		Custom	Towed:	[Off]	
Friction line:		ITTC-57	Margin:	[Off]	
Hull form factor:	[On]	1.421	**Water properties**		
Speed corr:	[Off]		Water type:		Salt
Spray drag corr:	[Off]		Density:		1026.00 kg/m3
Corr allowance:		0.000344	Viscosity:		1.18920e-6 m2/s
Roughness [mm]:	[On]	0.15			

Prediction method check [Andersen]

Parameters	FN [design]	CVOL	CB	LWL/BWL
Value	0.25	5.00	0.75	4.51*
Range	0.05··0.33	4.00··6.00	0.55··0.85	5.00··8.00

Prediction results

	SPEED COEFS		ITTC-78 COEFS						
SPEED [kt]	FN	FV	RN	CF	[CV/CF]	CR	dCF	CA	CT
10.00	0.166	0.372	4.22e8	0.001709	1.421	0.000020	0.000000	0.000344	0.002792
11.00	0.183	0.409	4.64e8	0.001687	1.421	0.000259	0.000000	0.000344	0.003001
12.00	0.200	0.446	5.06e8	0.001668	1.421	0.000585	0.000000	0.000344	0.003300
13.00	0.216	0.483	5.49e8	0.001651	1.421	0.001044	0.000000	0.000344	0.003735
14.00	0.233	0.521	5.91e8	0.001636	1.421	0.001721	0.000000	0.000344	0.004389
14.50	0.241	0.539	6.12e8	0.001628	1.421	0.002187	0.000000	0.000344	0.004845
+ 15.00 +	0.249	0.558	6.33e8	0.001621	1.421	0.002778	0.000000	0.000344	0.005426
15.50	0.258	0.576	6.54e8	0.001614	1.421	0.003542	0.000000	0.000344	0.006180
16.00	0.266	0.595	6.75e8	0.001608	1.421	0.004542	0.000000	0.000344	0.007171
17.00	0.283	0.632	7.18e8	0.001596	1.421	0.007559	0.000000	0.000344	0.010170

	RESISTANCE							
SPEED [kt]	RBARE [kN]	RAPP [kN]	RWIND [kN]	RSEAS [kN]	RCHAN [kN]	RTOWED [kN]	RMARGIN [kN]	RTOTAL [kN]
10.00	73.21	24.65	0.00	0.00	0.00	0.00	0.00	97.86
11.00	95.22	29.50	0.00	0.00	0.00	0.00	0.00	124.71
12.00	124.60	34.75	0.00	0.00	0.00	0.00	0.00	159.35
13.00	165.49	40.41	0.00	0.00	0.00	0.00	0.00	205.91
14.00	225.56	46.48	0.00	0.00	0.00	0.00	0.00	272.04
14.50	267.07	49.66	0.00	0.00	0.00	0.00	0.00	316.72
+ 15.00 +	320.11	52.94	0.00	0.00	0.00	0.00	0.00	373.04
15.50	389.31	56.31	0.00	0.00	0.00	0.00	0.00	445.63
16.00	481.32	59.79	0.00	0.00	0.00	0.00	0.00	541.11
17.00	770.66	67.04	0.00	0.00	0.00	0.00	0.00	837.70

	EFFECTIVE POWER		OTHER		
SPEED [kt]	PEBARE [kW]	PETOTAL [kW]	CTLR	CTLT	RBARE/W
10.00	376.6	503.5	0.00026	0.03542	0.00098
11.00	538.8	705.7	0.00329	0.03807	0.00127
12.00	769.2	983.7	0.00742	0.04186	0.00167
13.00	1106.8	1377.1	0.01325	0.04738	0.00221
14.00	1624.5	1959.3	0.02183	0.05568	0.00302
14.50	1992.2	2362.6	0.02774	0.06146	0.00357
+ 15.00 +	2470.2	2878.7	0.03525	0.06884	0.00428
15.50	3104.3	3553.4	0.04493	0.07840	0.00521
16.00	3961.8	4453.9	0.05762	0.09097	0.00644
17.00	6739.8	7326.2	0.09589	0.12902	0.01031

Resistance

26 Apr 2021 12:32 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Andersen**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Andersen.hcnc**

Hull data

General		
Configuration:		**Monohull**
Chine type:		**Round/multiple**
Length on WL:		**97.569 m**
Max beam on WL:	[LWL/BWL 4.513]	**21.620 m**
Max molded draft:	[BWL/T 4.581]	**4.720 m**
Displacement:	[CB 0.746]	**7619.69 t**
Wetted surface:	[CS 2.269]	**1931.231 m2**
ITTC-78 (CT)		
LCB fwd TR:	[XCB/LWL 0.516]	**50.320 m**
LCF fwd TR:	[XCF/LWL 0.484]	**47.220 m**
Max section area:	[CX 0.931]	**95.036 m2**
Waterplane area:	[CWP 0.918]	**1936.740 m2**
Bulb section area:		**0.000 m2**
Bulb ctr below WL:		**0.000 m**
Bulb nose fwd TR:		**0.000 m**
Imm transom area:	[ATR/AX 0.000]	**0.000 m2**
Transom beam WL:	[BTR/BWL 0.000]	**0.000 m**
Transom immersion:	[TTR/T 0.000]	**0.000 m**
Half entrance angle:		**55.58 deg**
Bow shape factor:	[WL flow]	**1.0**
Stern shape factor:	[WL flow]	**1.0**

Planing		
Proj chine length:		***0.000 m***
Proj bottom area:		***0.000 m2***
LCG fwd TR:	*[XCG/LP 0.000]*	***0.000 m***
VCG below WL:		***0.000 m***
Aft station (fwd TR):		***0.000 m***
Deadrise:		***0.00 deg***
Chine beam:		***0.000 m***
Chine ht below WL:		***0.000 m***
Fwd station (fwd TR):		***0.000 m***
Deadrise:		***0.00 deg***
Chine beam:		***0.000 m***
Chine ht below WL:		***0.000 m***
Propulsor type:		***Propeller***
Max prop diameter:		***3048.0 mm***
Shaft angle to WL:		***10.00 deg***
Position fwd TR:		***0.000 m***
Position below WL:		***0.000 m***
Transom lift device:		***Flap***
Device count:		***0***
Span:		***0.000 m***
Chord length:		***0.000 m***
Deflection angle:		***0.00 deg***
Tow point fwd TR:		***0.000 m***
Tow point below WL:		***0.000 m***
Foil assist (planing)		
Foil count:		***0***
Total planform area:		***0.000 m2***
LCE fwd TR:		***0.000 m***
VCE below WL:		***0.000 m***
Lift-drag ratio:		***0.0***
Lift fraction (design):		***0.00***
Design speed:		***0.00 kt***

Resistance

26 Apr 2021 12:32 PM
HydroComp NavCad 2020 [Premium]

Project ID	**MV Yahtse Andersen**
Description	**Ro-Ro Car/Cargo Alaskan Ferry**
File name	**NAME 4175 MV Yahtse NavCAD Andersen.hcnc**

Appendage data

General		**Skeg/Keel**	
Definition:	**Component**	Count:	**1**
Percent of hull drag:	**0.00 %**	Type:	**Skeg**
Planing influence		Mean length:	**0.000 m**
LCE fwd TR:	**0.000 m**	Mean width:	**0.000 m**
VCE below WL:	**0.000 m**	Height aft:	**0.000 m**
Shafting		Height mid:	**0.000 m**
Count:	**2**	Height fwd:	**0.000 m**
Max prop diameter:	**3048.0 mm**	Projected area:	**0.000 m2**
Shaft angle to WL:	**10.00 deg**	Wetted surface:	**87.330 m2**
Exposed shaft length:	**0.000 m**	**Stabilizer**	
Shaft diameter:	**0.000 m**	Count:	**0**
Wetted surface:	**14.860 m2**	Root chord:	**0.000 m**
Strut bossing length:	**0.000 m**	Tip chord:	**0.000 m**
Bossing diameter:	**0.000 m**	Span:	**0.000 m**
Wetted surface:	**6.690 m2**	T/C ratio:	**0.000**
Hull bossing length:	**0.000 m**	LE sweep:	**0.00 deg**
Bossing diameter:	**0.000 m**	Wetted surface:	**0.000 m2**
Wetted surface:	**9.480 m2**	Projected area:	**0.000 m2**
Strut (per shaft line)		Dynamic multiplier:	**1.00**
Count:	**2**	**Bilge keel**	
Root chord:	**0.000 m**	Count:	**2**
Tip chord:	**0.000 mm**	Mean length:	**0.000 m**
Span:	**0.000 m**	Mean base width:	**0.000 m**
T/C ratio:	**0.000**	Mean projection:	**0.000 m**
Projected area:	**0.000 m2**	Wetted surface:	**196.030 m2**
Wetted surface:	**3.723 m2**	**Tunnel thruster**	
Exposed palm depth:	**0.000 m**	Count:	**2**
Exposed palm width:	**0.000 m**	Diameter:	**0.000 m**
Rudder		**Sonar dome**	
Count:	**1**	Count:	**0**
Rudder location:	**Behind propeller**	Wetted surface:	**0.000 m2**
Type:	**Balanced foil**	**Miscellaneous**	
Root chord:	**0.000 m**	Count:	**0**
Tip chord:	**0.000 m**	Drag area:	**0.000 m2**
Span:	**0.000 m**	Drag coef:	**0.00**
T/C ratio:	**0.000**		
LE sweep:	**0.00 deg**		
Projected area:	**0.000 m2**		
Wetted surface:	**16.720 m2**		

Environment data

Wind		**Seas**	
Wind speed:	**0.00 kt**	Significant wave ht:	**0.000 m**
Angle off bow:	**0.00 deg**	Modal wave period:	**0.0 sec**
Gradient correction:	**Off**	**Shallow/channel**	
Exposed hull		Water depth:	**0.000 m**
Transverse area:	**0.000 m2**	Type:	**Shallow water**
VCE above WL:	**0.000 m**	Channel width:	**0.000 m**
Profile area:	**66.890 m2**	Channel side slope:	**0.00 deg**
Superstructure		Hull girth:	**0.000 m**
Superstructure shape:	**Ferry/Liner**		
Transverse area:	**0.000 m2**		
VCE above WL:	**0.000 m**		
Profile area:	**41.810 m2**		

Resistance

26 Apr 2021 12:32 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Andersen**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Andersen.hcnc**

Symbols and values

SPEED = Vessel speed
FN = Froude number [LWL]
FV = Froude number [VOL]

RN = Reynolds number [LWL]
CF = Frictional resistance coefficient
CV/CF = Viscous/frictional resistance coefficient ratio [dynamic form factor]
CR = Residuary resistance coefficient
dCF = Added frictional resistance coefficient for roughness
CA = Correlation allowance [dynamic]
CT = Total bare-hull resistance coefficient

RBARE = Bare-hull resistance
RAPP = Additional appendage resistance
RWIND = Additional wind resistance
RSEAS = Additional sea-state resistance
RCHAN = Additional shallow/channel resistance
RTOWED = Additional towed object resistance
RMARGIN = Resistance margin
RTOTAL = Total vessel resistance

PEBARE = Bare-hull effective power
PETOTAL = Total effective power

CTLR = Telfer residuary resistance coefficient
CTLT = Telfer total bare-hull resistance coefficient
RBARE/W = Bare-hull resistance to weight ratio

+ = Design speed indicator
* = Exceeds parameter limit

Appendix G: NavCAD Fung (CRTS) Results

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (CRTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (CRTS).hcnc**

Analysis parameters

Vessel drag		ITTC-78 (CT)
Technique:	[Calc]	Prediction
Prediction:		Fung (CRTS)
Reference ship:		
Model LWL:		
Expansion:		Custom
Friction line:		ITTC-57
Hull form factor:	[On]	1.421
Speed corr:	[Off]	
Spray drag corr:	[Off]	
Corr allowance:		0.000344
Roughness [mm]:	[On]	0.15

Added drag		
Appendage:	[Calc]	Holtrop (Component)
Wind:	[Off]	
Seas:	[Off]	
Shallow/channel:	[Off]	
Towed:	[Off]	
Margin:	[Off]	

Water properties	
Water type:	Salt
Density:	1026.00 kg/m3
Viscosity:	1.18920e-6 m2/s

Prediction method check [Fung (CRTS)]

Parameters	FN [design]	CVOL	CP	BWL/T	IE	ABT/AX	ATR/AX	BTR/BWL	TTR/T
Value	0.25	5.00	0.80*	4.58	23.5*	0.00	0.00	0.00	0.00
Range	0.18··0.40	4.85··11.27	0.52··0.70	2.20··5.20	4.0··20.0	0.00··0.10	0.00··0.40	0.00··0.85	0.00··0.42

Prediction results

	SPEED COEFS		ITTC-78 COEFS						
SPEED [kt]	FN	FV	RN	CF	[CV/CF]	CR	dCF	CA	CT
10.00 !	0.166	0.372	4.22e8	0.001709	1.421	0.000001	0.000000	0.000344	0.002773
11.00	0.183	0.409	4.64e8	0.001687	1.421	0.000001	0.000000	0.000344	0.002743
12.00	0.200	0.446	5.06e8	0.001668	1.421	0.000196	0.000000	0.000344	0.002911
13.00	0.216	0.484	5.49e8	0.001651	1.421	0.000530	0.000000	0.000344	0.003220
14.00	0.233	0.521	5.91e8	0.001636	1.421	0.000799	0.000000	0.000344	0.003467
14.50	0.241	0.539	6.12e8	0.001628	1.421	0.000948	0.000000	0.000344	0.003606
+ 15.00 +	0.249	0.558	6.33e8	0.001621	1.421	0.001131	0.000000	0.000344	0.003779
15.50	0.258	0.576	6.54e8	0.001614	1.421	0.001372	0.000000	0.000344	0.004011
16.00	0.266	0.595	6.75e8	0.001608	1.421	0.001694	0.000000	0.000344	0.004323
17.00	0.283	0.632	7.18e8	0.001596	1.421	0.002594	0.000000	0.000344	0.005206

	RESISTANCE								
SPEED [kt]	RBARE [kN]	RAPP [kN]	RWIND [kN]	RSEAS [kN]	RCHAN [kN]	RTOWED [kN]	RMARGIN [kN]	RTOTAL [kN]	
10.00 !	72.70	24.65	0.00	0.00	0.00	0.00	0.00	97.36	
11.00	87.02	29.50	0.00	0.00	0.00	0.00	0.00	116.52	
12.00	109.90	34.75	0.00	0.00	0.00	0.00	0.00	144.65	
13.00	142.69	40.41	0.00	0.00	0.00	0.00	0.00	183.11	
14.00	178.17	46.48	0.00	0.00	0.00	0.00	0.00	224.65	
14.50	198.77	49.66	0.00	0.00	0.00	0.00	0.00	248.43	
+ 15.00 +	222.95	52.94	0.00	0.00	0.00	0.00	0.00	275.89	
15.50	252.64	56.31	0.00	0.00	0.00	0.00	0.00	308.95	
16.00	290.18	59.79	0.00	0.00	0.00	0.00	0.00	349.97	
17.00	394.48	67.04	0.00	0.00	0.00	0.00	0.00	461.52	

	EFFECTIVE POWER		OTHER		
SPEED [kt]	PEBARE [kW]	PETOTAL [kW]	CTLR	CTLT	RBARE/W
10.00 !	374.0	500.8	0.00001	0.03518	0.00097
11.00	492.4	659.4	0.00001	0.03480	0.00116
12.00	678.4	893.0	0.00248	0.03693	0.00147
13.00	954.3	1224.6	0.00672	0.04086	0.00191
14.00	1283.2	1618.0	0.01014	0.04399	0.00238
14.50	1482.7	1853.1	0.01203	0.04575	0.00266
+ 15.00 +	1720.4	2128.9	0.01436	0.04795	0.00298
15.50	2014.5	2463.5	0.01741	0.05089	0.00338
16.00	2388.5	2880.7	0.02150	0.05485	0.00388
17.00	3449.9	4036.2	0.03292	0.06605	0.00528

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (CRTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (CRTS).hcnc**

Hull data

General		
Configuration:		**Monohull**
Chine type:		**Round/multiple**
Length on WL:		**97.569 m**
Max beam on WL:	[LWL/BWL 4.514]	**21.616 m**
Max molded draft:	[BWL/T 4.580]	**4.720 m**
Displacement:	[CB 0.746]	**7618.28 t**
Wetted surface:	[CS 2.269]	**1931.231 m2**
ITTC-78 (CT)		
LCB fwd TR:	[XCB/LWL 0.516]	**50.320 m**
LCF fwd TR:	[XCF/LWL 0.484]	**47.220 m**
Max section area:	[CX 0.931]	**95.036 m2**
Waterplane area:	[CWP 0.918]	**1936.740 m2**
Bulb section area:		**0.000 m2**
Bulb ctr below WL:		**0.000 m**
Bulb nose fwd TR:		**0.000 m**
Imm transom area:	[ATR/AX 0.000]	**0.000 m2**
Transom beam WL:	[BTR/BWL 0.000]	**0.000 m**
Transom immersion:	[TTR/T 0.000]	**0.000 m**
Half entrance angle:		**23.51 deg**
Bow shape factor:	[WL flow]	**1.0**
Stern shape factor:	[WL flow]	**1.0**

Planing		
Proj chine length:		0.000 m
Proj bottom area:		0.000 m2
LCG fwd TR:	[XCG/LP 0.000]	0.000 m
VCG below WL:		0.000 m
Aft station (fwd TR):		0.000 m
Deadrise:		0.00 deg
Chine beam:		0.000 m
Chine ht below WL:		0.000 m
Fwd station (fwd TR):		0.000 m
Deadrise:		0.00 deg
Chine beam:		0.000 m
Chine ht below WL:		0.000 m
Propulsor type:		Propeller
Max prop diameter:		3048.0 mm
Shaft angle to WL:		10.00 deg
Position fwd TR:		0.000 m
Position below WL:		0.000 m
Transom lift device:		Flap
Device count:		0
Span:		0.000 m
Chord length:		0.000 m
Deflection angle:		0.00 deg
Tow point fwd TR:		0.000 m
Tow point below WL:		0.000 m
Foil assist (planing)		
Foil count:		0
Total planform area:		0.000 m2
LCE fwd TR:		0.000 m
VCE below WL:		0.000 m
Lift-drag ratio:		0.0
Lift fraction (design):		0.00
Design speed:		0.00 kt

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (CRTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (CRTS).hcnc**

Appendage data

General	
Definition:	**Component**
Percent of hull drag:	**0.00 %**
Planing influence	
LCE fwd TR:	**0.000 m**
VCE below WL:	**0.000 m**
Shafting	
Count:	**2**
Max prop diameter:	**3048.0 mm**
Shaft angle to WL:	**10.00 deg**
Exposed shaft length:	**0.000 m**
Shaft diameter:	**0.000 m**
Wetted surface:	**14.860 m2**
Strut bossing length:	**0.000 m**
Bossing diameter:	**0.000 m**
Wetted surface:	**6.690 m2**
Hull bossing length:	**0.000 m**
Bossing diameter:	**0.000 m**
Wetted surface:	**9.480 m2**
Strut (per shaft line)	
Count:	**2**
Root chord:	**0.000 m**
Tip chord:	**0.000 mm**
Span:	**0.000 m**
T/C ratio:	**0.000**
Projected area:	**0.000 m2**
Wetted surface:	**3.723 m2**
Exposed palm depth:	**0.000 m**
Exposed palm width:	**0.000 m**
Rudder	
Count:	**1**
Rudder location:	**Behind propeller**
Type:	**Balanced foil**
Root chord:	**0.000 m**
Tip chord:	**0.000 m**
Span:	**0.000 m**
T/C ratio:	**0.000**
LE sweep:	**0.00 deg**
Projected area:	**0.000 m2**
Wetted surface:	**16.720 m2**

Skeg/Keel	
Count:	**1**
Type:	**Skeg**
Mean length:	**0.000 m**
Mean width:	**0.000 m**
Height aft:	**0.000 m**
Height mid:	**0.000 m**
Height fwd:	**0.000 m**
Projected area:	**0.000 m2**
Wetted surface:	**87.330 m2**
Stabilizer	
Count:	**0**
Root chord:	**0.000 m**
Tip chord:	**0.000 m**
Span:	**0.000 m**
T/C ratio:	**0.000**
LE sweep:	**0.00 deg**
Wetted surface:	**0.000 m2**
Projected area:	**0.000 m2**
Dynamic multiplier:	**1.00**
Bilge keel	
Count:	**2**
Mean length:	**0.000 m**
Mean base width:	**0.000 m**
Mean projection:	**0.000 m**
Wetted surface:	**196.030 m2**
Tunnel thruster	
Count:	**2**
Diameter:	**0.000 m**
Sonar dome	
Count:	**0**
Wetted surface:	**0.000 m2**
Miscellaneous	
Count:	**0**
Drag area:	**0.000 m2**
Drag coef:	**0.00**

Environment data

Wind	
Wind speed:	**0.00 kt**
Angle off bow:	**0.00 deg**
Gradient correction:	**Off**
Exposed hull	
Transverse area:	**0.000 m2**
VCE above WL:	**0.000 m**
Profile area:	**66.890 m2**
Superstructure	
Superstructure shape:	**Ferry/Liner**
Transverse area:	**0.000 m2**
VCE above WL:	**0.000 m**
Profile area:	**41.810 m2**

Seas	
Significant wave ht:	**0.000 m**
Modal wave period:	**0.0 sec**
Shallow/channel	
Water depth:	**0.000 m**
Type:	**Shallow water**
Channel width:	**0.000 m**
Channel side slope:	**0.00 deg**
Hull girth:	**0.000 m**

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (CRTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (CRTS).hcnc**

Symbols and values

SPEED = Vessel speed
FN = Froude number [LWL]
FV = Froude number [VOL]

RN = Reynolds number [LWL]
CF = Frictional resistance coefficient
CV/CF = Viscous/frictional resistance coefficient ratio [dynamic form factor]
CR = Residuary resistance coefficient
dCF = Added frictional resistance coefficient for roughness
CA = Correlation allowance [dynamic]
CT = Total bare-hull resistance coefficient

RBARE = Bare-hull resistance
RAPP = Additional appendage resistance
RWIND = Additional wind resistance
RSEAS = Additional sea-state resistance
RCHAN = Additional shallow/channel resistance
RTOWED = Additional towed object resistance
RMARGIN = Resistance margin
RTOTAL = Total vessel resistance

PEBARE = Bare-hull effective power
PETOTAL = Total effective power

CTLR = Telfer residuary resistance coefficient
CTLT = Telfer total bare-hull resistance coefficient
RBARE/W = Bare-hull resistance to weight ratio

+ = Design speed indicator
* = Exceeds parameter limit

Appendix H: NavCAD Fung (HSTS) Results

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID	**MV Yahtse Fung (HSTS)**
Description	**Ro-Ro Car/Cargo Alaskan Ferry**
File name	**NAME 4175 MV Yahtse NavCAD Fung (HSTS).hcnc**

Analysis parameters

Vessel drag		**ITTC-78 (CT)**
Technique:	**[Calc]**	**Prediction**
Prediction:		**Fung (HSTS)**
Reference ship:		
Model LWL:		
Expansion:		**Custom**
Friction line:		**ITTC-57**
Hull form factor:	**[On]**	**1.421**
Speed corr:	**[Off]**	
Spray drag corr:	**[Off]**	
Corr allowance:		**0.000344**
Roughness [mm]:	**[On]**	**0.15**

Added drag		
Appendage:	**[Calc]**	**Holtrop (Component)**
Wind:	**[Off]**	
Seas:	**[Off]**	
Shallow/channel:	**[Off]**	
Towed:	**[Off]**	
Margin:	**[Off]**	
Water properties		
Water type:		**Salt**
Density:		**1026.00 kg/m3**
Viscosity:		**1.18920e-6 m2/s**

Prediction method check [Fung (HSTS)]

Parameters	FN [design]	CVOL	CP	LWL/BWL	BWL/T	XCB/LWL	IE	ATR/AX	BTR/BWL
Value	**0.25**	**5.00**	**0.80***	**4.51**	**4.58**	**0.516***	**23.2**	**0.00**	**0.00**
Range	**0.15··0.40**	**4.73··10.60**	**0.55··0.72**	**3.40··12.10**	**2.10··6.90**	**0.440··0.510**	**3.7··26.0**	**0.00··0.54**	**0.00··0.95**

Prediction results

	SPEED COEFS		ITTC-78 COEFS						
SPEED [kt]	FN	FV	RN	CF	[CV/CF]	CR	dCF	CA	CT
10.00	**0.166**	**0.372**	**4.22e8**	**0.001709**	**1.421**	**0.006985**	**0.000000**	**0.000344**	**0.009757**
11.00	**0.183**	**0.409**	**4.64e8**	**0.001687**	**1.421**	**0.005689**	**0.000000**	**0.000344**	**0.008431**
12.00	**0.200**	**0.446**	**5.06e8**	**0.001668**	**1.421**	**0.005020**	**0.000000**	**0.000344**	**0.007735**
13.00	**0.216**	**0.484**	**5.49e8**	**0.001651**	**1.421**	**0.004491**	**0.000000**	**0.000344**	**0.007181**
14.00	**0.233**	**0.521**	**5.91e8**	**0.001636**	**1.421**	**0.004200**	**0.000000**	**0.000344**	**0.006869**
14.50	**0.241**	**0.539**	**6.12e8**	**0.001628**	**1.421**	**0.004342**	**0.000000**	**0.000344**	**0.007000**
+ 15.00 +	**0.249**	**0.558**	**6.33e8**	**0.001621**	**1.421**	**0.004474**	**0.000000**	**0.000344**	**0.007122**
15.50	**0.258**	**0.576**	**6.54e8**	**0.001614**	**1.421**	**0.004449**	**0.000000**	**0.000344**	**0.007087**
16.00	**0.266**	**0.595**	**6.75e8**	**0.001608**	**1.421**	**0.004271**	**0.000000**	**0.000344**	**0.006900**
17.00	**0.283**	**0.632**	**7.18e8**	**0.001596**	**1.421**	**0.003892**	**0.000000**	**0.000344**	**0.006503**

	RESISTANCE								
SPEED [kt]	RBARE [kN]	RAPP [kN]	RWIND [kN]	RSEAS [kN]	RCHAN [kN]	RTOWED [kN]	RMARGIN [kN]	RTOTAL [kN]	
10.00	**255.82**	**24.65**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**280.47**	
11.00	**267.49**	**29.50**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**296.99**	
12.00	**292.03**	**34.75**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**326.78**	
13.00	**318.21**	**40.41**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**358.62**	
14.00	**352.98**	**46.48**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**399.46**	
14.50	**385.90**	**49.66**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**435.56**	
+ 15.00 +	**420.14**	**52.94**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**473.07**	
15.50	**446.46**	**56.31**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**502.77**	
16.00	**463.17**	**59.79**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**522.96**	
17.00	**492.79**	**67.04**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**559.83**	

	EFFECTIVE POWER		OTHER		
SPEED [kt]	PEBARE [kW]	PETOTAL [kW]	CTLR	CTLT	RBARE/W
10.00	**1316.0**	**1442.9**	**0.08863**	**0.12380**	**0.00342**
11.00	**1513.7**	**1680.6**	**0.07219**	**0.10698**	**0.00358**
12.00	**1802.8**	**2017.3**	**0.06369**	**0.09814**	**0.00391**
13.00	**2128.1**	**2398.4**	**0.05698**	**0.09112**	**0.00426**
14.00	**2542.3**	**2877.0**	**0.05330**	**0.08715**	**0.00472**
14.50	**2878.6**	**3249.0**	**0.05510**	**0.08882**	**0.00517**
+ 15.00 +	**3242.0**	**3650.5**	**0.05677**	**0.09036**	**0.00562**
15.50	**3560.0**	**4009.1**	**0.05645**	**0.08993**	**0.00598**
16.00	**3812.4**	**4304.5**	**0.05420**	**0.08755**	**0.00620**
17.00	**4309.7**	**4896.0**	**0.04938**	**0.08252**	**0.00660**

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (HSTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (HSTS).hcnc**

Hull data

General		
Configuration:		**Monohull**
Chine type:		**Round/multiple**
Length on WL:		**97.569 m**
Max beam on WL:	[LWL/BWL 4.514]	**21.616 m**
Max molded draft:	[BWL/T 4.580]	**4.720 m**
Displacement:	[CB 0.746]	**7618.28 t**
Wetted surface:	[CS 2.269]	**1931.231 m2**
ITTC-78 (CT)		
LCB fwd TR:	[XCB/LWL 0.516]	**50.320 m**
LCF fwd TR:	[XCF/LWL 0.484]	**47.220 m**
Max section area:	[CX 0.931]	**95.036 m2**
Waterplane area:	[CWP 0.918]	**1936.740 m2**
Bulb section area:		**0.000 m2**
Bulb ctr below WL:		**0.000 m**
Bulb nose fwd TR:		**0.000 m**
Imm transom area:	[ATR/AX 0.000]	**0.000 m2**
Transom beam WL:	[BTR/BWL 0.000]	**0.000 m**
Transom immersion:	[TTR/T 0.000]	**0.000 m**
Half entrance angle:		**23.15 deg**
Bow shape factor:	[WL flow]	**1.0**
Stern shape factor:	[WL flow]	**1.0**

Planing		
Proj chine length:		***0.000 m***
Proj bottom area:		***0.000 m2***
LCG fwd TR:	*[XCG/LP 0.000]*	***0.000 m***
VCG below WL:		***0.000 m***
Aft station (fwd TR):		***0.000 m***
Deadrise:		***0.00 deg***
Chine beam:		***0.000 m***
Chine ht below WL:		***0.000 m***
Fwd station (fwd TR):		***0.000 m***
Deadrise:		***0.00 deg***
Chine beam:		***0.000 m***
Chine ht below WL:		***0.000 m***
Propulsor type:		***Propeller***
Max prop diameter:		***3048.0 mm***
Shaft angle to WL:		***10.00 deg***
Position fwd TR:		***0.000 m***
Position below WL:		***0.000 m***
Transom lift device:		***Flap***
Device count:		***0***
Span:		***0.000 m***
Chord length:		***0.000 m***
Deflection angle:		***0.00 deg***
Tow point fwd TR:		***0.000 m***
Tow point below WL:		***0.000 m***
Foil assist (planing)		
Foil count:		***0***
Total planform area:		***0.000 m2***
LCE fwd TR:		***0.000 m***
VCE below WL:		***0.000 m***
Lift-drag ratio:		***0.0***
Lift fraction (design):		***0.00***
Design speed:		***0.00 kt***

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (HSTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (HSTS).hcnc**

Appendage data

General		**Skeg/Keel**	
Definition:	**Component**	Count:	**1**
Percent of hull drag:	**0.00 %**	Type:	**Skeg**
Planing influence		Mean length:	**0.000 m**
LCE fwd TR:	**0.000 m**	Mean width:	**0.000 m**
VCE below WL:	**0.000 m**	Height aft:	**0.000 m**
Shafting		Height mid:	**0.000 m**
Count:	**2**	Height fwd:	**0.000 m**
Max prop diameter:	**3048.0 mm**	Projected area:	**0.000 m2**
Shaft angle to WL:	**10.00 deg**	Wetted surface:	**87.330 m2**
Exposed shaft length:	**0.000 m**	**Stabilizer**	
Shaft diameter:	**0.000 m**	Count:	**0**
Wetted surface:	**14.860 m2**	Root chord:	**0.000 m**
Strut bossing length:	**0.000 m**	Tip chord:	**0.000 m**
Bossing diameter:	**0.000 m**	Span:	**0.000 m**
Wetted surface:	**6.690 m2**	T/C ratio:	**0.000**
Hull bossing length:	**0.000 m**	LE sweep:	**0.00 deg**
Bossing diameter:	**0.000 m**	Wetted surface:	**0.000 m2**
Wetted surface:	**9.480 m2**	Projected area:	**0.000 m2**
Strut (per shaft line)		Dynamic multiplier:	**1.00**
Count:	**2**	**Bilge keel**	
Root chord:	**0.000 m**	Count:	**2**
Tip chord:	**0.000 mm**	Mean length:	**0.000 m**
Span:	**0.000 m**	Mean base width:	**0.000 m**
T/C ratio:	**0.000**	Mean projection:	**0.000 m**
Projected area:	**0.000 m2**	Wetted surface:	**196.030 m2**
Wetted surface:	**3.723 m2**	**Tunnel thruster**	
Exposed palm depth:	**0.000 m**	Count:	**2**
Exposed palm width:	**0.000 m**	Diameter:	**0.000 m**
Rudder		**Sonar dome**	
Count:	**1**	Count:	**0**
Rudder location:	**Behind propeller**	Wetted surface:	**0.000 m2**
Type:	**Balanced foil**	**Miscellaneous**	
Root chord:	**0.000 m**	Count:	**0**
Tip chord:	**0.000 m**	Drag area:	**0.000 m2**
Span:	**0.000 m**	Drag coef:	**0.00**
T/C ratio:	**0.000**		
LE sweep:	**0.00 deg**		
Projected area:	**0.000 m2**		
Wetted surface:	**16.720 m2**		

Environment data

Wind		**Seas**	
Wind speed:	**0.00 kt**	Significant wave ht:	**0.000 m**
Angle off bow:	**0.00 deg**	Modal wave period:	**0.0 sec**
Gradient correction:	**Off**	**Shallow/channel**	
Exposed hull		Water depth:	**0.000 m**
Transverse area:	**0.000 m2**	Type:	**Shallow water**
VCE above WL:	**0.000 m**	Channel width:	**0.000 m**
Profile area:	**66.890 m2**	Channel side slope:	**0.00 deg**
Superstructure		Hull girth:	**0.000 m**
Superstructure shape:	**Ferry/Liner**		
Transverse area:	**0.000 m2**		
VCE above WL:	**0.000 m**		
Profile area:	**41.810 m2**		

Resistance

26 Apr 2021 12:33 PM
HydroComp NavCad 2020 [Premium]

Project ID **MV Yahtse Fung (HSTS)**
Description **Ro-Ro Car/Cargo Alaskan Ferry**
File name **NAME 4175 MV Yahtse NavCAD Fung (HSTS).hcnc**

Symbols and values

SPEED = Vessel speed
FN = Froude number [LWL]
FV = Froude number [VOL]

RN = Reynolds number [LWL]
CF = Frictional resistance coefficient
CV/CF = Viscous/frictional resistance coefficient ratio [dynamic form factor]
CR = Residuary resistance coefficient
dCF = Added frictional resistance coefficient for roughness
CA = Correlation allowance [dynamic]
CT = Total bare-hull resistance coefficient

RBARE = Bare-hull resistance
RAPP = Additional appendage resistance
RWIND = Additional wind resistance
RSEAS = Additional sea-state resistance
RCHAN = Additional shallow/channel resistance
RTOWED = Additional towed object resistance
RMARGIN = Resistance margin
RTOTAL = Total vessel resistance

PEBARE = Bare-hull effective power
PETOTAL = Total effective power

CTLR = Telfer residuary resistance coefficient
CTLT = Telfer total bare-hull resistance coefficient
RBARE/W = Bare-hull resistance to weight ratio

+ = Design speed indicator
* = Exceeds parameter limit

Appendix I: Polynomial Code – WBPolynomials.py

```
# NAME 3150 Honors Work
# Date Last Modified: 05/13/2020

"""
Honors Assignment:
Given the Wageningen B-Series polynomials for determining
the K_T and K_Q curves and example data, program in Python open
water chart graphs for any given propeller data.
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

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

## 2. K_T polynomial values and sum

def K_Tfunc(J,PD,ar,Z):

    T1=0.00880496*(J**0)*(PD**0)*(ar**0)*(Z**0)
    T2=-0.204554*(J**1)*(PD**0)*(ar**0)*(Z**0)
    T3=0.166351*(J**0)*(PD**1)*(ar**0)*(Z**0)
    T4=0.158114*(J**0)*(PD**2)*(ar**0)*(Z**0)
    T5=-0.147581*(J**2)*(PD**0)*(ar**1)*(Z**0)
    T6=-0.481497*(J**1)*(PD**1)*(ar**1)*(Z**0)
    T7=0.415437*(J**0)*(PD**2)*(ar**1)*(Z**0)
    T8=0.0144043*(J**0)*(PD**0)*(ar**0)*(Z**1)
    T9=-0.0530054*(J**2)*(PD**0)*(ar**0)*(Z**1)
    T10=0.0143481*(J**0)*(PD**1)*(ar**0)*(Z**1)
    T11=0.0606826*(J**1)*(PD**1)*(ar**0)*(Z**1)
    T12=-0.0125894*(J**0)*(PD**0)*(ar**1)*(Z**1)
    T13=0.0109689*(J**1)*(PD**0)*(ar**1)*(Z**1)
    T14=-0.133698*(J**0)*(PD**3)*(ar**0)*(Z**0)
    T15=0.00638407*(J**0)*(PD**6)*(ar**0)*(Z**0)
    T16=-0.00132718*(J**2)*(PD**6)*(ar**0)*(Z**0)
    T17=0.168496*(J**3)*(PD**0)*(ar**1)*(Z**0)
```

```
    T18=-0.0507214*(J**0)*(PD**0)*(ar**2)*(Z**0)
    T19=0.0854559*(J**2)*(PD**0)*(ar**2)*(Z**0)
    T20=-0.0504475*(J**3)*(PD**0)*(ar**2)*(Z**0)
    T21=0.010465*(J**1)*(PD**6)*(ar**2)*(Z**0)
    T22=-0.00648272*(J**2)*(PD**6)*(ar**2)*(Z**0)
    T23=-0.00841728*(J**0)*(PD**3)*(ar**0)*(Z**1)
    T24=0.0168424*(J**1)*(PD**3)*(ar**0)*(Z**1)
    T25=-0.00102296*(J**3)*(PD**3)*(ar**0)*(Z**1)
    T26=-0.0317791*(J**0)*(PD**3)*(ar**1)*(Z**1)
    T27=0.018604*(J**1)*(PD**0)*(ar**2)*(Z**1)
    T28=-0.00410798*(J**0)*(PD**2)*(ar**2)*(Z**1)
    T29=-0.000606848*(J**0)*(PD**0)*(ar**0)*(Z**2)
    T30=-0.0049819*(J**1)*(PD**0)*(ar**0)*(Z**2)
    T31=0.0025983*(J**2)*(PD**0)*(ar**0)*(Z**2)
    T32=-0.000560528*(J**3)*(PD**0)*(ar**0)*(Z**2)
    T33=-0.00163652*(J**1)*(PD**2)*(ar**0)*(Z**2)
    T34=-0.000328787*(J**1)*(PD**6)*(ar**0)*(Z**2)
    T35=0.000116502*(J**2)*(PD**6)*(ar**0)*(Z**2)
    T36=0.000690904*(J**0)*(PD**0)*(ar**1)*(Z**2)
    T37=0.00421749*(J**0)*(PD**3)*(ar**1)*(Z**2)
    T38=0.0000565229*(J**3)*(PD**6)*(ar**1)*(Z**2)
    T39=-0.00146564*(J**0)*(PD**3)*(ar**2)*(Z**2)


K_T=T1+T2+T3+T4+T5+T6+T7+T8+T9+T10+T11+T12+T13+T14+T15+T16+T17+T18+T19
\
    +T20+T21+T22+T23+T24+T25+T26+T27+T28+T29+T30+T31+T32+T33+T34+T35+T36 \
    +T37+T38+T39

    return K_T


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

## 3. K_Q polynomial values and sum

def K_Qfunc(J,PD,ar,Z):

    Q1=0.00379368*(J**0)*(PD**0)*(ar**0)*(Z**0)
```

Q2=0.00886523*(J**2)*(PD**0)*(ar**0)*(Z**0)
Q3=-0.032241*(J**1)*(PD**1)*(ar**0)*(Z**0)
Q4=0.00344778*(J**0)*(PD**2)*(ar**0)*(Z**0)
Q5=-0.0408811*(J**0)*(PD**1)*(ar**1)*(Z**0)
Q6=-0.108009*(J**1)*(PD**1)*(ar**1)*(Z**0)
Q7=-0.0885381*(J**2)*(PD**1)*(ar**1)*(Z**0)
Q8=0.188561*(J**0)*(PD**2)*(ar**1)*(Z**0)
Q9=-0.00370871*(J**1)*(PD**0)*(ar**0)*(Z**1)
Q10=0.00513696*(J**0)*(PD**1)*(ar**0)*(Z**1)
Q11=0.0209449*(J**1)*(PD**1)*(ar**0)*(Z**1)
Q12=0.00474319*(J**2)*(PD**1)*(ar**0)*(Z**1)
Q13=-0.00723408*(J**2)*(PD**0)*(ar**1)*(Z**1)
Q14=0.00438388*(J**1)*(PD**1)*(ar**1)*(Z**1)
Q15=-0.0269403*(J**0)*(PD**2)*(ar**1)*(Z**1)
Q16=0.0558082*(J**3)*(PD**0)*(ar**1)*(Z**0)
Q17=0.0161886*(J**0)*(PD**3)*(ar**1)*(Z**0)
Q18=0.00318086*(J**1)*(PD**3)*(ar**1)*(Z**0)
Q19=0.015896*(J**0)*(PD**0)*(ar**2)*(Z**0)
Q20=0.0471729*(J**1)*(PD**0)*(ar**2)*(Z**0)
Q21=0.0196283*(J**3)*(PD**0)*(ar**2)*(Z**0)
Q22=-0.0502782*(J**0)*(PD**1)*(ar**2)*(Z**0)
Q23=-0.030055*(J**3)*(PD**1)*(ar**2)*(Z**0)
Q24=0.0417122*(J**2)*(PD**2)*(ar**2)*(Z**0)
Q25=-0.0397722*(J**0)*(PD**3)*(ar**2)*(Z**0)
Q26=-0.00350024*(J**0)*(PD**6)*(ar**2)*(Z**0)
Q27=-0.0106854*(J**3)*(PD**0)*(ar**0)*(Z**1)
Q28=0.00110903*(J**3)*(PD**3)*(ar**0)*(Z**1)
Q29=-0.000313912*(J**0)*(PD**6)*(ar**0)*(Z**1)
Q30=0.0035985*(J**3)*(PD**0)*(ar**1)*(Z**1)
Q31=-0.00142121*(J**0)*(PD**6)*(ar**1)*(Z**1)
Q32=-0.00383637*(J**1)*(PD**0)*(ar**2)*(Z**1)
Q33=0.0126803*(J**0)*(PD**2)*(ar**2)*(Z**1)
Q34=-0.00318278*(J**2)*(PD**3)*(ar**2)*(Z**1)
Q35=0.00334268*(J**0)*(PD**6)*(ar**2)*(Z**1)
Q36=-0.00183491*(J**1)*(PD**1)*(ar**0)*(Z**2)
Q37=0.000112451*(J**3)*(PD**2)*(ar**0)*(Z**2)
Q38=-0.0000297228*(J**3)*(PD**6)*(ar**0)*(Z**2)
Q39=0.000269551*(J**1)*(PD**0)*(ar**1)*(Z**2)
Q40=0.00083265*(J**2)*(PD**0)*(ar**1)*(Z**2)
Q41=0.00155334*(J**0)*(PD**2)*(ar**1)*(Z**2)

```
    Q42=0.000302683*(J**0)*(PD**6)*(ar**1)*(Z**2)
    Q43=-0.0001843*(J**0)*(PD**0)*(ar**2)*(Z**2)
    Q44=-0.000425399*(J**0)*(PD**3)*(ar**2)*(Z**2)
    Q45=0.0000869243*(J**3)*(PD**3)*(ar**2)*(Z**2)
    Q46=-0.0004659*(J**0)*(PD**6)*(ar**2)*(Z**2)
    Q47=0.0000554194*(J**1)*(PD**6)*(ar**2)*(Z**2)



K_Q=Q1+Q2+Q3+Q4+Q5+Q6+Q7+Q8+Q9+Q10+Q11+Q12+Q13+Q14+Q15+Q16+Q17+Q18
+Q19 \

+Q20+Q21+Q22+Q23+Q24+Q25+Q26+Q27+Q28+Q29+Q30+Q31+Q32+Q33+Q34+Q35+Q36
\
    +Q37+Q38+Q39+Q40+Q41+Q42+Q43+Q44+Q45+Q46+Q47

    return K_Q


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

## 4. eta_O function

def eta_Ofunc(J,PD,ar,Z):

    KT=K_Tfunc(J,PD,ar,Z)
    KQ=K_Qfunc(J,PD,ar,Z)

    eta_O=J/(2.*np.pi)*KT/KQ

    return eta_O

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

## 3. Open Water Chart graphing

def openwaterchart(J,PD,ar,Z):
    #plt.figure(figsize=(15,10))
    plt.plot(J,K_Tfunc(J,PD,ar,Z),lw=2,label=r"$[K_T]$")
    plt.plot(J,10.*K_Qfunc(J,PD,ar,Z),lw=2,label=r"$[10K_Q]$")
```

```
    plt.plot(J,eta_Ofunc(J,PD,ar,Z),lw=2,label=r"$[eta_O]$")
    plt.title("Open Water Chart for Wageningen B-Series Propeller")
    plt.xlabel("Advance Ratio, $J$ $[-]$")
    plt.ylabel("Thrust and Torque Coefficients, $K_T$, $10K_Q$ $[-]$")
    plt.legend()
    #plt.grid()
    #plt.show()

# use find function to find self prop point for dc_4
def findJTS2(dc_4,PD,ar,Z):
    """
    Find the self-propulsion point as intersection of
    parabola dc_4*J**2 and KT curve
    """
    J_0=0.7 #initial guess for J

    # solve for intersection point
    # needs to be , after JTS so that fsolve only
    # returns desired value
    JTS,=fsolve(lambda J:dc_4*(J**2)-K_Tfunc(J,PD,ar,Z),J_0)

    return JTS

# finding minimum area ratio from Burrill criterion
def ar_min(n,PD,D,T_req,v_as,e,rho):

    g = 9.807 #m/s^2
    rho=1026.021 #kg/m^3



    p_A=101325. #Pa
    p_v=1671.  #Pa
    p_0=p_A+rho*g*e

    v_1=np.sqrt((v_as**2)+(0.7*np.pi*n*D)**2)
    sigma_b=(p_0-p_v)/(0.5*rho*v_1**2)
    tau_c=0.715*(sigma_b**0.184)-0.437

    r1=0.5*rho*(v_1**2)*tau_c*(1.067- 0.229*PD)*np.pi*(D**2)/4.
    arm=T_req/r1
```

```
    return arm

# test
if __name__ == '__main__':

    Z=4.0
    PD=0.70
    ar=0.5500 #expanded area ratio
    J=np.array([0.0,0.2,0.4,0.6,0.8,1.0])

    plt.figure(figsize=(15,10))
    openwaterchart(J,PD,ar,Z)
    plt.grid()
    plt.show()
```

Appendix J: Optimization Code – WBOpt.py

```
# NAME 3155 Project: Propeller Optimization Tool
# Date Last Modified: 04/28/2021

"""
Propeller Selection Program:
   Use the Wageningen B-Series polynomials and Holtrop and
   Mennen's Resistance and Propulsion estimate method to
   vary propeller design parameters and optimize a propeller
   for any given vessel particulars.
"""


from WBPolynomials import eta_Ofunc,findJTS2,ar_min

g=9.807
rho=1027.8336 #kg/m^3; density at 4ºC
nu=1.6262e-6 #m^2/s; viscosity at 4ºC

## Step 1: Define Design Constants
# completed in project code

## Step 2: Open-Water Diagram for Chosen Design Constant
# completed in project code

## Step 3: Extract Max Efficiency from Diagram

def optimumprop(x,dc_4,Z,D,T_req,v_as,rho,e,proptype):
   #retreive free variables
   PD=x[0]
   ar=x[1]

   #find JTS for this prop
   JTS=findJTS2(dc_4,PD,ar,Z)

   #rate of revolution at self-propulsion point
   nTS=v_as/(JTS*D)

   #compute open water efficiency
```

```
eta_O=eta_Ofunc(JTS,PD,ar,Z)

#compute constraints
p=0. #initial value

if ar < ar_min(nTS,PD,D,T_req,v_as,e,rho):
    p=p+(ar_min(nTS,PD,D,T_req,v_as,e,rho)-ar)**2

print('p = ',p)

if proptype=='CPP':
    armax=0.75
    if ar > armax:
        p=p+7.*(ar-armax)**2
        print('p = ',p)

if PD > 1.4:
    p=p+(PD-1.4)**2

obj=1.-eta_O+ 10.*p

print('p = ',p)
print('')

return obj
```