5-14-2010

Hull Shape Optimization for Wave Resistance Using Panel Method

Krishna M. Karri
University of New Orleans

Hull Shape Optimization for Wave Resistance Using Panel Method

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Engineering
Naval Architecture and Marine Engineering

by

Krishna Murthy Karri

B.E. Andhra University College of Engineering, 2008

May, 2010

Dedication

I would like to dedicate this thesis to my parents and brother for all of their love and support.

Acknowledgments

I would like to give my sincere thanks to my advisor, Dr. Lothar Birk, Associate Professor of Naval Architecture and Marine Engineering at the University of New Orleans, under whose able guidance this thesis work has been conducted. Without his suggestions and mentor-ship this project would not have taken this form.

I would also like to thank my committee members Dr. William Vorus, Professor of Naval Architecture and Marine Engineering and Dr. Brandon M. Taravella, Assistant Professor of Naval Architecture and Marine Engineering for reviewing my thesis.

I express my gratitude to the Board of Regents, State of Louisiana for funding the project.

Table of Contents

Abstract x

1 Introduction 1
- 1.1 Purpose, Motivation and Objective of Study . . . 2
- 1.2 Proposed Method . . . 3

2 Background 4
- 2.1 Geometrical Variation of Ship Hull Forms . . . 4
- 2.2 Effect of Variation in LCB . . . 6
- 2.3 Panel Method . . . 7
- 2.4 Optimization . . . 8
  - 2.4.1 Algorithm . . . 8

3 Methodology 11
- 3.1 Optimization Loop . . . 11
- 3.2 Objective Function . . . 12
- 3.3 Geometry Acquisition . . . 12
  - 3.3.1 Geometry Input . . . 14
  - 3.3.2 Panel Zones . . . 15
  - 3.3.3 Panel Generation . . . 16
- 3.4 Panel Code . . . 19
- 3.5 Optimization Module . . . 19
  - 3.5.1 Function . . . 19
  - 3.5.2 Free Variable . . . 21
  - 3.5.3 Shape Transformation . . . 21

4 Results 23
- 4.1 Input Data . . . 24
  - 4.1.1 Geometry . . . 24
  - 4.1.2 Panel Code . . . 25
  - 4.1.3 Optimization . . . 26

4.2 Output 28
4.2.1 Optimization Output 28

5 Discussion **30**

6 Conclusions **33**

Bibliography **35**

APPENDIX: Python code **37**

Vita **83**

List of Figures

2.1 Nelder Mead Simplex Flow Chart [3] 10

3.1 Objective function flow chart 13
3.2 Geometry file data points 15
3.3 Different types of conventional vessels 17
3.4 Panel points 18
3.5 Optimizer Flow Chart 20
3.6 Wave field output from panel code 21
3.7 Top view: Wave field output from panel code 22

4.1 SUSAN MAERSK - 6600 TEU Container Ship [1] 25
4.2 Input geometry file 26
4.3 Waves generated along the ship's length 26
4.4 Waves generated at transom 27
4.5 Optimization convergence at speed of 24 knots 29

5.1 Kelvin wave angle 31
5.2 Higher wave elevation at transom 31
5.3 Froude Number vs LCB 32

Nomenclature

α	fraction for raise of free surface panels
$\bar{x}$	centroid of half body as a fraction of halflength
$\bar{z}$	LCB as a percentage of length
$\delta\bar{z}$	required shift of LCB as a percentage of length
$\delta\phi_a$	required change in prismatic coefficient of afterbody
$\delta\phi_f$	required change in prismatic coefficient of forebody
$\delta\phi_t$	required change in total prismatic coefficient
δp_a	required change in length of aft parallel middle body
δp_f	required change in length of forward parallel middle body
δx_a	resultant longitudinal shift of aft section at x_a
δx_f	resultant longitudinal shift of forward section at x_f
ϕ	prismatic coefficient of half body
ϕ_t	total prismatic coefficient
A	Lackenby constants for half body
A_a	Lackenby constants for afterbody
A_f	Lackenby constants for forebody
B	Lackenby constants for half body
B_a	Lackenby constants for afterbody
B_f	Lackenby constants for forebody
C	Lackenby constants for half body
C_a	Lackenby constants for afterbody

C_f	Lackenby constants for forebody
k	lever of the second moment of the original SAC about midships
LCB	Longitudinal center of buoyancy from origin
p	fractional length of parallel middle body of halflength
p_a	fractional length of parallel middle body of afterbody
p_f	fractional length of parallel middle body of forebody
SAC	sectional area curve
x_a	co-ordinates from midhips to the aft end
x_f	co-ordinates from midhips to the forward end
Cm	midship sectional area coefficient
Cp	prismatic coefficient

Abstract

A ship must be designed for efficiency and economy, thus there is an everlasting desire for the design of better and better ships. One of the important factors which directly influence the worthiness of a design is its resistance. Throughout decades of ship design, the resistance is given top most importance as a design objective. With the increase in computational speeds of both software and hardware, there has been an opportunity for optimizing ship hulls using iterative methods of design and modification.

A method for calculating resistance for a given hull geometry and to optimize it using optimization algorithms are required for achieving better hulls. The resistance is calculated using a panel method for a given hull and the hull geometry is later changed by applying Lackenby's method of longitudinal shift of stations.

An optimization algorithm extracts the best possible design out of the numerous design alternatives possible.

Keywords: Resistance, Panel Method, Lackenby Transformation, Optimization, Longitudinal Center of Buoyancy

Chapter 1

Introduction

Development of new hull shapes is based on a number of coupled characteristic features. They determine both technological and economic worthiness of a design. The development of a new hull, in general, can be approached in three different ways:

- Hull form from standard series.
- Form parameter based hull design.
- Hull form generation based on parent ship data.

Hull form design based on standard series are derived from empirical data, such as Todd's Series 60 [2], Taylor Standard Series [16], BSRA [7], MARAD [13] and Ridgley-Nevitt [12], resulted from model tests carried out in model basins. The individual models have been derived from a parent hull by systematic variation of geometric parameters. In many of the standard series length to beam ratio (L/B) and beam to draft ratio (B/T) have been varied. Various methods have been proposed to select a hull form from the standard series data. One of them is to select preliminary dimensions based on parent ship analysis. Parent ship analysis is a deterministic approach where a new hull is selected by using regression analysis on the available parent hull data. However, each of these series are developed for a specific type of vessel, thus making it difficult to achieve modern hull forms using series data.

Form parameter based hull design starts with a very basic approach of using a small number of control parameters for attaining final design requirements. The basic hull properties, such as center line buttock or the shape of the deck profile for example, are defined by curves created from form parameters. Once this set of basic hull curves is created from the control parameters, a numerical algorithm is applied to create a set of sections at selected locations. This is achieved by utilizing a parametric curve generation process where the vertices of all B-spline curves are computed from a geometric algorithm, which solves for a required fairness criteria and a global shape constraints [17]. As the design is completely based on the input form parameters, it may sometimes lead a designer into a techno-economically bad design. Better hull forms can be generated by incorporating a design optimization method within this design sequence.

The third alternative for designing hull forms is based on modifying an existing hull form. This method gives a partial confidence to the designer right from the beginning as he starts from a design which already performed well in real time. The selection of parent hull form is based on a preliminary analysis of available vessels which are close to the desired hull form with good design objectives. Even though the design is based on better parent hull forms, they have to be optimized for the changes in operational conditions. For a ship hull to be qualified as a good design, it has to meet the requirements such as minimal propulsion power, maximum internal volume, maximum stability, ease of construction etc. [8].

1.1 Purpose, Motivation and Objective of Study

The rising demand for better transportation and latest development in computing power has provided an opportunity to give automated design and optimization a thought. Efficient transportation by sea plays a vital role in intercontinental trade. So for future transportation we need a good design and optimization methods for achieving reliable hull forms. The preliminary design of hull form has already taken a strong form, with a very little scope to

get better, whereas the optimization is the one which is yet to be developed for reliability and validity. Thus a major consideration is given to the hull form optimization in present work.

In the present work, a design optimization method is developed by taking a standard hull of a KRISO container ship, provided by Korea Research Institute for Ships and Ocean Engineering [14], and is used as a baseline model for verifying the results.

1.2 Proposed Method

The method comprises of geometry acquisition, panel generation, panel method for resistance computation and the optimization. The process of hull optimization using a panel method definitely requires a programming language with a higher computational power and a good structure. To achieve the best out of available resources , FORTRAN is used for numerical operations of panel method and PYTHON is used for reading geometry, panel generation and integrating the process of optimization.

To make the process more generic, the hull geometry is considered as a standard GHS geometry file format. This file typically contains markers which specify the sections of a hull. The marker data is converted to a panel file using a shape function for panel distribution. The aspect ratio of hull panels is also a significant factor contributing to the accuracy of the panel method. The frictional resistance component remains almost the same and the major component to be minimized, is the wave making resistance. The wave making resistance is calculated by integrating the pressure distribution over wetted surface of the hull. It is used as objective function in the optimization. The simple yet efficient Nelder-Mead simplex algorithm is used for optimizing the hull for a better performance. The hull shape is varied using the Lackenby [5] transformation respective to a change in LCB position.

At the end of the optimization process, the hull is iterated to reach an optimum LCB position for which it has the least wave making resistance.

Chapter 2

Background

2.1 Geometrical Variation of Ship Hull Forms

H. Lackenby [5] proposed a method of designing a new vessel by systematic geometric variation of a parent ship form. His alternative to the "one minus prismatic" method varies the hull form with more control on design parameters. The "one minus prismatic" has a few limitations for achieving control over the independent variation of prismatic coefficient and parallel middle body, as well as restrictions on variation of fullness, entrance, run and maximum longitudinal shift of sections [5]. Lackenby's method allows to modify fullness, longitudinal position of buoyancy, and the extent of parallel middle in both fore and after bodies independently i.e. the variation can be carried in one or all of the above parameters at a time. The method is as follows,

Lackenby Transformation

Many design evaluation methods, such as resistance calculation techniques, use the major shape parameters of the hull to calculate their results. These parameters typically include length, beam, draft, prismatic coefficient (Cp), longitudinal center of buoyancy (LCB), and amidships coefficient (Cm). It is difficult to perform parametric or sensitivity studies using these techniques because all of the variables are interrelated and may change at the same

time. For example, when a vessel's length is varied, it's Cp, LCB and displacement values change at the same time. It would be desirable to be able to change any one of these major design variables, while maintaining constant values for the rest. Of course, this is impossible, because as one variable changes, at least one other variable must change to compensate. In the present approach Lackenby method for my hull variation is used. This method used the sectional area curve and allows for the modification of the following parameters The following form parameters are varied. They are

- Prismatic coefficient (Cp)
- Longitudinal center of buoyancy (LCB)
- Parallel middle body - forward (pf)
- Parallel middle body - aft (pa)

$$\delta\phi_f \;=\; \frac{2\left[\delta\phi_t\left(B_a+\bar{z}\right)\;+\;\delta\bar{z}\left(\phi_t+\delta\phi_t\right)\right]\;+\;C_f\cdot\delta p_f\;-\;C_a\cdot\delta p_a}{B_f+B_a} \tag{2.1}$$

$$\delta\phi_a \;=\; \frac{2\left[\delta\phi_t\left(B_f-\bar{z}\right)\;-\;\delta\bar{z}\left(\phi_t+\delta\phi_t\right)\right]\;-\;C_f\cdot\delta p_f\;+\;C_a\cdot\delta p_a}{B_f+B_a} \tag{2.2}$$

For a required change in hull form, the sectional area distribution is changed. The necessary change in station spacing is given by δx_f for stations forward of midships and by δx_a for stations aft of midships.

$$\delta x_f \;=\; (1-x_f)\left\{\frac{\delta p_f}{1-p_f}\;+\;\frac{(x_f-p_f)}{A_f}\left[\delta\phi_f-\delta p_f\frac{(1-\phi_f)}{(1-p_f)}\right]\right\} \tag{2.3}$$

$$\delta x_a \;=\; (1-x_a)\left\{\frac{\delta p_a}{1-p_a}\;+\;\frac{(x_a-p_a)}{A_a}\left[\delta\phi_a-\delta p_a\frac{(1-\phi_a)}{(1-p_a)}\right]\right\} \tag{2.4}$$

Where, the constants A, B and C are given by,

$$A = \phi\,(1 - 2\bar{x}) - p\,(1 - \phi) \tag{2.5}$$

$$B = \frac{\phi\,[2\bar{x} - 3k^2 - p\,(1 - 2\bar{x})]}{A} \tag{2.6}$$

$$C = \frac{B\,(1 - \phi) - \phi\,(1 - 2\bar{x})}{1 - p} \tag{2.7}$$

The practical limits for variation in prismatic coefficient, $\delta\phi_f$ and $\delta\phi_a$, to which the fullness of a given from can be varied without resulting in a very steep sectional area curve at the forward or aft ends is given by

$$\delta\phi = \frac{\delta p\,(1 - \phi) \pm \frac{1}{2}A\left(1 - \frac{\delta p}{1-p}\right)}{1 - p} \tag{2.8}$$

The various constants involved in the above equations refer to either the fore or the aft bodies as required. It is assumed that the reader has a basic knowledge of the coefficients of form of a ship hull. Otherwise, a detailed explanation can be found in the Principles of Naval Architecture, Vol 1 [6].

2.2 Effect of Variation in LCB

The effect of change in LCB position is very nominal upto a critical speed-length ratio, but above this critical limit the variation of LCB is very prominent [11].

Effect on Resistance

The effect of increase in speed is very prominent on the resistance of a vessel. The optimum position of LCB appears to move aft with increasing speed. Of course this depends on the hull shape.

Other Effects

Changes in position of LCB seem to have less influence on the factors such as wake fraction, thrust-deduction fraction, hull efficiency and quasi-propulsive coefficient. But as this evaluation is done on a specific series of hull model, the effect can be validated as an optimization constraint.

2.3 Panel Method

Boundary Element Method

In the last twenty years, the boundary-element method (BEM) has been established as a powerful numerical technique for dealing with a variety of problems in science and engineering involving elliptic partial differential equations. The strength of the method derives from its ability to solve, with remarkable efficiency and accuracy, problems in domains with complex and possibly evolving geometry where traditional methods can be inefficient, cumbersome, or unreliable.

Various elements (elementary solutions) exist to approximate the disturbance effect of the ship. These more or less complicated mathematical expressions are useful to model displacements(sources) or lift(vortices or dipoles). The basic idea of all the related boundary element methods is to superimpose elements in an unbounded fluid. Since the flow does not cross a streamline just as it does not cross real fluid boundary (hull), any unbounded flow field in which a stream line coincides with the actual flow boundaries in the bounded problem can be interpreted as a solution for bounded flow problem in the limited fluid domain.

Panel Methods for computing the ship wave resistance in potential flow is one form of a boundary element method where the ship hull is discretized into number of quadrilateral and triangular panels with rankine sources distributed at each panel centre.

This method assumes inviscid and non-lifting flows. Body boundary condition and a

linearised free surface boundary condition are applied in the current version of the panel method used. After solving for the source strengths at the panel centres. The pressure distribution is then integrated to get the net forces acting on the body, which is the sum of longitudinal force component, sinkage force and trimming moment of the ship moving in fluid domain. In solving the potential flow, the free surface panels are raised above the actual free surface to avoid singularities.

2.4 Optimization

Nelder Mead simplex, a simple yet efficient optimization algorithm is selected for this problem. The algorithm is very effective for the present problem. For the present case, it is a known fact that the optimum LCB is located near the midships. So, the initial guess for the optimization process is made at the midship.

2.4.1 Algorithm

The Nelder-Mead algorithm or simplex search algorithm, originally published in 1965 [9], is one of the best known algorithms for multidimensional unconstrained optimization. The basic algorithm is quite simple to understand and very easy to use. The method does not require any derivative information, which makes it suitable for problems with non-smooth functions. It is widely used to solve parameter estimation and similar statistical problems, where the function values are uncertain or subject to noise. [15].

Even though the method is quite simple, it is implemented in many different ways. Apart from some minor computational details in the basic algorithm, the main difference between various implementations lies in the construction of the initial simplex, and in the selection of convergence or termination tests used to end the iteration process. The general algorithm is given below (figure 2.1).

1. Construct the initial working simplex.

2. Repeat the following steps until the termination test is satisfied.

3. Calculate the termination test information.

4. If the termination test is not satisfied then transform the working simplex.

5. Return the best vertex of the current simplex and the associated function value.

In many practical problems a highly accurate solution is not necessary and may be impossible to compute. All that is desired is an improvement in function value, rather than full optimization. Though a true optimum is always better, there are limitations in lieu of the computation time required.

The Nelder-Mead method frequently gives significant improvements in the first few iteration and generally produces satisfactory results. Also, the method typically requires only one or two function evaluations per iteration, except in shrink transformations. This is very important in applications where each function evaluation is very expensive or time-consuming. For such problems, the method is often faster than other methods, especially those that require at least some function evaluations per iteration.

- **Pro:** In many numerical tests, the Nelder-Mead method succeeds in obtaining a good reduction in the function value using a relatively small number of function evaluations. Apart from being simple to understand and use, this is the main reason for its popularity in practice.

- **Con:** The method can take an enormous amount of iterations with negligible improvement in function value, despite being nowhere near to a minimum. This usually results in premature termination of iterations. A heuristic approach to deal with such cases is to restart the algorithm several times, with reasonably small number of allowed iterations per each run.

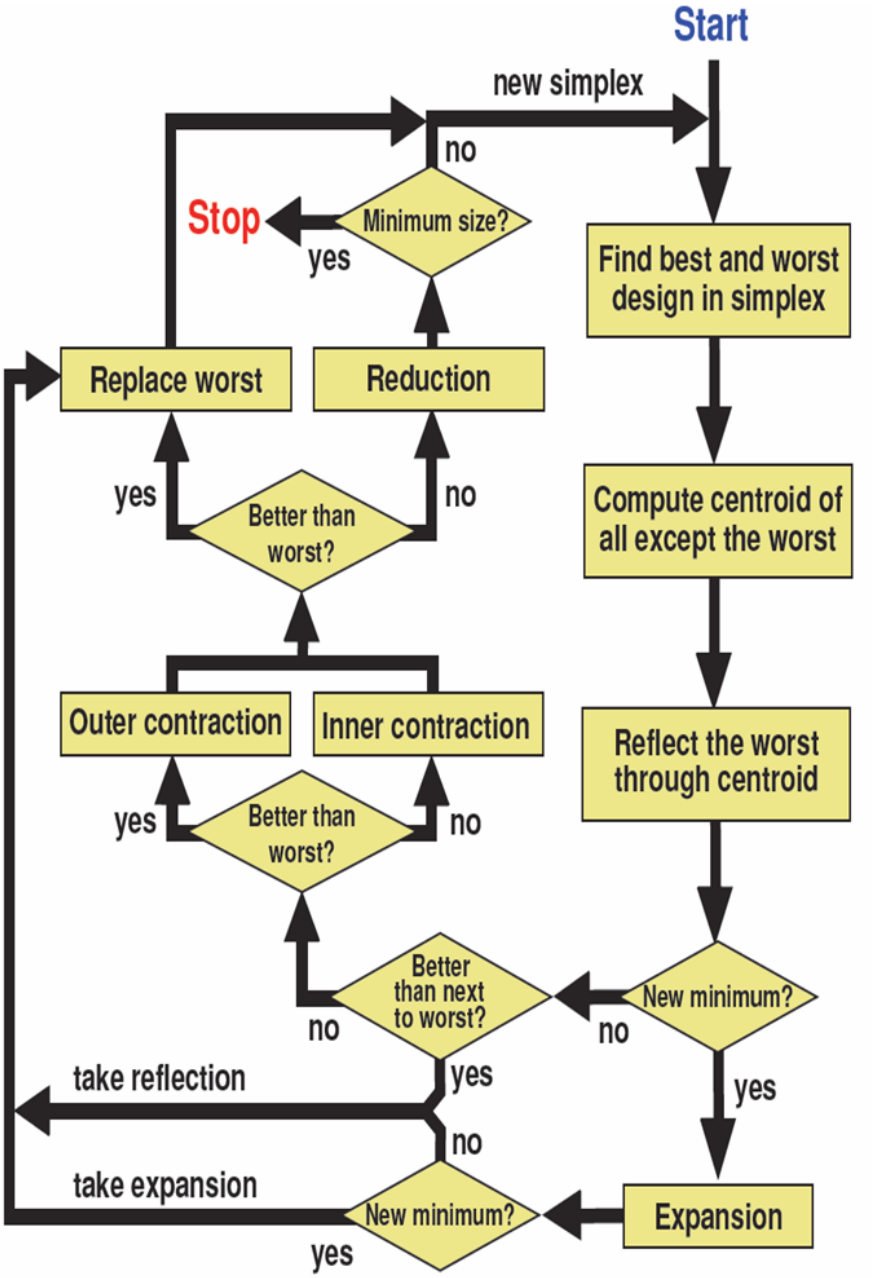

Figure 2.1: Nelder Mead Simplex Flow Chart [3]

Chapter 3

Methodology

Hull shape optimization for a minimum resistance can be achieved using any of the geometric variation processes as discussed in chapter 2. However, our goal is to optimize ship hull with a minimum deviation from the shape of the parent hull. For this reason, the variation of LCB using Lackenby's method of transformation [5] is considered appropriate.

3.1 Optimization Loop

The first step in the optimization loop is to read the hull geometry, where the hull offsets are defined and boundary surfaces are discretized. Surface discretization has a direct influence on the accuracy of any computational fluid dynamics solution. The ship hull geometry is represented as a point distribution along the girth at every station on the hull surface. For the purpose of discretization of hull surface into panels, the station spacing is changed and controlled using a shape function. The panels are divided at every station and are equally spaced girthwise. This kind of panel distribution will allow the optimization to work with Lackenby transformation without the necessity for panel generation during each and every iteration. Based on the type of the hull, subdivision of panels into zones is applied.

The second step is to find the resistance of the ship hull. This will serve as objective function for the optimization process. A panel code developed by Dr. Lothar Birk [4] is used

for calculation of wavemaking resistance of the ship hull. The panel code is a FORTRAN 90 program which accepts a panel input file along with other control parameters.

Finally, the optimization process is carried out using a PYTHON module which uses the Nelder - Mead simplex algorithm (section 2.4) for control. The algorithm find the optimized location of LCB for a particular speed.

3.2 Objective Function

The process of optimization iterates through the objective function (figure 3.1). The functional value of wave making resistance is calculated by

- Reading hull geometry.
- Creating hull patches.
- Hull and free surface panel generation.
- Generating panel input and control files.
- Computing wave making resistance.

3.3 Geometry Acquisition

The hull surface has to be represented with a panel distribution of varying panel sizes. For any computational fluid dynamics code to perform an accurate analysis, the panels have to be distributed uniformly and uniquely concentrated at locations where the flow is more complex. For a typical ship hull the flow near the bow and stern have large changes in flow velocities. Thus a concentrated panel distribution is applied to the ends of the hull.

The hull is typically divided into four primary panel zones. The zones are divided according to the boundary variables specified in the input. All the four zones has a typical

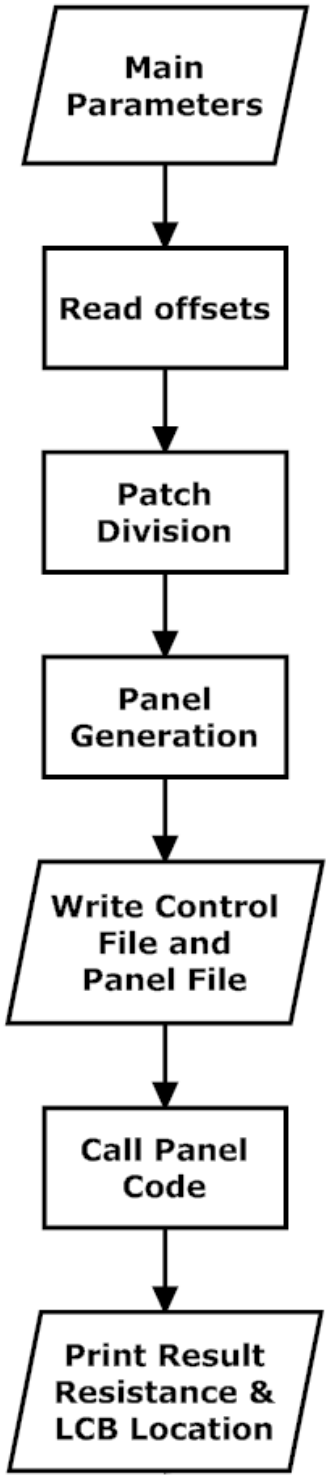

Figure 3.1: Objective function flow chart

panel generation scheme, which can be generated using a linear interpolation or a Lagrangian interpolation.

Panel zones maintain a consistent distribution by using a predefined global panel distribution shape function. The zone division process can be applied to any typical hull geometry, which are symmetric. Ships with a bulbous bow, transom stern, cruiser stern or centreline propeller bossing can be handled effectively by the panel zone division.

3.3.1 Geometry Input

The hull surface is defined by a standard GHS geometry file which contains all the information about the geometry of the vessel, represented in detail appropriate for hydrostatic and stability calculations (figure 3.2). It describes the volumes of the hull and various compartments. Each volume to be modelled consists of a series of transverse sections, arranged from bow to stern. A geometry file can be generated either by manually typing offsets or by using any other modelling software, Rhino3D is being used for this work. The geometry file should have a centerline symmetric marker distribution without repeated values. Centerline symmetric values can be defined on either port or starboard side.

Within the geometry file one or more components are defined together to form the complete watertight hull surface. This surface is specified in the input to make the process of hull generation more easier. Markers (offset points) in a geometry file are distributed along every station and they are stored into a standard offset array.

The number of markers may or may not be the same for every station. So, a girth wise subdivision is implemented to make them uniform. These offsets are stored in the offset variable, of dimensions " no. of points for a station $\times$ no. of stations $\times$ 2 (for y and z)". The y offsets are represented in Array [:, :, 0] and the corresponding z offsets in Array [:, :, 1].

The offsets are passed through a sanity check to verify that they are increasing in height. A warning shows up when such a problem is encountered. This helps is verifying the geometry

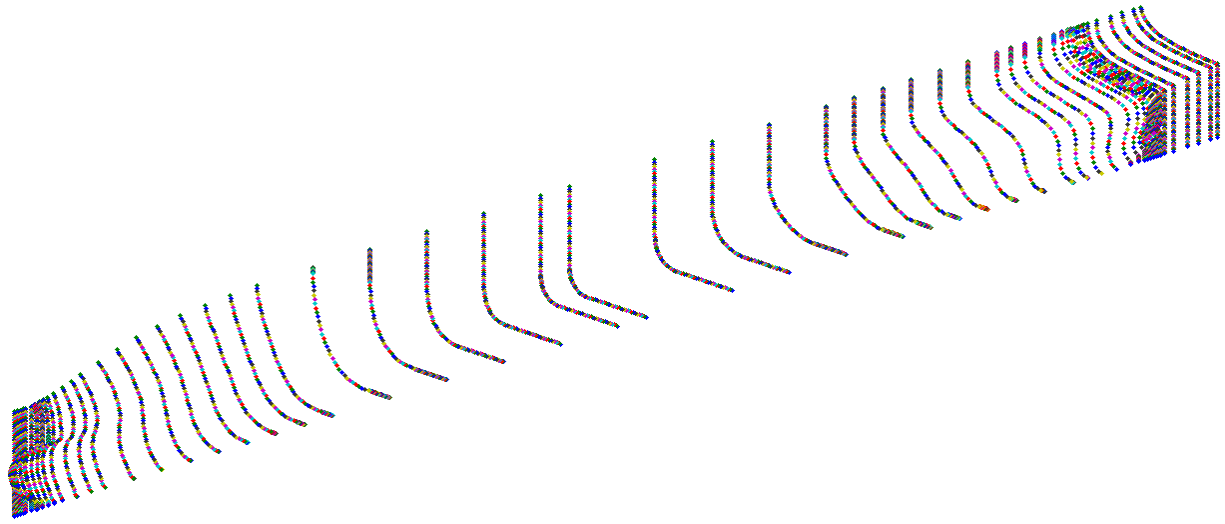

Figure 3.2: Geometry file data points

file for consistency.

3.3.2 Panel Zones

Every hull is divided into a four basic panel zones. This facilitates better way to generate panels and representation. Panel zones are defined on the basis of boundary lines and hull shape. Four variations in the zone distribution have been implemented for different types of conventional vessels (figure 3.3):

- Type 1:

  Center line propeller bosing - present.

  Bulbous bow - present.

- Type 2:

  Center line propeller bosing - absent.

  Bulbous bow - present.

- Type 3:

Center line propeller bosing - present.

Bulbous bow - absent.

- Type 4:

  Center line propeller bosing - absent.

  Bulbous bow - absent.

For each of the above cases the first zone will be the lower stern surface which includes the bossing if present. The second zone will be the upper stern surface. Third zone will be largest surface zone, the middle body surface. Last zone is defined for the bulbous bow surface.

3.3.3 Panel Generation

Though panels can be generated from the predefined stations, a controlled variation of panel distribution is essential for computational accuracy. A shape function is used to calculate and control the new distribution of panel stations. Panel stations are generated by a simple local interpolation of points in the three dimensional domain. Two approaches, linear interpolation and Lagrangian interpolation, are considered for interpolating along the local domain.

Linear Interpolation

For every new station, two nearest stations are selected and a new y and z ordinates are interpolated for the longitudinal (x ordinate) position of the station. As the stations are equally divided along the girth, the number of interpolated points will be the same in each station. Thus a vector interpolation is used to generate the same number of points at a new station location. The interpolation is operated independently over each girth point of same index along two selected stations. The two stations are adjacent to each other and are nearest to the required new station location.

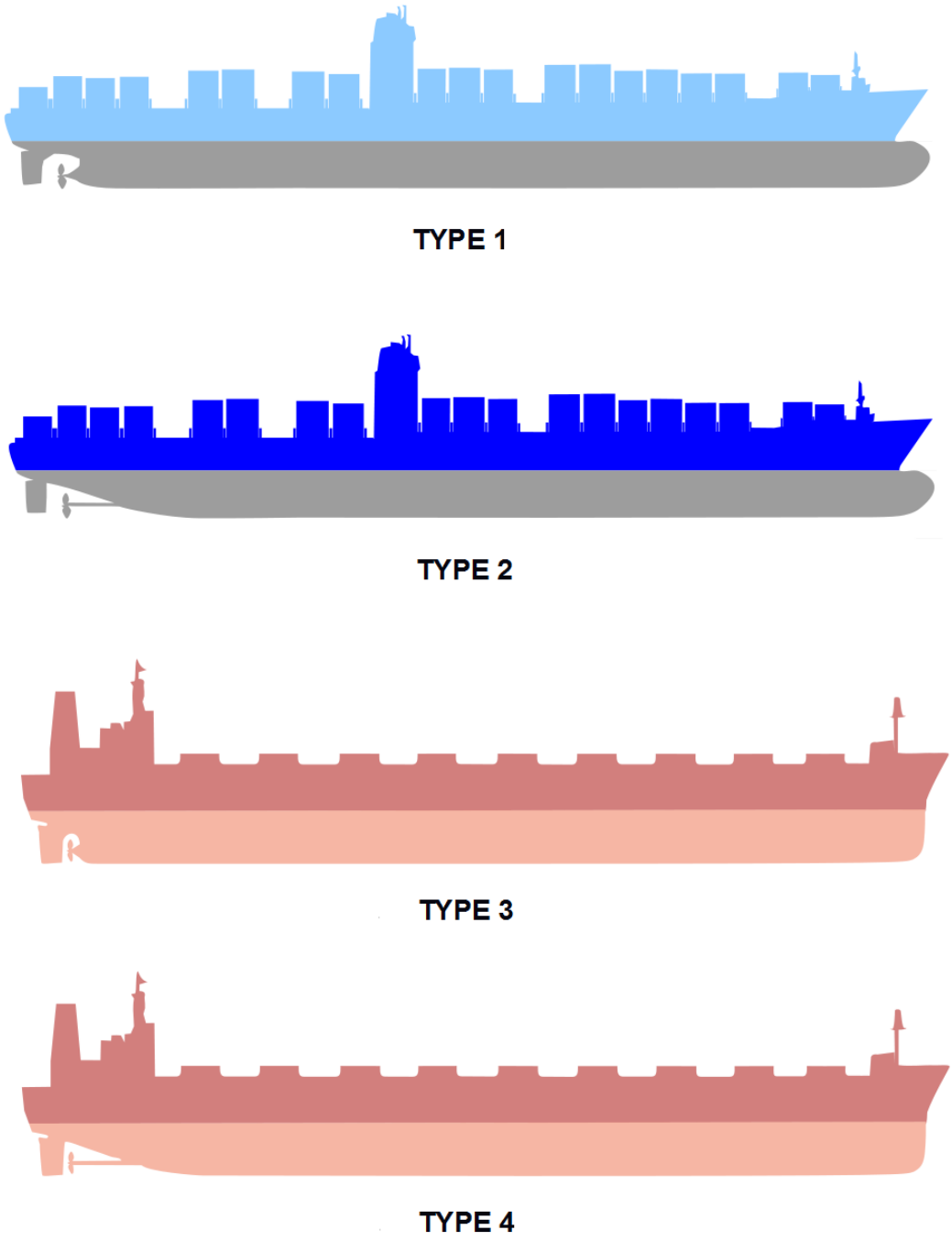

Figure 3.3: Different types of conventional vessels

Lagrangian Interpolation

This is a simple quadratic interpolation where three nearest stations are selected and an approach similar to linear interpolation is used to find the Lagrangian interpolation polynomials. For every new station location and for every girth interval, a new point is determined from the Lagrangian interpolation polynomial.

The old panel file (figure 3.2) is interpolated using any of the interpolation methods. The new panel points are more structured and cleaned up as shown in fig 3.4. The number of points in the new stations depend on the required aspect ratio and the input file specifications.

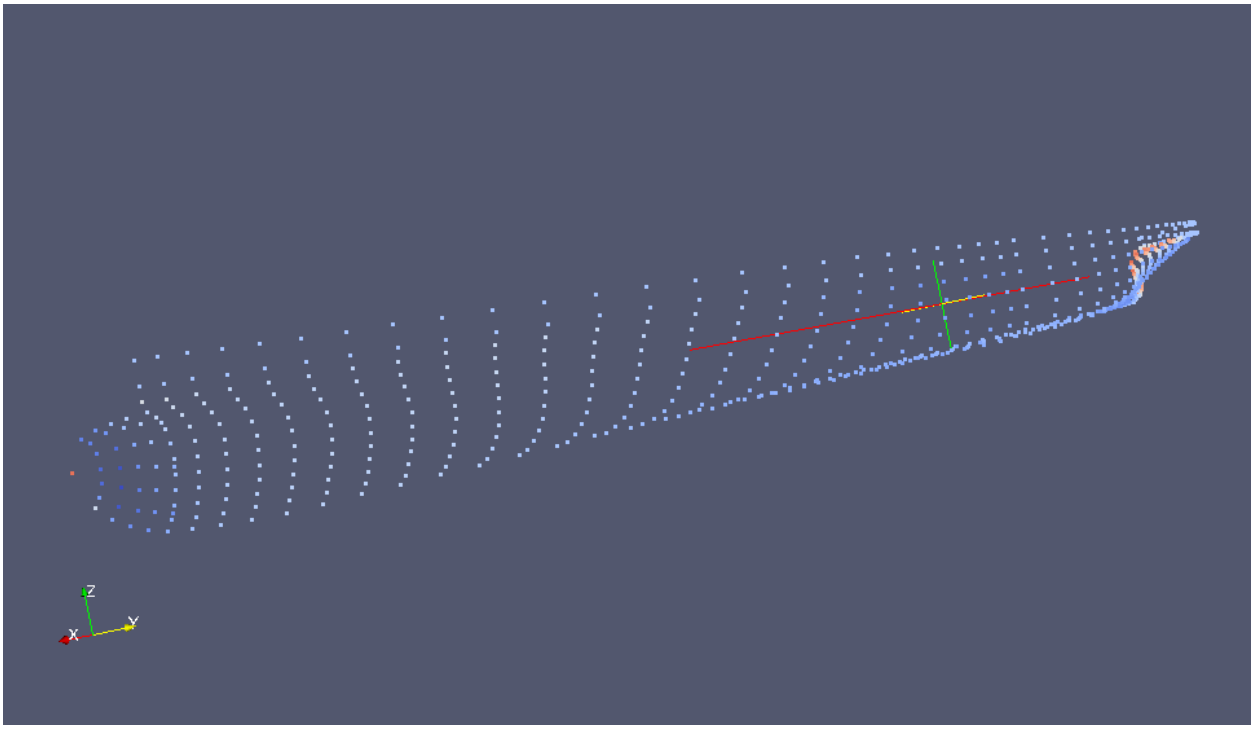

Figure 3.4: Panel points

3.4 Panel Code

The Panel method is used for finding [4] a solution to the potential flow around a ship hull (chapter 2.3). The panel code developed by Dr. Lothar Birk is designed to run using a panel file and a control file. The input panel file (*.pan) has a basic structure which first defines length, symmetry condition, number of hull patches, number of free surface patches, total number of hull points & panels and total number of free surface points & panels. The panel zone location and orientation is also specified. All the points are specified and the detailed orientation of every panel is listed.

The panel file is the most significant input that is defined after considering geometric approximations and appropriate free board panel raise. The rise in free board is specified by an α factor, which serves as a intermediate input for panel file generation (figure 3.5).

3.5 Optimization Module

A typical optimization requires an optimization method and a function, which has to be minimized. In this case the function is evaluated on the basis of resistance from panel code. The function is designed to take LCB position as a free variable. The optimizer will thus find the optimum LCB location at which the resistance is a minimum (figure 3.5).

3.5.1 Function

The function value is the coefficient of wave making resistance at a particular speed. This is evaluated using panel code [4] which writes the output to an output file. This value is used by the Nelder Mead simplex algorithm for its evaluations. The value of wave resistance will become higher as we move away from the optimum point and there is only one optimum location of LCB. The wave field created by the ship is sensitive to the panelization. The hull creates almost realistic wave field (figure 3.6), with a higher elevation at the transom.

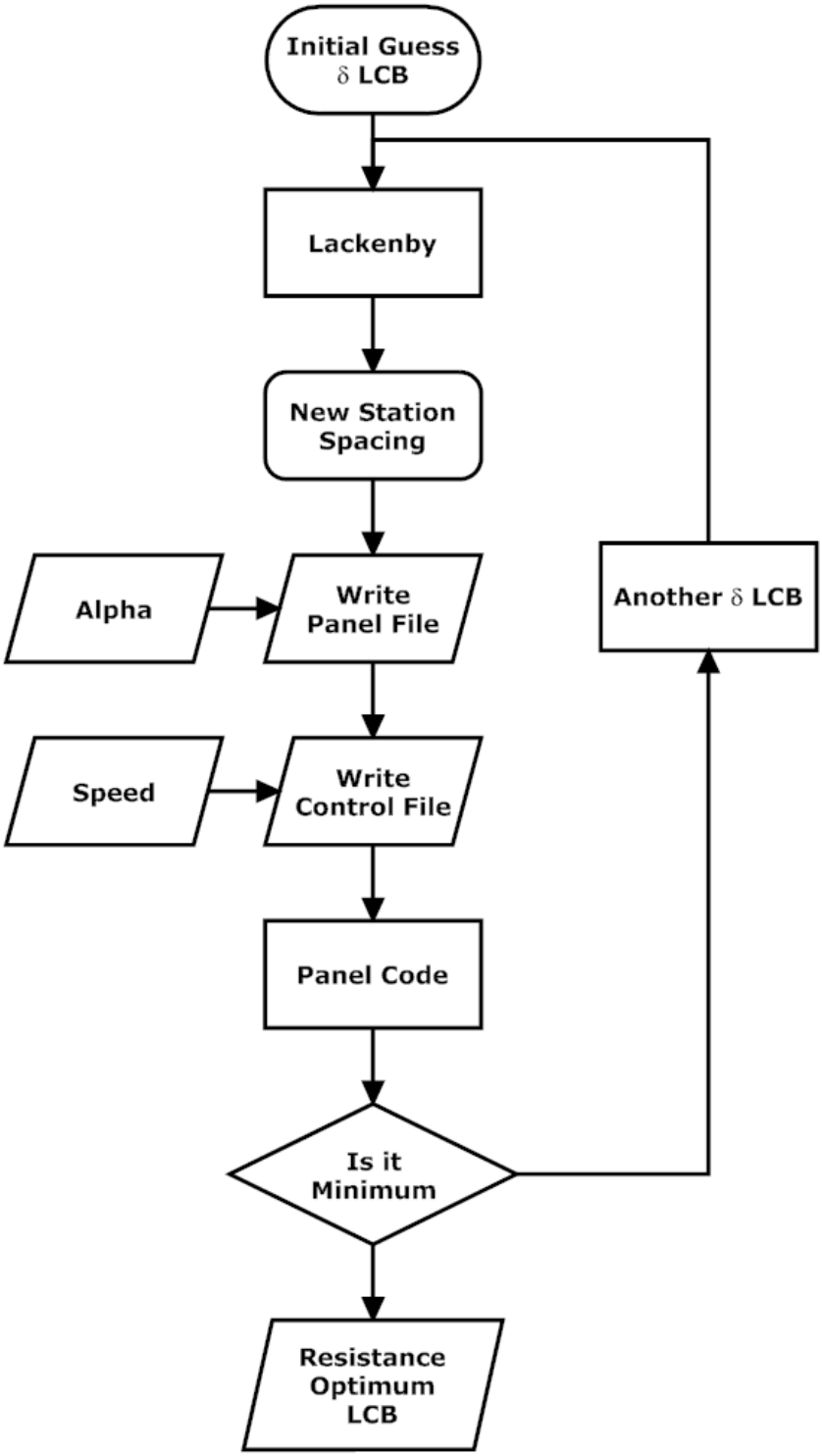

Figure 3.5: Optimizer Flow Chart

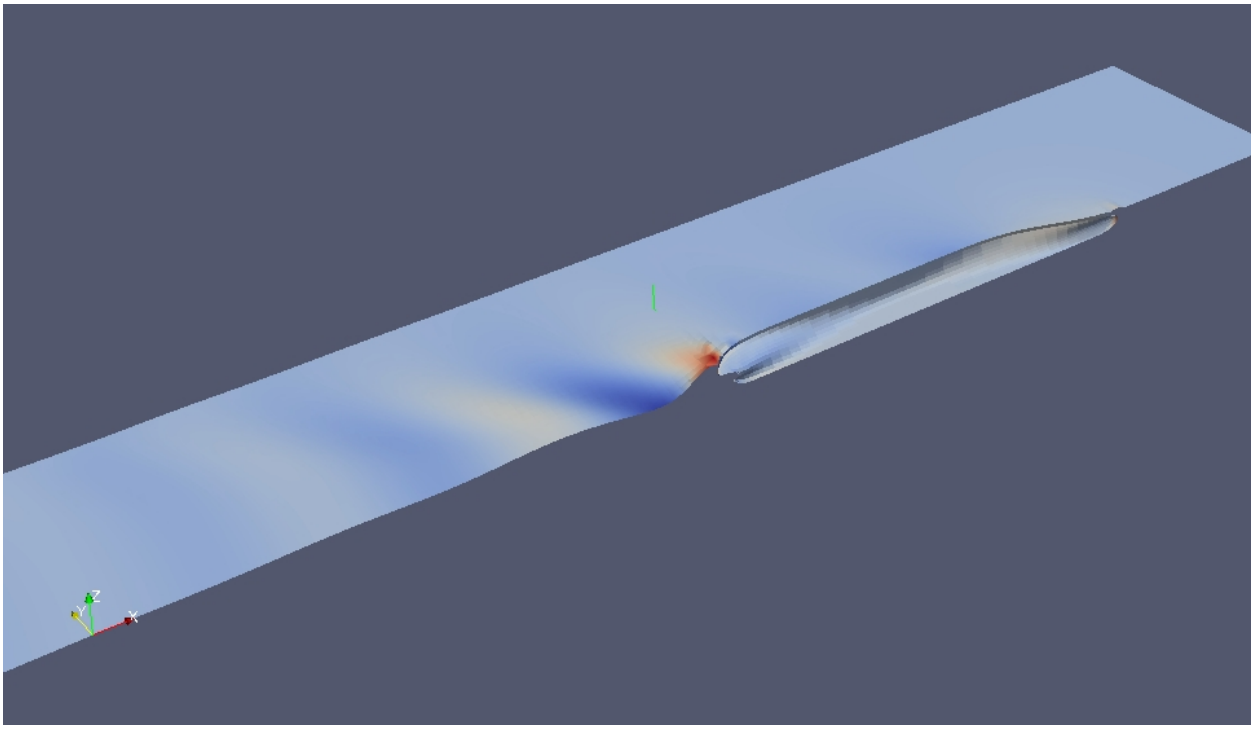

Figure 3.6: Wave field output from panel code

3.5.2 Free Variable

Longitudinal Center of Buoyancy is one of the critical parameters which influence the wave resistance of a ship moving in fluid domain. A general observation is that, at higher speeds the LCB shifts towards aft and the contrary for slow speeds. This a very sensitive variable which of course depends on the geometry and type of ship. So every ship has to be checked for an optimum LCB for the design operation conditions. The functional change in wave resistance for change in LCB is facilitated by changing hull form for required change in LCB by using Lackenby transformation [5].

3.5.3 Shape Transformation

In the present work, only LCB is considered as a free variable. When the LCB is shifted to one side, the sectional area curve is also shifted to the same side. This will impose a

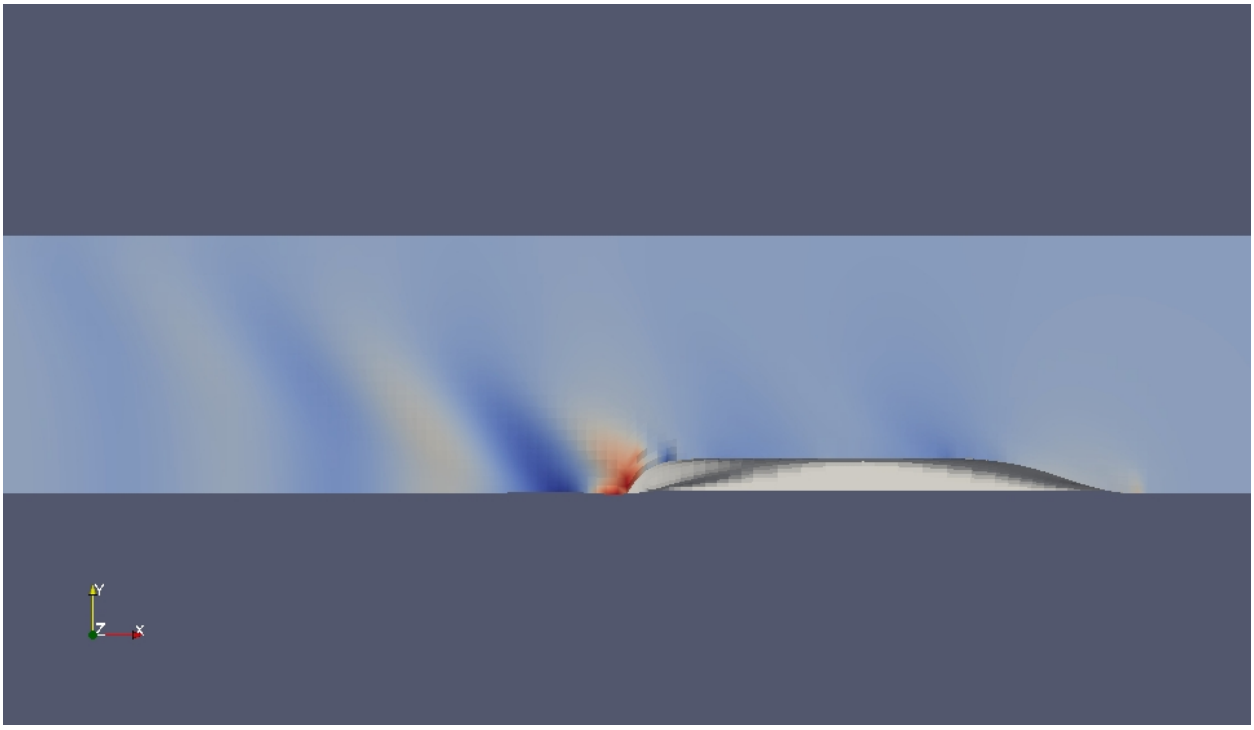

Figure 3.7: Top view: Wave field output from panel code

change in new station position, which is governed by Lackenby transformation method, thus changing the panel file. The panel file is deigned to take station data and station spacing as input. As the station location changes, the only change made is the station location input given to the panel file generation module. The rest of the process stays the same (figure 3.5).

Chapter 4

Results

The optimization of LCB position, for minimum wave resistance, using a panel method (section 3.3) was developed as a generic process. For the purpose of validation and discussion a standard hull form [14] is considered for analysis. The results thus obtained are compared with published technical data.

The optimization process has been carried out for a container ship hull of 53330 tons displacement, advancing in calm water at a speed of 24 knots (Fn = 0.26) in fixed condition (i.e., sinkage and trim not allowed). The optimization is applied only to the submerged hull. Resistance due to wind and appendages are not taken into consideration. More precisely, in the presented results the modification refers to just the submerged hull. Furthermore, in the hull transformations, only the x-coordinate of the hull is allowed to vary, while the transverse sections are kept fixed.

4.1 Input Data

As discussed earlier in 3.3, the process can be categorised into three sections. Each of them requires a specific input and they will generate output which will be passed on to the next. The geometry acquisition reads the hull geometry file and it generates a panel file. The panel code uses a panel file and a control file to generate the resistance. The resistance of the ship hull respective to a particular LCB position serves as an objective function for the optimization process.

4.1.1 Geometry

KRISO container ship (KCS) [14] is considered for analysis in this project. The KCS was conceived to provide data for validation of flow physics and CFD for a modern container ship with bulb bow and stern. Even though there is no existing ship exactly of same dimensions, but a similar ship is operated for Maersk Lines, the SUSAN MAERSK (figure 4.1) which is one of the largest container ship's in the world.

Korea Research Institute for Ships and Ocean Engineering performed towing-tank experiments to obtain resistance and wave field data for this ship. The data is available in the KCS website [14].

Main Particulars

- Length between Perpendiculars, Lpp = 230.0 m
- Breadth, B = 32.2 m
- Depth, D = 19 m
- Draft, T = 10.8 m
- Volume of displacement, V = 52030.0 m^3

Figure 4.1: SUSAN MAERSK - 6600 TEU Container Ship [1]

- Wetted surface area, $\Omega = 9424.0\ m^2$
- Block Coefficient, $C_b = 0.6505$
- Mid ship section area coefficient, $C_m = 0.9849$
- Longitudinal center of buoyancy, LCB = -1.48 % of Lpp from midships

Offsets Data: The offset file of KRISO container ship is provided as an input, either a GHS geometry file format or as a xyz offset data. The data provided has a distribution of one hundred points at ever station (figure 4.2). These stations are interpolated using linear interpolation along the length of the vessel. A numerical integration method is used for the calculation of form parameters of the vessel.

4.1.2 Panel Code

The panel code reads the geometry from a panel file (*.pan) and control parameters from a control file (*.in). A panel file is a detailed representation of all the panel points and their relative location. Whereas, the control file specifies the data which specifies the details of

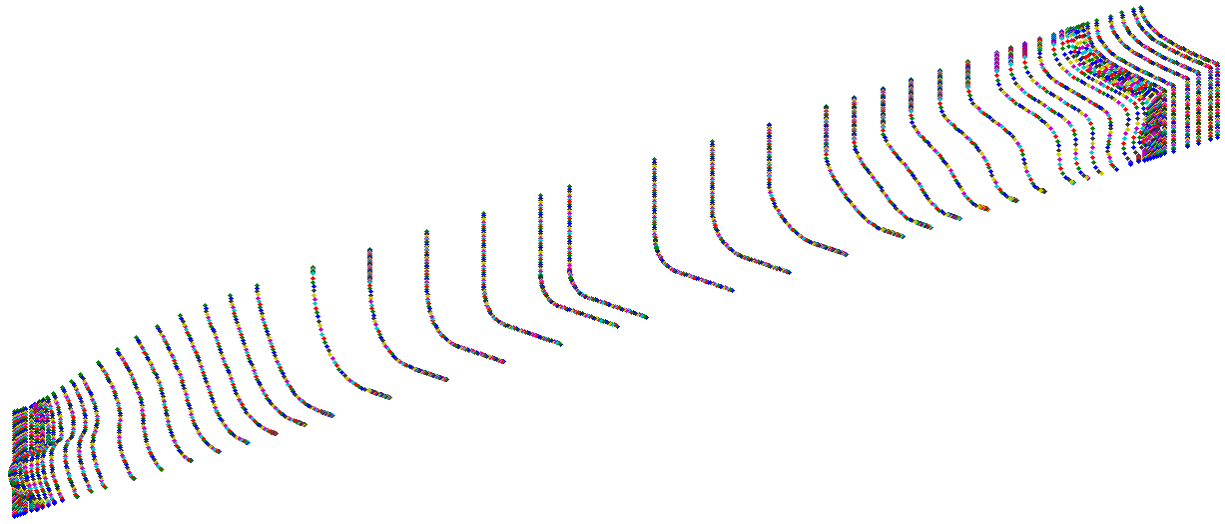

Figure 4.2: Input geometry file

the input and the output files, Froude number, free surface condition, flow condition etc. The code solves primarily for the pressure distribution. Later on wave making resistance, stream lines and free surface deformation are evaluated. Post processing of these output files is carried out using ParaView (figures 4.3 & 4.4).

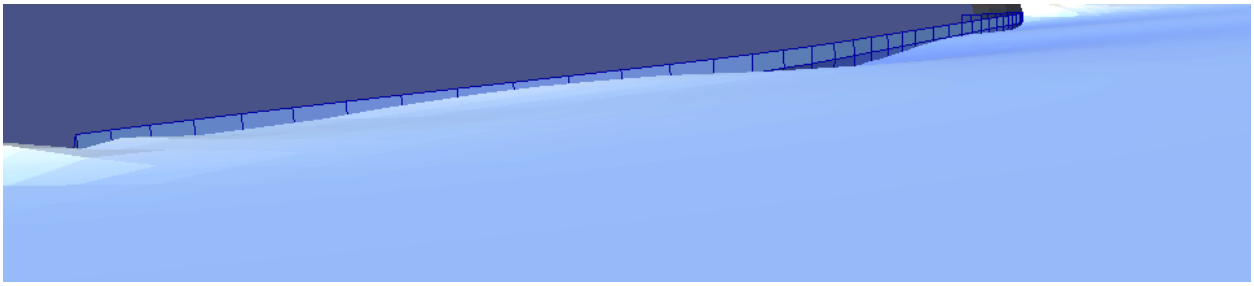

Figure 4.3: Waves generated along the ship's length

4.1.3 Optimization

As a basic optimization input a function to be optimized and an initial guess are defined. The function for optimization is the wave making resistance of the hull at design speed. The

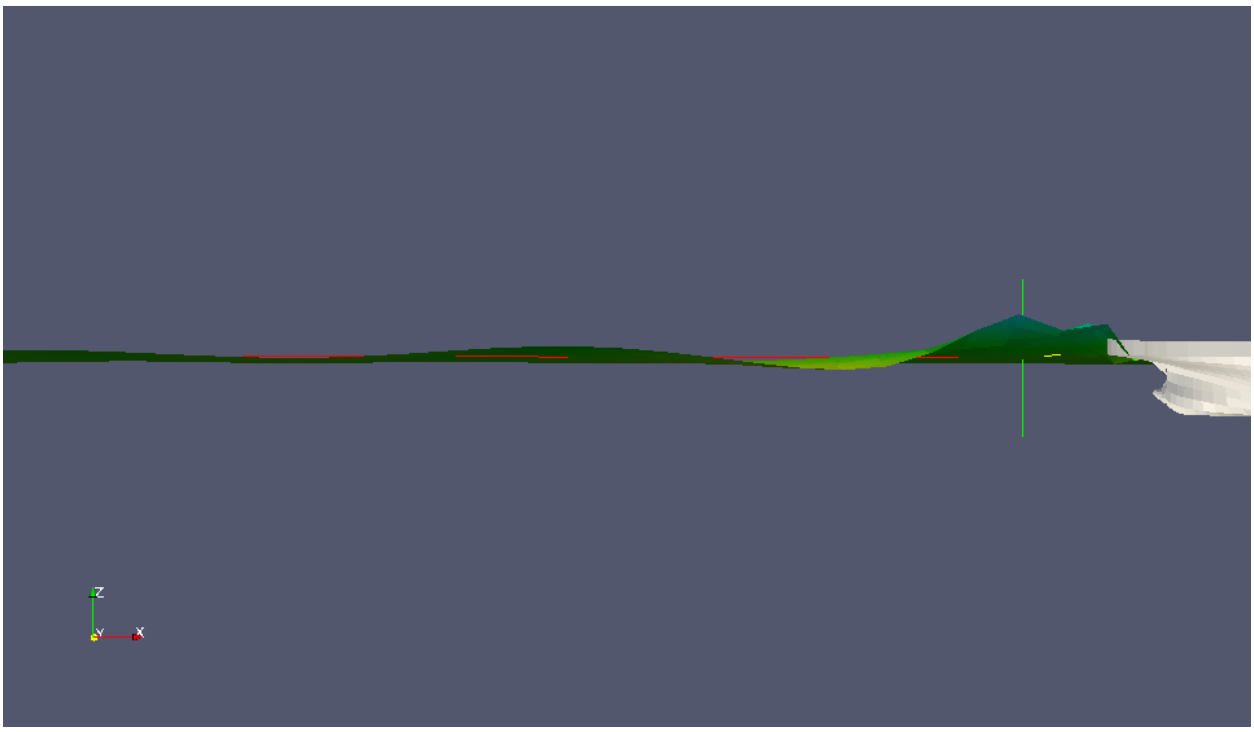

Figure 4.4: Waves generated at transom

panel file which represents the hull is already generated from geometry acquisition module. The panel stations are transformed using a lackenby transformation to achieve required LCB position for the hull. The LCB position is the variable for the optimization problem and at the end of optimization, an optimum location of LCB is determined.

Input for optimization module is:

- Speed, V = 24 knots
- Froude number, Fn = 0.26
- Parameter for raised panel height, $\alpha = 0.8$
- Initial guess of LCB shift of -0.05 %(of half length of Lpp) from midships. This position is definitely at a close proximity from the optimum location of LCB.

4.2 Output

Optimization algorithm searches for a optimum LCB position for a particular speed of ship. The coefficient of resistance starts at an initial guess value and gradually converges to a minimum (figure 4.5).

4.2.1 Optimization Output

- Number of Iterations = 9
- Total number of function evaluations = 19
- Minimum Coefficient of Wave Making Resistance, Cw = 0.00256509
- Optimized LCB location as a percentage of Lpp from midship = -1.65

The optimum LCB is 3.795 m aft of midships

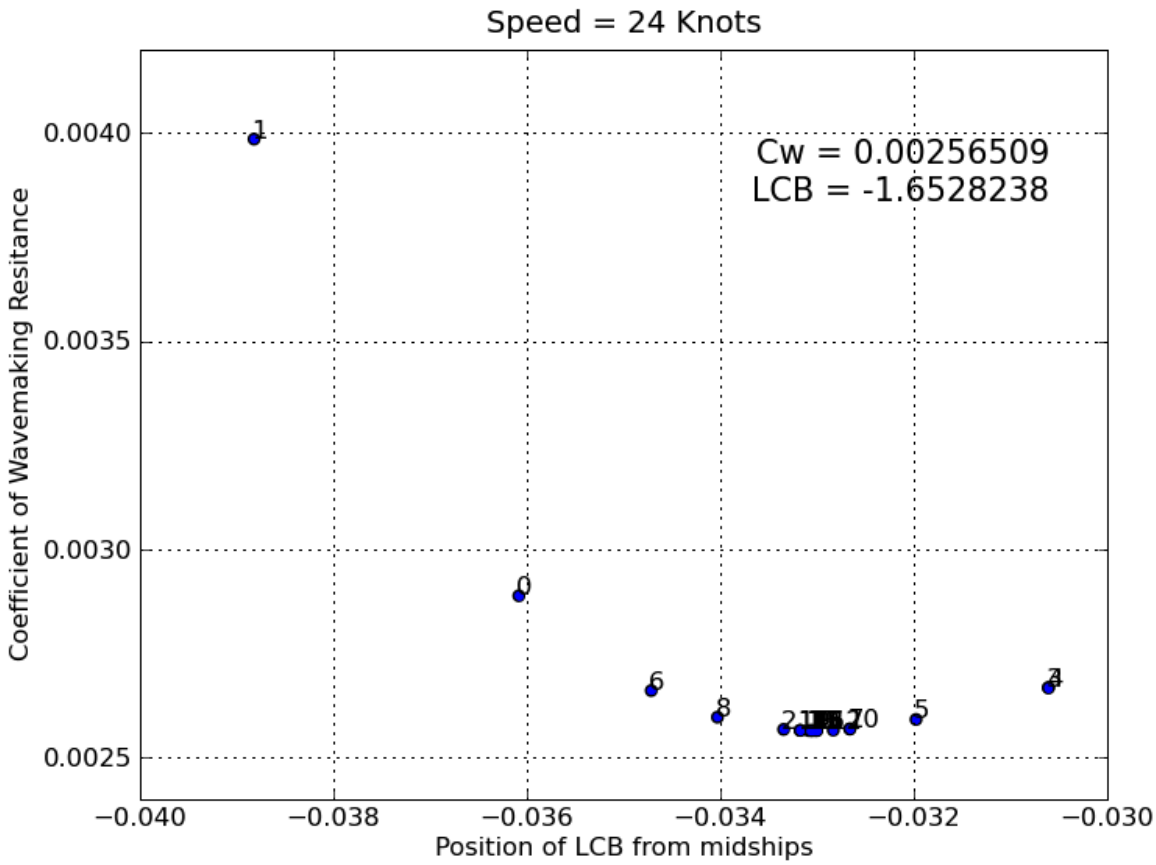

Figure 4.5: Optimization convergence at speed of 24 knots

Chapter 5

Discussion

1. **LCB position:** The location of optimized LCB is observed to be located aft of midships. This is as expected in view of hydrodynamics of a ship hull. The position of LCB as provided in a Flow Vision document [10] is -1.48 % of Lpp from midship. The observed value, -1.653 % of Lpp, is almost close to the published results and the deviation of 0.17 % with respect to Lpp. This might be for the reasons that the hull is approximated using linear panels.

2. **Wave making resistance:** The coefficient of wave making resistance, Cw = 0.002565, is more than three times of what it actually should be, value of Cw = 0.000731 as published [14].

   There are a lot of assumptions involved in calculating the resistance such as linearized free surface, geometry approximation using linear panels and the panel method itself. But, they will not effect the determination of optimized LCB position. The optimization algorithm only considers relative nature of resistance with change in LCB position.

3. **Wave pattern:** The wave pattern fromed by the ship hull is in agreement with the standard kelvin wave pattern with and half angle of 19.47 deg. The measured angle is 19.45 deg (figure 5.1)

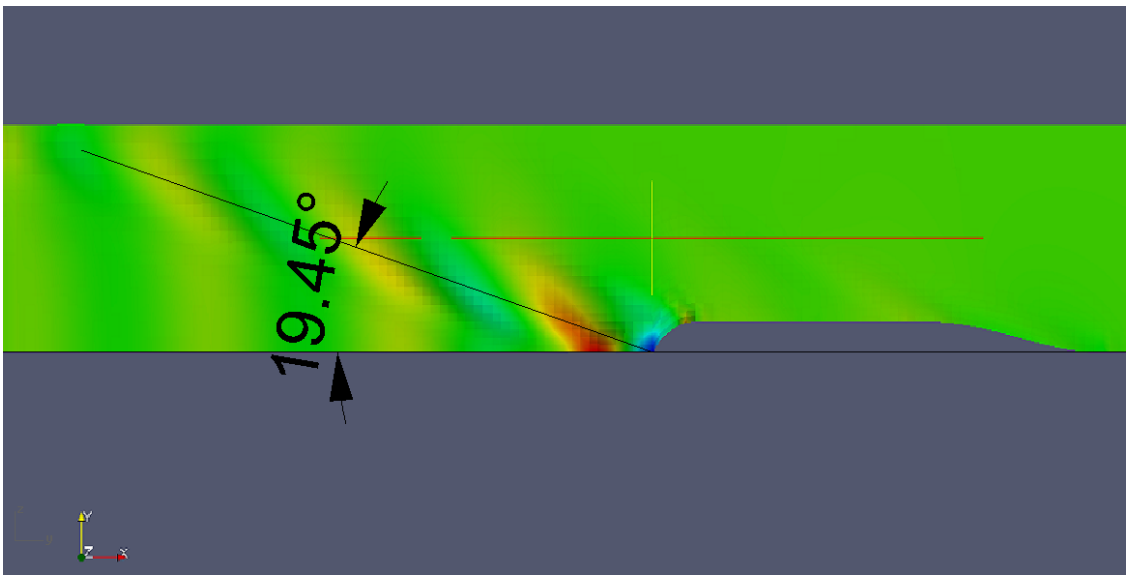

Figure 5.1: Kelvin wave angle

4. **Transom wave profile:** wave height at transom (figure 5.2) is higher than the measured wave profile data [14]. This is a general trend in panel methods for a ship hull.

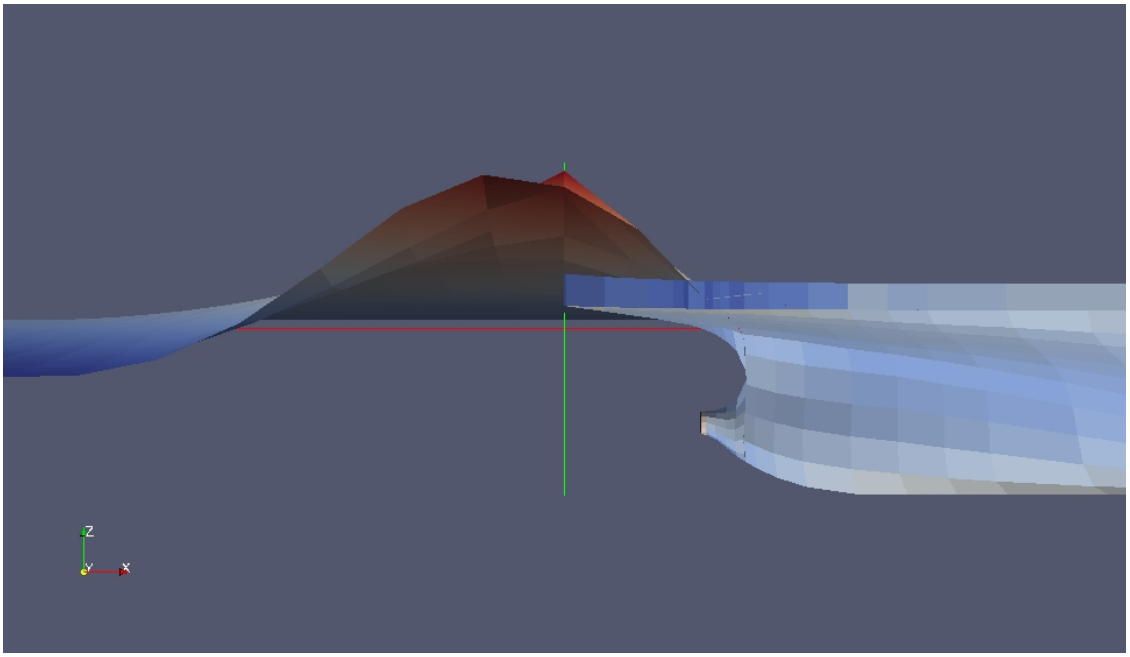

Figure 5.2: Higher wave elevation at transom

5. **Froude number:** The position of optimum LCB is observed to show a decreasing

trend (shifting towards aft) with increasing Froude number (figure 5.3). Further the LCB is expected to shift towards aft.

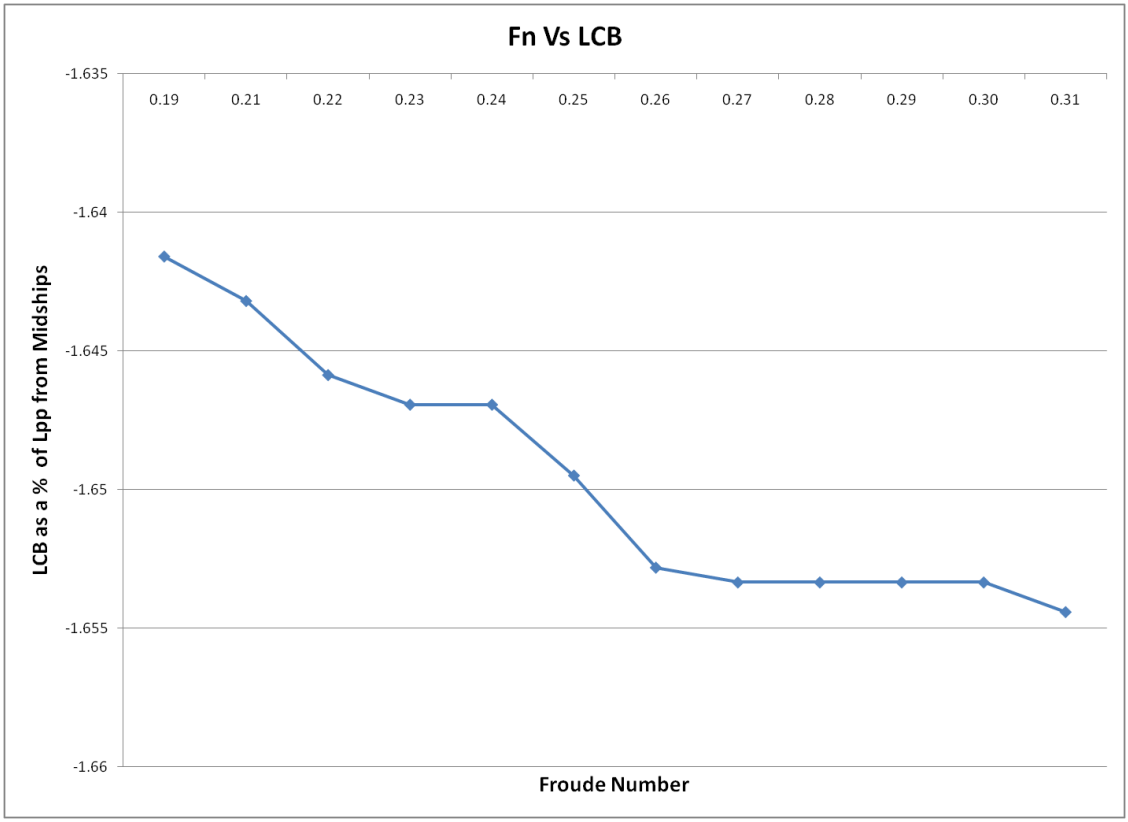

Figure 5.3: Froude Number vs LCB

Chapter 6

Conclusions

A generic method for optimization of ship hulls using panel method is presented. This method uses a simple interpolation scheme to generate panels for any conventional ship hull. The use of stations as a grid to develop panels eliminates the necessity to regenerate the panels for every iteration. This resulted in a significant improvement in computational speed. Lackenby transformation is used by the optimization algorithm to incorporate variation in hull geometry for a required LCB shift. A simple, yet efficient Nelder - Mead simplex method of optimization is used for this problem. In conclusion this approach has successfully determined the optimum location of LCB for a standard container ship hull.

The obvious advantages of this method include the ability to represent any conventional ship hull using a simple interpolation scheme for panel generation, an approximation which is reliable. Also, the use of lackenby method of transformation enables a control over preserving the volume and over all dimensions, while making a change in LCB position. The optimization algorithm selected is also very efficient in solving this kind of a problem.

The results from this work concur with published data and theory. The present approach focuses on locating an optimum LCB position for a ship hull and also observing the resistance characteristics of a ship hull at different speeds. The key point is achieving a best design lies in matching a ship hull with its operational conditions. The approach presented will enable a designer to identify the best possible design alternative for a selected condition.

Further, the panel method can be extended to handle transom immersion, appendages, trim and heel. The next milestone would be a panel code that includes non-linear free surface conditions. The range of free variable of optimization has to be extended to bulbous bow shape and also the over all dimensions. To add up, a stability approach would enhance the feasibility of the ship hull design.

Bibliography

[1] Susan maersk.

http://en.wikipedia.org/wiki/Emma_M%C3%A6rsk. vii, 25

[2] David W. Taylor Model Basin. and F. H. Todd. *Series 60 methodical experiments with models of single-screw merchant ships / by F. H. Todd.* [Washington,, 1964. 1

[3] Lothar Birk. Name 6145 course notes. Fall 09. vii, 10

[4] Lothar Birk. Panel code: nlhsfs v0.8-uno academic license. 2009. 11, 19

[5] H. Lackenby. On the systematic geometrical variation of ship forms. *INA*, 92:289–315, 1950. 3, 4, 11, 21

[6] Edward V. Lewis, editor. *Principles of naval architecture, Written by a group of authorities*, volume 1, pages 32–33. Society of Naval Architects and Marine Engineers, Jersey City, N.J., 1988. 6

[7] D. I. Moor, M. N. Parker, and R. N. M. Pattullo. The bsra methodical series: an overall presentation. *Transactions of the Royal Institution of Naval Architects*, 103:329–419, 1961. 1

[8] Ebru Narli and Kadir Sariöz. Geometrical variation and distorsion of ship hull forms. *Marine Technology*, 40(4):239–248, October 2003. 2

[9] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965. 8

[10] A. Pechenyuk. Computation of perspective kriso containership towing tests with the help of the complex of hydrodynamical analysis flowvision. Technical report, 2009. 30

[11] G. J. Goodrich R. E. Blackwell and D. J. Doust. The effect on resistance and propulsion of variation in lcb position. *Transactions of the Royal Institution of Naval Architects*, 99:367–406, 1957. 6

[12] C. RIDGLEY-NEVITT. The resistance of a high displacement-length ratio trawler series. *Transactions of the SNAME*, 75, 1967. 1

[13] D. P. Roseman. *Marad Systematic Series of Full Form Ship Models*. Society of Naval Architects and Marine Engineers, Jersy City, NJ, 1987. 1

[14] KRISO Container Ship. Gothenberg 2000. http://www.iihr.uiowa.edu/gothenburg2000/KCS/container.html. 3, 23, 24, 30, 31

[15] A. Singer and J. Nelder. Nelder-mead algorithm. *Scholarpedia*, 4(7):2928, 2009. 8

[16] D. W. Taylor. Calculation of ship's forms and the light thrown by model experiments upon resistance, propulsion and rolling of ships. *Transactions of the International Engineering Congress*, 1915. 1

[17] Ping ZHANG, De xiang ZHU, and Wen hao LENG. Parametric approach to design of hull forms. *Journal of Hydrodynamics, Ser. B*, 20(6):804 – 810, 2008. 2

Appendix: Python Source Code

```
# Mar 03, 2010
# kkarri, Hull shape optimization using lackenby transformation
# and Nelder-mead simplex algorithm

# importing required packages

import numpy as np
from scipy.integrate import trapz,simps
from scipy.io import savemat
from subprocess import call

#.......... Pre defined functions ..............

def readoffsets(filename, T):
    """
    Function to read offset file
    Input : filename of offsets file
            T is the draft
    Output: xlong  = station spacing along x axis from AP
            offsets = offsets stores half breadth as [station number, half
                breadth, 0]
                    offsets stores height as [station number, height above
                        baseline, 1]
            Across = Sectional area at each station.
```

```
    """
    f = open(filename,"r")
    lines=[]
    data=[]
    original_lines = f.readlines()
    for line in original_lines:
        line = line.strip()
        if line[0]!="#":
            lines.append(line)
    f.close()

    #create  storage space for data of dimension (number of stations, no of
        points,2)
    offsets = np.zeros((57,100,2),float)

    # offset lines read for data
    # xlong is longitudinal distance of section from AP,
    xlong = []
    yst = []
    zst = []
    for i in np.linspace(0,2296,57):    # for reading data in lines
        (0,41,82....)
        xlong.append(float(lines[int(i)].split()[0]))
        b = []
        h = []
        for j in np.linspace(1,20,20):
            b.append(lines[int(i)+int(j)].split())
            h.append(lines[20+int(i)+int(j)].split())

        offsets[int(i/41),:,0] = np.reshape(b,np.size(b))    #b assigned to
            offsets
```

```
        offsets[int(i/41),:,1] = np.reshape(h,np.size(h))    #h assigned to
            offsets
        yst.append(offsets[int(i/41),:,0])
        zst.append(offsets[int(i/41),:,1])
    xlong = np.asarray(xlong)

    # Across stores Sectional Area of each station
    Across = []
    Girth = []
    for i in range(0,57):  # runs through all stations
        A=0
        for j in range(0,99):  # runs through all points
            # The cross-sectional area is calculated upto draft
            if offsets[i,j+1,1] < T :
                # for h < T, sec area is calculated by adding up area of
                # Trapeziums
                A = A + 2*((offsets[i,j+1,1] - offsets[i,j,1])*0.5*\
                                (offsets[i,j,0]+offsets[i,j+1,0]))
            else :
                # b_T = b_previous +(delta_B / delta_h) * (T-b)
                b_T = offsets[i,j,0] +((offsets[i,j+1,0]-offsets[i,j,0])\
                      *(T-offsets[i,j,1])/(offsets[i,j+1,1] \
                                            - offsets[i,j,1]))
                A = A + 2*((T - offsets[i,j,1])*0.5*\
                           (offsets[i,j,0]+b_T))
                break
        Across.append(A)

        # generating offset array
        #offsets stores half breadth as [station number, half breadth, 0]
        #offsets stores height as [station number, height above baseline, 1]
```

```
81          y = offsets[i,:,0]
            z = offsets[i,:,1]
83          # v = number of points along girthwise
            v = 99
85          dist = ((np.diff(y))**2 + (np.diff(z))**2)**0.5
            girth = np.zeros(len(dist)+1)
87          delta = np.zeros(v+1)
            for k in range(len(dist)):
89              girth[k+1] = girth[k] + dist[k]
            zpan = []
91          ypan = []
            for k in range(int(v)):
93              delta[k] = sum(dist)*k/v
                for j in range(1,len(girth)):
95                  if (girth[j-1] <= delta[k])&(girth[j] > delta[k]):
                        ypan.append(y[j-1] + ((y[j]-y[j-1])*(delta[k]-girth[j-1])
                            /\
97                                                  (girth[j]-girth[j-1])))
                        zpan.append(z[j-1] + ((z[j]-z[j-1])*(delta[k]-girth[j-1])
                            /\
99                                                  (girth[j]-girth[j-1])))
            Girth.append(girth[-1])
101     Across = np.asarray(Across)
        Girth = np.asarray(Girth)
103
        return xlong, offsets,yst,zst, Across,Girth
105


107 def patch(xlong,offsets,sta,p,vp,r,vl,vu,sx,sy,sz,tx,ty,tz,bx,by,bz,beta,Lpp,T
        ):
        """
```

```
    INPUT:
    sta: station number or array. Specifies stations of a patch
    p:  patch number: 1-stern tube; 2-transom; 3-hull body; 4-bulb
    v:  no of panels girthwise ??
    r: ?

    """

    x = []
    y = []
    z = []

    if p == 1:
        # patch 1 (stern tube) is closed at aft end
        for i in range(vl+1):
            x.append(sx)
            y.append(sy)
            z.append(sz)
    if p == 2:
        # patch 2 (transom) is closed at aft end
        for i in range(vu+2):
            x.append(tx)
            y.append(ty)
            z.append(tz)
        # patch closure at the end for free board panel
        z[-1] = (tz + (beta * Lpp))

    for i in sta:  # runs through all stations
        z1,y1,ystart,yend = station(i,offsets,beta,Lpp,T)
        ypan, zpan = stadiv(z1,y1,ystart,yend,p,vp,vl,vu,i,sta[0],8.96)
        if p == 1:
```

```
            for j in range(len(ypan[0])):
                x.append(xlong[i])
                y.append(ypan[0][j])
                z.append(zpan[0][j])
        if p == 2:
            if np.size(ystart) == 1:
                for j in range(len(ypan)):
                    x.append(xlong[i])
                    y.append(ypan[j])
                    z.append(zpan[j])
            elif np.size(ystart) == 2:
                for j in range(len(ypan[1])):
                    x.append(xlong[i])
                    y.append(ypan[1][j])
                    z.append(zpan[1][j])
        if (p== 3) | (p ==4):
            for j in range(len(ypan)):
                x.append(xlong[i])
                y.append(ypan[j])
                z.append(zpan[j])
    if p == 4:
        # patch 4 (bulbous bow) is closed at fwd end
        for i in range(vp +1):
            x.append(bx)
            y.append(by)
            z.append(bz)
    return np.rot90(np.reshape(x,(-1,r+1)),-1),\
           np.rot90(np.reshape(y,(-1,r+1)),-1),\
           np.rot90(np.reshape(z,(-1,r+1)),-1)-T

def station(i,offsets,beta,Lpp,T):
```

```
171     """
        INPUT:
173     i = station number
        OUTPUT:
175     z = actual station water lines
        y = actual station offsets
177     z1 = station water lines upto Load Water Line
        y1 = station offsets upto Load Water Line
179     p = returns the number of discontinuous sections(i.e. for transom p =2
            as there are two discontinuous section lines
181     ystart = specifies the start points of sections
        yend  = specifies the end points of sections
183
        note: Sections are defined as continuous station lines,
185         for example: transom section, stern tube section, body sections,
                        bulbous bow section.
187     """
        z = []
189     y = []

191     # This loop selects the submerged portion of the hull and freeboard
        #i.e. upto water line and free board, stores offsets in y and z
193
        for j in range(0,99):# runs through all points
195         if offsets[i,j,1] < T:
                z.append(offsets[i,j,1])
197             y.append(offsets[i,j,0])

199         else :
                b_T = offsets[i,j,0] +((offsets[i,j+1,0]-offsets[i,j,0])\
201                     *(T-offsets[i,j,1])/(offsets[i,j+1,1] \
```

```
                                                - offsets[i,j,1]))
203             z.append(T)
                y.append(b_T)
205
                # for freeboard offsets
207             fb = beta * Lpp
                T_fb = T + fb
209             j = np.nonzero(offsets[i,:,1] >= T_fb)[0][0] -1
                b_fb = offsets[i,j,0] +((offsets[i,j+1,0]-offsets[i,j,0])\
211                   *(T_fb-offsets[i,j,1])/(offsets[i,j+1,1] \
                                                - offsets[i,j,1]))
213             z.append(T_fb)
                y.append(b_fb)
215             break

217     # y1 ans z1 returns the values of interpolated values upto WL
        z1 = []
219     y1 = []
        # this is returned with a value specifying no. of patches
221     ystart = []
        yend = []
223     for k in range(1,len(z)):
            if (y[k] > 0.) & (y[k-1] == 0.):
225             ystart.append(k-1)
                yend.append(len(z)-1)
227             y1.append(y[k-1])
                z1.append(z[k-1])
229             y1.append(y[k])
                z1.append(z[k])
231         elif (y[k] > 0.) & (y[k-1] >0.):
                y1.append(y[k])
```

```
233                 z1.append(z[k])
            elif (y[k] == 0.) &( y[k-1] >0.):
235                 yend[len(yend)-1] = k
                    y1.append(y[k])
237                 z1.append(z[k])
        return z1,y1,ystart,yend
239
    def stadiv(z1,y1,ystart,yend,p,v,vl,vu,i,bs,tb):
241     """
        This function divides a section into equal patches
243
        INPUT:
245     z1 = water line spacing
        y1 = half breadth
247     ystart = specifies the start points of sections
        yend  = specifies the end points of
249     p = no. of patches
        v = number of patches in verical direction
251     i = station number
        bs = station number at bulb closing, generally sta[0]
253     tb = trimming draft at bulb, points above tb are neglected in bulb patch

255     OUTPUT:
        ypan and zpan are the offsets for the panel points
257     """
        # division for stations with continious sections
259     if (np.size(ystart) == 1) | (p == 3):
            ypan, zpan = girthdiv(v,y1,z1,1)
261     # division for bulb sections
        if p == 4:
263         if (i == bs)&(p == 4):  # for bulb join station
```

```
            # this line makes the bulb an approximate cylinder
265             up = np.nonzero(np.array(z1)>tb)[0][0]
                ypan, zpan = girthdiv(v,y1[:up],z1[:up],0)
267         else:
                ypan, zpan = girthdiv(v,y1,z1,0)
269     # division when there are discontinious sections
        elif (np.size(ystart) == 2) & (p != 3)&(p!=4):
271         y1l = y1[0:yend[0]-ystart[0]+1]
            y1u = y1[yend[0]-ystart[0]+1:len(y1)]
273         z1l = z1[0:yend[0]-ystart[0]+1]
            z1u = z1[yend[0]-ystart[0]+1:len(z1)]
275
            # for lower patch
277         ypanl, zpanl = girthdiv(vl,y1l,z1l,0)
            # for upper patch
279         ypanu, zpanu = girthdiv(vu,y1u,z1u,1)

281         ypan = [ypanl,ypanu]
            zpan = [zpanl,zpanu]
283     return ypan, zpan

285 def girthdiv(v,y2,z2,wl):
        """
287     This function divides any given section patch into number of equal
        panels. The calcuation of patch length is based on the length along girth
289
        INPUT :
291     v = number of patches in verical direction
        y2 = half breadth
293     z2 =  water line spacing
        wl = 1 if patch includes a water line, 0 if it don't
```

```
    OUTPUT:
    ypan and zpan are the offsets for the divided section patches
    """
    if wl == 0:
        y = y2
        z = z2
    else:
        y = y2[:-1]
        z = z2[:-1]
    dist = ((np.diff(y))**2 + (np.diff(z))**2)**0.5
    girth = np.zeros(len(dist)+1)
    delta = np.zeros(v+1)
    for i in range(len(dist)):
        girth[i+1] = girth[i] + dist[i]
    zpan = []
    ypan = []
    for i in range(int(v)):
        delta[i] = sum(dist)*i/v
        for j in range(1,len(girth)):
            if (girth[j-1] <= delta[i])&(girth[j] > delta[i]):
                ypan.append(y[j-1] + ((y[j]-y[j-1])*(delta[i]-girth[j-1])/\
                                      (girth[j]-girth[j-1])))
                zpan.append(z[j-1] + ((z[j]-z[j-1])*(delta[i]-girth[j-1])/\
                                      (girth[j]-girth[j-1])))
    zpan.append(z[-1])
    ypan.append(y[-1])
    if wl==1:
        zpan.append(z2[-1])
        ypan.append(y2[-1])
    return ypan, zpan

```

```
327
def stainterp(xlong,zone,pandist,npan,x,y,z):
329     # number of points for new stations
        nlen = len(x)
331
        n_sta= sta_gen(zone,pandist,npan)
333
        nx= np.zeros([len(x),len(n_sta)],float)
335     ny= np.zeros([len(x),len(n_sta)],float)
        nz= np.zeros([len(x),len(n_sta)],float)
337
        descending = 0
339     if np.any(np.diff(x[0])<0):
            descending = 1
341         x = x[:,::-1]
            y = y[:,::-1]
343         z = z[:,::-1]

345     for i in range(len(n_sta)):

347         if n_sta[i] > x[0][-1]:
                xfloor = len(x) - 1
349         else:
                xfloor = np.searchsorted(x[0],n_sta[i])
351
            xl = x[:,xfloor]
353         xr = x[:,xfloor  -1]
            yl = y[:,xfloor]
355         yr = y[:,xfloor  -1]
            zl = z[:,xfloor]
```

```
357         zr = z[:,xfloor   -1]


359         for j in range(len(x)):
                delta = ((xr[j]-xl[j])**2 + (yr[j]-yl[j])**2 + \
361                         (zr[j]-zl[j])**2)**0.5
                t = (n_sta[i]-xl[j])*(delta/(xr[j]-xl[j]))
363             nx[j,i] = n_sta[i]
                ny[j,i] = yl[j]+ (t*((yr[j]-yl[j])/delta))
365             nz[j,i] = zl[j]+ (t*((zr[j]-zl[j])/delta))


367     if descending == 1:
            nx = nx[:,::-1]
369         ny = ny[:,::-1]
            nz = nz[:,::-1]
371
        return nx,ny,nz
373
    def sta_gen(zone, pandist,npan):
375     """
        Input:
377     zone = [aft ext, fwd ext, xstart, xend]
                aft and fwd ext are the extreme points on the hull
379             xstart is where the first station starts, xend it where it ends
        pandist = Panel distribution array - [[-1.0,-.9,-0.25,0.,0.25,0.9,1.0],\
381                         [0.0,0.25,1.0,1.0,1.0,0.25,0.0]]


383             First row - longitudinal axis (normalized
                Second row represents the variation of panel length form aft to
                    fwd,
385     npan = number of panels points
        OUTPUT:
```

```
387     n_sta = normalized new stations in the specified zone

389     """
        x = np.linspace(min(pandist[0]),max(pandist[0]),100)
391     y = np.interp(x,pandist[0],pandist[1])
        A = []
393     for i in range(len(x)):
            A.append(trapz(y[0:i+1],x[0:i+1]))
395     A = np.array(A)
        zx = np.interp(np.linspace(zone[2],zone[3],npan),\
397                    np.linspace(zone[0],zone[1],100),x)
        n_sta = np.interp(zx,x,A)
399     # Normalized
        n_sta = n_sta/(n_sta[-1]-n_sta[0])
401     n_sta = n_sta - min(n_sta)
        n_sta = zone[2] + (n_sta * abs(zone[3]-zone[2]))
403
        return n_sta
405
    def visual(x,y,z,p,disp):
407
        #save final patch offsets to *.arr
409     savemat('x_'+str(p)+'.mat', mdict={'arr':x})
        savemat('y_'+str(p)+'.mat', mdict={'arr':y})
411     savemat('z_'+str(p)+'.mat', mdict={'arr':z})

413     if disp == 1:
            # To view the panels geometry with MATLAB
415         fig = plt.figure()
            ax = Axes3D(fig,aspect = 'auto')
417         # importing plotting function if required
```

```
        from matplotlib import cm
419     from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.pyplot as plt
421     # creating surface plot
        ax.plot_surface(x,y,z,rstride = 1, cstride = 1,cmap = cm.jet)
423     # star board side of hull
        ax.plot_surface(x,-y,z,rstride = 1, cstride = 1,cmap = cm.jet)
425     ax.set_xlabel('X')
        ax.set_ylabel('Y')
427     ax.set_zlabel('Z')

429 def patchfs(xwl,ywl,fspan,v_fs,T,Lpp,Lwl,delta_s,beta,w_beta):
    """
431 initial free surface points generation

433 INPUT:
    xwl: x cordinates of waterline
435 ywl: y coordinates of water line
    fspan : Number of Free surface panels for one ship lenght
437 v_fs: transverse number of panel
    Lpp: Length between Perpendiculars
439 delta_s: Factor to determine initial panel offset from Hull water line
            (~ 0.015)
441 w_beta: Rate of increase in panel width away from hull. (~ 1.1)
    """
443 #Lwl local
    xfs_0 = np.linspace(xwl[0]+ (0.5*Lwl),xwl[-1]-(1.5*Lwl),(3*fspan)+1)
445 xfs = np.tile(xfs_0, (v_fs+1,1))
    # number of panels on WL, length wise
447 s_fs = np.shape(xfs)[1] -1
    # Interpolate halfbreadths for patch points of free surface
```

```
449     yfs_0 = np.interp(xfs_0[(0.5*fspan):-((1.5*fspan)+1)][::-1],\
                    xwl[::-1], ywl[::-1])[::-1]
451     #attaching panels fwd
        yfs_0 = np.append(np.zeros(0.5*fspan),yfs_0)
453     #attaching panels aft
        yfs_0 = np.append(yfs_0,np.zeros((1.5*fspan)+1))
455     yfs = []
        delta = (np.vander([w_beta],v_fs+1)*Lpp*delta_s).squeeze()
457
        for i in range(len(yfs_0)):
459         scale = (sum(delta)- yfs_0[i]) / sum(delta)
            delta1 = scale * delta[::-1]
461         yfs.append(yfs_0[i])
            for j in range(1,v_fs):
463             yfs.append(delta1[j]+yfs[-1])
            yfs.append(delta_s*w_beta*Lpp*(1- w_beta**(v_fs))/(1 -w_beta))
465
        yfs = np.array(yfs).reshape(s_fs+1,v_fs+1).T
467     zfs = np.zeros((v_fs +1,s_fs +1))
        fb = beta * Lpp
469     zfs[...] = T+ fb


471     return xfs,yfs,zfs-T


473 def resistance(x0):
        """
475     This function calculates the resist5ance for the modified hull form
        by dLCB
477     """
        desc = 0
479     if np.any(np.diff(nx3[0])<0):
```

```
        desc = 1
481         xparent = nx3[:,::-1][0,:]

483     sacparent = np.interp(xparent,xlong,Across)
        # distance of first station from A.P.
485     min_xp = min(xparent)
        # modifying xparent to have first station at x = 0
487     xparent = xparent - min_xp

489     # Part 2: Modifying Hull form
        xderived,sacparent,CPa,Cpf,pf,CP,LCB = \
491                     lackenby(xparent,sacparent,dpa=0,dpf=0,dCP=0, dLCB = x0)
        # modifying xderived to get back to normal state i.e. first station at
493     # x = min_xp
        if desc == 1:
495
            xderived = xderived[::-1] + min_xp
497     else:

499         xderived = xderived + min_xp

501     # Part3: Calculate resistance using Dr. Birk's panel code
        R = panelmethod(xderived,filename,origin,Lpp,Lwl,nx1,nx2,nx2t,nx3,nx4,nx5
            ,\
503         ny1,ny2,ny2t,ny3,ny4,ny5,nz1,nz2,nz2t,nz3,nz4,nz5,fspan,v_fs,T,delta_s
                ,beta,w_beta)

505     # Part4: Constraint application
        ## evaluate constraints for penalty function
507     c = []  # empty list for constraint values
```

```
509     # constraints
        c.append(lcbrange(LCB,xparent[-1]))
511
        # convert the list to a numpy array (for faster math)
513     c = np.asarray( c )

515     # Define and compute exterior penalty function
        r = 1.e10
517     """
            Multiplication by a huge number
519     """
        P = np.sum( c*c )
521
        print "Longitudinal center of Buoyancy, LCB = %12.12f"%LCB
523     print 'Resistance, R = %12.12f'%R
        print 'Penalty function, r*P = %f' % (r*P)
525     print "\n"
        fp = open("optimization"+str(Vs)+'_'+str(alpha)+".dat",'a')
527     fp.write(' %12.12f %12.12f\n' % (LCB, R))
        fp.close()
529     return R + r*P

531 def lackenby(xparent, sacparent,dpa,dpf,dCP, dLCB):

533     # parallel mid boy data
        p = 0.0   # total length of parallel midbody / half length
535     pa = 0.00   # par. midbodz aft of midship / half length
        pf = p - pa # par. midbodz forward of midship / half length
537
        #extract parent hull data
539     L, imidship, AM, V, CP, LCB, CPa, CPf, LCBa, LCBf, ka, kf = \
```

```
        gethulldata(xparent,sacparent)
    dx, xderived = getstationshifts(xparent, imidship, dpa, dpf, dCP, dLCB,L,\
                            p, pa, pf, CP, CPa, CPf, LCB, LCBa, LCBf, ka, kf)

    L, imidship, AM, V, CP, LCB, CPa, CPf, LCBa, LCBf, ka, kf = \
        gethulldata(xderived, sacparent)
    return xderived,sacparent,CPa,CPf,pf,CP,LCB

def gethulldata(xsta,sac):
    # length (x pointing positive forward)
    L = xsta[-1]

    # maximum sectional area
    AM = max(sac)

    # displacement
    # integration by Simpson's rule
    V = simps(sac, x=xsta, even='last')

    # Prismatic coefficient
    CP = V / (L*AM)

    # Compute longitudinal center of buoyancy (LCB)
    LCB = simps(sac*xsta, x=xsta, even='last') / V

    # Find index of midship or maximum section position
    for i in range(len(xsta)):
        if (sac == max(sac))[i]:
            imidship = i
            break
```

```
571     # xposition of midship
        xm = xsta[imidship]
573
575     # Compute prismatic coefficients, LCBs and radius of gyration
        # for fore body
577     Vf = simps(sac[imidship:], x=xsta[imidship:], even='last')
        CPf = Vf / ((L-xm)*AM)
579     LCBf = simps(sac[imidship:]*(xsta[imidship:]-xm), \
                     x=(xsta[imidship:]-xm), even='last') / Vf
581     kf = np.sqrt(simps(sac[imidship:]*(xsta[imidship:]-xm)**2, \
                     x=(xsta[imidship:]-xm), even='last') / Vf)
583
        # and aft body
585     Va = simps(sac[:imidship+1], x=xsta[:imidship+1], even='last')
        CPa = Va / ((xm)*AM)
587     LCBa = simps(sac[:imidship+1]*(xm-xsta[:imidship+1]), \
                     x=-(xm-xsta[:imidship+1]), even='last') / Va
589     ka = np.sqrt(simps(sac[:imidship+1]*(xm-xsta[:imidship+1])**2, \
                   x=-(xm-xsta[:imidship+1]), even='last') / Va)
591
        # refer LCB to midship and make dimensionless with half length
593     LCB = (LCB-xm)/xm

595     # make data dimensionless where still necessary
        LCBa = LCBa/xm
597     ka   = ka/xm
        LCBf = LCBf/(L-xm)
599     kf   = kf/(L-xm)

601     return L, imidship, AM, V, CP, LCB, CPa, CPf, LCBa, LCBf, ka, kf
```

```
603
def getstationshifts(xparent, imidship, dpa, dpf, dCP, dLCB, \
605                      L, p, pa, pf, CP, CPa, CPf, LCB, LCBa, LCBf, ka, kf):
    """
607         Computes station shifts using Lackenby's method

609         dx = e(1-x)(x+d)
    """
611     # Step 1: Compute constants A, B, C for fore and aft body
    Aa = A( pa, CPa, LCBa )
613     Af = A( pf, CPf, LCBf )
    Ba = B1( pa, Aa, CPa, LCBa, ka )
615     Bf = B1( pf, Af, CPf, LCBf, kf )
    Ca = C( pa, Ba, CPa, LCBa )
617     Cf = C( pf, Bf, CPf, LCBf )
    # Step 2: Compute necessary changes in CPa and CPf )
619     # Lackenby's equations
    dCPf = (2.*(dCP*(LCB+Ba) + dLCB*(CP+dCP)) + dpf*Cf - dpa*Ca) / (Ba+Bf)
621     dCPa = (2.*(dCP*(Bf-LCB) - dLCB*(CP+dCP)) - dpf*Cf + dpa*Ca) / (Ba+Bf)

623     # Step 3: Compute constants for shift function
    ea = e( dpa, dCPa, pa, Aa, CPa)
625     ef = e( dpf, dCPf, pf, Af, CPf)
    da = d( dpa, pa, ea)
627     df = d( dpf, pf, ef)

629     # Setp 4: compute shifs and new station positions
    xderived = np.zeros((len(xparent)), float)
631     dx = np.zeros((len(xparent)), float)
```

```
633     # aft ship
        xm = xparent[imidship]
635     for i in range(imidship):
            x = (xm - xparent[i]) / xm
637         dx[i] = (-ea * (1.-x)*(x + da))*xm
        # fore ship
639     for i in range(imidship+1,len(xparent)):
            x = (xparent[i]-xm) / (L-xm)
641         dx[i] = (ef * (1.-x)*(x + df))*(L-xm)

643     xderived = xparent + dx
        return dx, xderived
645
    def A(p, CP, LCB):
647
        return CP*(1.-2.*LCB) - p*(1.-CP)
649
    def B1(p, A, CP, LCB, k):
651
        return CP/A * (2.*LCB - 3.*k**2 - p*(1.-2.*LCB))
653
    def C(p, B, CP, LCB):
655     return (B*(1.-CP) - CP*(1.-2.*LCB)) / (1.-p)

657 def e( dp, dCP, p, A, CP):

659     return (dCP - dp/(1.-p)*(1.-CP)) / A

661 def d( dp, p, e):

663     return dp/(e*(1.-p)) - p
```

```
665 def panelmethod(xderived,filename,origin,Lpp,Lwl,tx1,tx2,tx2t,tx3,tx4,tx5,ty1,
    ty2,ty2t,\
                ty3,ty4,ty5,tz1,tz2,tz2t,tz3,tz4,tz5,fspan,v_fs,T,delta_s,beta
                    ,w_beta):
667
    tx3 = np.tile(xderived,(np.shape(tx3)[0],1))
669     # Waterline panel points
    txwl = np.append(tx3[-1],tx2[-1])
671     txwl = np.append(txwl,tx2t[-1][1:])
    tywl = np.append(ty3[-1],ty2[-1])
673     tywl = np.append(tywl,ty2t[-1][1:])

675     tx5,ty5,tz5 = patchfs(txwl,tywl,fspan,v_fs,T,Lpp,Lwl,delta_s,beta,w_beta)

677     # "panfile" creates the *.pan file
    panfile(filename,origin,Lwl,tx1,tx2,tx2t,tx3,tx4,tx5,ty1,ty2,ty2t,ty3,ty4,
        ty5,\
679             tz1,tz2,tz2t,tz3,tz4,tz5,fspan)

681     call(["nlhsfs","nlhsfsctrl.in","run.log"])

683     fp = open('run.log','r')
    lines = fp.readlines()
685     for line in lines:
        line = line.split()
687         if line[0]!="#":
            Cw = float(line[2])
689     fp.close()

691     return Cw
```

```
693 def panfile(filename,origin,Lwl,px1,px2,px2t,px3,px4,px5,py1,py2,py2t,py3,py4,
    py5,\
                pz1,pz2,pz2t,pz3,pz4,pz5,fspan):
695     reflength = Lwl
    grav = 9.81
697     symx = 0
    symy = 1
699     nhullpatches = 5
    nfspatches = 1
701     nhullpoints = np.size(px1) + np.size(px2)+np.size(px2t) + np.size(px3)+ np
        .size(px4)
    nfspoints = np.size(px5)
703     nhullpanels = (np.shape(px1)[0]-1)*(np.shape(px1)[1] -1) + \
                  (np.shape(px2)[0]-1)*(np.shape(px2)[1] -1) + \
705               (np.shape(px2t)[0]-1)*(np.shape(px2t)[1] -1)+\
                  (np.shape(px3)[0]-1)*(np.shape(px3)[1] -1)+\
707               (np.shape(px4)[0]-1)*(np.shape(px4)[1] -1)
    nfspanels = (np.shape(px5)[0]-1)*(np.shape(px5)[1] -1)
709     fp = open(filename, 'w')
    fp.close()
711     fp = open(filename, 'a')
    #lines 1 to 11
713     fp.write("##+\n# kriso container ship \n reflength %f\n grav %3.2f\n \
sym     %i     %i\n nhullpatches      %i\n nfspatches        %i\n \
715 nhullpoints     %i\n nfspoints      %i\n nhullpanels     %i\n \
nfspanels      %i\n" %(reflength,grav,symx,symy,nhullpatches,nfspatches,\
717                          nhullpoints,nfspoints,nhullpanels,nfspanels))

719     #lines 12 to 16
    st = np.array([0])
```

```
721     p4 = hull(fp,origin,st,4,0,px4,py4,pz4,1,4,0,0,0,0,0)
        p3 = hull(fp,origin,p4,3,1,px3,py3,pz3,1,3,0,\
723                 (0.5*fspan),(0.5*fspan)+np.shape(px3)[1]-1,489,526)
        p1 = hull(fp,origin,p3,1,0,px1,py1,pz1,1,1,0,0,0,0,0)
725     p2 = hull(fp,origin,p1,2,1,px2,py2,pz2,1,2,0,(0.5*fspan)+np.shape(px3)
            [1]-1,\
                    (0.5*fspan)+np.shape(px3)[1]-1+np.shape(px2)[1]-1,668,479)
727     p2t = hull(fp,origin,p2,2,1,px2t,py2t,pz2t,1,2,0,(0.5*fspan)+np.shape(px2)
            [1]-1,\
                    (0.5*fspan)+np.shape(px2)[1]-1+np.shape(px2)[1]-1,668,479)
729
        # lines 17 through 21
731     p5 = freesurf(fp,origin,st,1,px5,py5,pz5,1,1,0,(0.5*fspan),\
                        (0.5*fspan) + np.shape(px3)[1]-1+np.shape(px2)[1]-1,7,54)
733
        #define st =1
735     st4=1

737     st3 = points(fp,st4,px4,py4,pz4)
        st1 = points(fp,st3,px3,py3,pz3)
739     st2 = points(fp,st1,px1,py1,pz1)
        st2t = points(fp,st2,px2,py2,pz2)
741     st = points(fp,st2t,px2t,py2t,pz2t)
        # st is again defined as 1 for free surf
743     st5 =1
        st = points(fp,st5,px5,py5,pz5)
745     # pad arrays
        lpad4 = np.zeros((np.shape(p4)[0],2))
747     vpad4 = np.zeros((2,np.shape(px4)[1]))

749     lpad3 = np.zeros((np.shape(p3)[0],2))
```

```
    vpad3 = np.zeros((2,np.shape(p3)[1]))

    lpad2 = np.zeros((np.shape(p2)[0],2))
    vpad2 = np.zeros((2,np.shape(p2)[1]))

    lpad2t = np.zeros((np.shape(p2t)[0],2))
    vpad2t = np.zeros((2,np.shape(p2t)[1]))

    lpad1 = np.zeros((np.shape(p1)[0],2))
    vpad1 = np.zeros((2,np.shape(p1)[1]))

    lpad5 = np.zeros((np.shape(p5)[0],2))
    vpad5 = np.zeros((2,np.shape(p5)[1]))

    # nhullpanels
    panel(fp,p4,st4,lpad4,vpad4)
    panel(fp,p3,st3,lpad3,vpad3)
    panel(fp,p1,st1,lpad1,vpad1)
    panel(fp,p2,st2,lpad2,vpad2)
    tripanel(fp,p2t,st2t,lpad2t,vpad2t)
    panel(fp,p5,st5,lpad5,vpad5)

    fp.close()

def hull(fp,origin,pst,v1,v2,x,y,z,v1_15,v2_15,v3_15,v1_16,v2_16,v3_16,v4_16):
    """
    INPUT:
    fp = input pan file
    origin = origin for all the patches (Global)
    pst = previous panel array(output form last panel list);
        arrat([0]) for first hull patch
```

```
781     v1 = ID of the patch
        v2 = 0 for no water line, 1 for water line
783     x = x ordinates of i th panel
        y = y ordinates of i th panel
785     z = z ordinates of i th panel
        v1_15 = ?? ID of waterline this patch may have ??
787     v2_15 = ?? ID of hull patch this WL belongs to??
        v3_15 = 0 for WL on port side of hull; 1 for WL on Stbd side
789     v1_16 = index of first panel of WL
        v2_16 = index of last panel of WL
791     v3_16 = index of first point of WL
        v4_16 = index of last point of WL
793
        OUTPUT
795     panel list
        """
797     h = np.shape(x)[0]
        v = np.shape(x)[1]
799
        p = np.arange(1,1+ (h-1)*(v-1)).reshape(h-1,v-1)
801     # lines 12 to 16
        fp.write(" hull          %i  %i\n  %f %f %f\n  %i %i %i %i\n  \
803 %i %i %i\n  %i %i %i %i\n"% (v1,v2,origin[0],origin[1],origin[2],\
                                  pst.max()+ p[0,0],pst.max()+ p[h-2,v-2],\
805                               v-1,h-1,v1_15,v2_15,v3_15,v1_16,v2_16,v3_16,v4_16
                                      ))
        return p + pst.max()
807
    def freesurf(fp,origin,pst,v1,x,y,z,v1_20,v2_20,v3_20,v1_21,v2_21,v3_21,v4_21)
        :
809     """
```

```
    INPUT:
811 fp = input pan file
    origin = origin for all the patches (Global)
813 pst = previous panel array(output form last panel list);
        arrat([0]) for first hull patch
815 v1 = ID of the patch
    x = x ordinates of i th panel
817 y = y ordinates of i th panel
    z = z ordinates of i th panel
819 v1_20 = ?? ID of waterline this patch may have ??
    v2_20 = ?? ID of hull patch this WL belongs to??
821 v3_20 = 0 for WL on port side of hull; 1 for WL on Stbd side
    v1_21 = index of first panel of WL
823 v2_22 = index of last panel of WL
    v3_23 = index of first point of WL
825 v4_24 = index of last point of WL

827 OUTPUT
    panel list
829 """
    h = np.shape(x)[0]
831 v = np.shape(x)[1]

833 v3_15 = 0
    p = np.arange(1,1+ (h-1)*(v-1)).reshape(h-1,v-1)
835 # lines 17 to 21
    fp.write(" fs          %i\n  %f %f %f\n  %i %i %i %i\n  \
837 %i %i %i\n  %i %i %i %i\n"% (v1,origin[0],origin[1],origin[2],\
                                  pst+ p[0,0],pst+ p[h-2,v-2],\
839                               v-1,h-1,v1_20,v2_20,v3_20,v1_21,v2_21,v3_21,v4_21
                                  ))
```

```
    return p + pst.max()

def points(fp,st,x,y,z):
    """
    fp : file for writing pan file
    st : starting point of number of points; initially it is 1
    x,y,z : offsets

    output:
    i : next point in global points set
    """
    x = x.flatten()
    y = y.flatten()
    z = z.flatten()
    for j in range(len(x)):
        fp.write("%f %f %f %i\n" %(x[j], y[j], z[j], st+j))

    st = st+j+1

    return st

def panel(fp,p,st,lpad,vpad):
    """
    fp : file for writing pan file
    x : offsets
    lpad: contains longitudinal panel set of adjacent patches
    vpad: contains vertical or transverse(for free surface) set of panels of
        adjacent patches
    p: panel number file
    k: start index
    """
```

```
    #correction for the points list
    st = st-p[0,0]+1
    for i in range(np.shape(p)[0]):
        for j in range(np.shape(p)[1]):
            if i == 0:
                if j == 0:
                    v6 = lpad[i,0]
                    v7 = p[i,j+1]
                    v8 = p[i+1,j]
                    v9 = vpad[0,j]
                elif j == np.shape(p)[1]-1:
                    v6 = p[i,j-1]
                    v7 = lpad[i,1]
                    v8 = p[i+1,j]
                    v9 = vpad[0,j]
                else:
                    v6 = p[i,j-1]
                    v7 = p[i,j+1]
                    v8 = p[i+1,j]
                    v9 = vpad[0,j]
            elif i == np.shape(p)[0]-1:
                if j == 0:
                    v6 = lpad[i,0]
                    v7 = p[i,j+1]
                    v8 = vpad[1,j]
                    v9 = p[i-1,j]
                elif j == np.shape(p)[1]-1:
                    v6 = p[i,j-1]
                    v7 = lpad[i,1]
                    v8 = vpad[1,j]
                    v9 = p[i-1,j]
```

```
                else:
903                 v6 = p[i,j-1]
                    v7 = p[i,j+1]
905                 v8 = vpad[1,j]
                    v9 = p[i-1,j]
907             else:
                    if j == 0:
909                     v6 = lpad[i,0]
                        v7 = p[i,j+1]
911                     v8 = p[i+1,j]
                        v9 = p[i-1,j]
913                 elif j == np.shape(p)[1]-1:
                        v6 = p[i,j-1]
915                     v7 = lpad[i,1]
                        v8 = p[i+1,j]
917                     v9 = p[i-1,j]
                    else:
919                     v6 = p[i,j-1]
                        v7 = p[i,j+1]
921                     v8 = p[i+1,j]
                        v9 = p[i-1,j]
923
                fp.write("%i %i %i %i %i %i %i %i %i\n" %\
925                      (p[i,j]+i+st-1,p[i,j]+np.shape(p)[1]+st+i,\
                       p[i,j]+np.shape(p)[1]+st+1+i,\
927                    p[i,j]+st+i,p[i,j],v6,v7,v8,v9))


929 def tripanel(fp,p,st,lpad,vpad):
    """
931 fp : file for writing pan file
    x : offsets
```

```
    lpad: contains longitudinal panel set of adjacent patches
    vpad: contains vertical or transverse(for free surface) set of panels of
        adjacent patches
    p: panel number file
    k: start index
    """
    #correction for the points list
    st = st-p[0,0]+1
    for i in range(np.shape(p)[0]):
        for j in range(np.shape(p)[1]):
            if i == 0:
                if j == 0:
                    v6 = lpad[i,0]
                    v7 = lpad[i,1]
                    v8 = p[i+1,j]
                    v9 = vpad[0,j]
                elif j == np.shape(p)[1]-1:
                    v6 = lpad[i,0]
                    v7 = lpad[i,1]
                    v8 = vpad[1,j]
                    v9 = p[i-1,j]
                else:
                    v6 = lpad[i,0]
                    v7 = lpad[i,1]
                    v8 = p[i+1,j]
                    v9 = p[i-1,j]
            fp.write("%i %i %i %i %i %i %i %i %i\n" %\
                     (p[i,j]+i+st-1,p[i,j]+np.shape(p)[1]+st+i,\
                      p[i,j]+np.shape(p)[1]+st+1+i,\
                      p[i,j]+st+i,p[i,j],v6,v7,v8,v9))

```

```
def lcbrange(LCB, L):

    if ((-0.03*L <= LCB)&(LCB <= 0.03*L)):
        g = 0.
    else:
        # constraint is violated
        print 'exceeded'
        g = abs(LCB) - 0.03

    return g


## Nelder-Mead simplex algorithm
## Original routine is from scipy/optimize/optimize.py version 0.7 by
## Travis E. Oliphant
## last update: 090906, lb
import numpy
from numpy import atleast_1d, eye, mgrid, argmin, zeros, shape, empty, \
     squeeze, vectorize, asarray, absolute, sqrt, Inf, asfarray, isinf

# helper function
def wrap_function(function, args):
    ncalls = [0]
    def function_wrapper(x):
        ncalls[0] += 1
        return function(x, *args)
    return ncalls, function_wrapper


def nmsimplex(func, x0,nonzdelt, zdelt, xtol=1e-4, args=(), ftol=1e-4, \
              maxiter=None, maxfun=None,
```

```
995         full_output=0, disp=1, retall=0, callback=None):
    """Minimize a function using the downhill simplex algorithm.
997
    :Parameters:
999
      func : callable func(x,*args)
1001      The objective function to be minimized.
      x0 : ndarray
1003      Initial guess.
      args : tuple
1005      Extra arguments passed to func, i.e. ``f(x,*args)``.
      callback : callable
1007      Called after each iteration, as callback(xk), where xk is the
          current parameter vector.
1009
    :Returns: (xopt, {fopt, iter, funcalls, warnflag})
1011
      xopt : ndarray
1013      Parameter that minimizes function.
      fopt : float
1015      Value of function at minimum: ``fopt = func(xopt)``.
      iter : int
1017      Number of iterations performed.
      funcalls : int
1019      Number of function calls made.
      warnflag : int
1021      1 : Maximum number of function evaluations made.
          2 : Maximum number of iterations reached.
1023  allvecs : list
          Solution at each iteration.
1025
```

```
    *Other Parameters*:

      xtol : float
          Relative error in xopt acceptable for convergence.
      ftol : number
          Relative error in func(xopt) acceptable for convergence.
      maxiter : int
          Maximum number of iterations to perform.
      maxfun : number
          Maximum number of function evaluations to make.
      full_output : bool
          Set to True if fval and warnflag outputs are desired.
      disp : bool
          Set to True to print convergence messages.
      retall : bool
          Set to True to return list of solutions at each iteration.

    :Notes:

        Uses a Nelder-Mead simplex algorithm to find the minimum of
        function of one or more variables.

    """
    fcalls, func = wrap_function(func, args)
    x0 = asfarray(x0).flatten()
    N = len(x0)
    rank = len(x0.shape)
    if not -1 < rank < 2:
        raise ValueError, "Initial guess must be a scalar or rank-1 sequence."
    if maxiter is None:
        maxiter = N * 200
```

```
1057        if maxfun is None:
                maxfun = N * 200
1059
            rho = 1; chi = 2; psi = 0.5; sigma = 0.5;
1061        one2np1 = range(1,N+1)

1063        if rank == 0:
                sim = numpy.zeros((N+1,), dtype=x0.dtype)
1065        else:
                sim = numpy.zeros((N+1,N), dtype=x0.dtype)
1067        fsim = numpy.zeros((N+1,), float)
            sim[0] = x0
1069        if retall:
                allvecs = [sim[0]]
1071
            # The initial design is evaluated
1073        fsim[0] = func(x0)

1075    ##      nonzdelt = 0.05    # default value for creating initial simplex
        ##      zdelt = 0.00025    # default value if x_k==0
1077
            # Loop over N additional design for initial simplex
1079        for k in range(0,N):
                y = numpy.array(x0,copy=True)
1081            if y[k] != 0:

1083                y[k] = (1+nonzdelt)*y[k]
                else:
1085
                    y[k] = zdelt
1087
```

```
            # Add new design to simplex and evaluate it
1089        sim[k+1] = y
            f = func(y)
1091        fsim[k+1] = f

1093    ind = numpy.argsort(fsim)
        fsim = numpy.take(fsim,ind,0)
1095    # sort so sim[0,:] has the lowest function value
        sim = numpy.take(sim,ind,0)
1097
        iterations = 1
1099
        while (fcalls[0] < maxfun and iterations < maxiter):
1101        if (max(numpy.ravel(abs(sim[1:]-sim[0]))) <= xtol \
                and max(abs(fsim[0]-fsim[1:])) <= ftol):
1103            break

1105        xbar = numpy.add.reduce(sim[:-1],0) / N
            xr = (1+rho)*xbar - rho*sim[-1]
1107        fxr = func(xr)
            doshrink = 0
1109
            if fxr < fsim[0]:
1111            xe = (1+rho*chi)*xbar - rho*chi*sim[-1]
                fxe = func(xe)
1113
                if fxe < fxr:
1115                sim[-1] = xe
                    fsim[-1] = fxe
1117            else:
                    sim[-1] = xr
```

```
                fsim[-1] = fxr
            else: # fsim[0] <= fxr
                if fxr < fsim[-2]:
                    sim[-1] = xr
                    fsim[-1] = fxr
                else: # fxr >= fsim[-2]
                    # Perform contraction
                    if fxr < fsim[-1]:
                        xc = (1+psi*rho)*xbar - psi*rho*sim[-1]
                        fxc = func(xc)

                        if fxc <= fxr:
                            sim[-1] = xc
                            fsim[-1] = fxc
                        else:
                            doshrink=1
                    else:
                        # Perform an inside contraction
                        xcc = (1-psi)*xbar + psi*sim[-1]
                        fxcc = func(xcc)

                        if fxcc < fsim[-1]:
                            sim[-1] = xcc
                            fsim[-1] = fxcc
                        else:
                            doshrink = 1

                    if doshrink:
                        for j in one2np1:
                            sim[j] = sim[0] + sigma*(sim[j] - sim[0])
                            fsim[j] = func(sim[j])
```

```
1151            ind = numpy.argsort(fsim)
                sim = numpy.take(sim,ind,0)
1153            fsim = numpy.take(fsim,ind,0)
                if callback is not None:
1155                callback(sim[0])
                iterations += 1
1157            if retall:
                    allvecs.append(sim[0])
1159
        x = sim[0]
1161    fval = min(fsim)
        warnflag = 0
1163
        if fcalls[0] >= maxfun:
1165            warnflag = 1
                if disp:
1167                print "Warning: Maximum number of function evaluations has "\
                          "been exceeded."
1169    elif iterations >= maxiter:
            warnflag = 2
1171        if disp:
                print "Warning: Maximum number of iterations has been exceeded"
1173    else:
            if disp:
1175            print "Optimization terminated successfully."
                print "         Current function value: %f" % fval
1177            print "         Iterations: %d" % iterations
                print "         Function evaluations: %d" % fcalls[0]
1179
```

```
1181        if full_output:
                retlist = x, fval, iterations, fcalls[0], warnflag
1183            if retall:
                    retlist += (allvecs,)
1185        else:
                retlist = x
1187            if retall:
                    retlist = (x, allvecs)
1189
            return retlist
1191



1193
    if __name__ == "__main__":
1195
            Vsa = np.array(range(15,34))
1197        alphaa = np.zeros(len(Vsa))
            alphaa[:] = 0.8
1199        for i in range(len(Vsa)):
                Vs = Vsa[i]
1201            alpha = alphaa[i]


1203            Lpp = 230.              # Lenght between perpendiculars [m]
                Lwl = 233.58          # Lenght on Load water line [m]
1205            B =  32.2               # Breadth[m]
                T = 10.8               # Draft [m]
1207            Cb = 0.6505            # Block Coefficient of Fineness
                Pf = 0.                 # extent of forward paralled middle body
1209            Pa = 0.                # extent of aft paralled middle body
                v = 11           # number of panels girthwise
1211            vl = 6           # number of stern tube panels girthwise
```

```
        vu = v - vl     # number of transom patch panels girthwise
        v_fs = 20       # number of free surface panels width wise
        # fspan = Number of panels for one lenght [even number]
        fspan = 30  # number of free surface points  per ship length

        # number of panels on each patch (length wise)
        npan1 = 5
        npan2 = 8
        npan3 = 44
        npan4 = 6

        delta_s = 0.015 # defines initial Free surface panel width
        w_beta = 1.05 #defines rate of increase in free surface panel widths
            sideways
        # factor for calculating the freeboard, 0.5 to 0.7
    ##    alpha = 0.2
        # beta = alpha / no of panels length wise
        beta = alpha / (npan1+npan2+npan3)
        # end sealing
        # stern boss closing point
        sx = 5.405
        sy = 0.
        sz = 4.1
        #transom closing
        tx = -2.43
        ty =  0.0
        tz = 10.80
        #bulb closing
        bx = 237.083
        by = 0.
        bz = 5.876
```

```
1243        Atp = 1.0
            Ftp = 1.0
1245
            # pan file name:
1247        filename = 'hull_SFB.pan'

1249        # Call the function to read the offsets and Station spacing
            xlong, offsets,yst,zst, Across,Girth = readoffsets("kcs_body.txt",T)
1251        sta = range(0,57)
            origin = [0,0,0] # Global origin for all the points and patches
1253        # Generation of hull patch offset data
            x1,y1,z1 = patch(xlong,offsets,sta[6:12],1,vl,vl,\
1255                         vl,vu,sx,sy,sz,tx,ty,tz,bx,by,bz,beta,Lpp,T)
            x2,y2,z2 = patch(xlong,offsets,sta[2:12],2,vu,vu +1,\
1257                         vl,vu,sx,sy,sz,tx,ty,tz,bx,by,bz,beta,Lpp,T)
            x3,y3,z3 = patch(xlong,offsets,sta[11:49],3,v,v+1,\
1259                         vl,vu,sx,sy,sz,tx,ty,tz,bx,by,bz,beta,Lpp,T)
            x4,y4,z4 = patch(xlong,offsets,sta[48:57],4,vl,vl,\
1261                         vl,vu,sx,sy,sz,tx,ty,tz,bx,by,bz,beta,Lpp,T)

1263        # WL aft most point, WL fwd most point
            WL = [-2.43,233.58]
1265
            # Point of fwd of Bulbous bow
1267        xBlb = 237.083

1269        #Transom Patch
            zone1 = [WL[0],xBlb,x1[0,-1],x1[0,0]]
1271        zone2 = [WL[0],xBlb,x2[0,-1],x2[0,0]]
            zone3 = [WL[0],xBlb,x3[0,-1],x3[0,0]]
```

```
        zone4 = [WL[0],xBlb,x4[0,-1],x4[0,0]]

        pandist = np.array([[-1.0,0.,1.0],\
                            [0.1,1.0,0.1]])


        nx1,ny1,nz1 = stainterp(xlong,zone1,pandist,npan1,x1,y1,z1)
        nx2,ny2,nz2 = stainterp(xlong,zone2,pandist,npan2,x2,y2,z2)
        nx3,ny3,nz3 = stainterp(xlong,zone3,pandist,npan3,x3,y3,z3)
        nx4,ny4,nz4 = stainterp(xlong,zone4,pandist,npan4,x4,y4,z4)

        desc = 0
        if np.any(np.diff(nx3[0])<0):
            desc = 1
            pos = np.searchsorted(nx2[0][::-1],-.8)

            # for aft triangular patch
            nx2t = np.array([[nx2[0,len(nx2[0])-pos],nx2[0,-1]],\
                    [nx2[-2,len(nx2[0])-pos],nx2[-2,-1]],\
                    [nx2[-1,len(nx2[0])-pos],nx2[-1,-1]]])
            ny2t = np.array([[ny2[0,len(nx2[0])-pos],ny2[0,-1]],\
                    [ny2[-2,len(nx2[0])-pos],ny2[-2,-1]],\
                    [ny2[-1,len(nx2[0])-pos],ny2[-1,-1]]])
            nz2t = np.array([[nz2[0,len(nx2[0])-pos],nz2[0,-1]],\
                    [nz2[-2,len(nz2[0])-pos],nz2[-2,-1]],\
                    [nz2[-1,len(nx2[0])-pos],nz2[-1,-1]]])


            nx2 = nx2[:,:1-pos]
            ny2 = ny2[:,:1-pos]
            nz2 = nz2[:,:1-pos]
```

```
        else:
1305        pos = np.searchsorted(nx2[0],-.8)
            nx2 = nx2[:,pos:]
1307        ny2 = ny2[:,pos:]
            nz2 = nz2[:,pos:]
1309    # Correction for Free board straight panel half breadths
        # Correction by Dr. Birk on 18 Nov 09
1311    ny2t[-1] = ny2t[-2]
        ny2[-1] = ny2[-2]
1313    ny3[-1] = ny3[-2]
        # Waterline panel points
1315    nxwl = np.append(nx3[-1],nx2[-1][1:])
        nxwl = np.append(nxwl,nx2t[-1][1:])
1317    nywl = np.append(ny3[-1],ny2[-1][1:])
        nywl = np.append(nywl,ny2t[-1][1:])
1319
        nx5,ny5,nz5 = patchfs(nxwl,nywl,fspan,v_fs,T,Lpp,Lwl,delta_s,beta,
            w_beta)
1321
        ###  initial guess of change in LCB
1323    x0 = -0.05

1325    # Controlling the size of the Initial simplex
        nonzdelt = 0.05
1327    zdelt = 0.00025
        xtol = 1.e-5
1329    # control file for panel code
        fp = open('nlhsfsctrl.in','w')
1331    fp.close()

1333    fp = open('nlhsfsctrl.in','a')
```

```
        fp.write('# This is a controlfile for hsfs\n')
1335    fp.write(filename)
        fp.write('\nflow.dat\n')
1337    fp.write('sigma.vtp\n')
        fp.write('hulls\n')
1339    fp.write('hullsm.vtp\n')
        fp.write('hullv.vtu\n')
1341    fp.write('fss\n')
        fp.write('fssm.vtp\n')
1343    fp.write('waveprof\n')
        fp.write('1      # linear free surface;    double body flow=0\n')
1345    fp.write('0\n')
        fp.write('0.8\n')
1347    fp.write('%f #Froude number (design speed)\n'% (Vs*0.5144/(9.81*Lpp)
            **0.5))
        fp.close()
1349
        fp = open("optimization"+str(Vs)+'_'+str(alpha)+".dat", 'w')
1351    fp.close()

1353 ##       resistance(-0.033056478)
        nmsimplex(resistance,x0,nonzdelt,zdelt,xtol)
1355
        fp = open("optimization"+str(Vs)+'_'+str(alpha)+".dat", 'r')
1357    lines = fp.readlines()
        fp.close()
1359
        nLCB = []
1361    nR =  []

1363    for line in lines:
```

```
            nLCB.append(float(line.split()[0]))
            nR.append(float(line.split()[1]))

        nLCB = np.asarray(nLCB)
        nR = np.asarray(nR)

##          leg = plt.scatter(nLCB,nR)
##          for i in range(len(nLCB)):
##              ilabel = str(i)
##              plt.text(nLCB[i], nR[i], ilabel)
##          plt.plot(nLCB,nR, '.')
##          plt.title('Speed = '+str(Vs)+' Knots')
##          plt.xlabel(r'Position of LCB from midships')
##          plt.ylabel(r'Coefficient of Wavemaking Resitance')
##
##          plt.text(max(nLCB),max(nR),'Cw = '+(str(min(nR))[:10])+\
##                  '\nLCB = '+(str(50*nLCB[np.argmin(nR)])[:10]),\
##            verticalalignment='top', horizontalalignment='right', fontsize=15)
##          plt.grid()
##          plt.savefig(str(Vs)+'knots_'+str(alpha)+'alpha.png')
##          plt.close()

        fp = open('results.dat','a')
        fp.write('%f %f %12.12f %12.12f\n'%(Vs,alpha,nLCB[np.argmin(nR)],min(
            nR)))
        fp.close()
```

Optimizer.py

Vita

Krishna Murthy Karri, born in Hyderabad, India, graduated in May 2008 from Andhra University College of Engineering, Visakhapatnam, India, with a Bachelors (Honors) degree in Naval Architecture. Immediately upon completion of the undergraduate degree, he enrolled in the graduate program at the University of New Orleans, where he is currently a candidate for Master of Science degree in Naval Architecture and Marine Engineering. He is currently working as an intern in Technology Associates INC, New Orleans, and after successful completion of Masters degree he will work as a Naval Architect.