12-20-2009

Interactive Optimization Programs for Initial Propeller Design

Richard Biven

University of New Orleans

Interactive Optimization Programs for Initial Propeller Design

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

Richard P. Biven

B.S. Christian Brothers University (2006)

December 2009

Dedicated to my parents, whose unyielding belief and faith has lead me to places I could only imagine.

Acknowledgements

I would like to thank Dr. William Vorus and Dr. Lothar Birk who have supported me throughout my thesis with patience and knowledge whilst allowing me the room to work in my own way.

Contents

List of Figures

List of Tables

Abbreviations

AMT	**A**nnular **M**omemtum **T**heory
BET	**B**lade **E**lement **T**heory
CV	**C**ontrol **V**olume
GUI	**G**raphic **U**ser **I**nterface
NACA	**N**ational **A**dvisory **C**ommittee for **A**eronautics
NAME	**N**aval **A**rchitecture & **M**arine **E**ngineering
NM	**N**elder - **M**ead
SUMT	**S**equential **U**nconstrained **M**inimization **T**echnique

Symbols

A_D	Developed area of blades	ft^2
A_E	Propeller expanded area	ft^2
A_o	Propeller disk area	ft^2
BHP	Brake horsepower	HP
$\frac{C(r)}{r}$	Chord distribution	-
C_d	Sectional drag coefficient	-
C_{P_d}	Total power coefficient	-
$C_{P_{di}}$	Ideal power coefficient	-
C_{pv}	Viscous power coefficient	-
$C_T(r)$	Total thrust coefficient	-
$C_{Ti}(r)$	Ideal thrust coefficient	-
$C_{Tv}(r)$	Viscous thrust coefficient	-
D	Propeller diameter	ft
DHP	Delivered horsepower	HP
EAR	Expanded area ratio	-
$ERPM$	Engine revolutions per minute	$\frac{rev}{min}$
f_{max}	Maximum camber	ft
GR	Gear ratio	-
h	Distance to propeller shaft centerline	ft
J	Advance coefficient	-
K_T	Thrust coefficient (B-series)	-
K_Q	Torque coefficient (B-series)	-
$nrpm$	Operating revolutions per minute	$\frac{rev}{min}$

$\frac{P(r)}{D}$	Segmental pitch ratio	-
P_a	Absorbed power	HP
PD	Pitch ratio	-
$PRPM$	Propeller revolutions per minute	$\frac{rev}{min}$
Q	Open water propeller torque	$\frac{lbs}{ft}$
Q_s	Source strength	$\frac{ft^3}{s}$
r	Segmental radius	ft
R	Radius	ft
r_h	Hub radius	ft
Re	Reynolds number	-
R_F	Frictional resistance	lbs
R_T	Total resistance	lbs
s	Slip ratio	-
t_{max}	Maximum propeller thickness	ft
$\frac{t_r}{D}$	Radial blade thickness per diameter	-
T	Open water propeller thrust	lbs
$Temp$	Temperature	^oF
$\frac{t}{c}$	Blade thickness per length radial chord	-
t	Thrust deduction	-
$TBR_a(r)$	Tan beta ratio	-
U	Relative ship velocity	ft/s
u	Axial disturbance velocity	ft/s
$\frac{u_i}{U}$	Axial induced velocity ratio	-
V	Fluid speed	ft/s
V_A	Speed of advance	ft/s
$\frac{V_a}{U}$	Wake velocity distribution	-
$\frac{v_i}{U}$	Tangential induced velocity ratio	-
$\frac{V_r}{U}$	Resulting wake velocity distribution	-
w	Wave fraction	-
Z	Number of blades	-

β_a	Hydrodynamic advance angle	$degrees$
β_i	Hydrodynamic pitch angle	$degrees$
ϵ	Tolerance	-
η_R	Relative rotative efficiency	-
η_T	Transmission efficiency	-
η_o	Open water efficiency	-
ρ	Density	$\frac{lbs-sec^2}{ft^4}$
Ω	Angular frequency	rad/s

Abstract

This thesis presents two methods for initial design propeller optimization using constrained non-linear programming. The process uses the Nelder-Mead simplex algorithm. The Wageningen B-series optimal propeller selection is presented along with the combined annular momentum theory and blade element theory optimization. Both techniques require preliminary hull and engine design characteristics, but do not necessitate extensive background knowledge of propellers and their calculations. A comparison of the two methods shows the combined annular momentum theory and blade element theory optimization produces the more efficient propeller. The optimization programs were designed with a graphic user interface implemented in the programming language Python.

Keywords: Propeller, Optimization, Wageningen, Nelder-Mead

Chapter 1

Introduction

In the marine industry, the selection of propulsors is an extremely important and difficult task. This report overviews the basic mathematics and analysis behind two methods of propeller design:

1. Wageningen B-series propeller optimization
2. Combined annular momentum and blade element theory propeller optimization

Usually in the preliminary design stages some thrust/speed relation representing the hull along with some power/revolution per minute (RPM) relation representing the engine is available. Therefore, the goal of the engineer is to simultaneously match the hull/thrust/speed characteristics and the engine power/RPM characteristics. This matching process is the basis for the optimization.

Each method is optimized using the Nelder-Mead optimization algorithm. It is necessary to begin with a look at the optimization technique and how it calculates an optimum value. Following the optimization, this thesis will give a background explanation of how each method calculates thrust and efficiency for propellers. Once a basic understanding of how each methods' equations and the optimizer work together, a brief description is given about how the results from each analysis compare.

1.1 Programming

One of the purposes of this study is the generation of a graphic user interface (GUI) for the propeller optimizing methods. A brief discussion of the programming tools used are discussed below.

1.1.1 Python, wxPython, NumPy, and SciPy

Python [4] is an object-oriented, interpreted, and interactive programming language. Other programming languages with optimization packages (Nelder-Mead algorithm) and GUI capabilities can be used, but Python is the preferred language for this thesis.

Python uses two extra computational packages, NumPy [5] and SciPy [6]. NumPy is the fundamental package for scientific computing in Python, whereas SciPy is an additional software package for mathematics, science, and engineering. The SciPy library is built to work with NumPy to provide many user-friendly and efficient numerical routines, such as those for numerical integration and optimization. SciPy has a Nelder-Mead optimization function called *FMIN*. *FMIN* follows the minimization technique discussed in Section 2.2.

In order to create the GUI, wxPython was used. wxPython [7] is a GUI toolkit for the Python programming language.

Chapter 2

Nelder-Mead Optimization

2.1 Nelder-Mead Method

As Wolff [8] describes, the Nelder-Mead (NM) method, also known as the downhill simplex method, is an optimization algorithm named after statisticians, John Nelder and Roger Mead. It is a numerical method for minimizing an objective function in multi-dimensional space. The NM method is one of the most commonly used nonlinear optimization algorithms.

The NM method is simple, intuitive, and fairly stable. It approaches the optimum in great steps initially, followed by smaller incremental steps. The initial step of the NM method is the construction of a simplex. An m-simplex is an m-dimensional convex hull polytope, where $m+1$ is the number of vertices. "A polytope is a two-dimensional polygon or three-dimensional polyhedron, or any of the various generalizations thereof" [9].

The Nelder-Mead method uses three construction principles to determine a new point. For a simplex with $m+1$ vertices and $x^0, \ldots, x^m$,

$$s^j = \frac{1}{m} \sum_{\substack{i=0 \\ i \neq j}}^{m} x^i \tag{2.1}$$

s^j denotes the center of gravity with respect to x^i. x^i are the initial guesses for the algorithm. These initial guesses will be discussed later in 3.3.

The three construction principles used in the NM method:

1. Reflection
2. Expansion
3. Contraction
   (a) Partial interior contraction
   (b) Partial exterior contraction
   (c) Total contraction or shrink

Reflection uses the corner x^j at s^j to determine a new point

$$x^r = s^j + r(s^j - x^j) \tag{2.2}$$

where the reflection constant $0 < r \leq 1$.

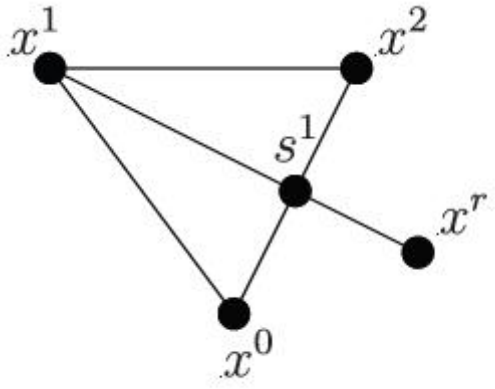

FIGURE 2.1: Reflection ($r = 1/2$)

Expansion uses the reflection point, x^r, in the direction $s^j - x^j$ to determine the new point

$$x^e = s^j + \gamma(x^r - s^j) \tag{2.3}$$

where the expansion constant $\gamma \leq 1$.

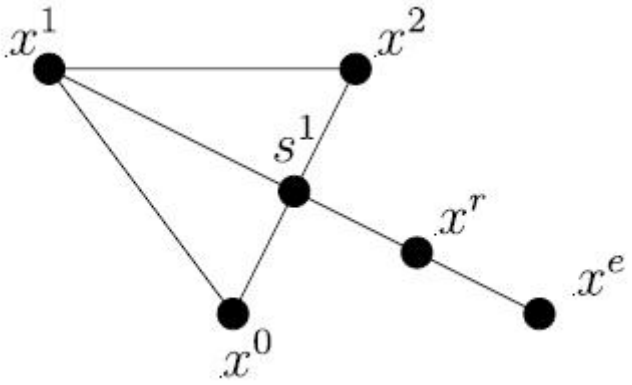

FIGURE 2.2: Expansion ($\gamma = 1$)

Contraction is distinguished into three different types and $0 < \beta \leq 1$ denotes the contraction constant.

1. Partial interior contraction uses x^j in the direction of $s^j - x^j$ to determine the new point from

$$x^c = s^j + \beta(x^j - s^j) \tag{2.4}$$

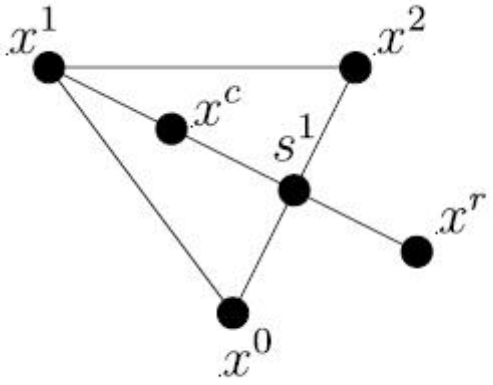

FIGURE 2.3: Partial Interior Contraction ($\beta = 1/2$)

2. Partial exterior contraction uses the reflection point, x^r, in the direction of $s^j - x^r$ to determine a new point

$$x^c = s^j + \beta(x^r - s^j) \tag{2.5}$$

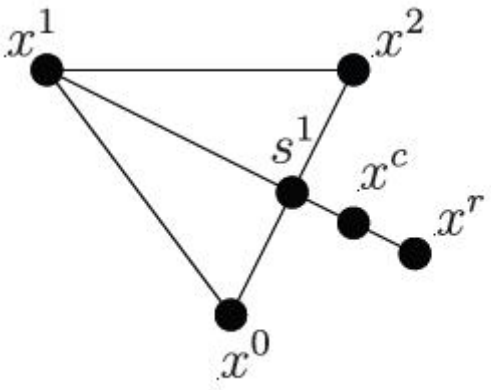

FIGURE 2.4: NM Partial Exterior Contraction ($\beta = 1/2$)

3. Total contraction uses x^j to replace all points

$$x^i = \frac{1}{2}(x^i - x^j) \tag{2.6}$$

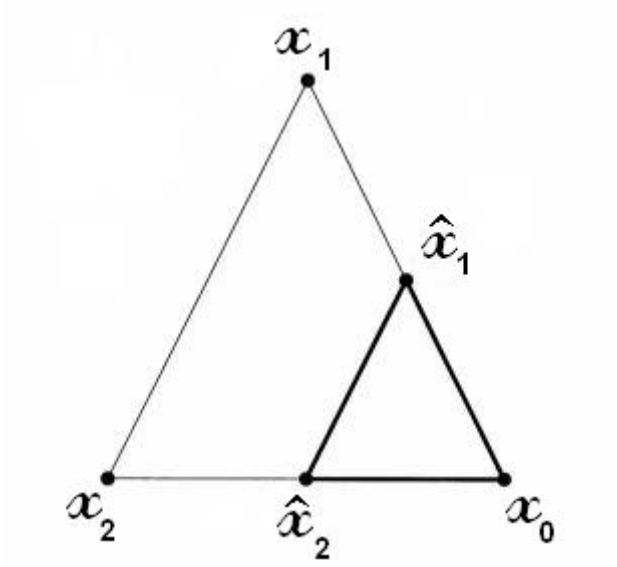

FIGURE 2.5: Total Contraction

2.2 Local, Bounded Nelder-Mead Algorithm

The Nelder-Mead principles are systematically used to minimized the objective function.

1. Start by selecting an initial point, $x^{(0,i)}$, and determining the vertices of the initial simplex.

2. From the simplex, determine the simplex point with the highest function value $x^{(k,h)}$, with the second largest function value $x^{(k,s)}$, and the vertex with the smallest function value $x^{(k,l)}$. The centroid, $s^{(k,m)}$, is also determined.

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{m+1}) \tag{2.7}$$

3. Apply a reflection point with respect to the smallest simplex function value, $x^{(k,l)}$.

$$x^r = x^{(k,s)} + r\left(s^{(k,m)} - x^{(k,l)}\right) \tag{2.8}$$

   Determine if the coordinates of the reflection are feasible, or within the constraints. If the coordinate is outside the constraints, a penalty is imposed[1].

4. After x^r is calculated, three cases are distinguished.

   (a) If the reflection created a new minimum, $f(x^r) < f^{(k,l)}$ (where the notation: $f\left(x^{(k,l)}\right) = f^{(k,l)}$), an even better function value is attempted through expansion of x^r in the direction of $x^r - s^{(k,m)}$

$$x^e = s^{(k,m)} + \gamma\left(x^r - s^{(k,m)}\right) \tag{2.9}$$

   If x^e is projected into an infeasible region, x^e is projected back onto the feasibility boundary with an imposed penalty. $x^{(k,h)}$ will be replaced by the smaller function value of x^r and x^e.

$$x^{(k+1,h)} = \begin{cases} x^e, & f(x^e) < f(x^r) \\ x^r, & \text{else} \end{cases}$$

[1]The constraints will be discussed in greater detail in Section 2.3

(b) If $f(x^r) \leq f^{(k,s)}$, x^r will have a better function value than all vertices except $x^{(k,l)}$; and better than $x^{(k,h)}$, which will be replaced.

$$x^{(k+1,h)} = x^r \tag{2.10}$$

(c) If $f(x^r) \geq f^{(k,h)}$, then reflection may have been the wrong selected direction. The opposite direction is attempted by using the interior contraction from $x^{(k,h)}$ in the direction of $s^{(k,m)} - x^{(k,h)}$

$$x^c = s^{(k,m)} + \beta \left(x^{(k,h)} - s^{(k,m)} \right) \tag{2.11}$$

However, if $f(x^r) < f\left(x^{(k,h)}\right)$ the selected direction might be right, but the function may be better if an exterior contraction is performed because all vertices except $x^{(k,h)}$ are better than x^r. An exterior contraction is performed from x^r in the direction of $s^{(k,m)} - x^r$

$$x^c = s^{(k,m)} + \beta \left(x^r - s^{(k,m)} \right) \tag{2.12}$$

The function value, $x^{(k,h)}$, will be replaced by x^c and the previous contraction equation is applied. If $f(x^c) < f\left(x^{(k,h)}\right)$ the resulting point is an improvement and $x^{(k,h)}$ will be replaced.

$$x^{(k+1,h)} = x^c \tag{2.13}$$

If $f(x^c) \geq f(x^{(k,h)})$ then all attempts for improvement failed. A total contraction is applied with respect to $x^{(k,l)}$. For $i \neq l$

$$x^{(k+1,i)} = \frac{1}{2} \left(x^{(k,i)} + x^{(k,l)} \right), i = 0 \ldots n \tag{2.14}$$

5. Continue with next iteration. Set $k = k + 1$

6. The NM simplex method never truly converges to an exact value or point. Therefore, an appropriate termination criteria must be employed to determine convergence. Convergence is achieved by setting the standard deviation of the objective function's simplex points to

a specified tolerance, ϵ.

$$\left(\frac{1}{n+1}\sum_{i=1}^{n} f\left(x^{(k,i)}\right) - \overline{f}_k\right)^{\frac{1}{2}} < \epsilon \tag{2.15}$$

with

$$\overline{f}_k = \frac{1}{n+1}\sum_{j=1}^{n} f\left(x^{(k,j)}\right)$$

Once the tolerance is reach (Equation 2.15 is true) and all constraints are not violated, the optimization is completed.

2.3 Constraints and Penalty Functions

In this report, the NM method is used with penalty functions and constraints. Numerous variations of penalty methods exist, all of which use the general formulations and constructions principles discussed in Section 2.2. For the optimization in this report, a static penalty function is applied as discussed by Smith and Coit [10].

The static penalty function is a simple method that penalizes infeasible solutions by applying a constant penalty to the solutions that violate the objective function's constraints. This is performed by adding a penalized function to the unpenalized objective function. For a problem with m constraints:

$$f_p(x) = f(x) + \sum_{i=1}^{m} C_i \delta_i \tag{2.16}$$

$$\text{where} \begin{cases} \delta_i = 1 & \text{if constraint } i \text{ is violated} \\ \delta_i = 0 & \text{if constraint } i \text{ is satisfied} \end{cases}$$

In (2.16) $f_p(x)$ is the penalized objective function, $f(x)$ is the unpenalized objective function, and C_i is a constant imposed for violations of the i^{th} constraint. The value of C is somewhat arbitrary and determined with a "*trail and error*" process. The value must be large enough to return an infeasible result back to regions of feasibility.

For this study a slightly modified version of the static penalty is used

$$f_p(x) = f(x) + C \sum_{i=1}^{m} X_c^2 \delta_i \tag{2.17}$$

$$\text{where} \begin{cases} \delta_i = 1 & \text{if constraint } i \text{ is violated} \\ \delta_i = 0 & \text{if constraint } i \text{ is satisfied} \end{cases}$$

$$X_c = X_{free} - X_m$$

where X_{free} is a free variable or function of a free variable and X_m is a minimum or maximum constraint value. A free variable is a notation, or place holder, in an expression or equation where calculations or substitutions take place. X_c is squared to verify that the penalty is always positive. Using (2.17) is beneficial because the more a constraint is violated, the larger the penalty. In both equations, (2.16) and (2.17), the penalty is only applied if a constraint is violated, as shown with the variable δ. Therefore, the construction principles of the NM method will run as stated in Section 2.1, unless an infeasible domain is encountered.

The NM optimization is used for both methods discussed in Chapters 3 & 4.

Chapter 3

B-Series Propeller

3.1 Background

For an complete description of the B-series propeller geometries and all other Wageningen propellers series, refer to Kuiper [1].

In 1936 W.P.A van Lammeren, director of the Netherlands Ship Model Basin (NSMB) from 1952 until 1972, published the results of open water tests for a series of five four-bladed propellers. These propellers were designed from Baker's [11] propeller geometry that was found to be very efficient. This group of propellers was designated $A4.40$, in which "A" was the series title, 4 was an indication of the number of blades, and .40 an indication of blade area ratio.

The A-series was susceptible to cavitation due to its airfoil sections towards the blade tip. As an improvement, the B-series was designed with circular segments at the blades' sections' tips, which are more common in modern propellers. At the time the B-series design was based on the knowledge of 90 propellers investigated and studied at the NSMB. The development of the B-series continued from 1936 until 1969. In 1969 a major review of the B-series data was given to van Lammeren, van Manen, and Oosterveld (former directors of NSMB). This review gave the necessary information for the development of polynomials for open water curves using regression analysis [2, 12, 13]. The B-series consists of a total of 120 propellers. The open water curves were later corrected for effects of changing Reynolds number, Re, because the original regression curves were derived with a constant Reynolds number of 2 million.

3.2 Blade Characteristics

The B-series has some common characteristics between each blade in the series, independent of the three characteristics most important to the prediction of thrust and torque: expanded area ratio EAR, pitch ratio PD, or number of blades Z.

3.2.1 Radial Pitch Distribution

The B-series has constant pitch at all radii, except the four-bladed series which has a pitch reduction from $0.5 \cdot R$ to the hub, so that the pitch at the hub is 80% of the pitch at the outer radius. This reduction was created to adapt the propeller to wake distributions around the hub and has negligible effects on the propulsive characteristics. A constant pitch should be accepted when B-series designs are made.

3.2.2 Blade Contour

Blade contours for the B-series are defined for a chord length at a specific radius with

$$c(r) = \frac{K(r) \times D \times EAR}{Z} \tag{3.1}$$

where the constant, $K(r)$, is a function of the radius. This helps in design by defining the blade contours for all area ratios with a single table. The B-series blade contour was modified to create the BB-series. The BB-series assumed the same performance characteristics as the B-series, but used a broader blade tip to enhance the propeller's cavitation characteristics. The open water regression polynomial uses the B-series blade contour table not shown in this thesis. But for design purposes, it is recommended to use the BB-series blade contour values of Table 3.1. The table also gives the skew per radial chord length.

TABLE 3.1: Blade Contour of the BB-series Propellers [1]

r/R	$K(r)$	$skew/c_r$
0.2	1.600	0.081
0.3	1.832	0.084
0.4	2.023	0.080
0.5	2.163	0.070
0.6	2.243	0.052
0.7	2.247	0.024
0.8	2.132	−0.020
0.85	2.005	−0.052
0.9	1.798	−0.098
0.95	1.434	−0.182
0.975	1.220	−0.273

3.2.3 Blade Thickness

The blade thickness for the B-series is defined as

$$\frac{t_r}{D} = A_r - B_r Z \tag{3.2}$$

where A_r and B_r are coefficients that have been tabulated in Table 3.2 as a function of radial position.

TABLE 3.2: Blade Thickness Coefficients [1]

r/R	A_r	B_r
0.2	0.0526	0.0040
0.3	0.0464	0.0035
0.4	0.0402	0.0030
0.5	0.0340	0.0025
0.6	0.0278	0.0020
0.7	0.0216	0.0015
0.8	0.0154	0.0010
0.9	0.0092	0.0005
1.0	0.0030	0.0000

3.2.4 Maximum Camber Distribution

Maximum camber of the B-series sections can be found to be half of the maximum thickness, except at inner radii. A correction for maximum camber at inner radii is tabulated like the blade thickness and contour. The maximum camber, f_{max}, is found by

$$f_{max} = \frac{t_{max}}{2} - K_f \times t_{max} \tag{3.3}$$

TABLE 3.3: Correction for Maximum Camber Calculations [1]

r/R	K_f
0.2	0.330
0.3	0.271
0.4	0.193
0.5	0.101
0.6	0.023

3.2.5 Rake

All B-series propellers have a rake of 15 degrees. This is somewhat high for present day propellers. The propeller's efficiency might increase slightly with a reduction or absence of rake; however, those calculations are outside this report's scope of work.

3.2.6 Hub Diameter

All hub diameters of the B-series are $\frac{1}{6}$ of the diameter, except the three-bladed series which is 18% the diameter. The effects of this difference is considered negligible.

3.2.7 B-Series Sections

The blade geometry at specified radii must be calculated for strength requirements. The blade shape can be determined with the tables for the values of V_1 and V_2 (as shown in Tables A.1 and A.2), which can then be used with following the equations to determine the pressure and

suction side geometries and thickness distributions. The values of V_1 and V_2 are determined with percentage distances from the location of maximum thickness. For the B-Series, the position of maximum thickness is determined with Table 3.4.

TABLE 3.4: Position of Maximum Thickness [1]

r/R	x_{tmax}/c_r
0.2	0.350
0.3	0.350
0.4	0.351
0.5	0.355
0.6	0.389
0.7	0.443
0.8	0.486
0.9	0.500
1.0	0.500

It should be noted that the leading edge thickness of the B-series propellers are close to $0.2 \cdot t_{max}$. The values of the leading and trailing edges are usually sized in accordance with classification societies or with manufacturer's requirements.

For $P \leq 0$

$$y_{face} = V_1(t_{max} - t_{t.e.}) \tag{3.4}$$

$$t_r = (V_1 + V_2)(t_{max} - t_{t.e.}) + t_{t.e.} \tag{3.5}$$

For $P > 0$

$$y_{face} = V_1(t_{max} - t_{l.e.}) \tag{3.6}$$

$$t_r = (V_1 + V_2)(t_{max} - t_{l.e.}) + t_{l.e.} \tag{3.7}$$

Where $P \leq 0$ represents the trailing edge side of the maximum thickness and $P > 0$ represents the leading edge side of the section.

Computation of maximum thickness is described in Section 3.2.3. All parameters for the strength calculations are defined and will be further discussed Chapter 4. The strength calculation are only used in the program *MTPROPOPT.py*.

3.3 Optimum Propeller Selection

In most ship propeller designs, certain information is usually given about the machinery, hull, and desired ship performance. One objective of this thesis is to create a program for selecting the optimum propeller for a set of given design inputs using the Wageningen B-series regression polynomials. Therefore, the program finds the optimal propeller for four basic design tasks:

1. For a given total resistance, propeller diameter, speed - optimize rate of revolution
2. For a given delivered horsepower, propeller diameter, speed - optimize rate of revolution
3. For a given total resistance, rate of revolution, speed - optimize propeller diameter
4. For a given delivered horsepower, rate of revolution, speed - optimize propeller diameter

Presented is the first design problem, the other design problems are solved with similar calculations. The open water propeller characteristics are given in terms of thrust and torque coefficients, K_T and K_Q, and the open water efficiency, η_o.

$$K_T = \frac{T}{\rho n^2 D^4} \tag{3.8}$$

$$K_Q = \frac{Q}{\rho n^2 D^5} \tag{3.9}$$

$$\eta_o = \frac{J K_T}{2\pi K_Q} \tag{3.10}$$

All three of these equations are functions of the same variables. This helps for the specification of free variables.

$$K_T = K_T(J, PD, EAR, Z, Re, \frac{t}{c}) \tag{3.11}$$

$$K_Q = K_Q(J, PD, EAR, Z, Re, \frac{t}{c}) \tag{3.12}$$

$$\eta_o = \eta_o(J, PD, EAR, Z, Re, \frac{t}{c}) \tag{3.13}$$

The coefficient of thrust K_T and coefficient of torque K_Q are calculated using a polynomial expression for two to seven blades at a Reynolds number Re of $2 * 10^6$, as shown in Figure A.3. The correction for Re effects was later published as shown in Figure A.4.

The coefficient of thrust K_T and coefficient of torque K_Q are dependent on six variables: speed of advance J, pitch ratio PD, expanded area ratio EAR, number of blades Z, Reynolds number Re, and blade thickness per chord length $\frac{t}{c}$. The maximum blade thickness per chord length, $\frac{t}{c}$, is varied to give the propeller suitable strength. Therefore, $\frac{t}{c}$ is dependent on blade number and the design coefficients, as shown in Section 3.2.3. The number of blades, Z, is usually not varying for the ship design and propeller selection. Z is selected based on vibrations due to the interaction of hull, shaft, propeller, and wake.

With blade thickness per chord length $\frac{t}{c}$ and blade number Z predetermined, the coefficient of thrust K_T and coefficient of torque K_Q are now changing with their dependence on the four remaining variables: J, PD, EAR, and Re. However, Re is determine by

$$Re = \frac{UC_{0.75}}{\nu} \tag{3.14}$$

where U is the speed of the vessel, $C_{0.75}$ is the chord length at $0.75R$, and ν is the kinematic viscosity. For each design task in the Wageningen optimization, Re incorporates the given input variables and the optimizing result, such as rate of revolution or propeller diameter

$$Re = \frac{C_{0.75} * \sqrt{V_A^2 + (0.75\pi nD)^2}}{\nu} \tag{3.15}$$

The rate of revolution, n (or diameter D, depending on the chosen method), is changing during the Nelder-Mead optimization algorithm; therefore, changing Re. The effects of blade thickness are taken into account in the Re calculation.

Therefore, the coefficients, K_T and K_Q, are changing by the three remaining variables: speed of advance J, expanded area ratio EAR, and pitch ratio PD. These are the free variables used in the optimization. An initial guess for each free variable is coded into the program. These values will setup the initial simplex discussed in Section 2.2. In addition to the 3 free variables, 14 (or 15 for method 2 & 4) static parameters are used as input.

1. Diameter $[D]$ for tasks 1 & 2, Operating rate of revolution $[nrpm]$ for tasks 3 & 4
2. Number of blades $[Z]$
3. Total resistance $[R_T]$ for tasks 1 & 3, Delivered horse power $[DHP]$ for tasks 2 & 4

4. Ship speed $[U]$
5. Wake fraction $[w]$
6. Thrust deduction $[t]$
7. Minimum pitch ratio $[PDmin]$
8. Maximum pitch ratio $[PDmax]$
9. Minimum advance coefficient $[Jmin]$
10. Maximum advance coefficient $[Jmax]$
11. Distance to propeller shaft centerline $[h]$
12. Minimum expanded area ratio $[EARmin]$
13. Maximum expanded area ratio $[EARmax]$
14. Temperature $[Temp]$
15. Relative rotation efficiency $[\eta_R]$ for methods 2 & 4

The temperature, $Temp$, is used to calculate water density, ρ, and kinematic viscosity, ν, using a regression equation [14]. These are all the necessary inputs needed to run *WageningenOpt.py*, as discussed in more detail in Chapter 5. The minimum and maximum input values are used as constraints. Based on the 120 propellers in the Wageningen B-series, the polynomial expressions for the coefficients, K_T and K_Q, are evaluated within the following limits.

$$0.0 < J < 1.8 \tag{3.16}$$

$$0.30 \leq EAR \leq 1.05 \tag{3.17}$$

$$0.50 \leq PD \leq 1.40 \tag{3.18}$$

These limits apply to all designs calculated by *WageningenOpt.py*, not just the 120 propellers in the B-series.

Another constraint used is based on cavitation. In practice B-series propellers are not adequate for the calculations of cavitation, usually a lifting line or lifting surface approach should be used. However, Burrill [15] did extensive research in the 40s and 50s for the necessary expanded area ratio, EAR, to prevent cavitation.

$$EAR = \frac{T}{\frac{1}{2}\rho v_1^2 A_o (1.067 - 0.229PD)\tau_c} \tag{3.19}$$

where τ_c is the thrust loading of the blades, v_1 is the sectional on-flow velocity at the radius $x = 0.7$, and A_o is the propeller disk area.

$$v_1 = \sqrt{V_A^2 + (\pi n 0.7D)^2} \tag{3.20}$$

$$A_o = \frac{\pi D^2}{4} \tag{3.21}$$

$$\tau_c = 0.3 * \sqrt{\sigma_{0.7} - 0.03} \tag{3.22}$$

where $\sigma_{0.7}$ is the cavitation number at $0.7 \cdot R$ and solved with the total pressure, p_0, and the vapor pressure, p_v.

$$\sigma_{0.7} = \frac{p_0 - p_v}{\frac{1}{2}\rho V_A^2} \tag{3.23}$$

Keller [16] proposed a simpler formula to estimate the necessary expanded area ratio EAR

$$EAR = \frac{(1.3 + 0.3Z)T}{(p_0 - p_v)D^2} + k \tag{3.24}$$

where k is a constant dependent on ship type

$k = 0$	for fast twin-screw ships
$k = 0.1$	for other twin-screw ships
$k = 0.2$	single screw, high loading

These two equations are used as constraints for the necessary expanded area ratio EAR.

The final constraint used is the thrust and torque requirements of the ship. The required thrust and torque is calculated to determine a propeller's efficiency for all pitch ratios PD, expanded area ratios EAR, and speeds of advance J within the limits stated above, as shown in Figure

A.5. The equations for the necessary thrust and torque changes slightly for each design task, shown in Table 3.5.

TABLE 3.5: Table for Thrust and Torque Requirements [1]

design task 1	$K_T/J^2 = \frac{R_T}{\rho(1-t)(1-w)^2V^2D^2}$
design task 2	$K_Q/J^3 = \frac{DHP\ \eta_R}{2\pi\rho(1-w)^3V^3D^2}$
design task 3	$K_T/J^4 = \frac{R_T\ n^2}{\rho(1-t)(1-w)^4V^4}$
design task 4	$K_Q/J^5 = \frac{DHP\ \eta_R\ n^2}{2\pi\rho(1-w)^5V^5}$

For design task 1

$$K_T - (CT * J_{free}^2) \geq 0 \tag{3.25}$$

Where CT is the K_T/J_{free}^x value from Table 3.5 and J_{free} is the free variable for advance coefficient. x is dependent on the selected design task, for design task 1: $x = 2$.

All constraints have been determined and all inputs are specific for optimization. The objective function is minimized

$$\min(\eta_o) \tag{3.26}$$

However, the maximum efficiency is desired, so the negative of the objective function will be minimized.

The GUI written for this analysis, *WageningenOpt.py*, and the results of the optimization are discussed in Chapter 5.

Chapter 4

Combined Annular Momentum Theory and Blade Element Theory

As discussed in the previous section, B-series propeller data is not ideal for all applications. Many important effects, such as cavitation, are not considered in the B-series data. Therefore, a propeller optimization is presented which uses a combined annular momentum theory (AMT) and blade element theory (BET). Like the B-series propellers, some thrust/speed relation representing the hull along with some power/RPM relation representing the engine is available in the preliminary design stage. Therefore, "the propulsors are designed to simultaneously match the hull/thrust/speed characteristics and the engine power/RPM characteristics" [17]. This matching process is the basis for the optimization and will be discuss in this chapter. But first a discussion of how the propulsor's characteristics are calculated with the AMT and BET is necessary.

For a complete review and discussion of AMT and BET, refer to Vorus [17] and van Manen and van Oossanen [14].

4.1 Annular Momentum Theory

The annular momentum theory, AMT, is based on the equations of continuity and momentum. For a control volume (CV), as shown in Figure 4.1, the continuity and momentum equations

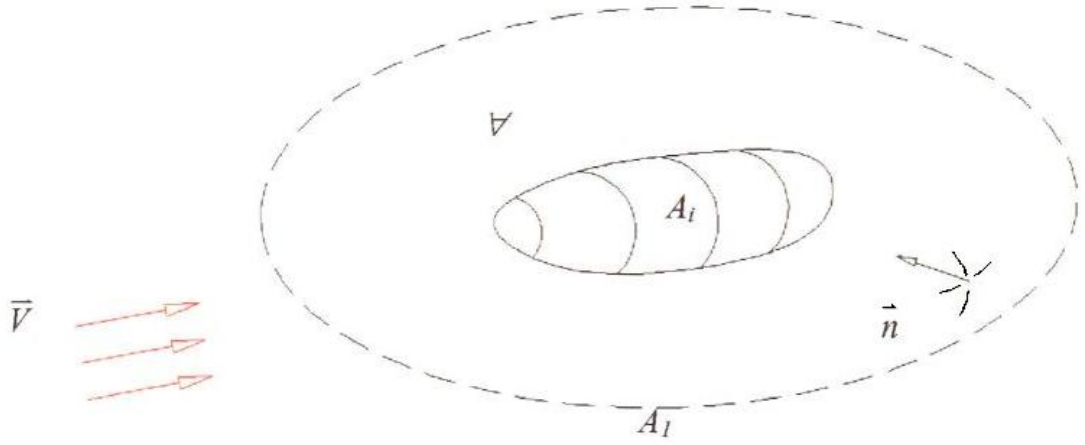

FIGURE 4.1: Control Volume [17]

are

$$\text{Continuity} \qquad \iint_{\sum A} \vec{V} \cdot \vec{n} dA = 0 \tag{4.1}$$

$$\text{Momentum} \qquad \rho \iint_{\sum A} \vec{V} \left(\vec{V} \cdot \vec{V} \right) dA - \rho g \forall \vec{k} + \iint_{\sum A} \vec{\sigma} dA = 0 \tag{4.2}$$

where ρ is the fluid density, $\vec{V}$ is the velocity vector, $\vec{\sigma}$ is the stress vector acting on the surfaces A of $\forall$, $\vec{k}$ is the unit vector with a positive up direction, and $\vec{n}$ is the unit vector in the normal direction pointing into $\forall$.

A good way to describe the benefits of using the AMT is to look at a brief description of a marine propeller in a CV. Consider a propeller advancing at a speed of V in a CV, similar to Figure 4.1. The fluid is contracted as it is accelerated through the rotating propeller, as shown in Figure 4.2. This contraction results in a velocity increase downstream of the propeller. The increase in speed is calculated with $V(1-s)$, where $s \equiv$ slip ratio. Slip ratio can be formulated in terms of speed V, where

$$\frac{V_2 - V_1}{V_1} = \frac{V(1-s) - V}{V} \equiv s \tag{4.3}$$

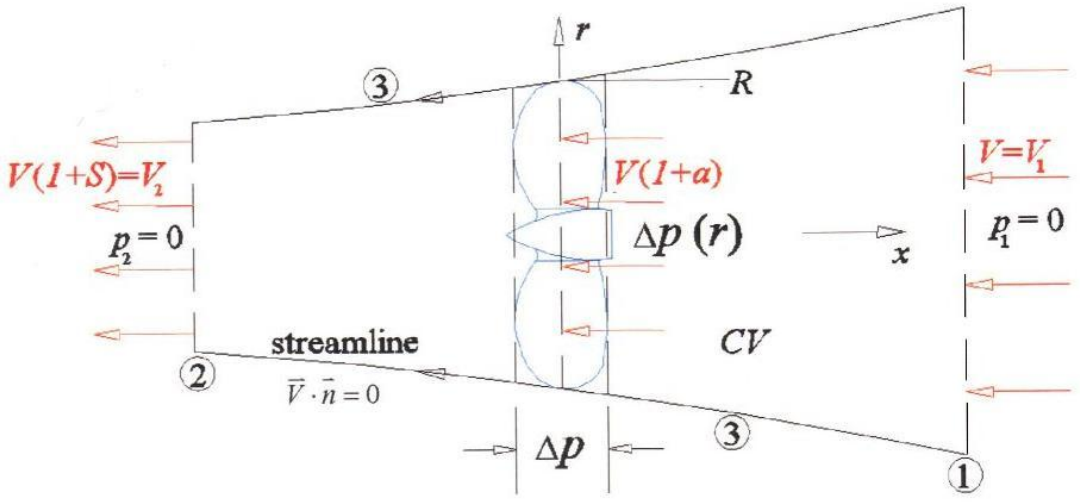

FIGURE 4.2: Propeller in Control Volume [17]

Using the fluid momentum passing through the propeller, thrust is calculated in terms of density ρ, source strength Q_s, speed V, and slip ratio s.

$$T = \rho Q_s V s \tag{4.4}$$

where Q_s, the "source strength," is defined as $Q_s = \iint_A u dA$, and u is the axial disturbance velocity.

By setting up a fraction of slip, a, occurring at the propeller plane; a is determined using the formulated thrust T and absorb power P_a. Absorb power, the rate of work done by the water, is calculated

$$P_a = TV(1+a) \tag{4.5}$$

"which must equal the time rate of change in kinetic energy upstream and downstream" [17]; therefore,

$$a = \frac{s}{2} \tag{4.6}$$

It is determined that half of the slip occurs upstream of the propeller and half downstream. With a known in terms of s, the coefficient of thrust can be determined

$$C_T = \frac{T}{\frac{1}{2}\rho V^2 A_D} = 2\left(1 + \frac{s}{2}\right)s \tag{4.7}$$

Efficiency is calculated as useful power divided by absorbed power; therefore, the ideal open water efficiency can be calculated in terms of C_T.

$$\eta_I = \frac{TV}{P_a} = \frac{1}{1+\frac{s}{2}} = \frac{2}{1+\sqrt{1+C_T}} \tag{4.8}$$

Equation (4.8) establishes the upper limit on available thrust. However, viscous forces have not been added, which will lower the open water efficiency. The addition of viscous forces are necessary to calculate the real power and thrust, as discussed later in this chapter.

4.2 Blade Element Theory

Conventional annular momentum theory has a circumferentially averaging effect that excludes geometric details, such as blade number, blade size and shape, etc. BET is an adapted approach from "turbo-machinery via velocity vector diagrams and lifting hydrofoils sections" [17].

This theory considers a rotating propeller at expanded sections of radius, r. With derivations not discussed in this thesis[1], the ideal efficiency is calculated using momentum analysis:

$$\eta_I = \frac{TV}{P_a} = \frac{N\Omega V \int_{r_h}^{R} \frac{l(r)r}{\sqrt{\Omega^2 r^2+V^2}}}{N\Omega V \int_{r_h}^{R} \frac{l(r)r}{\sqrt{\Omega^2 r^2+V^2}}} = 1.0 \tag{4.9}$$

The problem with blade elemental theory is the exclusion of induced velocity. Therefore, a combination of BET and AMT with the addition of viscous forces, such as skin friction, are used to correctly calculate propeller characteristics.

4.3 Combined AMT and BET

The combined AMT/BET method considers a momentum CV at annulus radii, as shown in Figure 4.3. This method was used for the programming of *MTPROPOPT.py*, which was written based off the programming work of Dr. William Vorus. *MTPROP* was written in Fortran by Dr. Vorus in October of 1996. He improved the viscous effects analysis in February of 1997.

[1] refer to [17]

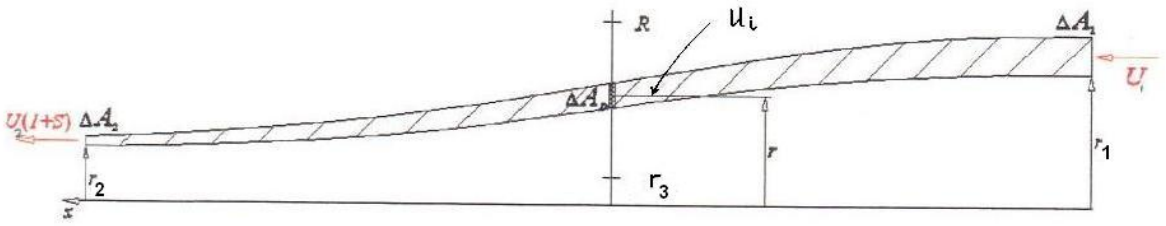

FIGURE 4.3: Annulus Control Volume at Radius r(x) [17]

Off-design calculations were added in November of 1999[2]. The program written for this thesis added an optimization algorithm to the original code. *MTPROP* calculates the ideal and real thrust of a propeller for a set of inputs. *MTPROP* works by specifying a pitch ratio $\frac{P_i}{D}$ and determining a thrust coefficient C_{Ti}. $\frac{P_i}{D}$ becomes a shape function for the analysis. As shown in [17], the propeller's real thrust, power, and efficiency is calculated for a ship speed U, propeller revolutions per minute $PRPM$, wake velocity distribution $\frac{V_a(r)}{U}$, propeller diameter D, propeller hub diameter D_h, chord distribution $\frac{C(r)}{R}$, and pitch ratio $\frac{P_i(r)}{D}$.

1. Calculate $C_{Ti}(r)$

   (a) Calculate the tan beta ratio, $TBR_a(r)$. The tan beta ratio is the ratio of the tangent of hydrodynamic pitch angle over the tangent of hydrodynamic advance angle.

$$TBR_a(r) \equiv \frac{\tan\left(\beta_i(r)\right)}{\tan\left(\beta_a(r)\right)} \tag{4.10}$$

   with

$$\tan(\beta_i(r)) = \frac{\frac{P_i}{D}}{\pi \overline{r}} \tag{4.11}$$

   where $\overline{r} = \frac{r}{R}$ and $R = \frac{D}{2}$

$$\tan(\beta_a(r)) = \frac{V_a}{\Omega r} = \frac{U}{\Omega r}\left(\frac{V_a}{U}\right) \tag{4.12}$$

   with the angular frequency $\Omega = \frac{2\pi}{60} PRPM$

[2] Off-design is not presented in this report

(b) Calculate the ideal sectional thrust coefficient, C_{Ti}, using the $TBR_a(r)$

$$C_{Tia}(r) = \frac{1 - \frac{1}{TBR_a} + (TBR_a - 1)\tan^2\beta_a}{\tan^2\beta_a + \frac{1}{4TBR_a^2}(1 - TBR_a^2\ \tan^2\beta_a)^2} \tag{4.13}$$

$$C_{Ti}(r) = \left(\frac{V_a}{U}\right)^2 C_{Tia}(r) \tag{4.14}$$

$$C_{Ti}(r) = \left(\frac{V_a}{U}\right)^2 \left[\frac{1 - \frac{1}{TBR_a} + (TBR_a - 1)\tan^2\beta_a}{\tan^2\beta_a + \frac{1}{4TBR_a^2}(1 - TBR_a^2\ \tan^2\beta_a)^2}\right] \tag{4.15}$$

2. Calculate the ideal sectional power coefficient, $C_{Pdi}(r)$, using the calculated $C_{Ti}(r)$ in the previous step

$$C_{P_{di}}(r) = \frac{\Omega R}{\pi U} C_{Ti}(r) \frac{P_i(r)}{D} \tag{4.16}$$

3. With the ideal sectional thrust and power coefficients determined, the viscous components are calculated. These are the necessary steps excluded in the AMT and BET methods previously discussed. Both viscous thrust and power coefficients are calculated using the radial distribution of slip. Calculate the resulting wake velocity distribtion $\frac{V_r}{U}$ to simplify calculations.

$$\frac{V_r}{U} = \sqrt{\left(\frac{V_a}{U} + \frac{u_i}{U}\right)^2 + \left(\frac{\Omega r}{U} - \frac{v_i}{U}\right)^2} \tag{4.17}$$

(a) Calculate $C_{Tv}(r)$

$$C_{Tv}(r) = -ZC_d \left(\frac{V_r}{U}\right)^2 \frac{C}{R} \frac{\sin\beta_i}{2\pi\bar{r}} \tag{4.18}$$

where C_d is the section drag coefficient.

$$\frac{u_i}{U} = \frac{1}{2}\frac{V_a}{U}\left[-1 + \sqrt{1 + C_{Ti}\left(\frac{U}{V_a}\right)^2}\right] \tag{4.19}$$

$\frac{u_i}{U}$ is the axial induced velocity in the propeller plane.

$$\frac{v_i}{U} = \frac{1}{2}\left[\frac{\Omega r}{U} - \sqrt{\left(\frac{\Omega r}{U}\right)^2 + C_{Ti}}\right] \tag{4.20}$$

where $\frac{\Omega r}{U} = \cot\beta$ and $\frac{v_i}{U}$ is the tangential velocity which is "determined by considering the pressure rise across the propeller plane" [17].

(b) Next, calculate the viscous power coefficient, C_{pv}

$$C_{pv} = ZC_d \left(\frac{V_r}{U}\right)^2 \frac{C}{R} \frac{\cos\beta_i}{2\pi\overline{r}} \cot\beta \tag{4.21}$$

4. With the viscous effects determined, the real coefficients of thrust and power can be calculated

$$C_T(r) = C_{Ti}(r) + C_{Tv}(r) \tag{4.22}$$

$$C_{P_d}(r) = C_{P_{di}}(r) + C_{pv}(r) \tag{4.23}$$

5. Integrate the coefficients to get the total values

$$C_T = \frac{2}{1-\overline{r_h}^2} \int_{\overline{r}=\overline{r_h}}^{1} C_T\left(\overline{r}\right) \overline{r} d\overline{r} \tag{4.24}$$

$$C_{P_d} = \frac{2}{1-\overline{r_h}^2} \int_{\overline{r_h}}^{1} C_{P_d}\left(\overline{r}\right) \overline{r} d\overline{r} \tag{4.25}$$

6. All variables have been determined to calculate real open water efficiency (η_o), power (DHP), and thrust (T)

$$\eta_o = \frac{C_T}{C_{P_d}} \tag{4.26}$$

$$DHP = C_{P_d}\left(\frac{1}{2}\rho U^3 A_D\right) \tag{4.27}$$

$$T = C_T\left(\frac{1}{2}\rho U^2 A_D\right) \tag{4.28}$$

where $A_D = \pi R^2\left(1-\overline{r_h}^2\right)$

7. Calculate the optimum propeller. This design technique uses $\frac{P_i}{D}$, along with speed (U), as a free variables for the above equations to find the optimum propeller. The same limits, given in Section 3.3 for pitch ratio, were used as a constraint. The NM algorithm is set up with four other constraints to maximize speed and efficiency.

4.4 Constraints and Strength Calculations

It is ideal to get the maximum output from the engine, without over-extending the engines capabilities. Therefore, two sets of constraints are constructed, two for each the engine and thrust capabilities. The first constraint verifies that the engine output is within the engine's physical capacity and not over. The second engine constraint is set-up to get the engine output to within 100% of the engines maximum capabilities. The second set of constraints is used to verify that the thrust is "as expected" from initial design specifications. The first constraint verifies that the thrust is positive and the second performs the same analysis as the second engine constraint.

Along with the thrust and engine constraints, the strength of the propeller must be taken into account. This is done by looking at each section of the blade as a hydrofoil, as shown in Figure 4.4. In order to calculate the strength of the propeller blade, the propeller type and material is

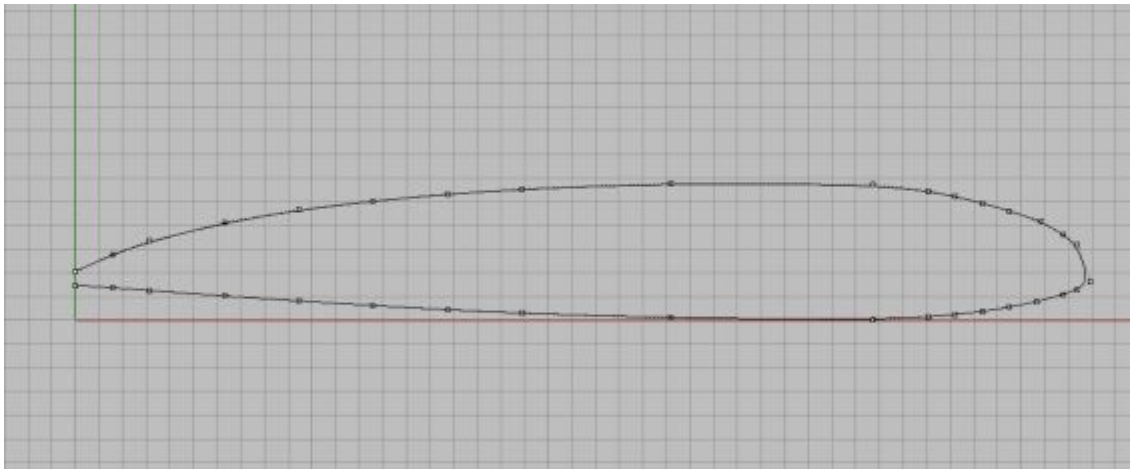

FIGURE 4.4: Radial Blade Section

needed. *MTPROPOPT.py* gives the user two options for blade type along with a few common blade materials. The blade material is needed to specify a yield stress for the propeller in order to compare to the calculated blade stress. If the blade's calculated stress is larger than the yield stress, the propeller fails. The stress of a propeller is calculated for each radial section with

$$|\sigma_{max}| = \frac{M}{SM_{min}} \tag{4.29}$$

in which σ_{max} is the maximum bending stress, M is the hydrofoil's moment, and SM_{min} is the minimum section modulus. The minimum section modulus is calculated

$$SM_{min} = \frac{I}{y} \tag{4.30}$$

where I is the moment of inertia and y is the vertical distance from the bending axis to the centroid of the cross section. These values are calculated uses a program written by Dr. Lothar Birk called *plasi.py*. For a closed figure, as shown in Figure 4.4; the area, centroid, and moment of inertia are calculated. This program is seen in Appendix B. The hydrofoils moment was calculated uses beam theory. The propeller's radial thrust distribution acted as a distributed load across the propeller, similar to a distributed load on a cantilevered beam.

However, in order to use the *plasi.py*, the propeller type must be determined. *MTPROPOPT.py* is set-up to calculate the stress for B-series propellers and user defined propellers. For a user defined propeller, some background information is needed to run the optimization. NACA propellers were used for tests in this thesis. Abbott and Doenhoff [3] gave all NACA propeller geometries, as seen in Table A.3.

The GUI written for this analysis, *MTPROPOPT.py*, and the results of the optimization are discussed in Chapter 5.

Chapter 5

Results

5.1 Wageningen B-series Optimization

The B-series optimization is coded into a graphic user interface (GUI) called *WageningenOpt.py*. The program's code is written in wxPython and displayed in Appendix B.2. As shown in Figure A.1, *WageningenOpt.py* is a GUI, similar to any windows based application. The program is created with 15 user inputs, 4 user buttons, and a list box of desired optimization tasks. *WageningenOpt.py* runs an optimization with four steps[1]:

1. Select the desired optimization design task and insert the design inputs into the provided spaces
2. With the desired task selected and all input values entered, select the **SAVE** button to save a *.txt* file. The program reads from this file for all calculations within the optimization.
3. After the file is saved, select the **RUN** button which will initiate the optimization.
4. When the optimization is completed, select the **View Results** button and the optimum values will be displayed.

The results for propeller optimization using the B-series have been published in past referenced works [18, 19]. The documentation in these reports select the optimum propeller for a constant

[1]Similar directions are given in the program's tool bar

expanded area ratio (EAR). Also, corrections for the effects of Reynolds number (Re) were not incorporated. *WageningenOpt.py* verified these results by limiting the EAR's maximum and minimum values to a small tolerance around the designed constant EAR and removing the corrections for Re. For each task explained in Section 3.3, a propeller with 5 blades and an EAR of 0.65 was used. All other input values are identical to those used in [18, 19]. The results are displayed in Table 5.1. It can be seen that the values of *WageningenOpt.py* are

TABLE 5.1: Results from Constant EAR Optimization

Design Task 1		
	University of Michagan	*WageningenOpt.py*
η	0.69	0.69
J	0.88	0.90
PD	1.20	1.20
CT/J^2	0.278	0.270
Design Task 2		
η	0.69	0.695
J	0.88	0.90
PD	1.20	1.18
CT/J^3	0.0641	0.0623
Design Task 3		
η	0.70	0.698
J	0.85	0.878
PD	1.10	1.12
CT/J^4	0.3611	0.3533
Design Task 4		
η	0.70	0.70
J	0.85	0.881
PD	1.10	1.11
CT/J^5	0.0822	0.0815

very close to the calculated reference values. One reason for the small difference could come from the fact that even thought the EAR is constrained in *WageningenOpt.py*, it still is not true a constant 0.65. Also, *WageningenOpt.py* uses a regression equation to calculate kinematic viscosity and density based on water temperature. The values in the reference material [18, 19] used a standard kinematic viscosity and density, which would cause a small difference in the final values.

Along with the verification of propeller selection for a constant EAR, the program was checked against another reference [20] which used a constrained EAR for an optimal propeller selection. Radojčić [20] used a Sequential Unconstrained Minimization Technique (SUMT) optimization

algorithm, very similar to the Nelder-Mead (NM) optimization algorithm used for this thesis. Again, corrections for Re were not taken into account. The results from both studies were similar, as shown in Table 5.2. The NM method was able to find higher efficiencies.

TABLE 5.2: Results from SUMT vs NM Optimization

Design Task 1		
	University of Southampton	*WageningenOpt.py*
η	0.6849	0.6922
J	0.8617	0.8974
PD	1.1805	1.1988
EAR	0.6398	0.6405
$n[RPM]$	78.141	75.0377
$D[ft]$	18.0	18.0
Design Task 2		
η	0.6866	0.6948
J	0.8679	0.8993
PD	1.1852	1.1824
EAR	0.6432	0.6486
$n[RPM]$	77.580	74.881
$D[ft]$	18.0	18.0
Design Task 3		
η	0.6909	0.6988
J	0.8302	0.8777
PD	1.0882	1.1154
EAR	0.6792	0.6627
$n[RPM]$	77.000	77.000
$D[ft]$	18.960	17.936
Design Task 4		
η	0.6915	0.6992
J	0.8316	0.8783
PD	1.0872	1.1123
EAR	0.6574	0.6633
$n[RPM]$	77.000	77.000
$D[ft]$	18.927	17.923

5.2 Combined AMT and BET Optimization

MTPROPOPT.py, illustrated in Figure A.2, creates a familiar GUI for the propeller optimization. Also written in wxPython, the direction for optimization are similar to *WageningenOpt.py*.

1. Insert design inputs into the provided spaces.

2. Select the **SAVE** button to save a *.txt* file. The program reads from the data file for all calculations within the optimization.

3. Select the designed blade type and material type.

4. Select the **Run** button. This will initiate the programming optimization.

5. After the optimization is completed, select the **View Results** button to view the final results of the optimization.

6. Along with viewing the results, the **Output to File** button will produce, *MTPresults.txt*, a text file with all the calculated results from the optimization. The text file output is shown in Appendix A.

Reference material for the AMT/BET optimization was not available like the references for the Wageningen B-series optimization. However, *MTPROP* was used for an example problem for the University of New Orleans NAME academia. *MTPPROPOPT.py* produced an optimum efficiency and speed similar to the example problem's results.

5.3 Comparison of the Two Methods

It is appropriate at this point to compare the two propeller optimization programs. Each program used the exact same input values. Since the vessel speed is not a free variable in *WageningenOpt.py* the optimum speed produced from *MTPROPOPT.py* was used. The results in Table 5.3 used input data for a high speed ship. The table displays how each design task in *WageningenOpt.py* compares against *MTPROPOPT.py*. For all design tasks in *WageningenOpt.py*, *MTPROPOPT.py* produced a more efficient propeller. This is expected because *WageningenOpt.py* is not design for high speed crafts like *MTPROPOPT.py*. "The basic propeller of the B-series typically is a merchant ship propeller" [1]. Therefore, a comparison of *MTPROPOPT.py* and *WageningenOpt.py* for a merchant ship is necessary. As just discussed, *WageningenOpt.py* uses constant speed, so *MTPROPOPT.py* had to be modified to optimize pitch-ratio for a constant speed and total resistance. Also, the wave velocity distribution was set to unity for the calculations. Again, *MTPROPOPT.py* produces the more efficient propeller, as shown in Table 5.4.

TABLE 5.3: Results from Presented Programs - High Speed Craft

	WageningenOpt.py				*MTPROPOPT.py*
	Design Task 1	Design Task 2	Design Task 3	Design Task 4	
η	0.6520	0.7268	0.6455	0.7359	0.8357
U	32.0	32.0	32.0	32.0	32.0
J	0.6493	0.9848	0.8098	0.9906	1.0472
PD	0.9399	1.2434	0.9459	1.1593	1.3
EAR	0.7882	0.7333	1.0400	0.9349	0.7161
$n[RPM]$	1259.1578	830.1440	2000.0	2000.0	2000.0
$D[ft]$	4.0	4.0	5.1443	4.2057	4.0
Thrust	40729.9877	17060.3498	22349.1875	12583.3477	23726.55
Torque	25522.3814	14399.5597	21669.8614	11184.1359	18928.0000

It should be noted that an efficiency of 0.8357 is high for typical marine propulsors. The maximum open-water efficiency in marine propellers is in the range of 0.6 to 0.8. A couple reasons for the high efficiency in Table 5.3 comes from the input resistance data and the wake velocity distribution. The given thrust is calculated from the given frictional resistance. The ships total resistance will be higher; therefore, lowering the open water efficiency. Also, the wake velocity distribution was unknown, so an estimation equation was used.

$$\frac{V_a}{U} = 0.75 + 0.20\left(\frac{\overline{r} - \overline{r_h}}{1. - \overline{r_h}}\right) \tag{5.1}$$

This equation provides realistic values for the distribution, but not exact. These two estimation are believed to be the reason for the high efficiency shown in Table 5.3.

TABLE 5.4: Results from Presented Programs - Merchant Ship

	WageningenOpt.py				*MTPROPOPT.py*
	Design Task 1	Design Task 2	Design Task 3	Design Task 4	
η	0.7174	0.7139	0.7282	0.7293	0.7940
U	16.0	16.0	16.0	16.0	16.0
J	0.8454	0.8670	0.8427	0.8526	1.1699
PD	1.1054	1.229	1.0524	1.0560	1.31
EAR	0.7958	0.7761	0.7877	0.7858	0.7166
$n[RPM]$	79.6506	77.6745	77.0	77.0	77.0
$D[ft]$	18.0	18.0	18.6797	18.4638	18.0
Thrust	65327.4075	61713.8085	59509.3485	55576.9018	61922.36
Torque	219076.6631	210817.1993	203717.0616	189625.3821	261377.8288

Chapter 6

Conclusion

After comparing the two methods, it is easy to see that *MTPROPOPT.py* will produce the more efficient propeller. However, *MTPROPOPT.py* requires more preliminary design information. If an engineer has minimum information at the beginning of the design process, *WageningenOpt.py* will probably be the ideal program. One benefit of *WageningenOpt.py*'s constrained optimization, it requires very little preliminary information.

With this in mind, if the design process has the time and money to receive the necessary preliminary information, such as a wake velocity distribution, propeller type, or chord distribution; *MTPROPOPT.py* will be the ideal optimization method.

6.1 Future Research

MTPROPOPT.py could be used for further research. One future consideration for an extension to *MTPROPOPT.py* is optimizing with rack and skew as added free variables. *MTPROPOPT.py* currently does not take the effects of skew and rack into account. Also, the strength calculations are not set-up as constraints, just a "*PASS/FAIL*" warning. This added constraint will simplify another necessary calculation for propeller design.

Another idea for future research would be to integrate *MTPROPOPT.py* with a program for calculating the hull's resistance based on input geometries. This would allow for the generation of

an optimum propeller for any hull shape, along with possibly giving stability and hydrodynamic calculations. Overall, *MTPROPOPT.py* has room to expand and hopefully will in the future.

Appendix A

Figures and Tables

TABLE A.1: Values of V_1 for Blade Sections [2]

r/R	-1.0	-0.95	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.2	0.0
$1.0-0.7$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	0.0522	0.0420	0.0330	0.0190	0.0040	0.0012	0.0	0.0	0.0	0.0
0.4	0.1467	0.1200	0.0972	0.0630	0.0395	0.0214	0.0116	0.0044	0.0	0.0
0.3	0.2306	0.2040	0.1790	0.1333	0.0943	0.0623	0.0376	0.0202	0.0033	0.0
0.25	0.2598	0.2372	0.2115	0.1651	0.1246	0.0899	0.0576	0.0350	0.0084	0.0
0.2	0.2826	0.2630	0.2400	0.1967	0.1570	0.1207	0.0880	0.0592	0.0172	0.0
0.15	0.3000	0.2824	0.2650	0.2300	0.1950	0.1610	0.1280	0.0955	0.0363	0.0
r/R	$+1.0$	$+0.95$	$+0.9$	$+0.8$	$+0.7$	$+0.6$	$+0.5$	$+0.4$	$+0.2$	0.0
$1.0-0.7$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.6	0.0382	0.0169	0.0067	0.0006	0.0	0.0	0.0	0.0	0.0	0.0
0.5	0.1278	0.0778	0.0500	0.0211	0.0085	0.0034	0.0008	0.0	0.0	0.0
0.4	0.2181	0.1467	0.1088	0.0637	0.0357	0.0189	0.0090	0.0033	0.0	0.0
0.3	0.2923	0.2186	0.1760	0.1191	0.0790	0.0503	0.0300	0.0148	0.0027	0.0
0.25	0.3256	0.2513	0.2068	0.1465	0.1008	0.0669	0.0417	0.0224	0.0031	0.0
0.2	0.3560	0.2821	0.2353	0.1685	0.1180	0.0804	0.0520	0.0304	0.0049	0.0
0.15	0.3860	0.3150	0.2642	0.1870	0.1320	0.0920	0.0615	0.0384	0.0096	0.0

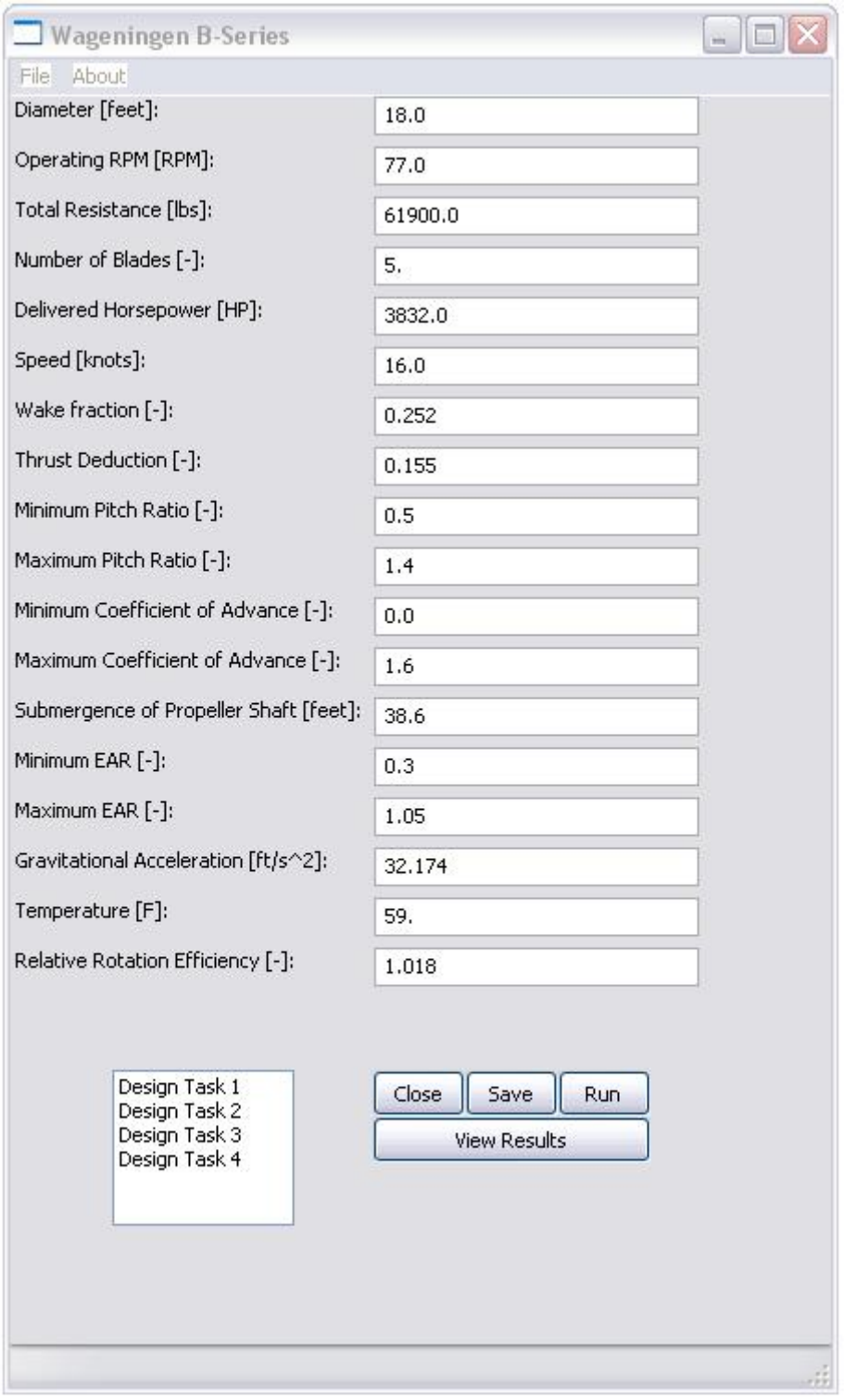

FIGURE A.1: WageningenOpt.py GUI

TABLE A.2: Values of V_2 for Blade Sections [2]

r/R	-1.0	-0.95	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.2	0.0
$1.0-0.9$	0.0	0.0975	0.19	0.36	0.51	0.64	0.75	0.84	0.96	1.0
0.85	0.0	0.0975	0.19	0.36	0.51	0.64	0.75	0.84	0.96	1.0
0.8	0.0	0.0975	0.19	0.36	0.51	0.64	0.75	0.84	0.96	1.0
0.7	0.0	0.0975	0.19	0.36	0.51	0.64	0.75	0.84	0.96	1.0
0.6	0.0	0.0965	0.1885	0.3585	0.5110	0.6415	0.7530	0.8426	0.9613	1.0
0.5	0.0	0.0950	0.1865	0.3569	0.5140	0.6439	0.7580	0.8456	0.9639	1.0
0.4	0.0	0.0905	0.1810	0.3500	0.5040	0.6353	0.7525	0.8415	0.9645	1.0
0.3	0.0	0.0800	0.1670	0.3360	0.4885	0.6195	0.7335	0.8265	0.9583	1.0
0.25	0.0	0.0725	0.1567	0.3228	0.4740	0.6050	0.7184	0.8139	0.9519	1.0
0.2	0.0	0.0640	0.1455	0.3060	0.4535	0.5842	0.6995	0.7984	0.9446	1.0
0.15	0.0	0.0540	0.1325	0.2870	0.4285	0.5585	0.6770	0.7805	0.9360	1.0
r/R	$+1.0$	$+0.95$	$+0.9$	$+0.8$	$+0.7$	$+0.6$	$+0.5$	$+0.4$	$+0.2$	0.0
$1.0-0.9$	0.0	0.0975	0.1900	0.3600	0.5100	0.6400	0.7500	0.8400	0.9600	1.0
0.85	0.0	0.1000	0.2028	0.3660	0.5160	0.6455	0.7550	0.8450	0.9615	1.0
0.8	0.0	0.1050	0.2337	0.3765	0.5265	0.6545	0.7635	0.8520	0.9635	1.0
0.7	0.0	0.1240	0.2720	0.4140	0.5615	0.6840	0.7850	0.8660	0.9675	1.0
0.6	0.0	0.1485	0.3056	0.4620	0.6060	0.7200	0.8090	0.8790	0.9690	1.0
0.5	0.0	0.1750	0.3235	0.5039	0.6430	0.7478	0.8275	0.8880	0.9710	1.0
0.4	0.0	0.1935	0.3197	0.5220	0.6590	0.7593	0.8345	0.8933	0.9725	1.0
0.3	0.0	0.1890	0.3197	0.5130	0.6505	0.7520	0.8315	0.8920	0.9750	1.0
0.25	0.0	0.1758	0.3042	0.4982	0.6359	0.7415	0.8259	0.8899	0.9751	1.0
0.2	0.0	0.1560	0.2840	0.4777	0.6190	0.7277	0.8170	0.8875	0.9750	1.0
0.15	0.0	0.1300	0.2600	0.4520	0.5995	0.7105	0.8055	0.8825	0.9760	1.0

TABLE A.3: Geometry Values for NACA 63-206 Blade Section [3]

Stations and ordinates given in percent of airfoil chord			
Upper Surface		Lower Surface	
Station	Ordinate	Station	Ordinate
0.0000	0.0000	0.0000	0.0000
0.5000	0.5745	0.5000	−0.4330
0.7500	0.6967	0.7500	−0.5225
1.2500	0.8950	1.2500	−0.6498
2.5000	1.2551	2.5000	−0.8608
5.0000	1.7893	5.0000	−1.1385
7.5000	2.1994	7.5000	−1.3368
15.0000	3.0650	15.0000	−1.7108
20.0000	3.4570	20.0000	−1.8589
25.0000	3.7393	25.0000	−1.9464
30.0000	3.9289	30.0000	−1.9833
35.0000	4.0296	35.0000	−1.9687
40.0000	4.0422	40.0000	−1.9004
45.0000	3.9716	45.0000	−1.7812
50.0000	3.8250	50.0000	−1.6186
55.0000	3.6112	55.0000	−1.4207
60.0000	3.3396	60.0000	−1.1963
65.0000	3.0161	65.0000	−0.9529
70.0000	2.6462	70.0000	−0.6993
75.0000	2.2396	75.0000	−0.4470
80.0000	1.8055	80.0000	−0.2101
85.0000	1.3547	85.0000	−0.0067
90.0000	0.9001	90.0000	0.1358
95.0000	0.4529	95.0000	0.1790
100.0000	0.0000	100.0000	0.0000

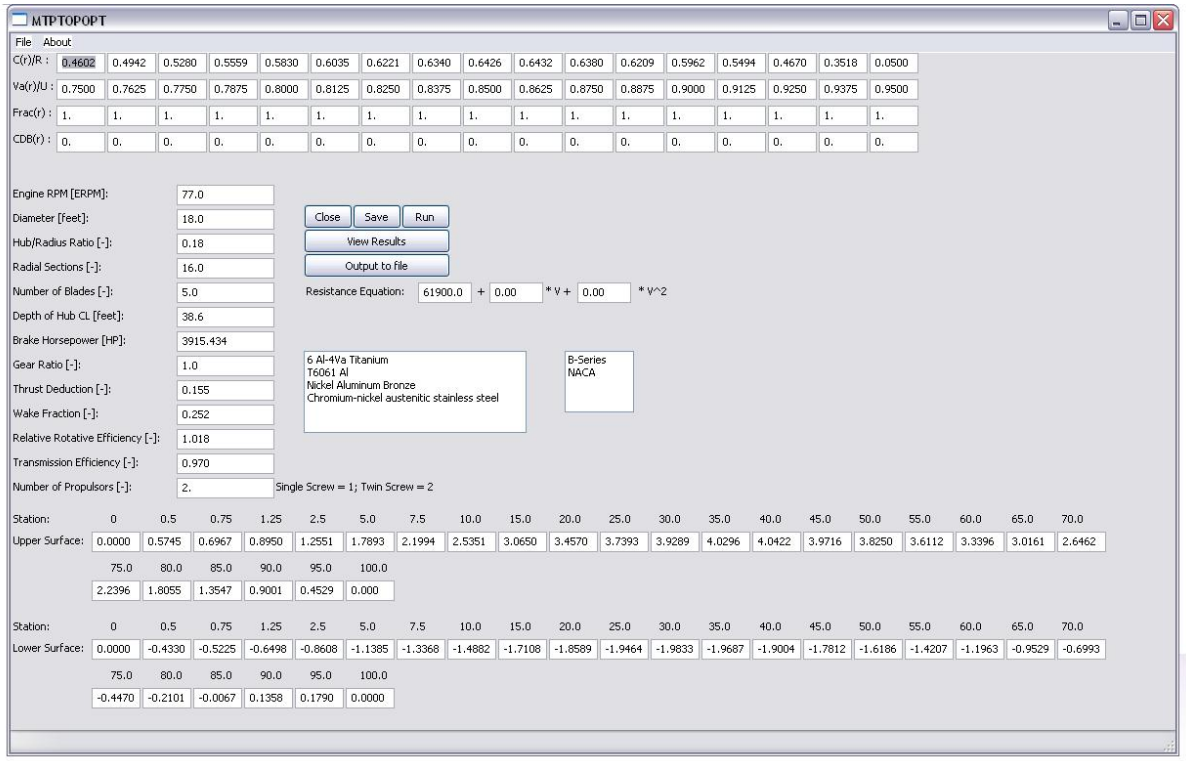

FIGURE A.2: MTPROPOPT.py GUI

TABLE 1

Coefficients and terms of the K_T and K_Q polynomials for the Wageningen B-screw Series for $R_n=2\times10^6$. Reproduced from [1]

$$K_T = \sum_{s,t,u,v} C^T_{s,t,u,v} \cdot (J)^s \cdot (P/D)^t \cdot (A_E/A_O)^u \cdot (z^v)$$

$$K_Q = \sum_{s,t,u,v} C^Q_{s,t,u,v} \cdot (J)^s \cdot (P/D)^t \cdot (A_E/A_O)^u \cdot (z^v)$$

K_T	$C^T_{s,t,u,v}$	s (J)	t (P/D)	u (A_E/A_O)	v (Z)	$C^Q_{s,t,u,v}$	s (J)	t (P/D)	u (A_E/A_O)	v (Z)
	+0.00880496	0	0	0	0	+0.00379368	0	0	0	0
	-0.204554	1	0	0	0	+0.00886523	2	0	0	0
	+0.166351	0	1	0	0	-0.032241	1	1	0	0
	+0.158114	0	2	0	0	+0.00344778	0	2	0	0
	-0.147581	2	0	1	0	-0.0408811	0	1	1	0
	-0.481497	1	1	1	0	-0.108009	1	1	1	0
	+0.415437	0	2	1	0	-0.0885381	2	1	1	0
	+0.0144043	0	0	0	1	+0.188561	0	2	1	0
	-0.0530054	2	0	0	1	-0.00370871	1	0	0	1
	+0.0143481	0	1	0	1	+0.00513696	0	1	0	1
	+0.0606826	1	1	0	1	+0.0209449	1	1	0	1
	-0.0125894	0	0	1	1	+0.00474319	2	1	0	1
	+0.0109689	1	0	1	1	-0.00723408	2	0	1	1
	-0.133698	0	3	0	0	+0.00438388	1	1	1	1
	+0.00638407	0	6	0	0	-0.0269403	0	2	1	1
	-0.00132718	2	6	0	0	+0.0558082	3	0	1	0
	+0.168496	3	0	1	0	+0.0161886	0	3	1	0
	-0.0507214	0	0	2	0	+0.00318086	1	3	1	0
	+0.0854559	2	0	2	0	+0.015896	0	0	2	0
	-0.0504475	3	0	2	0	+0.0471729	1	0	2	0
	+0.010465	1	6	2	0	+0.0196283	3	0	2	0
	-0.00648272	2	6	2	0	-0.0502782	0	1	2	0
	-0.00841728	0	3	0	1	-0.030055	3	1	2	0
	+0.0168424	1	3	0	1	+0.0417122	2	2	2	0
	-0.00102296	3	3	0	1	-0.0397722	0	3	2	0
	-0.0317791	0	3	1	1	-0.00350024	0	6	2	0
	+0.018604	1	0	2	1	-0.0106854	3	0	0	1
	-0.00410798	0	2	2	1	+0.00110903	3	3	0	1
	-0.000606848	0	0	0	2	-0.000313912	0	6	0	1
	-0.0049819	1	0	0	2	+0.0035985	3	0	1	1
	+0.0025983	2	0	0	2	-0.00142121	0	6	1	1
	-0.000560528	3	0	0	2	-0.00383637	1	0	2	1
	-0.00163652	1	2	0	2	+0.0126803	0	2	2	1
	-0.000328787	1	6	0	2	-0.00318278	2	3	2	1
	+0.000116502	2	6	0	2	+0.00334268	0	6	2	1
	+0.000690904	0	0	1	2	-0.00183491	1	1	0	2
	+0.00421749	0	3	1	2	+0.000112451	3	2	0	2
	+0.0000565229	3	6	1	2	-0.0000297228	3	6	0	2
	-0.00146564	0	3	2	2	+0.000269551	1	0	1	2
						+0.00083265	2	0	1	2
						+0.00155334	0	2	1	2
						+0.000302683	0	6	1	2
						-0.0001843	0	0	2	2
						-0.000425399	0	3	2	2
$R_n = 2 \times 10^6$						+0.0000869243	3	3	2	2
						-0.0004659	0	6	2	2
						+0.0000554194	1	6	2	2

FIGURE A.3: KT-KQ Polynomials

TABLE 2

Polynomials for Reynolds number effect (above $R_n = 2\times10^6$) on K_T and K_Q

$$\begin{aligned}
\Delta K_T = \; & 0.000353485 \\
& -0.00333758(A_E/A_O)J^2 \\
& -0.00478125(A_E/A_O)(P/D)J \\
& +0.000257792(\log R_n-0.301)^2(A_E/A_O)J^2 \\
& +0.0000643192(\log R_n-0.301)(P/D)^6J^2 \\
& -0.0000110636(\log R_n-0.301)^2(P/D)^6J^2 \\
& -0.0000276305(\log R_n-0.301)^2z(A_E/A_O)J^2 \\
& +0.0000954(\log R_n-0.301)z(A_E/A_O)(P/D)J \\
& +0.0000032049(\log R_n-0.301)z^2(A_E/A_O)(P/D)^3J
\end{aligned}$$

$$\begin{aligned}
\Delta K_Q = \; & -0.000591412 \\
& +0.00696898(P/D) \\
& -0.0000666654z(P/D)^6 \\
& +0.0160818(A_E/A_O)^2 \\
& -0.000938091(\log R_n-0.301)(P/D) \\
& -0.00059593(\log R_n-0.301)(P/D)^2 \\
& +0.0000782099(\log R_n-0.301)^2(P/D)^2 \\
& +0.0000052199(\log R_n-0.301)z(A_E/A_O)J^2 \\
& -0.00000088528(\log R_n-0.301)^2z(A_E/A_O)(P/D)J \\
& +0.0000230171(\log R_n-0.301)z(P/D)^6 \\
& -0.00000184341(\log R_n-0.301)^2z(P/D)^6 \\
& -0.00400252(\log R_n-0.301)(A_E/A_O)^2 \\
& +0.000220915(\log R_n-0.301)^2(A_E/A_O)^2
\end{aligned}$$

FIGURE A.4: KT-KQ Polynomials for Effects of Reynolds Number

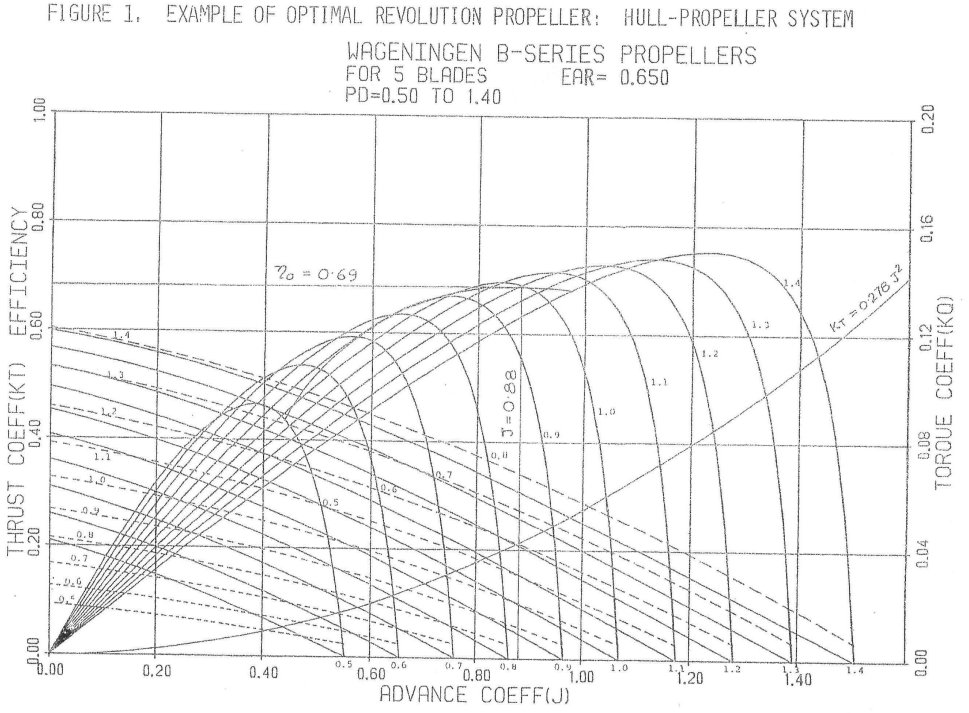

FIGURE A.5: KT-KQ Coefficients vs. J

MTPresults.txt

```
CRR    VaU    PrD    FRAC   CDBr
0.4602 1.0000 1.3101 1.0000 0.0000
0.4942 1.0000 1.3101 1.0000 0.0000
0.5280 1.0000 1.3101 1.0000 0.0000
0.5559 1.0000 1.3101 1.0000 0.0000
0.5830 1.0000 1.3101 1.0000 0.0000
0.6035 1.0000 1.3101 1.0000 0.0000
0.6221 1.0000 1.3101 1.0000 0.0000
0.6340 1.0000 1.3101 1.0000 0.0000
0.6426 1.0000 1.3101 1.0000 0.0000
0.6432 1.0000 1.3101 1.0000 0.0000
0.6380 1.0000 1.3101 1.0000 0.0000
0.6209 1.0000 1.3101 1.0000 0.0000
0.5962 1.0000 1.3101 1.0000 0.0000
0.5494 1.0000 1.3101 1.0000 0.0000
0.4670 1.0000 1.3101 1.0000 0.0000
0.3518 1.0000 1.3101 1.0000 0.0000
0.0500 1.0000 1.3101 1.0000 0.0000

 'Ideal Performance'
rb    CTi    Cpdi   PrD   C(r)/R EFFr   u/U    v/U
0.206 0.0960 0.1075 1.310 0.4772 0.8930 0.0234 0.0234
0.257 0.1362 0.1526 1.310 0.5111 0.8930 0.0330 0.0330
0.308 0.1765 0.1977 1.310 0.5419 0.8930 0.0423 0.0423
0.359 0.2148 0.2405 1.310 0.5695 0.8930 0.0511 0.0511
0.411 0.2500 0.2799 1.310 0.5933 0.8930 0.0590 0.0590
0.462 0.2816 0.3154 1.310 0.6128 0.8930 0.0660 0.0660
0.513 0.3096 0.3467 1.310 0.6281 0.8930 0.0722 0.0722
0.564 0.3342 0.3743 1.310 0.6383 0.8930 0.0775 0.0775
0.616 0.3557 0.3983 1.310 0.6429 0.8930 0.0822 0.0822
```

```
0.667 0.3744 0.4193 1.310 0.6406 0.8930 0.0862 0.0862
0.718 0.3907 0.4376 1.310 0.6295 0.8930 0.0896 0.0896
0.769 0.4050 0.4535 1.310 0.6085 0.8930 0.0927 0.0927
0.821 0.4174 0.4674 1.310 0.5728 0.8930 0.0953 0.0953
0.872 0.4283 0.4796 1.310 0.5082 0.8930 0.0976 0.0976
0.923 0.4379 0.4904 1.310 0.4094 0.8930 0.0996 0.0996
0.974 0.4463 0.4998 1.310 0.2009 0.8930 0.1013 0.1013

CTTI     CPPI     EFI
0.358302 0.401253 0.892957

Ideal Thrust     Ideal Power
64106.51         3527.433

 'Design Calculations'
 'Real Performance'
rb    CTi    Cpdi   EFF    CD          CL     REYN        V*
0.206 0.0770 0.1127 0.6833 8.811e-003 0.0693 1.2040e+007 1.1411
0.257 0.1190 0.1599 0.7443 8.684e-003 0.0757 1.3710e+007 1.2132
0.308 0.1603 0.2076 0.7723 8.566e-003 0.0752 1.5530e+007 1.2960
0.359 0.1992 0.2535 0.7860 8.458e-003 0.0704 1.7470e+007 1.3875
0.411 0.2348 0.2964 0.7924 8.360e-003 0.0635 1.9495e+007 1.4862
0.462 0.2668 0.3357 0.7947 8.273e-003 0.0561 2.1554e+007 1.5908
0.513 0.2951 0.3714 0.7945 8.196e-003 0.0490 2.3608e+007 1.7000
0.564 0.3199 0.4036 0.7927 8.129e-003 0.0427 2.5590e+007 1.8132
0.616 0.3417 0.4325 0.7900 8.072e-003 0.0373 2.7429e+007 1.9296
0.667 0.3608 0.4584 0.7870 8.027e-003 0.0328 2.9016e+007 2.0486
0.718 0.3775 0.4813 0.7843 7.995e-003 0.0292 3.0199e+007 2.1698
0.769 0.3924 0.5015 0.7825 7.978e-003 0.0265 3.0853e+007 2.2930
0.821 0.4057 0.5184 0.7826 7.984e-003 0.0247 3.0619e+007 2.4176
0.872 0.4179 0.5306 0.7876 8.039e-003 0.0244 2.8582e+007 2.5437
```

```
0.923 0.4294 0.5369 0.7998 8.176e-003 0.0268 2.4177e+007 2.6708
0.974 0.4419 0.5270 0.8385 8.779e-003 0.0482 1.2433e+007 2.7990

 'Optimum Thickness/C for Cavitation'
 '(< To Avoid BB Inception)'
rb    SIGMA  TBC
0.206 4.6962 1.5450
0.257 4.1545 1.3630
0.308 3.6407 1.1927
0.359 3.1762 1.0400
0.411 2.7684 0.9066
0.462 2.4165 0.7917
0.513 2.1158 0.6937
0.564 1.8599 0.6102
0.616 1.6424 0.5392
0.667 1.4571 0.4786
0.718 1.2988 0.4267
0.769 1.1631 0.3822
0.821 1.0462 0.3436
0.872 0.9451 0.3100
0.923 0.8572 0.2802
0.974 0.7805 0.2498

EAR 0.716557

CTT      CPP      EF
0.346094 0.435899 0.793979

J        KT       KQ       EF
1.169870 0.179981 0.042206 0.793979
```

Real Thrust	Real Power
61922.36	3832.000

Appendix B

Programs' Codes

B.1 plasi.py

```
#!/usr/bin/env python
#
##
## Python tools
##
import numpy

def plasi(C_polygon):
    """
    c--------------------------------------------------------------------
    c  Purpose:
    c  Calculation of area, moment, moment of inertia, moment of deviation,
    c  and centre of area from a surface defined through a polyline from
    c  given base points.
    c
    c--------------------------------------------------------------------
    c  Arguments:
    c
    c    Name     Type           I/O   Description
    c    ------------------------------------------------------------
    c    C_polygon real           I    2D array (2,number of points)
```

```
c
c-----------------------------------------------------------------------
c  Local variables:
c
c    Name     Type                Description
c    ------------------------------------------------------------------
c    i         integer            variable of counter
c    dx        real               difference between 2 following points
c    dy        real               difference between 2 following points
c    Di        real               part of one section to area
c    Db        real               sum of parts of area
c    Dn        real               part of the last section to the area
c    D         real               total area
c    Mx        real               total moment about x-axis
c    My        real               total moment about y-axis
c    Ix        real               total moment of inertia about x-axis
c    Iy        real               total moment of inertia about y-axis
c    Ixy       real               total moment of deviation
"""
#***** initialize ******************************************************
# could need some error checking
xi = numpy.zeros((len(C_polygon[0,:])), numpy.float)
xi1 = numpy.zeros((len(C_polygon[0,:])), numpy.float)
yi = numpy.zeros((len(C_polygon[1,:])), numpy.float)
yi1 = numpy.zeros((len(C_polygon[1,:])), numpy.float)
#print C_polygon[0,:-1]
xi[0:-1] = C_polygon[0,0:-1]
xi1[0:-1] = C_polygon[0,1:]
xi[-1] = xi1[-2]
xi1[-1] = xi[0]
yi[0:-1] = C_polygon[1,0:-1]
yi1[0:-1] = C_polygon[1,1:]
yi[-1] = yi1[-2]
yi1[-1] = yi[0]
#print xi, xi1
dx = xi1 - xi
#print dx
dy = yi1 - yi
Area = 0.5 * numpy.sum(dy *(xi1+xi))
```

```
    My = numpy.sum(dy * (xi1*xi1 + xi1*xi + xi*xi))/6.
    Mx = numpy.sum(-dx * (yi1*yi1 + yi1*yi + yi*yi))/6.

    Ix = -numpy.sum( (4.*yi**3+6.*dy*yi**2+4.*dy**2*yi+dy**3) * dx ) / 12.
    Iy =  numpy.sum( (4.*xi**3+6.*dx*xi**2+4.*dx**2*xi+dx**3) * dy ) / 12.

    ## Deviationsmoment
    Ixy = 1./24. * numpy.sum(dy * (3.*dx**2*(yi1+3.*yi) \
                + 4.*dx*xi*(2.*yi1+yi) + 6.*xi**2*(yi1+yi)))

    #---- Calculation if centre of area in x direction --------------------
    xs=My/Area

    #---- Calculation of centre of area in y direction --------------------
    ys=Mx/Area



    #---------------------------------------------------------------------
    #*****Control if area is positiv, e.i. if points are defined in *******
    #*****mathematical positiv way ****************************************
    if Area < .0:

        Area=(-1)*Area
        Mx=(-1)*Mx
        My=(-1)*My
        Ix=(-1)*Ix
        Iy=(-1)*Iy
        Ixy=(-1)*Ixy

    #***** for debugging purposes ******************************************
##      print 'area                          A = ', Area
##      print 'static moment about x-axis  Mx = ',Mx
##      print 'static moment about y-axis  My = ',My
##      print 'area center about x-axis    xs = ',xs
##      print 'area center about y-axis    ys = ',ys
##      print 'Traegheitsmoment um x-Achse    = ',Ix
##      print 'Traegheitsmoment um y-Achse    = ',Iy
##      print 'Deviationsmoment               = ',Ixy
    return Area, xs, ys, Mx, My, Ix, Iy, Ixy
```

```
## test tool
if __name__ == '__main__':

    pts = numpy.array([[ 0., 1., 1.,0.],[0.,0.,1.,1.]])

    #print pts[:,1]

    Area, xs, ys, Mx, My, Ix, Iy, Ixy = plasi(pts)
    #print Area, xs, ys, Mx, My, Ix, Iy, Ixy

    #print katalogue(list, 0.00001)
```

B.2 WageningenOpt.py

```
#!/usr/bin/env python

# Getting the GUI stuff
import wx
import wx.lib.dialogs
# Optimazation files
from scipy.optimize import *
from numpy import *
import KTKQ_Evaluation23

# Handling files and directories
import os

class TextFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Wageningen B-Series', pos=(0,0),
                size=(450, 750))
        self.filename = ""
#Create Status bar
        self.CreateStatusBar()
```

```
# Begin Toolbar / Menubar
        menubar = wx.MenuBar()

        file = wx.Menu()
        file.Append(-1, '&New')
        menuItem1 = file.Append(-1, "&Save")
        menuItem2 = file.Append(-1, "&Quit")
        file.AppendSeparator()

        helpMenu  = wx.Menu()
        menuItem3 = helpMenu.Append(-1, 'Design Tasks')
        menuItem4 = helpMenu.Append(-1, '&Directions')



        menubar.Append(file, '&File')
        menubar.Append(helpMenu, '&About')
        self.SetMenuBar(menubar)
        self.Bind(wx.EVT_MENU, self.OnSave, menuItem1)
        self.Bind(wx.EVT_MENU, self.OnCloseMe, menuItem2)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
        self.Bind(wx.EVT_MENU, self.OnMethods, menuItem3)
        self.Bind(wx.EVT_MENU, self.OnAbout, menuItem4)

 # Begin List of Input lines

        panel = wx.Panel(self, -1)

        Input0Label = wx.StaticText(panel, -1, " Diameter [feet]:")
        self.Input0Text = wx.TextCtrl(panel, -1, "18.0", size=(175, -1))
        self.Input0Text.SetInsertionPoint(0)

        Input1Label = wx.StaticText(panel, -1, " Operating RPM [RPM]:")
        self.Input1Text = wx.TextCtrl(panel, -1, "77.0", size=(175, -1))
        self.Input1Text.SetInsertionPoint(0)

        Input2Label = wx.StaticText(panel, -1, " Total Resistance [lbs]:")
        self.Input2Text = wx.TextCtrl(panel, -1, "61900.0", size=(175, -1))
        self.Input2Text.SetInsertionPoint(0)
```

```
Input3Label = wx.StaticText(panel, -1, " Number of Blades [-]:")
self.Input3Text = wx.TextCtrl(panel, -1, "5.", size=(175, -1))
self.Input3Text.SetInsertionPoint(0)

Input4Label = wx.StaticText(panel, -1, " Delivered Horsepower [HP]:")
self.Input4Text = wx.TextCtrl(panel, -1, "3832.0", size=(175, -1))
self.Input4Text.SetInsertionPoint(0)

Input5Label = wx.StaticText(panel, -1, " Speed [knots]:")
self.Input5Text = wx.TextCtrl(panel, -1, "16.0", size=(175, -1))
self.Input5Text.SetInsertionPoint(0)

Input6Label = wx.StaticText(panel, -1, " Wake fraction [-]:")
self.Input6Text = wx.TextCtrl(panel, -1, "0.252", size=(175, -1))
self.Input6Text.SetInsertionPoint(0)

Input7Label = wx.StaticText(panel, -1, " Thrust Deduction [-]:")
self.Input7Text = wx.TextCtrl(panel, -1, "0.155", size=(175, -1))
self.Input7Text.SetInsertionPoint(0)

Input8Label = wx.StaticText(panel, -1, " Minimum Pitch Ratio [-]:")
self.Input8Text = wx.TextCtrl(panel, -1, "0.5", size=(175, -1))
self.Input8Text.SetInsertionPoint(0)

Input9Label = wx.StaticText(panel, -1, " Maximum Pitch Ratio [-]:")
self.Input9Text = wx.TextCtrl(panel, -1, "1.4", size=(175, -1))
self.Input9Text.SetInsertionPoint(0)

Input10Label = wx.StaticText(panel, -1, " Minimum Coefficient of Advance [-]:")
self.Input10Text = wx.TextCtrl(panel, -1, "0.0", size=(175, -1))
self.Input10Text.SetInsertionPoint(0)

Input11Label = wx.StaticText(panel, -1, " Maximum Coefficient of Advance [-]:")
self.Input11Text = wx.TextCtrl(panel, -1, "1.6", size=(175, -1))
self.Input11Text.SetInsertionPoint(0)

Input12Label = wx.StaticText(panel, -1, " Submergence of Propeller Shaft [feet]:")
self.Input12Text = wx.TextCtrl(panel, -1, "38.6", size=(175, -1))
```

```
        self.Input12Text.SetInsertionPoint(0)

        Input13Label = wx.StaticText(panel, -1, " Minimum EAR [-]:")
        self.Input13Text = wx.TextCtrl(panel, -1, "0.3", size=(175, -1))
        self.Input13Text.SetInsertionPoint(0)

        Input14Label = wx.StaticText(panel, -1, " Maximum EAR [-]:")
        self.Input14Text = wx.TextCtrl(panel, -1, "1.05", size=(175, -1))
        self.Input14Text.SetInsertionPoint(0)

        Input15Label = wx.StaticText(panel, -1, " Gravitational Acceleration [ft/s^2]:")
        self.Input15Text = wx.TextCtrl(panel, -1, "32.174", size=(175, -1))
        self.Input15Text.SetInsertionPoint(0)

        Input16Label = wx.StaticText(panel, -1, " Temperature [F]:")
        self.Input16Text = wx.TextCtrl(panel, -1, "59.", size=(175, -1))
        self.Input16Text.SetInsertionPoint(0)

        Input17Label = wx.StaticText(panel, -1, " Relative Rotation Efficiency [-]:")
        self.Input17Text = wx.TextCtrl(panel, -1, "1.018", size=(175, -1))
        self.Input17Text.SetInsertionPoint(0)

        sizer = wx.FlexGridSizer(cols=2, hgap=6, vgap=6)
        sizer.AddMany([Input0Label, self.Input0Text, Input1Label, self.Input1Text,\
                       Input2Label, self.Input2Text, Input3Label, self.Input3Text,\
                       Input4Label, self.Input4Text, Input5Label, self.Input5Text,\
                       Input6Label, self.Input6Text, Input7Label, self.Input7Text,\
                       Input8Label, self.Input8Text, Input9Label, self.Input9Text,\
                       Input10Label, self.Input10Text, Input11Label, self.Input11Text,\
                       Input12Label, self.Input12Text, Input13Label, self.Input13Text,\
                       Input14Label, self.Input14Text, Input15Label, self.Input15Text,\
                       Input16Label, self.Input16Text, Input17Label, self.Input17Text])
        panel.SetSizer(sizer)

# Create Check boxes

        List = ['Design Task 1', 'Design Task 2', 'Design Task 3', 'Design Task 4']

        self.listbox = wx.ListBox(panel, -1, (55, 525), wx.DefaultSize,
```

```
                        List, wx.LB_SINGLE)

        self.Bind(wx.EVT_LISTBOX, self.OnList, self.listbox)

# dirname is an APPLICATION variable that we're choosing to store
# in with the frame - it's the parent directory for any file we
# choose to edit in this frame
        self.dirname = ''

 # Begin buttons
        button = wx.Button(panel, label="Close", pos=(195,525), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
        button1 = wx.Button(panel, label="Save", pos=(245,525), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnSave, button1)
        button2 = wx.Button(panel, label="Run", pos=(295,525), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnRun, button2)
        button3 = wx.Button(panel, label="View Results", pos=(195,550), size=(150,25))
        self.Bind(wx.EVT_BUTTON, self.OnResults, button3)

    def OnList(self, event):
        for eachText11 in [self.Input0Text, self.Input1Text,\
                           self.Input2Text, self.Input3Text, self.Input4Text,\
                            self.Input5Text,\
                           self.Input6Text, self.Input7Text, self.Input8Text, \
                           self.Input9Text,\
                           self.Input10Text, self.Input11Text, self.Input12Text,\
                            self.Input13Text,\
                           self.Input14Text, self.Input15Text, self.Input16Text, \
                           self.Input17Text]:
                eachText11.Enable(True)
        self.lselect1 = self.listbox.GetSelection()
        if self.lselect1 == 0:
            for eachText1 in [self.Input1Text, self.Input4Text, self.Input17Text]:
                eachText1.Enable(False)
            self.methodnumber = "method_1"

        elif self.lselect1 == 1:
            for eachText2 in [self.Input1Text, self.Input2Text]:
```

```
                eachText2.Enable(False)
            self.methodnumber = "method_2"

        elif self.lselect1 == 2:
            for eachText3 in [self.Input0Text, self.Input4Text, self.Input17Text]:
                eachText3.Enable(False)
            self.methodnumber = "method_3"

        elif self.lselect1 == 3:
            for eachText4 in [self.Input0Text, self.Input2Text]:
                eachText4.Enable(False)
            self.methodnumber = "method_4"

        return self.methodnumber, self.lselect1

# Event for closing
    def OnCloseMe(self, event):
        self.Close(True)

    def OnCloseWindow(self, event):
        self.Destroy()

    def OnMethods(self, event):
        methods ="Design Task 1 = Given RT, D, V.       Find optimum Revolutions \
        Per Minute - RPMopt\
                \
    Design Task 2 = Given Pd, D, V.       Find optimum RPM - RPMopt\
                \
                            \
    Design Task 3 = Given RT, N, V.       Find optimum Diameter - Dopt\
                \
                \
                Design Task 4 = Given Pd, N, V.       Find optimum Diameter - Dopt"

        dialog1 = wx.lib.dialogs.ScrolledMessageDialog(self, methods, "Design Task Options")
        dialog1.ShowModal()


    def OnAbout(self, event):
```

```
        directions ="""This program is pretty straight forward and can be completed\
         in a few simply steps.\
                     1.) Select the desired calculation task and insert your inputs\
                      into the provided spaces.\
            2.) Save the file.  The program names the file based on the selected List\
             Item.  Just save, do not change the file name.\
                           \
            3.) After the file is saved, select the 'RUN' button.\
                                                    \
                     4.) Select "View Results" button and the optimum values will \
                     be displayed."""
        dialog = wx.lib.dialogs.ScrolledMessageDialog(self, directions, "Directions")
        dialog.ShowModal()

# Event for saving

    def OnSave(self,e):
        #Specify values to save for list item selected
        if self.lselect1 == 0:
            v0  = float(self.Input0Text.GetValue())
            v2  = float(self.Input2Text.GetValue())
            v3  = float(self.Input3Text.GetValue())
            v5  = float(self.Input5Text.GetValue())
            v6  = float(self.Input6Text.GetValue())
            v7  = float(self.Input7Text.GetValue())
            v8  = float(self.Input8Text.GetValue())
            v9  = float(self.Input9Text.GetValue())
            v10 = float(self.Input10Text.GetValue())
            v11 = float(self.Input11Text.GetValue())
            v12 = float(self.Input12Text.GetValue())
            v13 = float(self.Input13Text.GetValue())
            v14 = float(self.Input14Text.GetValue())
            v15 = float(self.Input15Text.GetValue())
            v16 = float(self.Input16Text.GetValue())
            self.values =  v0, v2, v3, v5, v6, v7, v8, v9, v10, v11,\
                    v12, v13, v14, v15, v16

        elif self.lselect1 == 1:
            v0  = float(self.Input0Text.GetValue())
```

```
        v3  = float(self.Input3Text.GetValue())
        v4  = float(self.Input4Text.GetValue())
        v5  = float(self.Input5Text.GetValue())
        v6  = float(self.Input6Text.GetValue())
        v7  = float(self.Input7Text.GetValue())
        v8  = float(self.Input8Text.GetValue())
        v9  = float(self.Input9Text.GetValue())
        v10 = float(self.Input10Text.GetValue())
        v11 = float(self.Input11Text.GetValue())
        v12 = float(self.Input12Text.GetValue())
        v13 = float(self.Input13Text.GetValue())
        v14 = float(self.Input14Text.GetValue())
        v15 = float(self.Input15Text.GetValue())
        v16 = float(self.Input16Text.GetValue())
        v17 = float(self.Input17Text.GetValue())
        self.values =  v0, v3, v4, v5, v6, v7, v8, v9, v10, v11,\
                v12, v13, v14, v15, v16, v17

    elif self.lselect1 == 2:
        for eachText3 in [self.Input0Text, self.Input4Text, self.Input17Text]:
            eachText3.Enable(False)
        v1  = float(self.Input1Text.GetValue())
        v2  = float(self.Input2Text.GetValue())
        v3  = float(self.Input3Text.GetValue())
        v5  = float(self.Input5Text.GetValue())
        v6  = float(self.Input6Text.GetValue())
        v7  = float(self.Input7Text.GetValue())
        v8  = float(self.Input8Text.GetValue())
        v9  = float(self.Input9Text.GetValue())
        v10 = float(self.Input10Text.GetValue())
        v11 = float(self.Input11Text.GetValue())
        v12 = float(self.Input12Text.GetValue())
        v13 = float(self.Input13Text.GetValue())
        v14 = float(self.Input14Text.GetValue())
        v15 = float(self.Input15Text.GetValue())
        v16 = float(self.Input16Text.GetValue())
        self.values =  v1, v2, v3, v5, v6, v7, v8, v9, v10, v11,\
                v12, v13, v14, v15, v16
```

```
elif self.lselect1 == 3:
    v1  = float(self.Input1Text.GetValue())
    v3  = float(self.Input3Text.GetValue())
    v4  = float(self.Input4Text.GetValue())
    v5  = float(self.Input5Text.GetValue())
    v6  = float(self.Input6Text.GetValue())
    v7  = float(self.Input7Text.GetValue())
    v8  = float(self.Input8Text.GetValue())
    v9  = float(self.Input9Text.GetValue())
    v10 = float(self.Input10Text.GetValue())
    v11 = float(self.Input11Text.GetValue())
    v12 = float(self.Input12Text.GetValue())
    v13 = float(self.Input13Text.GetValue())
    v14 = float(self.Input14Text.GetValue())
    v15 = float(self.Input15Text.GetValue())
    v16 = float(self.Input16Text.GetValue())
    v17 = float(self.Input17Text.GetValue())
    self.values =  v1, v3, v4, v5, v6, v7, v8, v9, v10, v11,\
            v12, v13, v14, v15, v16, v17

self.sv = asarray(self.values)

# Save away the edited text
# Open the file, do an RU sure check for an overwrite!
dlg = wx.FileDialog(self, "Choose a file", self.dirname, self.methodnumber, \
                    ".txt", \
        wx.SAVE | wx.OVERWRITE_PROMPT)
if dlg.ShowModal() == wx.ID_OK:
    # Grab the content to be saved

      # Open the file for write, write, close
    self.filename=dlg.GetFilename()
    self.dirname=dlg.GetDirectory()
    filehandle=open(os.path.join(self.dirname, self.filename),"w")
    for i in range(len(self.sv)):
        filehandle.write("%f " %(self.sv[i]))
    filehandle.close()
# Get rid of the dialog to keep things tidy
dlg.Destroy()
```

```
################################################################################
    def OnRun(self, event):
        if self.lselect1 == 0:

            # Our given variables will be imported from a set-up set: method1.txt

            def objmethod1(xfree, args = None):

                f = open("method_1.txt")
                lines = f.readlines()
                f.close()

            # Make data from input file into usable data
                for line in lines:
                    strlist = line.split()
                    D       = float(strlist[0])         # Diameter              [Feet]
                    RT      = float(strlist[1])         # Resistance            [lbs]
                    Z       = float(strlist[2])         # Number of Blades
                    V       = float(strlist[3])         # Ship Speed            [knots]
                    w       = float(strlist[4])         # Wake Fraction         \
                    Typical value for Sub Cavitating Prop w = 0.1 - 0.3
                    t       = float(strlist[5])         # Thrust Deduction      \
                    Typical value for Sub Cavitating Prop t = 0.05 - 0.1
                    PDmin   = float(strlist[6])         # Minimum Pitch Ratio    \
                    Constraint
                    PDmax   = float(strlist[7])         # Maximum Pitch Ratio    \
                    Constraint
                    Jmin    = float(strlist[8])         # Minimum Advance Coefficient\
                    Constraint
                    Jmax    = float(strlist[9])         # Maximum Advance Coefficient\
                    Constraint
                    h       = float(strlist[10])        # Distance to propeller shaft\
                    centerline
                    EARmin  = float(strlist[11])        # min EAR
                    EARmax  = float(strlist[12])        # max EAR
                    g       = float(strlist[13])        # gravitational acceleration
                    T       = float(strlist[14])        # temperature
```

```
V1  = V*1.688           # convert speed from knots to fps
J   = xfree[0]
PD  = xfree[1]
EAR = xfree[2]
# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-\
0.0001*(T**2)+0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+\
2e-08*(T**2)-1e-06*T + 4e-05

# Calculated data for givens:
VA  = V1*(1-w)
EHP = RT* V1

n = VA / (xfree[0]* D)

# Compute chord length
CR  = 2.073*xfree[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12   # Convert to inches

# compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

KTC = KTKQ_Evaluation23.KTC(xfree[0], xfree[1], xfree[2], Z)
KQC = KTKQ_Evaluation23.KQC(xfree[0], xfree[1], xfree[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xfree[0], xfree[1], xfree[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xfree[0], xfree[1], xfree[2], Z, RN)

KT  = KTC + DKT
KQ  = KQC + DKQ

# Calculate Thrust and Torque
Th   = KT*rho*(n**2)*(D**4)
Q    = KQ*rho*(n**2)*(D**5)

CT   = EHP / (rho*(1-t)*((1-w)**2)*(V1**3)*(D**2))
```

```
pai = 14.7                      # [psi]
pa  = pai*144                   # [psf]
pvi = (2e-11)*(T**5) - (1e-9)*(T**4) + (1e-6)*T**3 - (7e-5)*(T**2)+ \
0.0061*T - 0.0726  # [psi]
pv  = pvi*144                   # [psf]

#Th   = KT*rho*(n**2)*(D**4)      # [lbs]
p0   = pa + (rho * g * h)        # [psf]

# k = 0 for normal vessels and k = 0.2 for single screw, high loading
k = 0.2

EARc = (((1.3 + 0.3 * Z)*Th)/((p0-pv)*((D)**2)))+k

p07  = pa + (rho * g * h)
VA2  = (VA**2)
VA22 = (pi*0.7*D*n)**2
v1   = sqrt(VA2 + VA22)
q07  = (0.5*rho*(v1**2))
sigma07 = (p07-pv)/q07
tau = 0.3*sqrt(sigma07) - 0.03
Ap = Th / (q07 * tau)
Ad = Ap / (1.067-0.229 * xfree[1])
Ao = pi*(D**2)/4

EARb = Ad/Ao

# Our given range constants:
# remain in these ranges
# 2 <= Z <= 7
# 0.3 <= EAR <= 1.05
# 0.5 <= PD <= 1.4
# Outside these ranges the values become unrealiable

# Set-up Constraints for Propellers
constrvalue = []

g1 = xfree[1] - PDmin
```

```
        constrvalue.append(g1)

        g2 = PDmax - xfree[1]
        constrvalue.append(g2)

        g3 = xfree[0] - Jmin
        constrvalue.append(g3)

        g4 = Jmax - xfree[0]
        constrvalue.append(g4)

        g5 = KT - (CT * xfree[0]**2)
        constrvalue.append(g5)

        g6 = xfree[2] - EARmin
        constrvalue.append(g6)

        g7 = EARmax - xfree[2]
        constrvalue.append(g7)

        g8 = xfree[2] - EARc
        constrvalue.append(g8)

        g9 = xfree[2] - EARb
        constrvalue.append(g9)

        objmethod1 = -(xfree[0]* KT / (2*pi*KQ))

    #    print objmethod1, EARb, EARc
    #    print objmethod1
    #    print xfree
    #    print topt, tmin, BSHP

        for constr in constrvalue:
            if constr<0:
                objmethod1 = objmethod1 + 11.0 * ((constr**2))
        return objmethod1

    # Begin Optimization
```

```
if __name__ == "__main__":



    x0 = [ .7, .7, .7 ]
    xsolution, fopt, niter, ncalls, error \
            = fmin(objmethod1, x0, xtol=1e-8, full_output=1, disp=0)
xs = abs(fopt)
# print xsolution, xs, niter, ncalls, error
print "J                ", xsolution[0]
print "PD               ", xsolution[1]
print "EAR              ", xsolution[2]
print "Max Efficency    ", xs

f = open("method_1.txt")
lines = f.readlines()
f.close()

for line in lines:
    strlist = line.split()
    D   = float(strlist[0])         # Diameter              [Feet]
    Z   = float(strlist[2])
    V   = float(strlist[3])         # Ship Speed            [knots]
    w   = float(strlist[4])         # Wake Fraction         Typical \
    value for Sub Cavitating Prop w = 0.1 - 0.3
    h   = float(strlist[10])        # Distance to propeller shaft \
    centerline
    T   = float(strlist[14])

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*\
(T**2)+0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*\
(T**2)-1e-06*T + 4e-05

V1  = V*1.688               # convert speed from knots to fps
VA  = V1*(1-w)
n   = VA / (xsolution[0]*  D)
```

```
            print "opt RPM          =", n*60, "[rpm]"

            # Compute chord length
            CR  = 2.073*xsolution[2]*D/Z   # Minimum chord length at R-0.75
            tc075 = (0.0185-0.00125*Z)*D/CR

            th = tc075 * CR * 12.   # Convert to inches

                # compute Reynolds number at R-0.75
            RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

            KTC = KTKQ_Evaluation23.KTC(xsolution[0], xsolution[1], xsolution[2], Z)
            KQC = KTKQ_Evaluation23.KQC(xsolution[0], xsolution[1], xsolution[2], Z)
            DKT = KTKQ_Evaluation23.KTCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)
            DKQ = KTKQ_Evaluation23.KQCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)

            KT  = KTC + DKT
            KQ  = KQC + DKQ

            # Calculate Thrust and Torque
            Th  = KT*rho*(n**2)*(D**4)
            Q   = KQ*rho*(n**2)*(D**5)

            self.Jopt    = xsolution[0]
            self.PDopt   = xsolution[1]
            self.EARopt  = xsolution[2]
            self.maxeff  = xs
            self.rpm     = n*60
            self.Thrust  = Th
            self.Torque  = Q


################################################################################
        elif self.lselect1 == 1:
            # Our given variables will be imported from a set-up set: method2.txt

            def objmethod2(xfree, args = None):

                f = open("method_2.txt")
```

```
    lines = f.readlines()
    f.close()

# Make data from input file into usable data
    for line in lines:
        strlist = line.split()
        D       = float(strlist[0])         # Diameter              [Feet]
        Z       = float(strlist[1])         # Number of Blades
        DHP     = float(strlist[2])         # Delivered Horse Power [HP]
        V       = float(strlist[3])         # Ship Speed            [knots]
        w       = float(strlist[4])         # Wake Fraction         Typical\
         value for Sub Cavitating Prop w = 0.1 - 0.3
        t       = float(strlist[5])         # Thrust Deduction      Typical\
         value for Sub Cavitating Prop t = 0.05 - 0.1
        PDmin   = float(strlist[6])         # Minimum Pitch Ratio \
        Constraint
        PDmax   = float(strlist[7])         # Maximum Pitch Ratio  \
        Constraint
        Jmin    = float(strlist[8])         # Minimum Advance Coefficient\
        Constraint
        Jmax    = float(strlist[9])         # Maximum Advance Coefficient\
        Constraint
        h       = float(strlist[10])        # Distance to propeller shaft \
        centerline
        EARmin  = float(strlist[11])        # min EAR
        EARmax  = float(strlist[12])        # max EAR
        g       = float(strlist[13])        # gravitational acceleration
        T       = float(strlist[14])        # temperature
        nr      = float(strlist[15])        # Relative Rotation Efficiency

    V1    = V*1.688       # convert speed from knots to fps
    DHP1  = DHP*550       # convert hp to foot-lbs/second
    J = xfree[0]
    PD = xfree[1]
    EAR = xfree[2]
    # Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
    rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*\
    (T**2)+0.003*T + 1.9679
    # Kinematic viscosity [ft^2/s]
```

```
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*\
(T**2)-1e-06*T + 4e-05

# Calculated data for givens:
VA  = V1*(1-w)
n = xfree[0]* D / VA

# Compute chord length
CR  = 2.073*xfree[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12   # Convert to inches

# compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

KTC = KTKQ_Evaluation23.KTC(xfree[0], xfree[1], xfree[2], Z)
KQC = KTKQ_Evaluation23.KQC(xfree[0], xfree[1], xfree[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xfree[0], xfree[1], xfree[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xfree[0], xfree[1], xfree[2], Z, RN)

KT  = KTC + DKT
KQ  = KQC + DKQ

#    T = KT*rho[0]*(n**2)*(D[0]**4)
#    Q = KQ*rho[0]*(n**2)*(D[0]**5)

CQ = (DHP1*nr) / (2*pi*rho*((1-w)**3)*(V1**3)*(D**2))

pai = 14.7                      # [psi]
pa  = pai*144                   # [psf]
pvi = (2.e-11)*(T**5) - (1.e-9)*(T**4) + (1.e-6)*T**3 - (7.e-5)*(T**2)\
         + 0.0061*T - 0.0726    # [psi]
pv  = pvi*144                   # [psf]

Th   = KT*rho*(n**2)*(D**4)     # [lbs]
p0  = pa + (rho * g * h)        # [psf]

# k = 0 for normal vessels and k = 0.2 for single screw, high loading
```

```
k = 0.2

EARc = (((1.3 + 0.3 * Z)*Th)/((p0-pv)*((D)**2)))+k

p07  = pa + (rho * g * h)
VA2  = (VA**2)
VA22 = (pi*0.7*D*n)**2
v1   = sqrt(VA2 + VA22)
q07  = (0.5*rho*(v1**2))
sigma07 = (p07-pv)/q07
tau = 0.3*sqrt(sigma07) - 0.03
Ap = Th / (q07 * tau)
Ad = Ap / (1.067-0.229 * xfree[1])
Ao = pi*(D**2)/4

EARb = Ad/Ao

# Our given range constants:
# remain in these ranges
# 2 <= Z <= 7
# 0.3 <= EAR <= 1.05
# 0.5 <= PD <= 1.4
# Outside these ranges the values become unrealiable

# Set-up Constraints for Propellers
constrvalue = []

g1 = xfree[1] - PDmin
constrvalue.append(g1)

g2 = PDmax - xfree[1]
constrvalue.append(g2)

g3 = xfree[2] - EARmin
constrvalue.append(g3)

g4 = EARmax - xfree[2]
constrvalue.append(g4)
```

```
    g5 = xfree[0] - Jmin
    constrvalue.append(g5)

    g6 = Jmax - xfree[0]
    constrvalue.append(g6)

    g7 = KQ - (CQ * xfree[0]**3)
    constrvalue.append(g7)

    g8 = xfree[2] - EARc
    constrvalue.append(g8)

    g9 = xfree[2] - EARb
    constrvalue.append(g9)

    objmethod2 = -(xfree[0]* KT / (2*pi*KQ))

#    print objmethod2
#    print xfree, CQ

    for constr in constrvalue:
        if constr<0:
            objmethod2 = objmethod2 + 122.8 * ((constr**2))
    return objmethod2

# Begin Optimization
if __name__ == "__main__":

    x0 = [ .6, .6, .6]
    xsolution, fopt, niter, ncalls, error \
            = fmin(objmethod2, x0, xtol = 1e-08, full_output=1, disp=0)
xs = abs(fopt)
# print xsolution, fopt, niter, ncalls, error
print "J               =", xsolution[0]
print "PD              =", xsolution[1]
print "EAR             =", xsolution[2]
print "max NO          =", xs

f = open("method_2.txt")
```

```
lines = f.readlines()
f.close()

for line in lines:
    strlist = line.split()
    D   = float(strlist[0])         # Diameter              [Feet]
    Z   = float(strlist[1])         # Number of Blades
    V   = float(strlist[3])         # Ship Speed            [knots]
    w   = float(strlist[4])         # Wake Fraction         Typical \
    value for Sub Cavitating Prop w = 0.1 - 0.3
    h   = float(strlist[10])        # Distance to propeller shaft centerline
    T   = float(strlist[14])        # temperature

V1  = V*1.688               # convert speed from knots to fps
VA  = V1*(1-w)
n   = VA / (xsolution[0]*  D)

print "opt RPM          =", n*60, "[rpm]"

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*(T**2)+\
0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*(T**2)-\
1e-06*T + 4e-05

# Compute chord length
CR  = 2.073*xsolution[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12.   # Convert to inches

    # compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

KTC = KTKQ_Evaluation23.KTC(xsolution[0], xsolution[1], xsolution[2], Z)
KQC = KTKQ_Evaluation23.KQC(xsolution[0], xsolution[1], xsolution[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)
```

```
            KT  = KTC + DKT
            KQ  = KQC + DKQ

            # Calculate Thrust and Torque
            Th  = KT*rho*(n**2)*(D**4)
            Q   = KQ*rho*(n**2)*(D**5)

            self.Jopt    = xsolution[0]
            self.PDopt   = xsolution[1]
            self.EARopt  = xsolution[2]
            self.maxeff  = xs
            self.rpm     = n*60
            self.Thrust  = Th
            self.Torque  = Q

########################################################################################################################
        elif self.lselect1 == 2:

            # Our given variables will be imported from a set-up set: method3.txt

            def objmethod3(xfree, args = None):

                f = open("method_3.txt")
                lines = f.readlines()
                f.close()

            # Make data from input file into usable data
                for line in lines:
                    strlist = line.split()
                    nrpm    = float(strlist[0])         # Operating RPM
                    RT      = float(strlist[1])         # Resistance            [lbs]
                    Z       = float(strlist[2])         # Number of Blades
                    V       = float(strlist[3])         # Ship Speed            [knots]
                    w       = float(strlist[4])         # Wake Fraction         Typical\
                     value for Sub Cavitating Prop w = 0.1 - 0.3
                    t       = float(strlist[5])         # Thrust Deduction      Typical\
                     value for Sub Cavitating Prop t = 0.05 - 0.1
                    PDmin   = float(strlist[6])         # Minimum Pitch Ratio         \
```

```
     Constraint
    PDmax   = float(strlist[7])         # Maximum Pitch Ratio         \
    Constraint
    Jmin    = float(strlist[8])         # Minimum Advance Coefficient \
    Constraint
    Jmax    = float(strlist[9])         # Maximum Advance Coefficient \
    Constraint
    h       = float(strlist[10])        # Distance to propeller shaft \
    centerline
    EARmin  = float(strlist[11])        # min EAR
    EARmax  = float(strlist[12])        # max EAR
    g       = float(strlist[13])        # Gravitational acceleration
    T       = float(strlist[14])        # Temperature

V1  = V*1.688                # convert speed from knots to fps
J   = xfree[0]
PD  = xfree[1]
EAR = xfree[2]
# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*(T**2)+\
0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*(T**2)-\
1e-06*T + 4e-05

n   = nrpm/60               # convert rpm to rps

# Calculated data for givens:
VA = V1 * (1-w)
D  = VA / (xfree[0] * n)

# Compute chord length
CR  = 2.073*xfree[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12   # Convert to inches

#compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv
```

```
KTC = KTKQ_Evaluation23.KTC(xfree[0], xfree[1], xfree[2], Z)
KQC = KTKQ_Evaluation23.KQC(xfree[0], xfree[1], xfree[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xfree[0], xfree[1], xfree[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xfree[0], xfree[1], xfree[2], Z, RN)

KT  = KTC + DKT
KQ  = KQC + DKQ

#Calculate Thrust and Torque
#    T = KT*rho[0]*(n**2)*(D[0]**4)
#    Q = KQ*rho[0]*(n**2)*(D[0]**5)

CT = (RT*n**2) / (rho*(1-t)*((1-w)**4)*(V1**4))

pai = 14.7                      # [psi]
pa  = pai*144                   # [psf]
pvi = (2.e-11)*(T**5) - (1.e-9)*(T**4) + (1.e-6)*T**3 - (7.e-5)*(T**2)\
         + 0.0061*T - 0.0726    # [psi]
pv  = pvi*144                   # [psf]

Th   = KT*rho*(n**2)*(D**4)      # [lbs]
p0  = pa + (rho * g * h)        # [psf]

# k = 0 for normal vessels and k = 0.2 for single screw, high loading
k = 0.2

EARc = (((1.3 + 0.3 * Z) * Th) / ((p0-pv)*((D)**2))) + k

p07  = pa + (rho * g * h)
VA2  = (VA**2)
VA22 = (pi*0.7*D*n)**2
v1   = sqrt(VA2 + VA22)
q07  = (0.5*rho*(v1**2))
sigma07 = (p07-pv)/q07
tau = 0.3*sqrt(sigma07) - 0.03
Ap = Th / (q07 * tau)
Ad = Ap / (1.067-0.229 * xfree[1])
Ao = pi*(D**2)/4
```

```
EARb = Ad/Ao

# Our given range constants:
# remain in these ranges
# 2 <= Z <= 7
# 0.3 <= EAR <= 1.05
# 0.5 <= PD <= 1.4
# Outside these ranges the values become unrealiable

# Set-up Constraints for Propellers
constrvalue = []

g1 = xfree[2] - EARmin
constrvalue.append(g1)

g2 = EARmax - xfree[2]
constrvalue.append(g2)

g3 = xfree[1] - PDmin
constrvalue.append(g3)

g4 = PDmax - xfree[1]
constrvalue.append(g4)

g5 = xfree[0] - Jmin
constrvalue.append(g5)

g6 = Jmax - xfree[0]
constrvalue.append(g6)

g7 = xfree[2] - EARc
constrvalue.append(g7)

g8 = xfree[2] - EARb
constrvalue.append(g8)

g9 = KT - (CT * xfree[0]**4)
constrvalue.append(g9)
```

```
    objmethod3 = -(xfree[0]* KT / (2*pi*KQ))

#    print objmethod3
#    print xfree,CT

    for constr in constrvalue:
        if constr<0:
            objmethod3 = objmethod3 + 3.3 * ((constr**2))
    return objmethod3

if __name__ == "__main__":

    x0 = [ .6, .6, .6]
    xsolution, fopt, niter, ncalls, error \
            = fmin(objmethod3, x0, xtol = 1e-8, full_output=1, disp=0)
xs = abs(fopt)
# print xsolution, xs, niter, ncalls, error
print "J                =", xsolution[0]
print "PD               =", xsolution[1]
print "EAR              =", xsolution[2]
print "max Efficiency   =", xs

f = open("method_3.txt")
lines = f.readlines()
f.close()
for line in lines:
    strlist = line.split()
    nrpm    = float(strlist[0])
    Z       = float(strlist[2])         # Number of Blades
    V       = float(strlist[3])
    w       = float(strlist[4])
    h       = float(strlist[10])        # Distance to propeller shaft \
    centerline
    T       = float(strlist[14])        # Temperature

n  = nrpm/60
V1 = V*1.688
VA = V1 * (1-w)
```

```
D  = VA / (xsolution[0] * n)

print "optimum diameter =", D, "[feet]"

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*(T**2)\
+0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*(T**2)\
-1e-06*T + 4e-05

# Compute chord length
CR  = 2.073*xsolution[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12.   # Convert to inches

# compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

KTC = KTKQ_Evaluation23.KTC(xsolution[0], xsolution[1], xsolution[2], Z)
KQC = KTKQ_Evaluation23.KQC(xsolution[0], xsolution[1], xsolution[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)

KT  = KTC + DKT
KQ  = KQC + DKQ

# Calculate Thrust and Torque
Th  = KT*rho*(n**2)*(D**4)
Q   = KQ*rho*(n**2)*(D**5)

self.Jopt    = xsolution[0]
self.PDopt   = xsolution[1]
self.EARopt  = xsolution[2]
self.maxeff  = xs
self.D       = D
self.Thrust  = Th
self.Torque  = Q
```

```
################################################################################
        elif self.lselect1 == 3:

            # Our given variables will be imported from a set-up set: method4.txt

            def objmethod4(xfree, args = None):

                f = open("method_4.txt")
                lines = f.readlines()
                f.close()

            # Make data from input file into usable data
                for line in lines:
                    strlist = line.split()
                    nrpm    = float(strlist[0])         # Operating RPM
                    Z       = float(strlist[1])         # Number of Blades
                    DHP     = float(strlist[2])         # Delivered Horse Power [HP]
                    V       = float(strlist[3])         # Ship Speed            [knots]
                    w       = float(strlist[4])         # Wake Fraction         Typical \
                    value for Sub Cavitating Prop w = 0.1 - 0.3
                    t       = float(strlist[5])         # Thrust Deduction      Typical \
                    value for Sub Cavitating Prop t = 0.05 - 0.1
                    PDmin   = float(strlist[6])         # Minimum Pitch Ratio           \
                    Constraint
                    PDmax   = float(strlist[7])         # Maximum Pitch Ratio           \
                    Constraint
                    Jmin    = float(strlist[8])         # Minimum Advance Coefficient   \
                    Constraint
                    Jmax    = float(strlist[9])         # Maximum Advance Coefficient   \
                    Constraint
                    h       = float(strlist[10])        # Distance to propeller shaft \
                    centerline
                    EARmin  = float(strlist[11])        # min EAR
                    EARmax  = float(strlist[12])        # max EAR
                    g       = float(strlist[13])        # Gravitational acceleration
                    T       = float(strlist[14])        # Temperature
                    nr      = float(strlist[15])        # Relative Rotation Efficiency
```

```
V1   = V*1.688                # convert speed from knots to fps
DHP1 = DHP*550               # convert hp to foot-lbs/second
J = xfree[0]
PD = xfree[1]
EAR = xfree[2]
# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*\
(T**2)+0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*\
(T**2)-1e-06*T + 4e-05

n = nrpm/60

# Calculated data for givens:
VA = V1 * (1-w)
D  = VA / (xfree[0] * n)

# Compute chord length
CR  = 2.073*xfree[2]*D/Z   # Minimum chord length at R-0.75
tc075 = (0.0185-0.00125*Z)*D/CR

th = tc075 * CR * 12   # Convert to inches

# compute Reynolds number at R-0.75
RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv



KTC = KTKQ_Evaluation23.KTC(xfree[0], xfree[1], xfree[2], Z)
KQC = KTKQ_Evaluation23.KQC(xfree[0], xfree[1], xfree[2], Z)
DKT = KTKQ_Evaluation23.KTCRC(xfree[0], xfree[1], xfree[2], Z, RN)
DKQ = KTKQ_Evaluation23.KQCRC(xfree[0], xfree[1], xfree[2], Z, RN)

KT  = KTC + DKT
KQ  = KQC + DKQ



#    T = KT*rho[0]*(n**2)*(D[0]**4)
#    Q = KQ*rho[0]*(n**2)*(D[0]**5)
```

```
CQ = (DHP1*(n**2)*nr) / (2*pi*rho*((1-w)**5)*(V1**5))

pai = 14.7                      # [psi]
pa  = pai*144                   # [psf]
pvi = (2.e-11)*(T**5) - (1.e-9)*(T**4) + (1.e-6)*T**3 - (7.e-5)*(T**2)\
         + 0.0061*T - 0.0726    # [psi]
pv  = pvi*144                   # [psf]

Th   = KT*rho*(n**2)*(D**4)      # [lbs]
p0  = pa + (rho * g * h)        # [psf]

# k = 0 for normal vessels and k = 0.2 for single screw, high loading
k = 0.2

EARc = (((1.3 + 0.3 * Z) * Th) / ((p0-pv)*((D)**2))) + k

p07  = pa + (rho * g * h)
VA2  = (VA**2)
VA22 = (pi*0.7*D*n)**2
v1   = sqrt(VA2 + VA22)
q07  = (0.5*rho*(v1**2))
sigma07 = (p07-pv)/q07
tau = 0.3*sqrt(sigma07) - 0.03
Ap = Th / (q07 * tau)
Ad = Ap / (1.067-0.229 * xfree[1])
Ao = pi*(D**2)/4

EARb = Ad/Ao

# Our given range constants:
# remain in these ranges
# 2 <= Z <= 7
# 0.3 <= EAR <= 1.05
# 0.5 <= PD <= 1.4
# Outside these ranges the values become unrealiable

# Set-up Constraints for Propellers
constrvalue = []
```

```
    g1 = xfree[2] - EARmin
    constrvalue.append(g1)

    g2 = EARmax - xfree[2]
    constrvalue.append(g2)

    g3 = xfree[1] - PDmin
    constrvalue.append(g3)

    g4 = PDmax - xfree[1]
    constrvalue.append(g4)

    g5 = xfree[0] - Jmin
    constrvalue.append(g5)

    g6 = Jmax - xfree[0]
    constrvalue.append(g6)

    g7 = xfree[2] - EARc
    constrvalue.append(g7)

    g8 = xfree[2] - EARb
    constrvalue.append(g8)

    g9 = KQ - (CQ * xfree[0]**5)
    constrvalue.append(g9)

    objmethod4 = -(xfree[0]* KT / (2*pi*KQ))

#    print objmethod4
#    print xfree,CT

    for constr in constrvalue:
        if constr<0:
            objmethod4 = objmethod4 + 44.5 * ((constr**2))
    return objmethod4

if __name__ == "__main__":
```

```
    x0 = [ .7, .7, .7]
    xsolution, fopt, niter, ncalls, error \
            = fmin(objmethod4, x0, xtol = 1e-8, full_output=1, disp=0)
xs = abs(fopt)
# print xsolution, xs, niter, ncalls, error

print "J                ="", xsolution[0]
print "PD               ="", xsolution[1]
print "EAR              ="", xsolution[2]
print "Max Efficiency   ="", xs


f = open("method_4.txt")
lines = f.readlines()
f.close()
for line in lines:
    strlist = line.split()
    nrpm    = float(strlist[0])
    Z       = float(strlist[1])         # Number of Blades
    V       = float(strlist[3])
    w       = float(strlist[4])
    h       = float(strlist[10])        # Distance to propeller shaft \
    centerline
    T       = float(strlist[14])        # Temperature

n  = nrpm/60
V1 = V*1.688
VA = V1 * (1.-w)
D  = VA / (xsolution[0] * n)

print "optimum diameter =", D, "[feet]"

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = -5e-13*(T**6) + 2e-10*(T**5) -4e-08*(T**4)+3e-06*(T**3)-0.0001*\
(T**2)+0.003*T + 1.9679
# Kinematic viscosity [ft^2/s]
kv = 3e-17*(T**6) - 1e-14*(T**5) + 3e-12*(T**4)-3e-10*(T**3)+2e-08*\
(T**2)-1e-06*T + 4e-05
```

```
            # Calculated data for givens:
            VA = V1 * (1-w)
            D  = VA / (xsolution[0] * n)

            # Compute chord length
            CR  = 2.073*xsolution[2]*D/Z   # Minimum chord length at R-0.75
            tc075 = (0.0185-0.00125*Z)*D/CR

            th = tc075 * CR * 12   # Convert to inches

            # compute Reynolds number at R-0.75
            RN  = CR * ((VA**2 + ((0.75*pi*n*D)**2))**0.5) / kv

            KTC = KTKQ_Evaluation23.KTC(xsolution[0], xsolution[1], xsolution[2], Z)
            KQC = KTKQ_Evaluation23.KQC(xsolution[0], xsolution[1], xsolution[2], Z)
            DKT = KTKQ_Evaluation23.KTCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)
            DKQ = KTKQ_Evaluation23.KQCRC(xsolution[0], xsolution[1], xsolution[2], Z, RN)

            KT  = KTC + DKT
            KQ  = KQC + DKQ

            Th = KT*rho*(n**2)*(D**4)
            Q = KQ*rho*(n**2)*(D**5)

            self.Jopt    = xsolution[0]
            self.PDopt   = xsolution[1]
            self.EARopt  = xsolution[2]
            self.maxeff  = xs
            self.D       = D
            self.Thrust  = Th
            self.Torque  = Q


# Need some help here!
   def OnResults(self, event):
       if self.lselect1 == 0:
           results ="%15s = %6.4f\n\
 %15s = %6.4f\n\
```

```
%15s = %6.4f\n\
           %15s = %6.4f\n\
        %15s = %6.4f\n\
         %15s = %6.4f\n\
           %15s = %6.4f" \
         % ('J', self.Jopt, 'P/D', self.PDopt, 'EAR', self.EARopt,'Max Efficiency',\
             self.maxeff, 'Optimum RPM', self.rpm, 'Thrust [lbs]', self.Thrust, \
             'Torque [ft lb]', self.Torque )
          dialog = wx.MessageBox (results, 'Results', style = wx.OK )
      elif self.lselect1 == 1:
          results1 ="%15s = %6.4f\n\
%15s = %6.4f\n\
%15s = %6.4f\n\
           %15s = %6.4f\n\
         %15s = %6.4f\n\
          %15s = %6.4f\n\
           %15s = %6.4f" \
          % ('J', self.Jopt, 'P/D', self.PDopt, 'EAR', self.EARopt,'Max Efficiency', \
             self.maxeff, 'Optimum RPM', self.rpm, 'Thrust [lbs]', self.Thrust,\
              'Torque [ft lb]', self.Torque )
          dialog1 = wx.MessageBox (results1, 'Results', style = wx.OK )
      elif self.lselect1 == 2:
          results2 ="%15s = %6.4f\n\
%15s = %6.4f\n\
%15s = %6.4f\n\
           %15s = %6.4f\n\
            %15s = %6.4f\n\
          %15s = %6.4f\n\
           %15s = %6.4f" \
          % ('J', self.Jopt, 'P/D', self.PDopt, 'EAR', self.EARopt,'Max Efficiency', \
             self.maxeff, 'Optimum Diameter', self.D, 'Thrust [lbs]', self.Thrust, \
             'Torque [ft lb]', self.Torque )
          dialog2 = wx.MessageBox (results2, 'Results', style = wx.OK )
      elif self.lselect1 == 3:
          results3 ="%15s = %6.4f\n\
%15s = %6.4f\n\
%15s = %6.4f\n\
           %15s = %6.4f\n\
            %15s = %6.4f\n\
```

```
            %15s = %6.4f\n\
              %15s = %6.4f" \
             % ('J', self.Jopt, 'P/D', self.PDopt, 'EAR', self.EARopt,'Max Efficiency',\
                 self.maxeff, 'Optimum Diameter', self.D, 'Thrust [lbs]', self.Thrust,\
                  'Torque [ft lb]', self.Torque  )
             dialog3 = wx.MessageBox (results3, 'Results', style = wx.OK )

if __name__ == '__main__':
    app = wx.PySimpleApp()
    TextFrame().Show()
    app.MainLoop()
```

B.3 KTKQEvaluation.py

```
#Calculation of KT from B-series data
#J=input("Enter advance co-efficient 'J' : ")
#PD=input("Enter pitch-diamter ratio 'P/D' : ")
#EAR=input("Enter expanded area ratio 'Ae/Ao' : ")
#Z=input("Enter number of blades 'z' : ")


def KTC(J, PD, EAR, Z):


    KT1 = 0.00880496
    KT2 = -0.204554*(J**1)*(PD**0)*(EAR**0)*(Z**0)
    KT3 = 0.166351*(J**0)*(PD**1)*(EAR**0)*(Z**0)
    KT4 = 0.158114*(J**0)*(PD**2)*(EAR**0)*(Z**0)
    KT5 = -0.147581*(J**2)*(PD**0)*(EAR**1)*(Z**0)

    KT6 = -0.481497*(J**1)*(PD**1)*(EAR**1)*(Z**0)
    KT7 = 0.415437*(J**0)*(PD**2)*(EAR**1)*(Z**0)
    KT8 = 0.0144043*(J**0)*(PD**0)*(EAR**0)*(Z**1)
    KT9 = -0.0530054*(J**2)*(PD**0)*(EAR**0)*(Z**1)
    KT10 = 0.0143481*(J**0)*(PD**1)*(EAR**0)*(Z**1)

    KT11 = 0.0606826*(J**1)*(PD**1)*(EAR**0)*(Z**1)
```

```
KT12 = -0.0125894*(J**0)*(PD**0)*(EAR**1)*(Z**1)
KT13 = 0.0109689*(J**1)*(PD**0)*(EAR**1)*(Z**1)
KT14 = -0.133698*(J**0)*(PD**3)*(EAR**0)*(Z**0)
KT15 = 0.00638407*(J**0)*(PD**6)*(EAR**0)*(Z**0)

KT16 = -0.00132718*(J**2)*(PD**6)*(EAR**0)*(Z**0)
KT17 = 0.168496*(J**3)*(PD**0)*(EAR**1)*(Z**0)
KT18 = -0.0507214*(J**0)*(PD**0)*(EAR**2)*(Z**0)
KT19 = 0.0854559*(J**2)*(PD**0)*(EAR**2)*(Z**0)
KT20 = -0.0504475*(J**3)*(PD**0)*(EAR**2)*(Z**0)

KT21 = 0.010465*(J**1)*(PD**6)*(EAR**2)*(Z**0)
KT22 = -0.00648272*(J**2)*(PD**6)*(EAR**2)*(Z**0)
KT23 = -0.00841728*(J**0)*(PD**3)*(EAR**0)*(Z**1)
KT24 = 0.0168424*(J**1)*(PD**3)*(EAR**0)*(Z**1)
KT25 = -0.00102296*(J**3)*(PD**3)*(EAR**0)*(Z**1)

KT26 = -0.0317791*(J**0)*(PD**3)*(EAR**1)*(Z**1)
KT27 = 0.018604*(J**1)*(PD**0)*(EAR**2)*(Z**1)
KT28 = -0.00410798*(J**0)*(PD**2)*(EAR**2)*(Z**1)
KT29 = -0.000606848*(J**0)*(PD**0)*(EAR**0)*(Z**2)
KT30 = -0.0049819*(J**1)*(PD**0)*(EAR**0)*(Z**2)

KT31 = 0.0025983*(J**2)*(PD**0)*(EAR**0)*(Z**2)
KT32 = -0.000560528*(J**3)*(PD**0)*(EAR**0)*(Z**2)
KT33 = -0.00163652*(J**1)*(PD**2)*(EAR**0)*(Z**2)
KT34 = -0.000328787*(J**1)*(PD**6)*(EAR**0)*(Z**2)
KT35 = 0.000116502*(J**2)*(PD**6)*(EAR**0)*(Z**2)

KT36 = 0.000690904*(J**0)*(PD**0)*(EAR**1)*(Z**2)
KT37 = 0.00421749*(J**0)*(PD**3)*(EAR**1)*(Z**2)
KT38 = 0.0000565229*(J**3)*(PD**6)*(EAR**1)*(Z**2)
KT39 = -0.00146564*(J**0)*(PD**3)*(EAR**2)*(Z**2)

KTTOTAL = KT1+KT2+KT3+KT4+KT5+KT6+KT7+KT8+KT9+KT10\
          +KT11+KT12+KT13+KT14+KT15+KT16+KT17+KT18+KT19\
          +KT20+KT21+KT22+KT23+KT24+KT25+KT26+KT27+KT28\
          +KT29+KT30+KT31+KT32+KT33+KT34+KT35+KT36+KT37\
          +KT38+KT39
```

```
    return KTTOTAL

def KQC(J, PD, EAR, Z):

    KQ1 = 0.00379368
    KQ2 = 0.00886523*(J**2)*(PD**0)*(EAR**0)*(Z**0)
    KQ3 = -0.032241*(J**1)*(PD**1)*(EAR**0)*(Z**0)
    KQ4 = 0.00344778*(J**0)*(PD**2)*(EAR**0)*(Z**0)
    KQ5 = -0.0408811*(J**0)*(PD**1)*(EAR**1)*(Z**0)

    KQ6 = -0.108009*(J**1)*(PD**1)*(EAR**1)*(Z**0)
    KQ7 = -0.0885381*(J**2)*(PD**1)*(EAR**1)*(Z**0)
    KQ8 = 0.188561*(J**0)*(PD**2)*(EAR**1)*(Z**0)
    KQ9 = -0.00370871*(J**1)*(PD**0)*(EAR**0)*(Z**1)
    KQ10 = 0.00513696*(J**0)*(PD**1)*(EAR**0)*(Z**1)

    KQ11 = 0.0209449*(J**1)*(PD**1)*(EAR**0)*(Z**1)
    KQ12 = 0.00474319*(J**2)*(PD**1)*(EAR**0)*(Z**1)
    KQ13 = -0.00723408*(J**2)*(PD**0)*(EAR**1)*(Z**1)
    KQ14 = 0.00438388*(J**1)*(PD**1)*(EAR**1)*(Z**1)
    KQ15 = -0.0269403*(J**0)*(PD**2)*(EAR**1)*(Z**1)

    KQ16 = 0.0558082*(J**3)*(PD**0)*(EAR**1)*(Z**0)
    KQ17 = 0.0161886*(J**0)*(PD**3)*(EAR**1)*(Z**0)
    KQ18 = 0.00318086*(J**1)*(PD**3)*(EAR**1)*(Z**0)
    KQ19 = 0.015896*(J**0)*(PD**0)*(EAR**2)*(Z**0)
    KQ20 = 0.0471729*(J**1)*(PD**0)*(EAR**2)*(Z**0)

    KQ21 = 0.0196283*(J**3)*(PD**0)*(EAR**2)*(Z**0)
    KQ22 = -0.0502782*(J**0)*(PD**1)*(EAR**2)*(Z**0)
    KQ23 = -0.030055*(J**3)*(PD**1)*(EAR**2)*(Z**0)
    KQ24 = 0.0417122*(J**2)*(PD**2)*(EAR**2)*(Z**0)
    KQ25 = -0.0397722*(J**0)*(PD**3)*(EAR**2)*(Z**0)

    KQ26 = -0.00350024*(J**0)*(PD**6)*(EAR**2)*(Z**0)
    KQ27 = -0.0106854*(J**3)*(PD**0)*(EAR**0)*(Z**1)
    KQ28 = 0.00110903*(J**3)*(PD**3)*(EAR**0)*(Z**1)
    KQ29 = -0.000313912*(J**0)*(PD**6)*(EAR**0)*(Z**1)
```

```
    KQ30 = 0.0035985*(J**3)*(PD**0)*(EAR**1)*(Z**1)

    KQ31 = -0.00142121*(J**0)*(PD**6)*(EAR**1)*(Z**1)
    KQ32 = -0.00383637*(J**1)*(PD**0)*(EAR**2)*(Z**1)
    KQ33 = 0.0126803*(J**0)*(PD**2)*(EAR**2)*(Z**1)
    KQ34 = -0.00318278*(J**2)*(PD**3)*(EAR**2)*(Z**1)
    KQ35 = 0.00334268*(J**0)*(PD**6)*(EAR**2)*(Z**1)

    KQ36 = -0.00183491*(J**1)*(PD**1)*(EAR**0)*(Z**2)
    KQ37 = 0.000112451*(J**3)*(PD**2)*(EAR**0)*(Z**2)
    KQ38 = -0.0000297228*(J**3)*(PD**6)*(EAR**0)*(Z**2)
    KQ39 = 0.000269551*(J**1)*(PD**0)*(EAR**1)*(Z**2)
    KQ40 = 0.00083265*(J**2)*(PD**0)*(EAR**1)*(Z**2)

    KQ41 = 0.00155334*(J**0)*(PD**2)*(EAR**1)*(Z**2)
    KQ42 = 0.000302683*(J**0)*(PD**6)*(EAR**1)*(Z**2)
    KQ43 = -0.0001843*(J**0)*(PD**0)*(EAR**2)*(Z**2)
    KQ44 = -0.000425399*(J**0)*(PD**3)*(EAR**2)*(Z**2)
    KQ45 = 0.0000869243*(J**3)*(PD**3)*(EAR**2)*(Z**2)

    KQ46 = -0.0004659*(J**0)*(PD**6)*(EAR**2)*(Z**2)
    KQ47 = 0.0000554194*(J**1)*(PD**6)*(EAR**2)*(Z**2)


    KQTOTAL = KQ1+KQ2+KQ3+KQ4+KQ5+KQ6+KQ7+KQ8+KQ9+KQ10\
              +KQ11+KQ12+KQ13+KQ14+KQ15+KQ16+KQ17+KQ18+KQ19\
              +KQ20+KQ21+KQ22+KQ23+KQ24+KQ25+KQ26+KQ27+KQ28\
              +KQ29+KQ30+KQ31+KQ32+KQ33+KQ34+KQ35+KQ36+KQ37\
              +KQ38+KQ39+KQ40+KQ41+KQ42+KQ43+KQ44+KQ45+KQ46+KQ47

    return KQTOTAL

# Calculation of KT and KQ corrections for Reynolds number
#   and blade thickness from B-series data
import math

def KTCRC(J, PD, EAR, Z, RN):

    LRN  = math.log(RN,10)
```

```
    DKT1 =  0.000353485
    DKT2 = -0.00333758*EAR*(J**2)
    DKT3 = -0.00478125*EAR*PD*J
    DKT4 =  0.000257792*((LRN-0.301)**2)*EAR*(J**2)
    DKT5 =  0.0000643192*(LRN-0.301)*(PD**6)*(J**2)
    DKT6 = -0.0000110636*((LRN-0.301)**2)*(PD**6)*(J**2)
    DKT7 = -0.0000276305*((LRN-0.301)**2)*Z*EAR*(J**2)
    DKT8 =  0.0000954*(LRN-0.301)*Z*EAR*PD*J
    DKT9 =  0.0000032049*(LRN-0.301)*(Z**2)*EAR*(PD**3)*J

    DKT = DKT1+DKT2+DKT3+DKT4+DKT5+DKT6+DKT7+DKT8+DKT9

    return DKT

def KQCRC(J, PD, EAR, Z, RN):

    LRN   = math.log(RN,10)

    DKQ1  = -0.000591412
    DKQ2  =  0.00696898*PD
    DKQ3  = -0.0000666654*Z*(PD**6)
    DKQ4  =  0.0160818*(EAR**2)
    DKQ5  = -0.000938091*(LRN-0.301)*PD
    DKQ6  = -0.00059593*(LRN-0.301)*(PD**2)
    DKQ7  =  0.0000782099*((LRN-0.301)**2)*(PD**2)
    DKQ8  =  0.0000052199*(LRN-0.301)*Z*EAR*(J**2)
    DKQ9  = -0.00000088528*((LRN-0.301)**2)*Z*EAR*PD*J
    DKQ10 =  0.0000230171*(LRN-0.301)*Z*(PD**6)
    DKQ11 = -0.00000184341*((LRN-0.301)**2)*Z*(PD**6)
    DKQ12 = -0.00400252*(LRN-0.301)*(EAR**2)
    DKQ13 =  0.000220915*((LRN-0.301)**2)*(EAR**2)

    DKQ = DKQ1+DKQ2+DKQ3+DKQ4+DKQ5+DKQ6+DKQ7+DKQ8+DKQ9+DKQ10+DKQ11+DKQ12+DKQ13

    return DKQ
```

B.4 MTPROPOPT.py

```
#!/usr/bin/env python

# Getting the GUI stuff
import wx
import wx.lib.dialogs
# Optimazation files
from scipy.optimize import *
from numpy import *
from plasi import plasi

# Handling files and directories
import os

class TextFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, "MTPTOPOPT", pos=(0,0),
                size=(1200, 770))
        self.filename = ""
#Create Status bar
        self.CreateStatusBar()

# Begin Toolbar / Menubar
        menubar = wx.MenuBar()

        file = wx.Menu()
        file.Append(-1, '&New')
        menuItem1 = file.Append(-1, "&Save")
        menuItem2 = file.Append(-1, "&Quit")
        file.AppendSeparator()

        helpMenu  = wx.Menu()
        menuItem4 = helpMenu.Append(-1, '&Directions')

        menubar.Append(file, '&File')
        menubar.Append(helpMenu, '&About')
        self.SetMenuBar(menubar)
        self.Bind(wx.EVT_MENU, self.OnSave, menuItem1)
        self.Bind(wx.EVT_MENU, self.OnCloseMe, menuItem2)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
        self.Bind(wx.EVT_MENU, self.OnAbout, menuItem4)

 # Begin List of Input lines
```

```
panel = wx.Panel(self, -1)
ILabel0 = wx.StaticText(panel, -1, " C(r)/R :")
self.IText0  = wx.TextCtrl(panel, -1, "0.4602", size=(50, -1))
self.IText1  = wx.TextCtrl(panel, -1, "0.4942", size=(50, -1))
self.IText2  = wx.TextCtrl(panel, -1, "0.5280", size=(50, -1))
self.IText3  = wx.TextCtrl(panel, -1, "0.5559", size=(50, -1))
self.IText4  = wx.TextCtrl(panel, -1, "0.5830", size=(50, -1))
self.IText5  = wx.TextCtrl(panel, -1, "0.6035", size=(50, -1))
self.IText6  = wx.TextCtrl(panel, -1, "0.6221", size=(50, -1))
self.IText7  = wx.TextCtrl(panel, -1, "0.6340", size=(50, -1))
self.IText8  = wx.TextCtrl(panel, -1, "0.6426", size=(50, -1))
self.IText9  = wx.TextCtrl(panel, -1, "0.6432", size=(50, -1))
self.IText10 = wx.TextCtrl(panel, -1, "0.6380", size=(50, -1))
self.IText11 = wx.TextCtrl(panel, -1, "0.6209", size=(50, -1))
self.IText12 = wx.TextCtrl(panel, -1, "0.5962", size=(50, -1))
self.IText13 = wx.TextCtrl(panel, -1, "0.5494", size=(50, -1))
self.IText14 = wx.TextCtrl(panel, -1, "0.4670", size=(50, -1))
self.IText15 = wx.TextCtrl(panel, -1, "0.3518", size=(50, -1))
self.IText16 = wx.TextCtrl(panel, -1, "0.0500", size=(50, -1))
self.IText0.SetInsertionPoint(0)
self.IText1.SetInsertionPoint(0)
self.IText2.SetInsertionPoint(0)
self.IText3.SetInsertionPoint(0)
self.IText4.SetInsertionPoint(0)
self.IText5.SetInsertionPoint(0)
self.IText6.SetInsertionPoint(0)
self.IText7.SetInsertionPoint(0)
self.IText8.SetInsertionPoint(0)
self.IText9.SetInsertionPoint(0)
self.IText10.SetInsertionPoint(0)
self.IText11.SetInsertionPoint(0)
self.IText12.SetInsertionPoint(0)
self.IText13.SetInsertionPoint(0)
self.IText14.SetInsertionPoint(0)
self.IText15.SetInsertionPoint(0)
self.IText16.SetInsertionPoint(0)

I2Label0 = wx.StaticText(panel, -1, " Va(r)/U :")
self.I2Text0  = wx.TextCtrl(panel, -1, "0.7500", size=(50, -1))
self.I2Text1  = wx.TextCtrl(panel, -1, "0.7625", size=(50, -1))
self.I2Text2  = wx.TextCtrl(panel, -1, "0.7750", size=(50, -1))
self.I2Text3  = wx.TextCtrl(panel, -1, "0.7875", size=(50, -1))
self.I2Text4  = wx.TextCtrl(panel, -1, "0.8000", size=(50, -1))
self.I2Text5  = wx.TextCtrl(panel, -1, "0.8125", size=(50, -1))
self.I2Text6  = wx.TextCtrl(panel, -1, "0.8250", size=(50, -1))
```

```
self.I2Text7  = wx.TextCtrl(panel, -1, "0.8375", size=(50, -1))
self.I2Text8  = wx.TextCtrl(panel, -1, "0.8500", size=(50, -1))
self.I2Text9  = wx.TextCtrl(panel, -1, "0.8625", size=(50, -1))
self.I2Text10 = wx.TextCtrl(panel, -1, "0.8750", size=(50, -1))
self.I2Text11 = wx.TextCtrl(panel, -1, "0.8875", size=(50, -1))
self.I2Text12 = wx.TextCtrl(panel, -1, "0.9000", size=(50, -1))
self.I2Text13 = wx.TextCtrl(panel, -1, "0.9125", size=(50, -1))
self.I2Text14 = wx.TextCtrl(panel, -1, "0.9250", size=(50, -1))
self.I2Text15 = wx.TextCtrl(panel, -1, "0.9375", size=(50, -1))
self.I2Text16 = wx.TextCtrl(panel, -1, "0.9500", size=(50, -1))
self.I2Text0.SetInsertionPoint(0)
self.I2Text1.SetInsertionPoint(0)
self.I2Text2.SetInsertionPoint(0)
self.I2Text3.SetInsertionPoint(0)
self.I2Text4.SetInsertionPoint(0)
self.I2Text5.SetInsertionPoint(0)
self.I2Text6.SetInsertionPoint(0)
self.I2Text7.SetInsertionPoint(0)
self.I2Text8.SetInsertionPoint(0)
self.I2Text9.SetInsertionPoint(0)
self.I2Text10.SetInsertionPoint(0)
self.I2Text11.SetInsertionPoint(0)
self.I2Text12.SetInsertionPoint(0)
self.I2Text13.SetInsertionPoint(0)
self.I2Text14.SetInsertionPoint(0)
self.I2Text15.SetInsertionPoint(0)
self.I2Text16.SetInsertionPoint(0)

I3Label0 = wx.StaticText(panel, -1, " Frac(r) :")
self.I3Text0  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text1  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text2  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text3  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text4  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text5  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text6  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text7  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text8  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text9  = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text10 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text11 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text12 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text13 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text14 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text15 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
```

```
self.I3Text16 = wx.TextCtrl(panel, -1, "1.", size=(50, -1))
self.I3Text0.SetInsertionPoint(0)
self.I3Text1.SetInsertionPoint(0)
self.I3Text2.SetInsertionPoint(0)
self.I3Text3.SetInsertionPoint(0)
self.I3Text4.SetInsertionPoint(0)
self.I3Text5.SetInsertionPoint(0)
self.I3Text6.SetInsertionPoint(0)
self.I3Text7.SetInsertionPoint(0)
self.I3Text8.SetInsertionPoint(0)
self.I3Text9.SetInsertionPoint(0)
self.I3Text10.SetInsertionPoint(0)
self.I3Text11.SetInsertionPoint(0)
self.I3Text12.SetInsertionPoint(0)
self.I3Text13.SetInsertionPoint(0)
self.I3Text14.SetInsertionPoint(0)
self.I3Text15.SetInsertionPoint(0)
self.I3Text16.SetInsertionPoint(0)

I4Label0 = wx.StaticText(panel, -1, " CDB(r) :")
self.I4Text0  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text1  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text2  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text3  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text4  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text5  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text6  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text7  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text8  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text9  = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text10 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text11 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text12 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text13 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text14 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text15 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text16 = wx.TextCtrl(panel, -1, "0.", size=(50, -1))
self.I4Text0.SetInsertionPoint(0)
self.I4Text1.SetInsertionPoint(0)
self.I4Text2.SetInsertionPoint(0)
self.I4Text3.SetInsertionPoint(0)
self.I4Text4.SetInsertionPoint(0)
self.I4Text5.SetInsertionPoint(0)
self.I4Text6.SetInsertionPoint(0)
self.I4Text7.SetInsertionPoint(0)
```

```
self.I4Text8.SetInsertionPoint(0)
self.I4Text9.SetInsertionPoint(0)
self.I4Text10.SetInsertionPoint(0)
self.I4Text11.SetInsertionPoint(0)
self.I4Text12.SetInsertionPoint(0)
self.I4Text13.SetInsertionPoint(0)
self.I4Text14.SetInsertionPoint(0)
self.I4Text15.SetInsertionPoint(0)
self.I4Text16.SetInsertionPoint(0)

sizer1 = wx.FlexGridSizer(cols=18, hgap=2, vgap=6)
sizer1.AddMany([ILabel0, self.IText0, self.IText1,\
                self.IText2, self.IText3,\
                self.IText4, self.IText5,\
                self.IText6, self.IText7,\
                self.IText8, self.IText9,\
                self.IText10, self.IText11,\
                self.IText12, self.IText13,\
                self.IText14, self.IText15,\
                self.IText16,\
                I2Label0, self.I2Text0, self.I2Text1,\
                self.I2Text2, self.I2Text3,\
                self.I2Text4, self.I2Text5,\
                self.I2Text6, self.I2Text7,\
                self.I2Text8, self.I2Text9,\
                self.I2Text10, self.I2Text11,\
                self.I2Text12, self.I2Text13,\
                self.I2Text14, self.I2Text15,\
                self.I2Text16,\
                I3Label0, self.I3Text0, self.I3Text1,\
                self.I3Text2, self.I3Text3,\
                self.I3Text4, self.I3Text5,\
                self.I3Text6, self.I3Text7,\
                self.I3Text8, self.I3Text9,\
                self.I3Text10, self.I3Text11,\
                self.I3Text12, self.I3Text13,\
                self.I3Text14, self.I3Text15,\
                self.I3Text16,\
                I4Label0, self.I4Text0, self.I4Text1,\
                self.I4Text2, self.I4Text3,\
                self.I4Text4, self.I4Text5,\
                self.I4Text6, self.I4Text7,\
                self.I4Text8, self.I4Text9,\
                self.I4Text10, self.I4Text11,\
                self.I4Text12, self.I4Text13,\
```

```
                self.I4Text14, self.I4Text15,\
                self.I4Text16])
panel.SetSizer(sizer1)

Input0Label = wx.StaticText(panel, -1, " Engine RPM [ERPM]:", pos=(0,137))
self.Input0Text = wx.TextCtrl(panel, -1, "77.0", size=(100, -1), pos=(170,135))
self.Input0Text.SetInsertionPoint(0)

Input1Label = wx.StaticText(panel, -1, " Diameter [feet]:", pos=(0,162))
self.Input1Text = wx.TextCtrl(panel, -1, "18.0", size=(100, -1), pos=(170,160))
self.Input1Text.SetInsertionPoint(0)

Input2Label = wx.StaticText(panel, -1, " Hub/Radius Ratio [-]:", pos=(0,187))
self.Input2Text = wx.TextCtrl(panel, -1, "0.18", size=(100, -1), pos=(170,185))
self.Input2Text.SetInsertionPoint(0)

Input3Label = wx.StaticText(panel, -1, " Radial Sections [-]:", pos=(0,212))
self.Input3Text = wx.TextCtrl(panel, -1, "16.0", size=(100, -1), pos=(170,210))
self.Input3Text.SetInsertionPoint(0)

Input4Label = wx.StaticText(panel, -1, " Number of Blades [-]:", pos=(0,237))
self.Input4Text = wx.TextCtrl(panel, -1, "5.0", size=(100, -1), pos=(170,235))
self.Input4Text.SetInsertionPoint(0)

Input5Label = wx.StaticText(panel, -1, " Depth of Hub CL [feet]:", pos=(0,262))
self.Input5Text = wx.TextCtrl(panel, -1, "38.6", size=(100, -1), pos=(170,260))
self.Input5Text.SetInsertionPoint(0)

Input6Label = wx.StaticText(panel, -1, " Brake Horsepower [HP]:", pos=(0,287))
self.Input6Text = wx.TextCtrl(panel, -1, "3915.434", size=(100, -1), pos=(170,285))
self.Input6Text.SetInsertionPoint(0)

Input7Label = wx.StaticText(panel, -1, " Gear Ratio [-]:", pos=(0,312))
self.Input7Text = wx.TextCtrl(panel, -1, "1.0", size=(100, -1), pos=(170,310))
self.Input7Text.SetInsertionPoint(0)

Input8Label = wx.StaticText(panel, -1, " Thrust Deduction [-]:", pos=(0,337))
self.Input8Text = wx.TextCtrl(panel, -1, "0.155", size=(100, -1), pos=(170,335))
self.Input8Text.SetInsertionPoint(0)

Input9Label = wx.StaticText(panel, -1, " Wake Fraction [-]:", pos=(0,362))
self.Input9Text = wx.TextCtrl(panel, -1, "0.252", size=(100, -1), pos=(170,360))
self.Input9Text.SetInsertionPoint(0)

Input10Label = wx.StaticText(panel, -1, " Relative Rotative Efficiency [-]:", pos=(0,387))
```

```
self.Input10Text = wx.TextCtrl(panel, -1, "1.018", size=(100, -1), pos=(170,385))
self.Input10Text.SetInsertionPoint(0)

Input11Label = wx.StaticText(panel, -1, " Transmission Efficiency [-]:", pos=(0,412))
self.Input11Text = wx.TextCtrl(panel, -1, "0.970", size=(100, -1), pos=(170,410))
self.Input11Text.SetInsertionPoint(0)

Input12Label = wx.StaticText(panel, -1, " Number of Propulsors [-]:", pos=(0,437))
self.Input12Text = wx.TextCtrl(panel, -1, "2.", size=(100, -1), pos=(170,435))
Input12aLabel = wx.StaticText(panel, -1, "Single Screw = 1; Twin Screw = 2", pos=(272,437))
self.Input12Text.SetInsertionPoint(0)

Input13Label = wx.StaticText(panel, -1, " Resistance Equation:", pos=(300,237))
self.Input13Text = wx.TextCtrl(panel, -1, "61900.0", size=(55, -1), pos=(417,235))
self.Input13Text.SetInsertionPoint(0)

Input14Label = wx.StaticText(panel, -1, " + ", pos=(475,237))
self.Input14Text = wx.TextCtrl(panel, -1, "0.00", size=(55, -1), pos=(490,235))
self.Input14Text.SetInsertionPoint(0)

Input15Label = wx.StaticText(panel, -1, " * V + ", pos=(545,237))
self.Input15Text = wx.TextCtrl(panel, -1, "0.00", size=(55, -1), pos=(580,235))
self.Input15Text.SetInsertionPoint(0)

Input16Label = wx.StaticText(panel, -1, " * V^2 ", pos=(640,237))

# NACA blade geometries upper
NACAInputLabel    = wx.StaticText(panel, -1, " Station: ", pos=(0,470))
NACAInputUSLabel  = wx.StaticText(panel, -1, " Upper Surface: ", pos=(0,492))
NACAInputU0Label  = wx.StaticText(panel, -1, " 0 ", pos=(100,470))
self.NACAU0       = wx.TextCtrl(panel, -1, "0.0000", size=(50, -1), pos=(83,490))
NACAInput1Label   = wx.StaticText(panel, -1, " 0.5 ", pos=(151,470))
self.NACAU1       = wx.TextCtrl(panel, -1, "0.5745", size=(50, -1), pos=(135,490))
NACAInput2Label   = wx.StaticText(panel, -1, " 0.75 ", pos=(202,470))
self.NACAU2       = wx.TextCtrl(panel, -1, "0.6967", size=(50, -1), pos=(187,490))
NACAInput3Label   = wx.StaticText(panel, -1, " 1.25 ", pos=(253,470))
self.NACAU3       = wx.TextCtrl(panel, -1, "0.8950", size=(50, -1), pos=(239,490))
NACAInput4Label   = wx.StaticText(panel, -1, " 2.5 ", pos=(304,470))
self.NACAU4       = wx.TextCtrl(panel, -1, "1.2551", size=(50, -1), pos=(291,490))
NACAInput5Label   = wx.StaticText(panel, -1, " 5.0 ", pos=(355,470))
self.NACAU5       = wx.TextCtrl(panel, -1, "1.7893", size=(50, -1), pos=(343,490))
NACAInput6Label   = wx.StaticText(panel, -1, " 7.5 ", pos=(406,470))
self.NACAU6       = wx.TextCtrl(panel, -1, "2.1994", size=(50, -1), pos=(395,490))
NACAInput7Label   = wx.StaticText(panel, -1, " 10.0 ", pos=(457,470))
self.NACAU7       = wx.TextCtrl(panel, -1, "2.5351", size=(50, -1), pos=(447,490))
```

```
NACAInput8Label   = wx.StaticText(panel, -1, " 15.0 ", pos=(508,470))
self.NACAU8       = wx.TextCtrl(panel, -1, "3.0650", size=(50, -1), pos=(499,490))
NACAInput9Label   = wx.StaticText(panel, -1, " 20.0 ", pos=(559,470))
self.NACAU9       = wx.TextCtrl(panel, -1, "3.4570", size=(50, -1), pos=(551,490))
NACAInput10Label  = wx.StaticText(panel, -1, " 25.0 ", pos=(610,470))
self.NACAU10      = wx.TextCtrl(panel, -1, "3.7393", size=(50, -1), pos=(603,490))
NACAInput11Label  = wx.StaticText(panel, -1, " 30.0 ", pos=(661,470))
self.NACAU11      = wx.TextCtrl(panel, -1, "3.9289", size=(50, -1), pos=(655,490))
NACAInput12Label  = wx.StaticText(panel, -1, " 35.0 ", pos=(712,470))
self.NACAU12      = wx.TextCtrl(panel, -1, "4.0296", size=(50, -1), pos=(707,490))
NACAInput13Label  = wx.StaticText(panel, -1, " 40.0 ", pos=(764,470))
self.NACAU13      = wx.TextCtrl(panel, -1, "4.0422", size=(50, -1), pos=(759,490))
NACAInput14Label  = wx.StaticText(panel, -1, " 45.0 ", pos=(815,470))
self.NACAU14      = wx.TextCtrl(panel, -1, "3.9716", size=(50, -1), pos=(811,490))
NACAInput15Label  = wx.StaticText(panel, -1, " 50.0 ", pos=(866,470))
self.NACAU15      = wx.TextCtrl(panel, -1, "3.8250", size=(50, -1), pos=(863,490))
NACAInput16Label  = wx.StaticText(panel, -1, " 55.0 ", pos=(917,470))
self.NACAU16      = wx.TextCtrl(panel, -1, "3.6112", size=(50, -1), pos=(915,490))
NACAInput17Label  = wx.StaticText(panel, -1, " 60.0 ", pos=(969,470))
self.NACAU17      = wx.TextCtrl(panel, -1, "3.3396", size=(50, -1), pos=(967,490))
NACAInput18Label  = wx.StaticText(panel, -1, " 65.0 ", pos=(1022,470))
self.NACAU18      = wx.TextCtrl(panel, -1, "3.0161", size=(50, -1), pos=(1019,490))
NACAInput19Label  = wx.StaticText(panel, -1, " 70.0 ", pos=(1073,470))
self.NACAU19      = wx.TextCtrl(panel, -1, "2.6462", size=(50, -1), pos=(1071,490))
NACAInput20Label  = wx.StaticText(panel, -1, " 75.0 ", pos=(100,520))
self.NACAU20      = wx.TextCtrl(panel, -1, "2.2396", size=(50, -1), pos=(83,540))
NACAInput21Label  = wx.StaticText(panel, -1, " 80.0 ", pos=(151,520))
self.NACAU21      = wx.TextCtrl(panel, -1, "1.8055", size=(50, -1), pos=(135,540))
NACAInput22Label  = wx.StaticText(panel, -1, " 85.0 ", pos=(202,520))
self.NACAU22      = wx.TextCtrl(panel, -1, "1.3547", size=(50, -1), pos=(187,540))
NACAInput23Label  = wx.StaticText(panel, -1, " 90.0 ", pos=(253,520))
self.NACAU23      = wx.TextCtrl(panel, -1, "0.9001", size=(50, -1), pos=(239,540))
NACAInput24Label  = wx.StaticText(panel, -1, " 95.0 ", pos=(304,520))
self.NACAU24      = wx.TextCtrl(panel, -1, "0.4529", size=(50, -1), pos=(291,540))
NACAInput25Label  = wx.StaticText(panel, -1, " 100.0 ", pos=(355,520))
self.NACAU25      = wx.TextCtrl(panel, -1, "0.000", size=(50, -1), pos=(343,540))

#Specify values to save for list item selected
NAUW0      = float(self.NACAU0.GetValue())
NAUW1      = float(self.NACAU1.GetValue())
NAUW2      = float(self.NACAU2.GetValue())
NAUW3      = float(self.NACAU3.GetValue())
NAUW4      = float(self.NACAU4.GetValue())
NAUW5      = float(self.NACAU5.GetValue())
NAUW6      = float(self.NACAU6.GetValue())
```

```
NAUW7      = float(self.NACAU7.GetValue())
NAUW8      = float(self.NACAU8.GetValue())
NAUW9      = float(self.NACAU9.GetValue())
NAUW10     = float(self.NACAU10.GetValue())
NAUW11     = float(self.NACAU11.GetValue())
NAUW12     = float(self.NACAU12.GetValue())
NAUW13     = float(self.NACAU13.GetValue())
NAUW14     = float(self.NACAU14.GetValue())
NAUW15     = float(self.NACAU15.GetValue())
NAUW16     = float(self.NACAU16.GetValue())
NAUW17     = float(self.NACAU17.GetValue())
NAUW18     = float(self.NACAU18.GetValue())
NAUW19     = float(self.NACAU19.GetValue())
NAUW20     = float(self.NACAU20.GetValue())
NAUW21     = float(self.NACAU21.GetValue())
NAUW22     = float(self.NACAU22.GetValue())
NAUW23     = float(self.NACAU23.GetValue())
NAUW24     = float(self.NACAU24.GetValue())
NAUW25     = float(self.NACAU25.GetValue())

self.NUvalues =  NAUW0, NAUW1, NAUW2, NAUW3, NAUW4, NAUW5, NAUW6, NAUW7,\
               NAUW8, NAUW9, NAUW10, NAUW11, NAUW12, NAUW13, NAUW14,\
               NAUW15, NAUW16, NAUW17, NAUW18, NAUW19, NAUW20, NAUW21,\
               NAUW22, NAUW23, NAUW24, NAUW25

self.NU = asarray(self.NUvalues)

# NACA blade geometries lower
NACAInputLabel    = wx.StaticText(panel, -1, " Station: ", pos=(0,580))
NACAInputLSLabel  = wx.StaticText(panel, -1, " Lower Surface: ", pos=(0,602))
NACAInputL0Label  = wx.StaticText(panel, -1, " 0 ", pos=(100,580))
self.NACAL0       = wx.TextCtrl(panel, -1, "0.0000", size=(50, -1), pos=(83,600))
NACAInput1Label   = wx.StaticText(panel, -1, " 0.5 ", pos=(151,580))
self.NACAL1       = wx.TextCtrl(panel, -1, "-0.4330", size=(50, -1), pos=(135,600))
NACAInput2Label   = wx.StaticText(panel, -1, " 0.75 ", pos=(202,580))
self.NACAL2       = wx.TextCtrl(panel, -1, "-0.5225", size=(50, -1), pos=(187,600))
NACAInput3Label   = wx.StaticText(panel, -1, " 1.25 ", pos=(253,580))
self.NACAL3       = wx.TextCtrl(panel, -1, "-0.6498", size=(50, -1), pos=(239,600))
NACAInput4Label   = wx.StaticText(panel, -1, " 2.5 ", pos=(304,580))
self.NACAL4       = wx.TextCtrl(panel, -1, "-0.8608", size=(50, -1), pos=(291,600))
NACAInput5Label   = wx.StaticText(panel, -1, " 5.0 ", pos=(355,580))
self.NACAL5       = wx.TextCtrl(panel, -1, "-1.1385", size=(50, -1), pos=(343,600))
NACAInput6Label   = wx.StaticText(panel, -1, " 7.5 ", pos=(406,580))
self.NACAL6       = wx.TextCtrl(panel, -1, "-1.3368", size=(50, -1), pos=(395,600))
NACAInput7Label   = wx.StaticText(panel, -1, " 10.0 ", pos=(457,580))
```

```
self.NACAL7       = wx.TextCtrl(panel, -1, "-1.4882", size=(50, -1), pos=(447,600))
NACAInput8Label   = wx.StaticText(panel, -1, " 15.0 ", pos=(508,580))
self.NACAL8       = wx.TextCtrl(panel, -1, "-1.7108", size=(50, -1), pos=(499,600))
NACAInput9Label   = wx.StaticText(panel, -1, " 20.0 ", pos=(559,580))
self.NACAL9       = wx.TextCtrl(panel, -1, "-1.8589", size=(50, -1), pos=(551,600))
NACAInput10Label  = wx.StaticText(panel, -1, " 25.0 ", pos=(610,580))
self.NACAL10      = wx.TextCtrl(panel, -1, "-1.9464", size=(50, -1), pos=(603,600))
NACAInput11Label  = wx.StaticText(panel, -1, " 30.0 ", pos=(661,580))
self.NACAL11      = wx.TextCtrl(panel, -1, "-1.9833", size=(50, -1), pos=(655,600))
NACAInput12Label  = wx.StaticText(panel, -1, " 35.0 ", pos=(712,580))
self.NACAL12      = wx.TextCtrl(panel, -1, "-1.9687", size=(50, -1), pos=(707,600))
NACAInput13Label  = wx.StaticText(panel, -1, " 40.0 ", pos=(764,580))
self.NACAL13      = wx.TextCtrl(panel, -1, "-1.9004", size=(50, -1), pos=(759,600))
NACAInput14Label  = wx.StaticText(panel, -1, " 45.0 ", pos=(815,580))
self.NACAL14      = wx.TextCtrl(panel, -1, "-1.7812", size=(50, -1), pos=(811,600))
NACAInput15Label  = wx.StaticText(panel, -1, " 50.0 ", pos=(866,580))
self.NACAL15      = wx.TextCtrl(panel, -1, "-1.6186", size=(50, -1), pos=(863,600))
NACAInput16Label  = wx.StaticText(panel, -1, " 55.0 ", pos=(917,580))
self.NACAL16      = wx.TextCtrl(panel, -1, "-1.4207", size=(50, -1), pos=(915,600))
NACAInput17Label  = wx.StaticText(panel, -1, " 60.0 ", pos=(969,580))
self.NACAL17      = wx.TextCtrl(panel, -1, "-1.1963", size=(50, -1), pos=(967,600))
NACAInput18Label  = wx.StaticText(panel, -1, " 65.0 ", pos=(1022,580))
self.NACAL18      = wx.TextCtrl(panel, -1, "-0.9529", size=(50, -1), pos=(1019,600))
NACAInput19Label  = wx.StaticText(panel, -1, " 70.0 ", pos=(1073,580))
self.NACAL19      = wx.TextCtrl(panel, -1, "-0.6993", size=(50, -1), pos=(1071,600))
NACAInput20Label  = wx.StaticText(panel, -1, " 75.0 ", pos=(100,630))
self.NACAL20      = wx.TextCtrl(panel, -1, "-0.4470", size=(50, -1), pos=(83,650))
NACAInput21Label  = wx.StaticText(panel, -1, " 80.0 ", pos=(151,630))
self.NACAL21      = wx.TextCtrl(panel, -1, "-0.2101", size=(50, -1), pos=(135,650))
NACAInput22Label  = wx.StaticText(panel, -1, " 85.0 ", pos=(202,630))
self.NACAL22      = wx.TextCtrl(panel, -1, "-0.0067", size=(50, -1), pos=(187,650))
NACAInput23Label  = wx.StaticText(panel, -1, " 90.0 ", pos=(253,630))
self.NACAL23      = wx.TextCtrl(panel, -1, "0.1358", size=(50, -1), pos=(239,650))
NACAInput24Label  = wx.StaticText(panel, -1, " 95.0 ", pos=(304,630))
self.NACAL24      = wx.TextCtrl(panel, -1, "0.1790", size=(50, -1), pos=(291,650))
NACAInput25Label  = wx.StaticText(panel, -1, " 100.0 ", pos=(355,630))
self.NACAL25      = wx.TextCtrl(panel, -1, "0.0000", size=(50, -1), pos=(343,650))

#Specify values to save for list item selected
NALW0      = float(self.NACAL0.GetValue())
NALW1      = float(self.NACAL1.GetValue())
NALW2      = float(self.NACAL2.GetValue())
NALW3      = float(self.NACAL3.GetValue())
NALW4      = float(self.NACAL4.GetValue())
NALW5      = float(self.NACAL5.GetValue())
```

```
        NALW6      = float(self.NACAL6.GetValue())
        NALW7      = float(self.NACAL7.GetValue())
        NALW8      = float(self.NACAL8.GetValue())
        NALW9      = float(self.NACAL9.GetValue())
        NALW10     = float(self.NACAL10.GetValue())
        NALW11     = float(self.NACAL11.GetValue())
        NALW12     = float(self.NACAL12.GetValue())
        NALW13     = float(self.NACAL13.GetValue())
        NALW14     = float(self.NACAL14.GetValue())
        NALW15     = float(self.NACAL15.GetValue())
        NALW16     = float(self.NACAL16.GetValue())
        NALW17     = float(self.NACAL17.GetValue())
        NALW18     = float(self.NACAL18.GetValue())
        NALW19     = float(self.NACAL19.GetValue())
        NALW20     = float(self.NACAL20.GetValue())
        NALW21     = float(self.NACAL21.GetValue())
        NALW22     = float(self.NACAL22.GetValue())
        NALW23     = float(self.NACAL23.GetValue())
        NALW24     = float(self.NACAL24.GetValue())
        NALW25     = float(self.NACAL25.GetValue())

        self.NLvalues =  NALW0, NALW1, NALW2, NALW3, NALW4, NALW5, NALW6, NALW7,\
                       NALW8, NALW9, NALW10, NALW11, NALW12, NALW13, NALW14,\
                       NALW15, NALW16, NALW17, NALW18, NALW19, NALW20, NALW21,\
                       NALW22, NALW23, NALW24, NALW25

        self.NL = asarray(self.NLvalues)

# dirname is an APPLICATION variable that we're choosing to store
# in with the frame - it's the parent directory for any file we
# choose to edit in this frame
        self.dirname = ''

 # Begin buttons
        button = wx.Button(panel, label="Close", pos=(300,155), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
        button1 = wx.Button(panel, label="Save", pos=(350,155), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnSave, button1)
        button2 = wx.Button(panel, label="Run", pos=(400,155), size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.OnRun, button2)
        button3 = wx.Button(panel, label="View Results", pos=(300,180), size=(150,25))
        self.Bind(wx.EVT_BUTTON, self.OnResults, button3)
        button3 = wx.Button(panel, label="Output to file", pos=(300,205), size=(150,25))
        self.Bind(wx.EVT_BUTTON, self.OnOutput, button3)
```

```
        List = ['6 Al-4Va Titanium', 'T6061 Al', 'Nickel Aluminum Bronze', \
                'Chromium-nickel austenitic stainless steel']
        Listb = ['B-Series', 'NACA']

        self.listbox = wx.ListBox(panel, -1, (300, 305), wx.DefaultSize,
                        List, wx.LB_SINGLE)

        self.listboxb = wx.ListBox(panel, -1, (567, 305), wx.DefaultSize,
                                   Listb, wx.LB_SINGLE)

        self.Bind(wx.EVT_LISTBOX, self.OnList, self.listbox)
        self.Bind(wx.EVT_LISTBOX, self.OnList2, self.listboxb)

    def OnList(self, event):
        self.lselect1 = self.listbox.GetSelection()
        if self.lselect1 == 0:
            self.sigmamax = 135000.0           # material strength lbs/in^2
            self.methodnumber = "6 Al-4Va Titanium"

        elif self.lselect1 == 1:
            self.sigmamax = 40000.00           # material strength lbs/in^2
            self.methodnumber = "T6061 Al"

        elif self.lselect1 == 2:
            self.sigmamax = 75000.00           # material strength lbs/in^2
            self.methodnumber = "Nickel Aluminum Bronze"

        elif self.lselect1 == 3:
            self.sigmamax = 50000.00           # material strength lbs/in^2
            self.methodnumber = "Chromium-nickel austenitic stainless steel"

        return self.methodnumber, self.lselect1, self.sigmamax

    def OnList2(self, event):
        self.lselect2 = self.listboxb.GetSelection()
        if self.lselect2 == 0:
            self.methodblade = "B-Series"

        elif self.lselect2 == 1:
            self.methodblade = "User Defined"

        return self.methodblade, self.lselect2
```

```
# Event for closing
    def OnCloseMe(self, event):
        self.Close(True)

    def OnCloseWindow(self, event):
        self.Destroy()

    def OnAbout(self, event):
        directions ="""
        MTproppy.py can be run in a few simple steps.
        1.) Fill in input boxes.  Use float values (0. or 12.)
        2.) Once input boxes are filled, select the "Save" button.  This will \
        save the input file for the program to read.  Similar to MTp2.in file \
        written for Dr. Vorus's MTprop.exe
        3.) Select the type of propeller desired for optimization.  The B-series \
        propeller is coded into the program and its selection is all that is necessary.  \
        If NACA is selected,
            the input values listed at the bottom of the screen as upper and lower are needed.
        4.) The material desired for the propeller is needed.  The program is \
        set-up for strength calculation based only those materials listed.
        3.) When the input file is saved, and material and blade type selected, \
        select the "Run" button.  This will initiate the optimization programming.
        4.) Select the "View Results" button to view the final results of the optimization.
        5.) Final, the button "Output to File" to produce a "MTPresults.txt" \
        file with all the calculated values use to produce the optimized results.
        6.) The program must be run in this sequence.

        """
        dialog = wx.lib.dialogs.ScrolledMessageDialog(self, directions, "Directions")
        dialog.ShowModal()

 # Event for saving

    def OnSave(self,e):
        #Specify values to save for list item selected
        v0      = float(self.Input0Text.GetValue())
        v1      = float(self.Input1Text.GetValue())
        v2      = float(self.Input2Text.GetValue())
        v3      = float(self.Input3Text.GetValue())
        v4      = float(self.Input4Text.GetValue())
        v5      = float(self.Input5Text.GetValue())
        v6      = float(self.Input6Text.GetValue())
        v7      = float(self.Input7Text.GetValue())
        v8      = float(self.Input8Text.GetValue())
        v9      = float(self.Input9Text.GetValue())
```

```
        v10     = float(self.Input10Text.GetValue())
        v11     = float(self.Input11Text.GetValue())
        v12     = float(self.Input12Text.GetValue())

        self.values =  v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12

        self.sv = asarray(self.values)

        # Save away the edited text
        # Open the file, do an RU sure check for an overwrite!
        dlg = wx.FileDialog(self, "Choose a file", self.dirname, "MTpropIN", ".txt", \
                wx.SAVE | wx.OVERWRITE_PROMPT)
        if dlg.ShowModal() == wx.ID_OK:
            # Grab the content to be saved

            self.filename=dlg.GetFilename()
            self.dirname=dlg.GetDirectory()
            filehandle=open(os.path.join(self.dirname, self.filename),"w")
            for i in range(len(self.sv)):
                filehandle.write("%f " %(self.sv[i]))
            filehandle.close()

        dlg.Destroy()


##########################################################################
    def OnRun(self, event):

        self.EQU1  = float(self.Input13Text.GetValue())
        self.EQU2  = float(self.Input14Text.GetValue())
        self.EQU3  = float(self.Input15Text.GetValue())

        C0   = float(self.IText0.GetValue())
        C1   = float(self.IText1.GetValue())
        C2   = float(self.IText2.GetValue())
        C3   = float(self.IText3.GetValue())
        C4   = float(self.IText4.GetValue())
        C5   = float(self.IText5.GetValue())
        C6   = float(self.IText6.GetValue())
        C7   = float(self.IText7.GetValue())
        C8   = float(self.IText8.GetValue())
        C9   = float(self.IText9.GetValue())
        C10  = float(self.IText10.GetValue())
        C11  = float(self.IText11.GetValue())
        C12  = float(self.IText12.GetValue())
```

```
C13  = float(self.IText13.GetValue())
C14  = float(self.IText14.GetValue())
C15  = float(self.IText15.GetValue())
C16  = float(self.IText16.GetValue())

self.CRRA =  C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12,\
C13, C14, C15, C16

self.CRR = asarray(self.CRRA)

VUV0    = float(self.I2Text0.GetValue())
VUV1    = float(self.I2Text1.GetValue())
VUV2    = float(self.I2Text2.GetValue())
VUV3    = float(self.I2Text3.GetValue())
VUV4    = float(self.I2Text4.GetValue())
VUV5    = float(self.I2Text5.GetValue())
VUV6    = float(self.I2Text6.GetValue())
VUV7    = float(self.I2Text7.GetValue())
VUV8    = float(self.I2Text8.GetValue())
VUV9    = float(self.I2Text9.GetValue())
VUV10   = float(self.I2Text10.GetValue())
VUV11   = float(self.I2Text11.GetValue())
VUV12   = float(self.I2Text12.GetValue())
VUV13   = float(self.I2Text13.GetValue())
VUV14   = float(self.I2Text14.GetValue())
VUV15   = float(self.I2Text15.GetValue())
VUV16   = float(self.I2Text16.GetValue())

self.VaU1 =  VUV0, VUV1, VUV2, VUV3, VUV4, VUV5, VUV6, VUV7, VUV8,\
VUV9, VUV10, VUV11, VUV12, VUV13, VUV14, VUV15, VUV16

self.VaU = asarray(self.VaU1)

FR0    = float(self.I3Text0.GetValue())
FR1    = float(self.I3Text1.GetValue())
FR2    = float(self.I3Text2.GetValue())
FR3    = float(self.I3Text3.GetValue())
FR4    = float(self.I3Text4.GetValue())
FR5    = float(self.I3Text5.GetValue())
FR6    = float(self.I3Text6.GetValue())
FR7    = float(self.I3Text7.GetValue())
FR8    = float(self.I3Text8.GetValue())
FR9    = float(self.I3Text9.GetValue())
FR10   = float(self.I3Text10.GetValue())
FR11   = float(self.I3Text11.GetValue())
```

```
FR12   = float(self.I3Text12.GetValue())
FR13   = float(self.I3Text13.GetValue())
FR14   = float(self.I3Text14.GetValue())
FR15   = float(self.I3Text15.GetValue())
FR16   = float(self.I3Text16.GetValue())

self.FRV1 =  FR0, FR1, FR2, FR3, FR4, FR5, FR6, FR7, FR8,\
FR9, FR10, FR11, FR12, FR13, FR14, FR15, FR16

self.FRACV = asarray(self.FRV1)

CB0    = float(self.I4Text0.GetValue())
CB1    = float(self.I4Text1.GetValue())
CB2    = float(self.I4Text2.GetValue())
CB3    = float(self.I4Text3.GetValue())
CB4    = float(self.I4Text4.GetValue())
CB5    = float(self.I4Text5.GetValue())
CB6    = float(self.I4Text6.GetValue())
CB7    = float(self.I4Text7.GetValue())
CB8    = float(self.I4Text8.GetValue())
CB9    = float(self.I4Text9.GetValue())
CB10   = float(self.I4Text10.GetValue())
CB11   = float(self.I4Text11.GetValue())
CB12   = float(self.I4Text12.GetValue())
CB13   = float(self.I4Text13.GetValue())
CB14   = float(self.I4Text14.GetValue())
CB15   = float(self.I4Text15.GetValue())
CB16   = float(self.I4Text16.GetValue())

self.CBV1 =  CB0, CB1, CB2, CB3, CB4, CB5, CB6, CB7, CB8,\
CB9, CB10, CB11, CB12, CB13, CB14, CB15, CB16

self.CDBR = asarray(self.CBV1)

def MTPROPOPT(xfree, args = None):
    f = open("MTpropIN.txt")
    lines = f.readlines()
    f.close()

    # Make data from input file into usable data
    for line in lines:
        strlist = line.split()
        erpm    = float(strlist[0])       # Engine Revolutions per Minute
        D       = float(strlist[1])       # Diameter [feet]
        RHOR    = float(strlist[2])       # Hub Radius Ratio
```

```
    rn      = float(strlist[3])       # Radial Sections, at least \
                16 should be used for serious calculations (Vorus, 2007)
    Z       = float(strlist[4])       # Number of Blades
    ZH      = float(strlist[5])       # Depth of Hub Centerline
    BHP     = float(strlist[6])       # Brake Horsepower - Used as a Constraint
    GR      = float(strlist[7])       # Gear Ratio
    td      = float(strlist[8])       # Thrust deduction
    wf      = float(strlist[9])       # Wave fraction
    EFrot   = float(strlist[10])      # Relative Rotative Efficieny
    EFtrans = float(strlist[11])      # Transmission Efficiency
    screws  = float(strlist[12])      # Propulsors - Single or Double screw

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = 1.9900
# Kinematic viscosity [ft^2/s]
kv  = 0.000011

prpm = erpm/GR

U1 = xfree[0]
PA = xfree[1]

# Computed values from givens:
U       = xfree[0] * 1.689          # Convert speed from knots to ft/s
R       = D/2.                      # Radius calculated from inputs
Rh      = RHOR*R                    # Radius of Hub
AD      = pi*((R)**2-(Rh)**2)       # Propeller disk area
Omega   = (2*pi*prpm)/60.           # Rate of Revolutions converted rad/s
DR      = (1-RHOR)/ rn              # Sectional separation
TBR     = U/(Omega*R)               # TBR is equal to tan Beta at R
RR      = RHOR - 0.5*DR
RR1     = RR + DR
DHP     = BHP*EFtrans*(EFrot**0.5)  # Delivered Horsepower - Constraint

# List of all the the arrays created
i           = 0
j           = 0
k           = 0
COR         = []                # Chord/ R distribution
VAOU        = []                # Wake Velocity distribution estimate
RHOR1       = []
rb          = []                # Rbar = ROR in Vorus code
PrDS        = []                # Scaled Pitch distribution [VaU * PA]
PIOD        = []                # Hydrodynamic Pitch Distribution
FRAC        = []                # Cavitation Pressure [psia] - cavity gas pressure
```

```
CDBr        = []
TBRR        = []
C           = []
C2          = []
TB2         = []
P1          = []
P2          = []
CTII        = []                # This array and equation makes CTII \
                                nondimensional on VaU
CTI         = []                # Nondimensional on U
UI          = []                # Induced velocity
VI          = []                # Induced tangential velocity
WI          = []
EFFr        = []                # radial efficiency
CPI         = []
TBI         = []
CTTI1       = []                # Variable made for CTTI calculation
CPPI1       = []                # Variable made for CPPI calculation
CC          = []                # Variable made for CTTI calculation
PP          = []                # Variable made for CPPI calculation

## Set-up resistance data from determined resistance curves
V = (U/1.689)

RF = self.EQU1 + (self.EQU2*V) + (self.EQU3*(V**2.))
T = ((1./screws)*RF)/((1.-td)*(EFrot**0.5))

####### IDEAL CALCULATIONS  #########
# Specified chord radius ratio
#CRR  = [0.4937, 0.5148, 0.5360, 0.5573, 0.5783, 0.5985, 0.6174, \
0.6338, 0.6464, 0.6536, 0.6527, 0.6387, 0.6055, 0.5424, 0.4358, 0.2648, 0.0500]
#VaU  = [0.7500, 0.7625, 0.7750, 0.7875, 0.8000, 0.8125, 0.8250, \
0.8375, 0.8500, 0.8625, 0.8750, 0.8875, 0.9000, 0.9125, 0.9250, 0.9375, 0.9500]
CRR  = self.CRR
VaU  = self.VaU
FRAC = self.FRACV
CDBr = self.CDBR

while k <= rn:
    PrDS.append(VaU[k]*xfree[1])                    # S stands for SCALED
    k += 1

# Array set up to find midpoint location for COR
CRR1  = [CRR[1],  CRR[2],  CRR[3],  CRR[4],  CRR[5],  CRR[6],  \
         CRR[7],  CRR[8],  CRR[9],  CRR[10],  CRR[11],  CRR[12],  \
```

```
         CRR[13],  CRR[14],  CRR[15],  CRR[16],  0.0000]
# Array set up to find midpoint location for PIOD
PrDS1 = [PrDS[1], PrDS[2], PrDS[3], PrDS[4], PrDS[5], PrDS[6], \
         PrDS[7], PrDS[8], PrDS[9], PrDS[10], PrDS[11], PrDS[12], \
         PrDS[13], PrDS[14], PrDS[15], PrDS[16], 0.0000]
# Array set up to find midpoint location for Hydrodynamic wake
VaU1  = [VaU[1],  VaU[2],  VaU[3],  VaU[4],  VaU[5],  VaU[6],  \
         VaU[7],  VaU[8],  VaU[9],  VaU[10],  VaU[11],  VaU[12], \
          VaU[13],  VaU[14],  VaU[15],  VaU[16],  0.0000]

while j <= rn:
    COR.append(0.5*(CRR[j]+CRR1[j]))
    PIOD.append(0.5*(PrDS[j]+PrDS1[j]))
    VAOU.append(0.5*(VaU[j]+VaU1[j]))
    RHOR1.append(RHOR + DR*j)
    rb.append(RR1+DR*j)
    TBRR.append(TBR*VAOU[j])
    C.append(PIOD[j]/(pi*TBRR[j]))
    C2.append(C[j]*C[j])
    TB2.append((TBRR[j]/rb[j])**2)
    P1.append(((1.-C2[j]*TB2[j])**2)/(4.*C2[j]))
    P2.append(rb[j]/TBR)
    CTII.append((1.-1./C[j]+(C[j]-1.)*TB2[j])/(TB2[j]+P1[j]))
    CTI.append(CTII[j]*(VAOU[j]**2))
    UI.append(0.5*VAOU[j]*(-1.+(sqrt(1.+CTII[j]))))
    VI.append(0.5*(P2[j]-sqrt((P2[j]**2)-CTI[j])))
    WI.append(VI[j]/P2[j])
    EFFr.append((1. - WI[j])/(1.+UI[j]))
    TBI.append((PIOD[j]/(pi*rb[j])))
    CPI.append((CTI[j]*TBI[j]*rb[j])/TBR)
    CC.append(CTI[j]*rb[j])
    PP.append(CPI[j]*rb[j])
    j+=1

CTTI1 = CC[0]+CC[1]+CC[2]+CC[3]+CC[4]+CC[5]+CC[6]+CC[7]+CC[8]+\
CC[9]+CC[10]+CC[11]+CC[12]+CC[13]+CC[14]+CC[15]
CTTI  = 2.*CTTI1*DR/(1.-RHOR**2)
CPPI1 = PP[0]+PP[1]+PP[2]+PP[3]+PP[4]+PP[5]+PP[6]+PP[7]+PP[8]+\
PP[9]+PP[10]+PP[11]+PP[12]+PP[13]+PP[14]+PP[15]
CPPI  = 2.*CPPI1*DR/(1.-RHOR**2)
EFI = CTTI/CPPI

THSTI = CTTI*0.5*rho*(U**2)*AD
PWRI  = CPPI*0.5*rho*(U**3)*AD/550
```

```
####### REAL PERFORMANCE CALCULATIONS  #########

XJD = U*60. / (prpm*D)                  # J - Advance Coefficient

#Add viscous forces
TB          = []
BI          = []
SIBI        = []
COBI        = []
COTBI       = []
VSTAR2      = []                        # Vstar squared
VSTAR       = []                        # Vr/U - Hypotenuse of \
                                      Blade Element Velocity Diagram
REYN        = []                        # Reynolds Number
CDRAG       = []                        # Coefficient of Drag
CTV         = []                        # Viscous Thrust Coefficient
CT          = []                        # Total Thrust Coefficient
CPV         = []                        # Viscous Power Coefficient
CP          = []                        # Total Power Coefficient
CL          = []                        # Coefficient of lift
EFF         = []                        # Efficiency
SIGMA       = []                        # Cavitation Number
CMBR        = []                        # Camber
TBC         = []
CCC         = []                        # Variable made for CTT calculation
PPP         = []                        # Variable made for CPP calculation
EARC        = []                        # Variable made for EAR calculation

#Strength Calculations
Chrd        = []            # Chord length [ft]
tn          = []            # Blade thickness
tmax        = []            # Maximum blade thickness from B-series equations
Ar          = []            # Variable used for tmin calculations
Br          = []            # Variable used for tmin calculations
xcr         = []            # Position of maximum thickness from the leading edge
xcrr        = []
self.tle    = []            # Thickness at leading edge, B-series only
self.tte    = []            # Thickness at trailing edge, B-Series only

while i <= rn:
    TB.append(TBR/rb[i])
    BI.append(arctan(TBI[i]))
    SIBI.append(sin(BI[i]))
    COBI.append(cos(BI[i]))
    COTBI.append(1./TBI[i])
```

```
    VSTAR2.append(((VAOU[i]+UI[i])**2)+(1./TB[i]-VI[i])**2)
    VSTAR.append(sqrt(VSTAR2[i]))
    REYN.append(R*COR[i]*U*VSTAR[i]/kv)
    CDRAG.append(CDBr[i]+0.15/(((log10(REYN[i]))-2.)**2)+0.003)
    CTV.append(-Z*CDRAG[i]*COR[i]*VSTAR2[i]*SIBI[i]/(2.*pi*rb[i]))
    CPV.append(Z*CDRAG[i]*COR[i]*VSTAR2[i]*COBI[i]/(2.*pi*rb[i]*TB[i]))
    CL.append((2.*pi*rb[i]*CTI[i])/(((VSTAR2[i])**2)*COR[i]*Z*COBI[i]))
    CT.append(CTI[i]+CTV[i])
    CP.append(CPI[i]+CPV[i])
    EFF.append(CT[i]/CP[i])
    SIGMA.append((2117. + 64.*ZH-FRAC[i]*144.)/(0.5*rho*(U*VSTAR[i])**2))
    CMBR.append(0.0679*CL[i])
    TBC.append(-3.008*CMBR[i]+(0.334-0.425*CMBR[i])*SIGMA[i])
    CCC.append(CT[i]*rb[i])
    PPP.append(CP[i]*rb[i])
    EARC.append(COR[i])

    Chrd.append(COR[i]*(D/2))
    tn.append(TBC[i]*Chrd[i]*12)            # Cavitation thicknesses calculated
    Ar.append(-0.062*rb[i] + 0.065)
    Br.append(-0.005*rb[i] + 0.005)
    tmax.append((Ar[i]-(Br[i]*Z))*D)
    xcrr.append(35.37*rb[i]**6 - 118.29*rb[i]**5 + 152.53*rb[i]**4\
                - 96.125*rb[i]**3 + 31.458*rb[i]**2 - 5.1109*rb[i] + 0.6743)
    xcr.append(xcrr[i]*Chrd[i])

    i+=1

CTT1 = CCC[0]+CCC[1]+CCC[2]+CCC[3]+CCC[4]+CCC[5]+CCC[6]+CCC[7]+\
CCC[8]+CCC[9]+CCC[10]+CCC[11]+CCC[12]+CCC[13]+CCC[14]+CCC[15]
CTT  = 2.*CTT1*DR/(1.-RHOR**2)
CPP1 = PPP[0]+PPP[1]+PPP[2]+PPP[3]+PPP[4]+PPP[5]+PPP[6]+PPP[7]+\
PPP[8]+PPP[9]+PPP[10]+PPP[11]+PPP[12]+PPP[13]+PPP[14]+PPP[15]
CPP  = 2.*CPP1*DR/(1.-RHOR**2)
EAR1 = EARC[0]+EARC[1]+EARC[2]+EARC[3]+EARC[4]+EARC[5]+EARC[6]+\
EARC[7]+EARC[8]+EARC[9]+EARC[10]+EARC[11]+EARC[12]+EARC[13]+EARC[14]+EARC[15]
EAR  = Z * EAR1 * DR / pi
EF   = CTT/CPP

XKT = CTT * XJD**2 * pi * (1.-RHOR**2) / 8.
XKQ = CPP * XJD**3 * (1.-RHOR**2) / 16.

THST = CTT*0.5*rho*(U**2)*AD
PWR  = CPP*0.5*rho*(U**3)*AD/550.
```

```
#print "THRST COF", CTT

ii = 0

# Hydrofoil section calculations - V1 negative values
self.rb1n1       = []            # -1.0
self.rb95n1      = []            # -0.95
self.rb9n1       = []            # -0.90
self.rb8n1       = []            # -0.8
self.rb7n1       = []            # -0.7
self.rb6n1       = []            # -0.6
self.rb5n1       = []            # -0.5
self.rb4n1       = []            # -0.4
self.rb2n1       = []            # -0.2

# Hydrofoil section calculations - V1 positive values
self.rb1p1       = []            # 1.0
self.rb95p1      = []            # 0.95
self.rb9p1       = []            # 0.90
self.rb85p1      = []            # 0.85
self.rb8p1       = []            # 0.8
self.rb7p1       = []            # 0.7
self.rb6p1       = []            # 0.6
self.rb5p1       = []            # 0.5
self.rb4p1       = []            # 0.4
self.rb2p1       = []            # 0.2

# Hydrofoil section calculations - V2 negative values
self.rb1n2       = []            # -1.0
self.rb95n2      = []            # -0.95
self.rb9n2       = []            # -0.90
self.rb8n2       = []            # -0.8
self.rb7n2       = []            # -0.7
self.rb6n2       = []            # -0.6
self.rb5n2       = []            # -0.5
self.rb4n2       = []            # -0.4
self.rb2n2       = []            # -0.2

# Hydrofoil section calculations - V2 positive values
self.rb1p2       = []            # -1.0
self.rb95p2      = []            # 0.95
self.rb9p2       = []            # 0.90
self.rb85p2      = []            # 0.85
self.rb8p2       = []            # 0.8
self.rb7p2       = []            # 0.7
```

```
self.rb6p2       = []            # 0.6
self.rb5p2       = []            # 0.5
self.rb4p2       = []            # 0.4
self.rb2p2       = []            # 0.2

Upper0           = []
Upper1           = []
Upper2           = []
Upper3           = []
Upper4           = []
Upper5           = []
Upper6           = []
Upper7           = []
Upper8           = []
Upper9           = []
Upper10          = []
Upper11          = []
Upper12          = []
Upper13          = []
Upper14          = []
Upper15          = []

Lower0           = []
Lower1           = []
Lower2           = []
Lower3           = []
Lower4           = []
Lower5           = []
Lower6           = []
Lower7           = []
Lower8           = []
Lower9           = []
Lower10          = []
Lower11          = []
Lower12          = []
Lower13          = []
Lower14          = []
Lower15          = []

if self.lselect2 ==0:
    while ii <= rn:
        self.tle.append(0.2*tmax[ii])
        self.tte.append(0.1*tmax[ii])
        self.rb1n1.append(abs(15.13651574*rb[ii]**8-34.29336202*\
        rb[ii]**7+38.43403424*rb[ii]**6-62.63339931*rb[ii]**5+84.53854182\
```

```
*rb[ii]**4-55.07323020*rb[ii]**3+15.77797977*rb[ii]**2-2.327732128\
*rb[ii]+.4409705328))
self.rb95n1.append(abs(8.575313847*rb[ii]**8-20.60991836*\
rb[ii]**7+29.68905017*rb[ii]**6-50.82503833*rb[ii]**5+58.30929982\
*rb[ii]**4-30.91981308*rb[ii]**3+6.149730292*rb[ii]**2-.6942295710\
*rb[ii]+.3258323036))
self.rb9n1.append(abs(5.497643016*rb[ii]**8-15.99107858*rb[ii]**7\
+31.49120993*rb[ii]**6-53.13046197*rb[ii]**5+52.14603925*rb[ii]**4-\
23.38651031*rb[ii]**3+3.472073204*rb[ii]**2-.4009626055*rb[ii]+\
.3022409421))
self.rb8n1.append(abs(1.779196375*rb[ii]**8-9.282920961*rb[ii]**7\
+28.29492223*rb[ii]**6-46.51788168*rb[ii]**5+37.53087498*rb[ii]**4\
-12.55990785*rb[ii]**3+.8033266969*rb[ii]**2-.3352025014*rb[ii]+\
.2877262417))
self.rb7n1.append(abs(0.7907434879e-1*rb[ii]**8-4.601950356*\
rb[ii]**7+19.45462799*rb[ii]**6-31.02215947*rb[ii]**5+\
21.62920464*rb[ii]**4-5.466133078*rb[ii]**3+.2686815286*rb[ii]**2\
-.6341881276*rb[ii]+.2929213301))
self.rb6n1.append(abs(-.8154282486*rb[ii]**8-1.691817720*rb[ii]**7\
+12.48823911*rb[ii]**6-18.86108012*rb[ii]**5+10.67302602*rb[ii]**4\
-1.755223090*rb[ii]**3+.6442194500*rb[ii]**2-.9759930144*rb[ii]+\
.2940938997))
self.rb5n1.append(abs(.7407177955*rb[ii]**8-3.449431236*rb[ii]**7+\
6.830935334*rb[ii]**6-7.741737467*rb[ii]**5+6.699199978*rb[ii]**4-\
5.927353604*rb[ii]**3+4.426525143*rb[ii]**2-1.910862054*rb[ii]+\
.3320296729))
self.rb4n1.append(abs(1.433400787*rb[ii]**8-4.513152228*rb[ii]**7+\
5.521961253*rb[ii]**6-5.064383659*rb[ii]**5+7.253227925*rb[ii]**4-\
9.015491763*rb[ii]**3+6.296203951*rb[ii]**2-2.226500358*rb[ii]+\
.3147503647))
self.rb2n1.append(abs(1.948646712*rb[ii]**8-5.429471356*rb[ii]**7+\
6.121691684*rb[ii]**6-6.922030298*rb[ii]**5+10.81343425*rb[ii]**4-\
11.18283160*rb[ii]**3+6.159356196*rb[ii]**2-1.693092334*rb[ii]+\
.1843175380))

self.rb1p1.append(abs(10.25226170*rb[ii]**8-3.761689131*rb[ii]**7-\
51.69628729*rb[ii]**6+73.14881926*rb[ii]**5-19.22843892*rb[ii]**4-\
18.57376761*rb[ii]**3+12.39421862*rb[ii]**2-3.189760449*rb[ii]+\
.6544484890))
self.rb95p1.append(abs(7.888898834*rb[ii]**8-8.161623551*rb[ii]**7\
-17.75528024*rb[ii]**6+22.52397903*rb[ii]**5+7.740439149*rb[ii]**4-\
19.60988130*rb[ii]**3+9.193827237*rb[ii]**2-2.340763866*rb[ii]+\
.5203713403))
self.rb9p1.append(abs(6.265483353*rb[ii]**8-9.913596162*rb[ii]**7+\
.5535162992*rb[ii]**6-4.801769667*rb[ii]**5+20.93640507*rb[ii]**4-\
```

```
18.15256063*rb[ii]**3+6.212812829*rb[ii]**2-1.500710856*rb[ii]+\
.4004757394))
self.rb85p1.append(abs(3.592100743*rb[ii]**8-6.673301592*rb[ii]**7+\
6.419999336*rb[ii]**6-13.91828062*rb[ii]**5+18.91444376*rb[ii]**4-\
9.575768227*rb[ii]**3+1.348910219*rb[ii]**2-.3814060671*rb[ii]+\
.2733793865))
self.rb8p1.append(abs(1.602493688*rb[ii]**8-4.894827656*rb[ii]**7+\
12.77414826*rb[ii]**6-23.44316036*rb[ii]**5+19.91919405*rb[ii]**4-\
4.921038943*rb[ii]**3-1.537792495*rb[ii]**2+.3196870140*rb[ii]+\
.1813853546))
self.rb7p1.append(abs(-1.206980125*rb[ii]**8-.7107303019*rb[ii]**7+\
14.51383728*rb[ii]**6-25.54091555*rb[ii]**5+13.92903874*rb[ii]**4+\
2.659266525*rb[ii]**3-4.681464815*rb[ii]**2+.9594359006*rb[ii]+\
0.07857806213))
self.rb6p1.append(abs(-2.286986183*rb[ii]**8+2.141788490*rb[ii]**7+\
9.608575585*rb[ii]**6-17.28673707*rb[ii]**5+5.463249133*rb[ii]**4+\
6.539951960*rb[ii]**3-5.189158111*rb[ii]**2+.9702566103*rb[ii]+\
0.03908928388))
self.rb5p1.append(abs(-2.151739510*rb[ii]**8+2.613762839*rb[ii]**7+\
5.846312040*rb[ii]**6-10.57886300*rb[ii]**5+1.265656829*rb[ii]**4+\
6.375645106*rb[ii]**3-4.094517080*rb[ii]**2+.6963433450*rb[ii]+\
0.02741016557))
self.rb4p1.append(abs(-1.485246548*rb[ii]**8+2.525922765*rb[ii]**7+\
.7419019803*rb[ii]**6-1.765749029*rb[ii]**5-3.150174653*rb[ii]**4+\
5.230841867*rb[ii]**3-2.417211026*rb[ii]**2+.2858212248*rb[ii]+\
0.03388713915))
self.rb2p1.append(abs(.5847114699*rb[ii]**8-2.425410760*rb[ii]**7+\
5.127113307*rb[ii]**6-7.358102110*rb[ii]**5+7.272920648*rb[ii]**4-\
4.687463633*rb[ii]**3+1.869016360*rb[ii]**2-.4267313064*rb[ii]+\
0.04396202800))

self.rb1n2.append(abs(0.0))
self.rb95n2.append(abs(.5345244882*rb[ii]**8-.8535005374*rb[ii]**7-\
0.7445379983e-1*rb[ii]**6-.4217139041*rb[ii]**5+2.641202656*rb[ii]**4-\
2.678776838*rb[ii]**3+.8220348784*rb[ii]**2+0.9988273293e-1*rb[ii]+\
0.02831096120))
self.rb9n2.append(abs(.7869564000*rb[ii]**8-1.304216845*rb[ii]**7+\
.6007645932*rb[ii]**6-2.720512168*rb[ii]**5+6.585463212*rb[ii]**4-\
5.753668370*rb[ii]**3+1.897333164*rb[ii]**2-0.9711187488e-2*rb[ii]+\
.1076139833))
self.rb8n2.append(abs(-0.05291401582*rb[ii]**8+.1382543867*rb[ii]**7+\
2.576382137*rb[ii]**6-9.308146896*rb[ii]**5+12.28890487*rb[ii]**4-\
7.225383494*rb[ii]**3+1.361607402*rb[ii]**2+.3603859446*rb[ii]+\
.2209360050))
self.rb7n2.append(abs(.5650190053*rb[ii]**8-.2033762112*rb[ii]**7-\
```

```
5.214567130*rb[ii]**6+12.15251206*rb[ii]**5-12.47834266*rb[ii]**4+\
7.871047978*rb[ii]**3-3.819318704*rb[ii]**2+1.346113853*rb[ii]+\
.2908638512))
self.rb6n2.append(abs(.4839771746*rb[ii]**8-.1773265160*rb[ii]**7-\
4.087527112*rb[ii]**6+9.077821099*rb[ii]**5-8.945636861*rb[ii]**4+\
5.810025855*rb[ii]**3-3.212353887*rb[ii]**2+1.265872190*rb[ii]+\
.4251091385))
self.rb5n2.append(abs(2.331871509*rb[ii]**8-3.154070140*rb[ii]**7-\
6.016123425*rb[ii]**6+13.49158999*rb[ii]**5-7.224891559*rb[ii]**4-\
0.001686767694*rb[ii]**3+.2745327405*rb[ii]**2+.4411961237*rb[ii]+\
.6075501137))
self.rb4n2.append(abs(1.446677211*rb[ii]**8-1.922554135*rb[ii]**7-\
3.001814692*rb[ii]**6+5.350322500*rb[ii]**5+0.1391802322e-1*rb[ii]**4-\
3.101591427*rb[ii]**3+1.086517571*rb[ii]**2+.2380060733*rb[ii]+\
.7305040678))
self.rb2n2.append(abs(.9866482156*rb[ii]**8-1.405067293*rb[ii]**7-\
.3388428028*rb[ii]**6-1.334406671*rb[ii]**5+6.078742699*rb[ii]**4-\
5.855478532*rb[ii]**3+2.028442528*rb[ii]**2-.1261315999*rb[ii]+\
.9261044321))

self.rb1p2.append(abs(0.0))
self.rb95p2.append(abs(.6466110328*rb[ii]**8-1.358978603*rb[ii]**7+\
3.438038698*rb[ii]**6-11.86520822*rb[ii]**5+19.29510815*rb[ii]**4-\
13.55552608*rb[ii]**3+3.137149454*rb[ii]**2+.3107992090*rb[ii]+\
0.04956079205))
self.rb9p2.append(abs(-2.733613161*rb[ii]**8+5.613453536*rb[ii]**7+\
5.522761915*rb[ii]**6-20.38988508*rb[ii]**5+15.90246759*rb[ii]**4-\
2.729714829*rb[ii]**3-2.524304349*rb[ii]**2+1.423928273*rb[ii]+\
.1049785733))
self.rb85p2.append(abs(3.592100743*rb[ii]**8-6.673301592*rb[ii]**7+\
6.419999336*rb[ii]**6-13.91828062*rb[ii]**5+18.91444376*rb[ii]**4-\
9.575768227*rb[ii]**3+1.348910219*rb[ii]**2-.3814060671*rb[ii]+\
.2733793865))
self.rb8p2.append(abs(0.6082574542*rb[ii]**8-1.848169821*rb[ii]**7-\
3.550148086*rb[ii]**6+13.50188995*rb[ii]**5-11.45231416*rb[ii]**4+\
2.909235461*rb[ii]**3-.9096284577*rb[ii]**2+.7449985792*rb[ii]+\
.3558329744))
self.rb7p2.append(abs(0.5233641006*rb[ii]**8-1.778365257*rb[ii]**7-\
.8050688742*rb[ii]**6+4.712017551*rb[ii]**5+0.01906514336*rb[ii]**4-\
5.071225485*rb[ii]**3+2.332317032*rb[ii]**2+0.01654188602*rb[ii]+\
.5613293197))
self.rb6p2.append(abs(-.2920445899*rb[ii]**8-.09862560243*rb[ii]**7-\
1.674661791*rb[ii]**6+7.107371741*rb[ii]**5-7.075577220*rb[ii]**4+\
2.313746504*rb[ii]**3-.8194009453*rb[ii]**2+.5351983633*rb[ii]+\
.6439603166))
```

```
        self.rb5p2.append(abs(-.2319341368*rb[ii]**8+.2636820154*rb[ii]**7-\
        1.696854642*rb[ii]**6+6.018141843*rb[ii]**5-7.420706264*rb[ii]**4+\
        4.294600498*rb[ii]**3-1.830953337*rb[ii]**2+.6099717582*rb[ii]+\
        .7440306198))
        self.rb4p2.append(abs(0.6823356160*rb[ii]**8-.9723781215*rb[ii]**7-\
        1.436093355*rb[ii]**6+4.422788399*rb[ii]**5-4.073069237*rb[ii]**4+\
        1.754256219*rb[ii]**3-.6035871334*rb[ii]**2+.2043161922*rb[ii]+\
        .8614283218))
        self.rb2p2.append(abs(-.3477590237*rb[ii]**8+.5284678812*rb[ii]**7+\
        2.871125103*rb[ii]**6-8.833417770*rb[ii]**5+9.965919280*rb[ii]**4-\
        5.594850754*rb[ii]**3+1.616314926*rb[ii]**2-.2349596558*rb[ii]+\
        .9891879820))

        ii+=1

iii = 0
yfacef1         = []
yfacef95        = []
yfacef9         = []
yfacef8         = []
yfacef7         = []
yfacef6         = []
yfacef5         = []
yfacef4         = []
yfacef2         = []

yfacea1         = []
yfacea95        = []
yfacea9         = []
yfacea8         = []
yfacea7         = []
yfacea6         = []
yfacea5         = []
yfacea4         = []
yfacea2         = []

ybackf1         = []
ybackf95        = []
ybackf9         = []
ybackf8         = []
ybackf7         = []
ybackf6         = []
ybackf5         = []
ybackf4         = []
ybackf2         = []
```

```
ybacka1         = []
ybacka95        = []
ybacka9         = []
ybacka8         = []
ybacka7         = []
ybacka6         = []
ybacka5         = []
ybacka4         = []
ybacka2         = []

xf1             = []
xf2             = []
xf3             = []
xf4             = []
xf5             = []
xf6             = []
xf7             = []
xf8             = []
xf9             = []

xa1             = []
xa2             = []
xa3             = []
xa4             = []
xa5             = []
xa6             = []
xa7             = []
xa8             = []
xa9             = []

self.L          = []          # Lift at section
self.M          = []          # Moment at chord length

# Location of y coordinate
while iii <= rn:
    yfacef1.append(self.rb1p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef95.append(self.rb95p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef9.append(self.rb9p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef8.append(self.rb8p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef7.append(self.rb7p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef6.append(self.rb6p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef5.append(self.rb5p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef4.append(self.rb4p1[iii]*(tmax[iii]-self.tle[iii]))
    yfacef2.append(self.rb2p1[iii]*(tmax[iii]-self.tle[iii]))
```

```
ybackf1.append((self.rb1p1[iii]+self.rb1p2[iii])*(tmax[iii]\
            -self.tle[iii])+self.tle[iii])
ybackf95.append((self.rb95p1[iii]+self.rb95p2[iii])*(tmax[iii]\
            -self.tle[iii])+self.tle[iii])
ybackf9.append((self.rb9p1[iii]+self.rb9p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf8.append((self.rb8p1[iii]+self.rb8p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf7.append((self.rb7p1[iii]+self.rb7p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf6.append((self.rb6p1[iii]+self.rb6p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf5.append((self.rb5p1[iii]+self.rb5p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf4.append((self.rb4p1[iii]+self.rb4p2[iii])*(tmax[iii]\
         -self.tle[iii])+self.tle[iii])
ybackf2.append((self.rb2p1[iii]+self.rb2p2[iii])*(tmax[iii]\
        -self.tle[iii])+self.tle[iii])

yfacea1.append(self.rb1n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea95.append(self.rb95n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea9.append(self.rb9n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea8.append(self.rb8n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea7.append(self.rb7n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea6.append(self.rb6n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea5.append(self.rb5n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea4.append(self.rb4n1[iii]*(tmax[iii]-self.tte[iii]))
yfacea2.append(self.rb2n1[iii]*(tmax[iii]-self.tte[iii]))

ybacka1.append((self.rb1n1[iii]+self.rb1p2[iii])*(tmax[iii]\
        -self.tte[iii])+self.tte[iii])
ybacka95.append((self.rb95n1[iii]+self.rb95p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka9.append((self.rb9n1[iii]+self.rb9p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka8.append((self.rb8n1[iii]+self.rb8p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka7.append((self.rb7n1[iii]+self.rb7p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka6.append((self.rb6n1[iii]+self.rb6p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka5.append((self.rb5n1[iii]+self.rb5p2[iii])*(tmax[iii]-\
        self.tte[iii])+self.tte[iii])
ybacka4.append((self.rb4n1[iii]+self.rb4p2[iii])*(tmax[iii]-\
```

```
                self.tte[iii])+self.tte[iii])
        ybacka2.append((self.rb2n1[iii]+self.rb2p2[iii])*(tmax[iii]-\
                self.tte[iii])+self.tte[iii])

# Set up x coordinates
        xf1.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.2))
        xf2.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.4))
        xf3.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.5))
        xf4.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.6))
        xf5.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.7))
        xf6.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.8))
        xf7.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.9))
        xf8.append(xcr[iii]+((Chrd[iii]-xcr[iii])*.95))
        xf9.append(xcr[iii]+((Chrd[iii]-xcr[iii])*1.0))

        xa1.append(0.0)
        xa2.append((xcr[iii])*.05)
        xa3.append((xcr[iii])*.1)
        xa4.append((xcr[iii])*.2)
        xa5.append((xcr[iii])*.3)
        xa6.append((xcr[iii])*.4)
        xa7.append((xcr[iii])*.5)
        xa8.append((xcr[iii])*.6)
        xa9.append((xcr[iii])*.8)

        self.L.append((pi/4.)*CL[iii]*(1./2.)*rho*(D/2.)*Chrd[iii]*(xfree[0]**2.))
        self.M.append(self.L[iii]*(D/2.)/(3.*pi))

        iii += 1

# Set up coordinate system to calculate strenght characteristics
    cp0   = array([[xa1[0],xa2[0],xa3[0],xa4[0],xa5[0],xa6[0],xa7[0],\
        xa8[0],xa9[0],xf1[0],xf2[0],xf3[0],xf4[0],xf5[0],xf6[0],xf7[0],\
        xf8[0],xf9[0],xf8[0],xf7[0],xf6[0],xf5[0],xf4[0],xf3[0],xf2[0],\
        xf1[0],xa9[0],xa8[0],xa7[0],xa6[0],xa5[0],xa4[0],xa3[0],xa2[0],\
        xa1[0]],[yfacea1[0],yfacea95[0],yfacea9[0],yfacea8[0],yfacea7[0],\
        yfacea6[0],yfacea5[0],yfacea4[0],yfacea2[0],yfacef2[0],yfacef4[0],\
        yfacef5[0],yfacef6[0],yfacef7[0],yfacef8[0],yfacef9[0],yfacef95[0],\
        yfacef1[0],ybackf95[0],ybackf9[0],ybackf8[0],ybackf7[0],ybackf6[0],\
        ybackf5[0],ybackf4[0],ybackf2[0],ybacka2[0],ybacka4[0],ybacka5[0],\
        ybacka6[0],ybacka7[0],ybacka8[0],ybacka9[0],ybacka95[0],ybacka1[0]]])
    cp1   = array([[xa1[1],xa2[1],xa3[1],xa4[1],xa5[1],xa6[1],xa7[1],\
        xa8[1],xa9[1],xf1[1],xf2[1],xf3[1],xf4[1],xf5[1],xf6[1],xf7[1],\
        xf8[1],xf9[1],xf8[1],xf7[1],xf6[1],xf5[1],xf4[1],xf3[1],xf2[1],\
        xf1[1],xa9[1],xa8[1],xa7[1],xa6[1],xa5[1],xa4[1],xa3[1],xa2[1],\
```

```
    xa1[1]],[yfacea1[1],yfacea95[1],yfacea9[1],yfacea8[1],yfacea7[1],\
    yfacea6[1],yfacea5[1],yfacea4[1],yfacea2[1],yfacef2[1],yfacef4[1],\
    yfacef5[1],yfacef6[1],yfacef7[1],yfacef8[1],yfacef9[1],yfacef95[1],\
    yfacef1[0],ybackf95[1],ybackf9[1],ybackf8[1],ybackf7[1],ybackf6[1],\
    ybackf5[1],ybackf4[1],ybackf2[1],ybacka2[1],ybacka4[1],ybacka5[1],\
    ybacka6[1],ybacka7[1],ybacka8[1],ybacka9[1],ybacka95[1],ybacka1[1]]])
cp2   = array([[xa1[2],xa2[2],xa3[2],xa4[2],xa5[2],xa6[2],xa7[2],\
    xa8[2],xa9[2],xf1[2],xf2[2],xf3[2],xf4[2],xf5[2],xf6[2],xf7[2],\
    xf8[2],xf9[2],xf8[2],xf7[2],xf6[2],xf5[2],xf4[2],xf3[2],xf2[2],\
    xf1[2],xa9[2],xa8[2],xa7[2],xa6[2],xa5[2],xa4[2],xa3[2],xa2[2],\
    xa1[2]],[yfacea1[2],yfacea95[2],yfacea9[2],yfacea8[2],yfacea7[2],\
    yfacea6[2],yfacea5[2],yfacea4[2],yfacea2[2],yfacef2[2],yfacef4[2],\
    yfacef5[2],yfacef6[2],yfacef7[2],yfacef8[2],yfacef9[2],yfacef95[2],\
    yfacef1[0],ybackf95[2],ybackf9[2],ybackf8[2],ybackf7[2],ybackf6[2],\
    ybackf5[2],ybackf4[2],ybackf2[2],ybacka2[2],ybacka4[2],ybacka5[2],\
    ybacka6[2],ybacka7[2],ybacka8[2],ybacka9[2],ybacka95[2],ybacka1[2]]])
cp3   = array([[xa1[3],xa2[3],xa3[3],xa4[3],xa5[3],xa6[3],xa7[3],\
    xa8[3],xa9[3],xf1[3],xf2[3],xf3[3],xf4[3],xf5[3],xf6[3],xf7[3],\
    xf8[3],xf9[3],xf8[3],xf7[3],xf6[3],xf5[3],xf4[3],xf3[3],xf2[3],\
    xf1[3],xa9[3],xa8[3],xa7[3],xa6[3],xa5[3],xa4[3],xa3[3],xa2[3],\
    xa1[3]],[yfacea1[3],yfacea95[3],yfacea9[3],yfacea8[3],yfacea7[3],\
    yfacea6[3],yfacea5[3],yfacea4[3],yfacea2[3],yfacef2[3],yfacef4[3],\
    yfacef5[3],yfacef6[3],yfacef7[3],yfacef8[3],yfacef9[3],yfacef95[3],\
    yfacef1[0],ybackf95[3],ybackf9[3],ybackf8[3],ybackf7[3],ybackf6[3],\
    ybackf5[3],ybackf4[3],ybackf2[3],ybacka2[3],ybacka4[3],ybacka5[3],\
    ybacka6[3],ybacka7[3],ybacka8[3],ybacka9[3],ybacka95[3],ybacka1[3]]])
cp4   = array([[xa1[4],xa2[4],xa3[4],xa4[4],xa5[4],xa6[4],xa7[4],\
    xa8[4],xa9[4],xf1[4],xf2[4],xf3[4],xf4[4],xf5[4],xf6[4],xf7[4],\
    xf8[4],xf9[4],xf8[4],xf7[4],xf6[4],xf5[4],xf4[4],xf3[4],xf2[4],\
    xf1[4],xa9[4],xa8[4],xa7[4],xa6[4],xa5[4],xa4[4],xa3[4],xa2[4],\
    xa1[4]],[yfacea1[4],yfacea95[4],yfacea9[4],yfacea8[4],yfacea7[4],\
    yfacea6[4],yfacea5[4],yfacea4[4],yfacea2[4],yfacef2[4],yfacef4[4],\
    yfacef5[4],yfacef6[4],yfacef7[4],yfacef8[4],yfacef9[4],yfacef95[4],\
    yfacef1[0],ybackf95[4],ybackf9[4],ybackf8[4],ybackf7[4],ybackf6[4],\
    ybackf5[4],ybackf4[4],ybackf2[4],ybacka2[4],ybacka4[4],ybacka5[4],\
    ybacka6[4],ybacka7[4],ybacka8[4],ybacka9[4],ybacka95[4],ybacka1[4]]])
cp5   = array([[xa1[5],xa2[5],xa3[5],xa4[5],xa5[5],xa6[5],xa7[5],\
    xa8[5],xa9[5],xf1[5],xf2[5],xf3[5],xf4[5],xf5[5],xf6[5],xf7[5],\
    xf8[5],xf9[5],xf8[5],xf7[5],xf6[5],xf5[5],xf4[5],xf3[5],xf2[5],\
    xf1[5],xa9[5],xa8[5],xa7[5],xa6[5],xa5[5],xa4[5],xa3[5],xa2[5],\
    xa1[5]],[yfacea1[5],yfacea95[5],yfacea9[5],yfacea8[5],yfacea7[5],\
    yfacea6[5],yfacea5[5],yfacea4[5],yfacea2[5],yfacef2[5],yfacef4[5],
    yfacef5[5],yfacef6[5],yfacef7[5],yfacef8[5],yfacef9[5],yfacef95[5],\
    yfacef1[0],ybackf95[5],ybackf9[5],ybackf8[5],ybackf7[5],ybackf6[5],\
    ybackf5[5],ybackf4[5],ybackf2[5],ybacka2[5],ybacka4[5],ybacka5[5],\
```

```
    ybacka6[5],ybacka7[5],ybacka8[5],ybacka9[5],ybacka95[5],ybacka1[5]]])
cp6   = array([[xa1[6],xa2[6],xa3[6],xa4[6],xa5[6],xa6[6],xa7[6],\
    xa8[6],xa9[6],xf1[6],xf2[6],xf3[6],xf4[6],xf5[6],xf6[6],xf7[6],\
    xf8[6],xf9[6],xf8[6],xf7[6],xf6[6],xf5[6],xf4[6],xf3[6],xf2[6],\
    xf1[6],xa9[6],xa8[6],xa7[6],xa6[6],xa5[6],xa4[6],xa3[6],xa2[6],\
    xa1[6]],[yfacea1[6],yfacea95[6],yfacea9[6],yfacea8[6],yfacea7[6],\
    yfacea6[6],yfacea5[6],yfacea4[6],yfacea2[6],yfacef2[6],yfacef4[6],\
    yfacef5[6],yfacef6[6],yfacef7[6],yfacef8[6],yfacef9[6],yfacef95[6],\
    yfacef1[0],ybackf95[6],ybackf9[6],ybackf8[6],ybackf7[6],ybackf6[6],\
    ybackf5[6],ybackf4[6],ybackf2[6],ybacka2[6],ybacka4[6],ybacka5[6],\
    ybacka6[6],ybacka7[6],ybacka8[6],ybacka9[6],ybacka95[6],ybacka1[6]]])
cp7   = array([[xa1[7],xa2[7],xa3[7],xa4[7],xa5[7],xa6[7],xa7[7],\
    xa8[7],xa9[7],xf1[7],xf2[7],xf3[7],xf4[7],xf5[7],xf6[7],xf7[7],\
    xf8[7],xf9[7],xf8[7],xf7[7],xf6[7],xf5[7],xf4[7],xf3[7],xf2[7],\
    xf1[7],xa9[7],xa8[7],xa7[7],xa6[7],xa5[7],xa4[7],xa3[7],xa2[7],\
    xa1[7]],[yfacea1[7],yfacea95[7],yfacea9[7],yfacea8[7],yfacea7[7],\
    yfacea6[7],yfacea5[7],yfacea4[7],yfacea2[7],yfacef2[7],yfacef4[7],\
    yfacef5[7],yfacef6[7],yfacef7[7],yfacef8[7],yfacef9[7],yfacef95[7],\
    yfacef1[0],ybackf95[7],ybackf9[7],ybackf8[7],ybackf7[7],ybackf6[7],\
    ybackf5[7],ybackf4[7],ybackf2[7],ybacka2[7],ybacka4[7],ybacka5[7],\
    ybacka6[7],ybacka7[7],ybacka8[7],ybacka9[7],ybacka95[7],ybacka1[7]]])
cp8   = array([[xa1[8],xa2[8],xa3[8],xa4[8],xa5[8],xa6[8],xa7[8],\
    xa8[8],xa9[8],xf1[8],xf2[8],xf3[8],xf4[8],xf5[8],xf6[8],xf7[8],\
    xf8[8],xf9[8],xf8[8],xf7[8],xf6[8],xf5[8],xf4[8],xf3[8],xf2[8],\
    xf1[8],xa9[8],xa8[8],xa7[8],xa6[8],xa5[8],xa4[8],xa3[8],xa2[8],\
    xa1[8]],[yfacea1[8],yfacea95[8],yfacea9[8],yfacea8[8],yfacea7[8],\
    yfacea6[8],yfacea5[8],yfacea4[8],yfacea2[8],yfacef2[8],yfacef4[8],\
    yfacef5[8],yfacef6[8],yfacef7[8],yfacef8[8],yfacef9[8],yfacef95[8],\
    yfacef1[0],ybackf95[8],ybackf9[8],ybackf8[8],ybackf7[8],ybackf6[8],\
    ybackf5[8],ybackf4[8],ybackf2[8],ybacka2[8],ybacka4[8],ybacka5[8],\
    ybacka6[8],ybacka7[8],ybacka8[8],ybacka9[8],ybacka95[8],ybacka1[8]]])
cp9   = array([[xa1[9],xa2[9],xa3[9],xa4[9],xa5[9],xa6[9],xa7[9],\
    xa8[9],xa9[9],xf1[9],xf2[9],xf3[9],xf4[9],xf5[9],xf6[9],xf7[9],\
    xf8[9],xf9[9],xf8[9],xf7[9],xf6[9],xf5[9],xf4[9],xf3[9],xf2[9],\
    xf1[9],xa9[9],xa8[9],xa7[9],xa6[9],xa5[9],xa4[9],xa3[9],xa2[9],\
    xa1[9]],[yfacea1[9],yfacea95[9],yfacea9[9],yfacea8[9],yfacea7[9],\
    yfacea6[9],yfacea5[9],yfacea4[9],yfacea2[9],yfacef2[9],yfacef4[9],\
    yfacef5[9],yfacef6[9],yfacef7[9],yfacef8[9],yfacef9[9],yfacef95[9],\
    yfacef1[0],ybackf95[9],ybackf9[9],ybackf8[9],ybackf7[9],ybackf6[9],\
    ybackf5[9],ybackf4[9],ybackf2[9],ybacka2[9],ybacka4[9],ybacka5[9],\
    ybacka6[9],ybacka7[9],ybacka8[9],ybacka9[9],ybacka95[9],ybacka1[9]]])
cp10  = array([[xa1[10],xa2[10],xa3[10],xa4[10],xa5[10],xa6[10],\
    xa7[10],xa8[10],xa9[10],xf1[10],xf2[10],xf3[10],xf4[10],xf5[10],\
    xf6[10],xf7[10],xf8[10],xf9[10],xf8[10],xf7[10],xf6[10],xf5[10],\
    xf4[10],xf3[10],xf2[10],xf1[10],xa9[10],xa8[10],xa7[10],xa6[10],\
```

```
    xa5[10],xa4[10],xa3[10],xa2[10],xa1[10]],[yfacea1[10],yfacea95[10],\
    yfacea9[10],yfacea8[10],yfacea7[10],yfacea6[10],yfacea5[10],\
    yfacea4[10],yfacea2[10],yfacef2[10],yfacef4[10],yfacef5[10],\
    yfacef6[10],yfacef7[10],yfacef8[10],yfacef9[10],yfacef95[10],\
    yfacef1[10],ybackf95[10],ybackf9[10],ybackf8[10],ybackf7[10],\
    ybackf6[10],ybackf5[10],ybackf4[10],ybackf2[10],ybacka2[10],\
    ybacka4[10],ybacka5[10],ybacka6[10],ybacka7[10],ybacka8[10],\
    ybacka9[10],ybacka95[10],ybacka1[10]]])
cp11  = array([[xa1[11],xa2[11],xa3[11],xa4[11],xa5[11],xa6[11],\
    xa7[11],xa8[11],xa9[11],xf1[11],xf2[11],xf3[11],xf4[11],xf5[11],\
    xf6[11],xf7[11],xf8[11],xf9[11],xf8[11],xf7[11],xf6[11],xf5[11],\
    xf4[11],xf3[11],xf2[11],xf1[11],xa9[11],xa8[11],xa7[11],xa6[11],\
    xa5[11],xa4[11],xa3[11],xa2[11],xa1[11]],[yfacea1[11],yfacea95[11],\
    yfacea9[11],yfacea8[11],yfacea7[11],yfacea6[11],yfacea5[11],\
    yfacea4[11],yfacea2[11],yfacef2[11],yfacef4[11],yfacef5[11],\
    yfacef6[11],yfacef7[11],yfacef8[11],yfacef9[11],yfacef95[11],\
    yfacef1[11],ybackf95[11],ybackf9[11],ybackf8[11],ybackf7[11],\
    ybackf6[11],ybackf5[11],ybackf4[11],ybackf2[11],ybacka2[11],\
    ybacka4[11],ybacka5[11],ybacka6[11],ybacka7[11],ybacka8[11],\
    ybacka9[11],ybacka95[11],ybacka1[11]]])
cp12  = array([[xa1[12],xa2[12],xa3[12],xa4[12],xa5[12],xa6[12],\
    xa7[12],xa8[12],xa9[12],xf1[12],xf2[12],xf3[12],xf4[12],xf5[12],\
    xf6[12],xf7[12],xf8[12],xf9[12],xf8[12],xf7[12],xf6[12],xf5[12],\
    xf4[12],xf3[12],xf2[12],xf1[12],xa9[12],xa8[12],xa7[12],xa6[12],\
    xa5[12],xa4[12],xa3[12],xa2[12],xa1[12]],[yfacea1[12],yfacea95[12],\
    yfacea9[12],yfacea8[12],yfacea7[12],yfacea6[12],yfacea5[12],\
    yfacea4[12],yfacea2[12],yfacef2[12],yfacef4[12],yfacef5[12],\
    yfacef6[12],yfacef7[12],yfacef8[12],yfacef9[12],yfacef95[12],\
    yfacef1[12],ybackf95[12],ybackf9[12],ybackf8[12],ybackf7[12],\
    ybackf6[12],ybackf5[12],ybackf4[12],ybackf2[12],ybacka2[12],\
    ybacka4[12],ybacka5[12],ybacka6[12],ybacka7[12],ybacka8[12],\
    ybacka9[12],ybacka95[12],ybacka1[12]]])
cp13  = array([[xa1[13],xa2[13],xa3[13],xa4[13],xa5[13],xa6[13],\
    xa7[13],xa8[13],xa9[13],xf1[13],xf2[13],xf3[13],xf4[13],xf5[13],\
    xf6[13],xf7[13],xf8[13],xf9[13],xf8[13],xf7[13],xf6[13],xf5[13],\
    xf4[13],xf3[13],xf2[13],xf1[13],xa9[13],xa8[13],xa7[13],xa6[13],\
    xa5[13],xa4[13],xa3[13],xa2[13],xa1[13]],[yfacea1[13],yfacea95[13],\
    yfacea9[13],yfacea8[13],yfacea7[13],yfacea6[13],yfacea5[13],\
    yfacea4[13],yfacea2[13],yfacef2[13],yfacef4[13],yfacef5[13],\
    yfacef6[13],yfacef7[13],yfacef8[13],yfacef9[13],yfacef95[13],\
    yfacef1[13],ybackf95[13],ybackf9[13],ybackf8[13],ybackf7[13],\
    ybackf6[13],ybackf5[13],ybackf4[13],ybackf2[13],ybacka2[13],\
    ybacka4[13],ybacka5[13],ybacka6[13],ybacka7[13],ybacka8[13],\
    ybacka9[13],ybacka95[13],ybacka1[13]]])
cp14  = array([[xa1[14],xa2[14],xa3[14],xa4[14],xa5[14],xa6[14],\
```

```
    xa7[14],xa8[14],xa9[14],xf1[14],xf2[14],xf3[14],xf4[14],xf5[14],\
    xf6[14],xf7[14],xf8[14],xf9[14],xf8[14],xf7[14],xf6[14],xf5[14],\
    xf4[14],xf3[14],xf2[14],xf1[14],xa9[14],xa8[14],xa7[14],xa6[14],\
    xa5[14],xa4[14],xa3[14],xa2[14],xa1[14]],[yfacea1[14],yfacea95[14],\
    yfacea9[14],yfacea8[14],yfacea7[14],yfacea6[14],yfacea5[14],\
    yfacea4[14],yfacea2[14],yfacef2[14],yfacef4[14],yfacef5[14],\
    yfacef6[14],yfacef7[14],yfacef8[14],yfacef9[14],yfacef95[14],\
    yfacef1[14],ybackf95[14],ybackf9[14],ybackf8[14],ybackf7[14],\
    ybackf6[14],ybackf5[14],ybackf4[14],ybackf2[14],ybacka2[14],\
    ybacka4[14],ybacka5[14],ybacka6[14],ybacka7[14],ybacka8[14],\
    ybacka9[14],ybacka95[14],ybacka1[14]]])
cp15  = array([[xa1[15],xa2[15],xa3[15],xa4[15],xa5[15],xa6[15],\
    xa7[15],xa8[15],xa9[15],xf1[15],xf2[15],xf3[15],xf4[15],xf5[15],\
    xf6[15],xf7[15],xf8[15],xf9[15],xf8[15],xf7[15],xf6[15],xf5[15],\
    xf4[15],xf3[15],xf2[15],xf1[15],xa9[15],xa8[15],xa7[15],xa6[15],\
    xa5[15],xa4[15],xa3[15],xa2[15],xa1[15]],[yfacea1[15],yfacea95[15],\
    yfacea9[15],yfacea8[15],yfacea7[15],yfacea6[15],yfacea5[15],\
    yfacea4[15],yfacea2[15],yfacef2[15],yfacef4[15],yfacef5[15],\
    yfacef6[15],yfacef7[15],yfacef8[15],yfacef9[15],yfacef95[15],\
    yfacef1[15],ybackf95[15],ybackf9[15],ybackf8[15],ybackf7[15],\
    ybackf6[15],ybackf5[15],ybackf4[15],ybackf2[15],ybacka2[15],\
    ybacka4[15],ybacka5[15],ybacka6[15],ybacka7[15],ybacka8[15],\
    ybacka9[15],ybacka95[15],ybacka1[15]]])


Area0, xs0, ys0, Mxp0, Myp0, Ix0, Iy0, Ixy0 = plasi(cp0)
Area1, xs1, ys1, Mxp1, Myp1, Ix1, Iy1, Ixy1 = plasi(cp1)
Area2, xs2, ys2, Mxp2, Myp2, Ix2, Iy2, Ixy2 = plasi(cp2)
Area3, xs3, ys3, Mxp3, Myp3, Ix3, Iy3, Ixy3 = plasi(cp3)
Area4, xs4, ys4, Mxp4, Myp4, Ix4, Iy4, Ixy4 = plasi(cp4)
Area5, xs5, ys5, Mxp5, Myp5, Ix5, Iy5, Ixy5 = plasi(cp5)
Area6, xs6, ys6, Mxp6, Myp6, Ix6, Iy6, Ixy6 = plasi(cp6)
Area7, xs7, ys7, Mxp7, Myp7, Ix7, Iy7, Ixy7 = plasi(cp7)
Area8, xs8, ys8, Mxp8, Myp8, Ix8, Iy8, Ixy8 = plasi(cp8)
Area9, xs9, ys9, Mxp9, Myp9, Ix9, Iy9, Ixy9 = plasi(cp9)
Area10, xs10, ys10, Mxp10, Myp10, Ix10, Iy10, Ixy10 = plasi(cp10)
Area11, xs11, ys11, Mxp11, Myp11, Ix11, Iy11, Ixy11 = plasi(cp11)
Area12, xs12, ys12, Mxp12, Myp12, Ix12, Iy12, Ixy12 = plasi(cp12)
Area13, xs13, ys13, Mxp13, Myp13, Ix13, Iy13, Ixy13 = plasi(cp13)
Area14, xs14, ys14, Mxp14, Myp14, Ix14, Iy14, Ixy14 = plasi(cp14)
Area15, xs15, ys15, Mxp15, Myp15, Ix15, Iy15, Ixy15 = plasi(cp15)


# Moment Calculations
```

```
Mx0  = (0.7*(D/2)-((0./16.)*(D/2)))*THST
Mx1  = (0.7*(D/2)-((1./16.)*(D/2)))*THST
Mx2  = (0.7*(D/2)-((2./16.)*(D/2)))*THST
Mx3  = (0.7*(D/2)-((3./16.)*(D/2)))*THST
Mx4  = (0.7*(D/2)-((4./16.)*(D/2)))*THST
Mx5  = (0.7*(D/2)-((5./16.)*(D/2)))*THST
Mx6  = (0.7*(D/2)-((6./16.)*(D/2)))*THST
Mx7  = (0.7*(D/2)-((7./16.)*(D/2)))*THST
Mx8  = (0.7*(D/2)-((8./16.)*(D/2)))*THST
Mx9  = (0.7*(D/2)-((9./16.)*(D/2)))*THST
Mx10 = (0.7*(D/2)-((10./16.)*(D/2)))*THST
Mx11 = (0.7*(D/2)-((11./16.)*(D/2)))*THST
Mx12 = 0.*(D/2)*THST
Mx13 = 0.*(D/2)*THST
Mx14 = 0.*(D/2)*THST
Mx15 = 0.*(D/2)*THST

#############################################################################
# Moment of Inertia

Ixc0     = Ix0 + Area0*ys0
Ixc1     = Ix1 + Area1*ys1
Ixc2     = Ix2 + Area2*ys2
Ixc3     = Ix3 + Area3*ys3
Ixc4     = Ix4 + Area4*ys4
Ixc5     = Ix5 + Area5*ys5
Ixc6     = Ix6 + Area6*ys6
Ixc7     = Ix7 + Area7*ys7
Ixc8     = Ix8 + Area8*ys8
Ixc9     = Ix9 + Area9*ys9
Ixc10    = Ix10 + Area10*ys10
Ixc11    = Ix11 + Area11*ys11
Ixc12    = Ix12 + Area12*ys12
Ixc13    = Ix13 + Area13*ys13
Ixc14    = Ix14 + Area14*ys14
Ixc15    = Ix15 + Area15*ys15

 ############################################################################
# Section Modulus - may need - Parallel Axis Theorem
SM0     = Ixc0/ys0
SM1     = Ixc1/ys1
SM2     = Ixc2/ys2
SM3     = Ixc3/ys3
SM4     = Ixc4/ys4
SM5     = Ixc5/ys5
```

```
            SM6     = Ixc6/ys6
            SM7     = Ixc7/ys7
            SM8     = Ixc8/ys8
            SM9     = Ixc9/ys9
            SM10    = Ixc10/ys10
            SM11    = Ixc11/ys11
            SM12    = Ixc12/ys12
            SM13    = Ixc13/ys13
            SM14    = Ixc14/ys14
            SM15    = Ixc15/ys15

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

            sigmamin0  = Mx0/(SM0*144.)
            sigmamin1  = Mx1/(SM1*144.)
            sigmamin2  = Mx2/(SM2*144.)
            sigmamin3  = Mx3/(SM3*144.)
            sigmamin4  = Mx4/(SM4*144.)
            sigmamin5  = Mx5/(SM5*144.)
            sigmamin6  = Mx6/(SM6*144.)
            sigmamin7  = Mx7/(SM7*144.)
            sigmamin8  = Mx8/(SM8*144.)
            sigmamin9  = Mx9/(SM9*144.)
            sigmamin10 = Mx10/(SM10*144.)
            sigmamin11 = Mx11/(SM11*144.)
            sigmamin12 = Mx12/(SM12*144.)
            sigmamin13 = Mx13/(SM13*144.)
            sigmamin14 = Mx14/(SM14*144.)
            sigmamin15 = Mx15/(SM15*144.)

            sigma = ([sigmamin0,sigmamin1,sigmamin2,sigmamin3,sigmamin4,\
                      sigmamin5,sigmamin6,sigmamin7,sigmamin8,sigmamin9,sigmamin10\
                      ,sigmamin11,sigmamin12,sigmamin13,sigmamin14,sigmamin15])

            print sigma
            print self.sigmamax

            if self.sigmamax < any(sigma):
                print "fail"
            elif self.sigmamax > any(sigma):
                print "Pass"

        elif self.lselect2 ==1:
            x0  = ([Chrd[0]*0.00,Chrd[0]*0.005,Chrd[0]*0.0075,Chrd[0]*0.0125,\
                    Chrd[0]*0.025,Chrd[0]*0.050,Chrd[0]*0.0750,Chrd[0]*0.100,Chrd[0]\
```

```
        *0.150,Chrd[0]*0.20,Chrd[0]*0.25,\
       Chrd[0]*0.30,Chrd[0]*0.35,Chrd[0]*0.40,Chrd[0]*0.45,Chrd[0]*0.50,\
       Chrd[0]*0.55,Chrd[0]*0.60,Chrd[0]*0.65,Chrd[0]*0.70,Chrd[0]*0.75,\
       Chrd[0]*0.80,Chrd[0]*0.85,Chrd[0]*0.90,Chrd[0]*0.95,Chrd[0]*1.00])
x1  = ([Chrd[1]*0.00,Chrd[1]*0.005,Chrd[1]*0.0075,Chrd[1]*0.0125,Chrd[1]*\
        0.025,Chrd[1]*0.050,Chrd[1]*0.0750,Chrd[1]*0.100,Chrd[1]*0.150,\
        Chrd[1]*0.20,Chrd[1]*0.25,\
       Chrd[1]*0.30,Chrd[1]*0.35,Chrd[1]*0.40,Chrd[1]*0.45,Chrd[1]*0.50,\
       Chrd[1]*0.55,Chrd[1]*0.60,Chrd[1]*0.65,Chrd[1]*0.70,Chrd[1]*0.75,\
       Chrd[1]*0.80,Chrd[1]*0.85,Chrd[1]*0.90,Chrd[1]*0.95,Chrd[1]*1.00])
x2  = ([Chrd[2]*0.00,Chrd[2]*0.005,Chrd[2]*0.0075,Chrd[2]*0.0125,Chrd[2]*\
        0.025,Chrd[2]*0.050,Chrd[2]*0.0750,Chrd[2]*0.100,Chrd[2]*0.150,\
        Chrd[2]*0.20,Chrd[2]*0.25,\
       Chrd[2]*0.30,Chrd[2]*0.35,Chrd[2]*0.40,Chrd[2]*0.45,Chrd[2]*0.50,\
       Chrd[2]*0.55,Chrd[2]*0.60,Chrd[2]*0.65,Chrd[2]*0.70,Chrd[2]*0.75,\
       Chrd[2]*0.80,Chrd[2]*0.85,Chrd[2]*0.90,Chrd[2]*0.95,Chrd[2]*1.00])
x3  = ([Chrd[3]*0.00,Chrd[3]*0.005,Chrd[3]*0.0075,Chrd[3]*0.0125,Chrd[3]*\
        0.025,Chrd[3]*0.050,Chrd[3]*0.0750,Chrd[3]*0.100,Chrd[3]*0.150,\
        Chrd[3]*0.20,Chrd[3]*0.25,\
       Chrd[3]*0.30,Chrd[3]*0.35,Chrd[3]*0.40,Chrd[3]*0.45,Chrd[3]*0.50,\
       Chrd[3]*0.55,Chrd[3]*0.60,Chrd[3]*0.65,Chrd[3]*0.70,Chrd[3]*0.75,\
       Chrd[3]*0.80,Chrd[3]*0.85,Chrd[3]*0.90,Chrd[3]*0.95,Chrd[3]*1.00])
x4  = ([Chrd[4]*0.00,Chrd[4]*0.005,Chrd[4]*0.0075,Chrd[4]*0.0125,Chrd[4]*\
        0.025,Chrd[4]*0.050,Chrd[4]*0.0750,Chrd[4]*0.100,Chrd[4]*0.150,\
        Chrd[4]*0.20,Chrd[4]*0.25,\
       Chrd[4]*0.30,Chrd[4]*0.35,Chrd[4]*0.40,Chrd[4]*0.45,Chrd[4]*0.50,\
       Chrd[4]*0.55,Chrd[4]*0.60,Chrd[4]*0.65,Chrd[4]*0.70,Chrd[4]*0.75,\
       Chrd[4]*0.80,Chrd[4]*0.85,Chrd[4]*0.90,Chrd[4]*0.95,Chrd[4]*1.00])
x5  = ([Chrd[5]*0.00,Chrd[5]*0.005,Chrd[5]*0.0075,Chrd[5]*0.0125,Chrd[5]*\
        0.025,Chrd[5]*0.050,Chrd[5]*0.0750,Chrd[5]*0.100,Chrd[5]*0.150,\
        Chrd[5]*0.20,Chrd[5]*0.25,\
       Chrd[5]*0.30,Chrd[5]*0.35,Chrd[5]*0.40,Chrd[5]*0.45,Chrd[5]*0.50,\
       Chrd[5]*0.55,Chrd[5]*0.60,Chrd[5]*0.65,Chrd[5]*0.70,Chrd[5]*0.75,\
       Chrd[5]*0.80,Chrd[5]*0.85,Chrd[5]*0.90,Chrd[5]*0.95,Chrd[5]*1.00])
x6  = ([Chrd[6]*0.00,Chrd[6]*0.005,Chrd[6]*0.0075,Chrd[6]*0.0125,Chrd[6]*\
        0.025,Chrd[6]*0.050,Chrd[6]*0.0750,Chrd[6]*0.100,Chrd[6]*0.150,\
        Chrd[6]*0.20,Chrd[6]*0.25,\
       Chrd[6]*0.30,Chrd[6]*0.35,Chrd[6]*0.40,Chrd[6]*0.45,Chrd[6]*0.50,\
       Chrd[6]*0.55,Chrd[6]*0.60,Chrd[6]*0.65,Chrd[6]*0.70,Chrd[6]*0.75,\
       Chrd[6]*0.80,Chrd[6]*0.85,Chrd[6]*0.90,Chrd[6]*0.95,Chrd[6]*1.00])
x7  = ([Chrd[7]*0.00,Chrd[7]*0.005,Chrd[7]*0.0075,Chrd[7]*0.0125,Chrd[7]*\
        0.025,Chrd[7]*0.050,Chrd[7]*0.0750,Chrd[7]*0.100,Chrd[7]*0.150,\
        Chrd[7]*0.20,Chrd[7]*0.25,\
       Chrd[7]*0.30,Chrd[7]*0.35,Chrd[7]*0.40,Chrd[7]*0.45,Chrd[7]*0.50,\
       Chrd[7]*0.55,Chrd[7]*0.60,Chrd[7]*0.65,Chrd[7]*0.70,Chrd[7]*0.75,\
```

```
       Chrd[7]*0.80,Chrd[7]*0.85,Chrd[7]*0.90,Chrd[7]*0.95,Chrd[7]*1.00])
x8  = ([Chrd[8]*0.00,Chrd[8]*0.005,Chrd[8]*0.0075,Chrd[8]*0.0125,Chrd[8]*\
        0.025,Chrd[8]*0.050,Chrd[8]*0.0750,Chrd[8]*0.100,Chrd[8]*0.150,\
        Chrd[8]*0.20,Chrd[8]*0.25,\
       Chrd[8]*0.30,Chrd[8]*0.35,Chrd[8]*0.40,Chrd[8]*0.45,Chrd[8]*0.50,\
       Chrd[8]*0.55,Chrd[8]*0.60,Chrd[8]*0.65,Chrd[8]*0.70,Chrd[8]*0.75,\
       Chrd[8]*0.80,Chrd[8]*0.85,Chrd[8]*0.90,Chrd[8]*0.95,Chrd[8]*1.00])
x9  = ([Chrd[9]*0.00,Chrd[9]*0.005,Chrd[9]*0.0075,Chrd[9]*0.0125,Chrd[9]*\
        0.025,Chrd[9]*0.050,Chrd[9]*0.0750,Chrd[9]*0.100,Chrd[9]*0.150,\
        Chrd[9]*0.20,Chrd[9]*0.25,\
       Chrd[9]*0.30,Chrd[9]*0.35,Chrd[9]*0.40,Chrd[9]*0.45,Chrd[9]*0.50,\
       Chrd[9]*0.55,Chrd[9]*0.60,Chrd[9]*0.65,Chrd[9]*0.70,Chrd[9]*0.75,\
       Chrd[9]*0.80,Chrd[9]*0.85,Chrd[9]*0.90,Chrd[9]*0.95,Chrd[9]*1.00])
x10 = ([Chrd[10]*0.00,Chrd[10]*0.005,Chrd[10]*0.0075,Chrd[10]*0.0125,\
        Chrd[10]*0.025,Chrd[10]*0.050,Chrd[10]*0.0750,Chrd[10]*0.100,\
        Chrd[10]*0.150,Chrd[10]*0.20,Chrd[10]*0.25,\
       Chrd[10]*0.30,Chrd[10]*0.35,Chrd[10]*0.40,Chrd[10]*0.45,Chrd[10]\
       *0.50,Chrd[10]*0.55,Chrd[10]*0.60,Chrd[10]*0.65,Chrd[10]*0.70,\
       Chrd[10]*0.75,\
       Chrd[10]*0.80,Chrd[10]*0.85,Chrd[10]*0.90,Chrd[10]*0.95,Chrd[10]*1.00])
x11 = ([Chrd[11]*0.00,Chrd[11]*0.005,Chrd[11]*0.0075,Chrd[11]*0.0125,Chrd[11]\
        *0.025,Chrd[11]*0.050,Chrd[11]*0.0750,Chrd[11]*0.100,Chrd[11]*0.150,\
        Chrd[11]*0.20,Chrd[11]*0.25,\
       Chrd[11]*0.30,Chrd[11]*0.35,Chrd[11]*0.40,Chrd[11]*0.45,Chrd[11]*0.50,\
       Chrd[11]*0.55,Chrd[11]*0.60,Chrd[11]*0.65,Chrd[11]*0.70,Chrd[11]*0.75,\
       Chrd[11]*0.80,Chrd[11]*0.85,Chrd[11]*0.90,Chrd[11]*0.95,Chrd[11]*1.00])
x12 = ([Chrd[12]*0.00,Chrd[12]*0.005,Chrd[12]*0.0075,Chrd[12]*0.0125,Chrd[12]\
        *0.025,Chrd[12]*0.050,Chrd[12]*0.0750,Chrd[12]*0.100,Chrd[12]*0.150,\
        Chrd[12]*0.20,Chrd[12]*0.25,\
       Chrd[12]*0.30,Chrd[12]*0.35,Chrd[12]*0.40,Chrd[12]*0.45,Chrd[12]*0.50,\
       Chrd[12]*0.55,Chrd[12]*0.60,Chrd[12]*0.65,Chrd[12]*0.70,Chrd[12]*0.75,\
       Chrd[12]*0.80,Chrd[12]*0.85,Chrd[12]*0.90,Chrd[12]*0.95,Chrd[12]*1.00])
x13 = ([Chrd[13]*0.00,Chrd[13]*0.005,Chrd[13]*0.0075,Chrd[13]*0.0125,Chrd[13]*\
        0.025,Chrd[13]*0.050,Chrd[13]*0.0750,Chrd[13]*0.100,Chrd[13]*0.150,\
        Chrd[13]*0.20,Chrd[13]*0.25,\
       Chrd[13]*0.30,Chrd[13]*0.35,Chrd[13]*0.40,Chrd[13]*0.45,Chrd[13]*0.50,\
       Chrd[13]*0.55,Chrd[13]*0.60,Chrd[13]*0.65,Chrd[13]*0.70,Chrd[13]*0.75,\
       Chrd[13]*0.80,Chrd[13]*0.85,Chrd[13]*0.90,Chrd[13]*0.95,Chrd[13]*1.00])
x14 = ([Chrd[14]*0.00,Chrd[14]*0.005,Chrd[14]*0.0075,Chrd[14]*0.0125,Chrd[14]*\
        0.025,Chrd[14]*0.050,Chrd[14]*0.0750,Chrd[14]*0.100,Chrd[14]*0.150,\
        Chrd[14]*0.20,Chrd[14]*0.25,\
       Chrd[14]*0.30,Chrd[14]*0.35,Chrd[14]*0.40,Chrd[14]*0.45,Chrd[14]*0.50,\
       Chrd[14]*0.55,Chrd[14]*0.60,Chrd[14]*0.65,Chrd[14]*0.70,Chrd[14]*0.75,\
       Chrd[14]*0.80,Chrd[14]*0.85,Chrd[14]*0.90,Chrd[14]*0.95,Chrd[14]*1.00])
x15 = ([Chrd[15]*0.00,Chrd[15]*0.005,Chrd[15]*0.0075,Chrd[15]*0.0125,Chrd[15]*\
```

```
         0.025,Chrd[15]*0.050,Chrd[15]*0.0750,Chrd[15]*0.100,Chrd[15]*0.150,\
         Chrd[15]*0.20,Chrd[15]*0.25,\
        Chrd[15]*0.30,Chrd[15]*0.35,Chrd[15]*0.40,Chrd[15]*0.45,Chrd[15]*0.50,\
        Chrd[15]*0.55,Chrd[15]*0.60,Chrd[15]*0.65,Chrd[15]*0.70,Chrd[15]*0.75,\
        Chrd[15]*0.80,Chrd[15]*0.85,Chrd[15]*0.90,Chrd[15]*0.95,Chrd[15]*1.00])


while ii <= 25:
    Upper0.append(self.NU[ii]*Chrd[0]/12.)
    Upper1.append(self.NU[ii]*Chrd[1]/12.)
    Upper2.append(self.NU[ii]*Chrd[2]/12.)
    Upper3.append(self.NU[ii]*Chrd[3]/12.)
    Upper4.append(self.NU[ii]*Chrd[4]/12.)
    Upper5.append(self.NU[ii]*Chrd[5]/12.)
    Upper6.append(self.NU[ii]*Chrd[6]/12.)
    Upper7.append(self.NU[ii]*Chrd[7]/12.)
    Upper8.append(self.NU[ii]*Chrd[8]/12.)
    Upper9.append(self.NU[ii]*Chrd[9]/12.)
    Upper10.append(self.NU[ii]*Chrd[10]/12.)
    Upper11.append(self.NU[ii]*Chrd[11]/12.)
    Upper12.append(self.NU[ii]*Chrd[12]/12.)
    Upper13.append(self.NU[ii]*Chrd[13]/12.)
    Upper14.append(self.NU[ii]*Chrd[14]/12.)
    Upper15.append(self.NU[ii]*Chrd[15]/12.)

    Lower0.append(self.NL[ii]*Chrd[0]/12.)
    Lower1.append(self.NL[ii]*Chrd[1]/12.)
    Lower2.append(self.NL[ii]*Chrd[2]/12.)
    Lower3.append(self.NL[ii]*Chrd[3]/12.)
    Lower4.append(self.NL[ii]*Chrd[4]/12.)
    Lower5.append(self.NL[ii]*Chrd[5]/12.)
    Lower6.append(self.NL[ii]*Chrd[6]/12.)
    Lower7.append(self.NL[ii]*Chrd[7]/12.)
    Lower8.append(self.NL[ii]*Chrd[8]/12.)
    Lower9.append(self.NL[ii]*Chrd[9]/12.)
    Lower10.append(self.NL[ii]*Chrd[10]/12.)
    Lower11.append(self.NL[ii]*Chrd[11]/12.)
    Lower12.append(self.NL[ii]*Chrd[12]/12.)
    Lower13.append(self.NL[ii]*Chrd[13]/12.)
    Lower14.append(self.NL[ii]*Chrd[14]/12.)
    Lower15.append(self.NL[ii]*Chrd[15]/12.)
    ii += 1

cp0    = array([[x0[0],x0[1],x0[2],x0[3],x0[4],x0[5],x0[6],x0[7]\
        ,x0[8],x0[9],x0[10],x0[11],x0[12],x0[13],x0[14],x0[15],\
```

```
        x0[16],x0[17],x0[18],x0[19],x0[20],x0[21],x0[22],x0[23],\
        x0[24],x0[25],x0[25],x0[24],x0[23],x0[22],x0[21],x0[20],\
        x0[19],x0[18],x0[17],x0[16],x0[15],x0[14],x0[13],x0[12],\
        x0[11],x0[10],x0[9],x0[8],x0[7],x0[6],x0[5],x0[4],x0[3],\
        x0[2],x0[1],x0[0]],[Upper0[0],Upper0[1],Upper0[2],Upper0[3],\
        Upper0[4],Upper0[5],Upper0[6],Upper0[7],Upper0[8],Upper0[9],\
        Upper0[10],Upper0[11],Upper0[12],Upper0[13],Upper0[14],\
        Upper0[15],Upper0[16],Upper0[17],Upper0[18],Upper0[19],\
        Upper0[20],Upper0[21],Upper0[22],Upper0[23],Upper0[24],\
        Upper0[25],Lower0[25],Lower0[24],Lower0[23],Lower0[22],\
        Lower0[21],Lower0[20],Lower0[19],Lower0[18],Lower0[17],\
        Lower0[16],Lower0[15],Lower0[14],Lower0[13],Lower0[12],\
        Lower0[11],Lower0[10],Lower0[9],Lower0[8],Lower0[7],Lower0[6],\
        Lower0[5],Lower0[4],Lower0[3],Lower0[2],Lower0[1],Lower0[0]]])
cp1    = array([[x1[0],x1[1],x1[2],x1[3],x1[4],x1[5],x1[6],x1[7],\
    x1[8],x1[9],x1[10],x1[11],x1[12],x1[13],x1[14],x1[15],x1[16],\
    x1[17],x1[18],x1[19],x1[20],x1[21],x1[22],x1[23],x1[24],x1[25],\
    x1[25],x1[24],x1[23],x1[22],x1[21],x1[20],x1[19],x1[18],x1[17],\
    x1[16],x1[15],x1[14],x1[13],x1[12],x1[11],x1[10],x1[9],x1[8],\
    x1[7],x1[6],x1[5],x1[4],x1[3],x1[2],x1[1],x1[0]],[Upper1[0],\
    Upper1[1],Upper1[2],Upper1[3],Upper1[4],Upper1[5],Upper1[6],\
    Upper1[7],Upper1[8],Upper1[9],Upper1[10],Upper1[11],Upper1[12],\
    Upper1[13],Upper1[14],Upper1[15],Upper1[16],Upper1[17],Upper1[18],\
    Upper1[19],Upper1[20],Upper1[21],Upper1[22],Upper1[23],Upper1[24],\
    Upper1[25],Lower1[25],Lower1[24],Lower1[23],Lower1[22],Lower1[21],\
    Lower1[20],Lower1[19],Lower1[18],Lower1[17],Lower1[16],Lower1[15],\
    Lower1[14],Lower1[13],Lower1[12],Lower1[11],Lower1[10],Lower1[9],\
    Lower1[8],Lower1[7],Lower1[6],Lower1[5],Lower1[4],Lower1[3],Lower1[2],\
    Lower1[1],Lower1[0]]])
cp2    = array([[x2[0],x2[1],x2[2],x2[3],x2[4],x2[5],x2[6],x2[7],x2[8],\
    x2[9],x2[10],x2[11],x2[12],x2[13],x2[14],x2[15],x2[16],x2[17],x2[18],\
    x2[19],x2[20],x2[21],x2[22],x2[23],x2[24],x2[25],x2[25],x2[24],x2[23],\
    x2[22],x2[21],x2[20],x2[19],x2[18],x2[17],x2[16],x2[15],x2[14],x2[13],\
    x2[12],x2[11],x2[10],x2[9],x2[8],x2[7],x2[6],x2[5],x2[4],x2[3],x2[2],\
    x2[1],x2[0]],[Upper2[0],Upper2[1],Upper2[2],Upper2[3],Upper2[4],Upper2[5],\
    Upper2[6],Upper2[7],Upper2[8],Upper2[9],Upper2[10],Upper2[11],Upper2[12],\
    Upper2[13],Upper2[14],Upper2[15],Upper2[16],Upper2[17],Upper2[18],Upper2[19],\
    Upper2[20],Upper2[21],Upper2[22],Upper2[23],Upper2[24],Upper2[25],Lower2[25],\
    Lower2[24],Lower2[23],Lower2[22],Lower2[21],Lower2[20],Lower2[19],Lower2[18],\
    Lower2[17],Lower2[16],Lower2[15],Lower2[14],Lower2[13],Lower2[12],Lower2[11],\
    Lower2[10],Lower2[9],Lower2[8],Lower2[7],Lower2[6],Lower2[5],Lower2[4],Lower2[3],\
    Lower2[2],Lower2[1],Lower2[0]]])
cp3    = array([[x3[0],x3[1],x3[2],x3[3],x3[4],x3[5],x3[6],x3[7],x3[8],\
    x3[9],x3[10],x3[11],x3[12],x3[13],x3[14],x3[15],x3[16],x3[17],x3[18],\
    x3[19],x3[20],x3[21],x3[22],x3[23],x3[24],x3[25],x3[25],x3[24],x3[23],\
```

```
    x3[22],x3[21],x3[20],x3[19],x3[18],x3[17],x3[16],x3[15],x3[14],x3[13],\
    x3[12],x3[11],x3[10],x3[9],x3[8],x3[7],x3[6],x3[5],x3[4],x3[3],x3[2],\
    x3[1],x3[0]],[Upper3[0],Upper3[1],Upper3[2],Upper3[3],Upper3[4],Upper3[5],\
    Upper3[6],Upper3[7],Upper3[8],Upper3[9],Upper3[10],Upper3[11],Upper3[12],\
    Upper3[13],Upper3[14],Upper3[15],Upper3[16],Upper3[17],Upper3[18],Upper3[19],\
    Upper3[20],Upper3[21],Upper3[22],Upper3[23],Upper3[24],Upper3[25],Lower3[25],\
    Lower3[24],Lower3[23],Lower3[22],Lower3[21],Lower3[20],Lower3[19],Lower3[18],\
    Lower3[17],Lower3[16],Lower3[15],Lower3[14],Lower3[13],Lower3[12],Lower3[11],\
    Lower3[10],Lower3[9],Lower3[8],Lower3[7],Lower3[6],Lower3[5],Lower3[4],Lower3[3],\
    Lower3[2],Lower3[1],Lower3[0]]])
cp4    = array([[x4[0],x4[1],x4[2],x4[3],x4[4],x4[5],x4[6],x4[7],\
    x4[8],x4[9],x4[10],x4[11],x4[12],x4[13],x4[14],x4[15],x4[16],\
    x4[17],x4[18],x4[19],x4[20],x4[21],x4[22],x4[23],x4[24],x4[25],\
    x4[25],x4[24],x4[23],x4[22],x4[21],x4[20],x4[19],x4[18],x4[17],\
    x4[16],x4[15],x4[14],x4[13],x4[12],x4[11],x4[10],x4[9],x4[8],\
    x4[7],x4[6],x4[5],x4[4],x4[3],x4[2],x4[1],x4[0]],[Upper4[0],\
    Upper4[1],Upper4[2],Upper4[3],Upper4[4],Upper4[5],Upper4[6],\
    Upper4[7],Upper4[8],Upper4[9],Upper4[10],Upper4[11],Upper4[12],\
    Upper4[13],Upper4[14],Upper4[15],Upper4[16],Upper4[17],Upper4[18],\
    Upper4[19],Upper4[20],Upper4[21],Upper4[22],Upper4[23],Upper4[24],\
    Upper4[25],Lower4[25],Lower4[24],Lower4[23],Lower4[22],Lower4[21],\
    Lower4[20],Lower4[19],Lower4[18],Lower4[17],Lower4[16],Lower4[15],\
    Lower4[14],Lower4[13],Lower4[12],Lower4[11],Lower4[10],Lower4[9],\
    Lower4[8],Lower4[7],Lower4[6],Lower4[5],Lower4[4],Lower4[3],Lower4[2],\
    Lower4[1],Lower4[0]]])
cp5    = array([[x5[0],x5[1],x5[2],x5[3],x5[4],x5[5],x5[6],x5[7],\
    x5[8],x5[9],x5[10],x5[11],x5[12],x5[13],x5[14],x5[15],x5[16],\
    x5[17],x5[18],x5[19],x5[20],x5[21],x5[22],x5[23],x5[24],x5[25],\
    x5[25],x5[24],x5[23],x5[22],x5[21],x5[20],x5[19],x5[18],x5[17],\
    x5[16],x5[15],x5[14],x5[13],x5[12],x5[11],x5[10],x5[9],x5[8],\
    x5[7],x5[6],x5[5],x5[4],x5[3],x5[2],x5[1],x5[0]],[Upper5[0],\
    Upper5[1],Upper5[2],Upper5[3],Upper5[4],Upper5[5],Upper5[6],\
    Upper5[7],Upper5[8],Upper5[9],Upper5[10],Upper5[11],Upper5[12],\
    Upper5[13],Upper5[14],Upper5[15],Upper5[16],Upper5[17],Upper5[18],\
    Upper5[19],Upper5[20],Upper5[21],Upper5[22],Upper5[23],Upper5[24],\
    Upper5[25],Lower5[25],Lower5[24],Lower5[23],Lower5[22],Lower5[21],\
    Lower5[20],Lower5[19],Lower5[18],Lower5[17],Lower5[16],Lower5[15],\
    Lower5[14],Lower5[13],Lower5[12],Lower5[11],Lower5[10],Lower5[9],\
    Lower5[8],Lower5[7],Lower5[6],Lower5[5],Lower5[4],Lower5[3],Lower5[2],\
    Lower5[1],Lower5[0]]])
cp6    = array([[x6[0],x6[1],x6[2],x6[3],x6[4],x6[5],x6[6],x6[7],\
    x6[8],x6[9],x6[10],x6[11],x6[12],x6[13],x6[14],x6[15],x6[16],\
    x6[17],x6[18],x6[19],x6[20],x6[21],x6[22],x6[23],x6[24],x6[25],\
    x6[25],x6[24],x6[23],x6[22],x6[21],x6[20],x6[19],x6[18],x6[17],\
    x6[16],x6[15],x6[14],x6[13],x6[12],x6[11],x6[10],x6[9],x6[8],\
```

```
    x6[7],x6[6],x6[5],x6[4],x6[3],x6[2],x6[1],x6[0]],[Upper6[0],\
    Upper6[1],Upper6[2],Upper6[3],Upper6[4],Upper6[5],Upper6[7],\
    Upper6[8],Upper6[9],Upper6[10],Upper6[11],Upper6[12],Upper6[13],\
    Upper6[14],Upper6[15],Upper6[16],Upper6[17],Upper6[18],Upper6[19],\
    Upper6[20],Upper6[21],Upper6[22],Upper6[23],Upper6[24],Upper6[25],\
    Lower6[25],Lower6[24],Lower6[23],Lower6[22],Lower6[21],Lower6[20],\
    Lower6[19],Lower6[18],Lower6[17],Lower6[16],Lower6[15],Lower6[14],\
    Lower6[13],Lower6[12],Lower6[11],Lower6[10],Lower6[9],Lower6[8],\
    Lower6[7],Lower6[6],Lower6[5],Lower6[4],Lower6[3],Lower6[2],Lower6[1],\
    Lower6[0]]])
cp7    = array([[x7[0],x7[1],x7[2],x7[3],x7[4],x7[5],x7[6],x7[7],\
    x7[8],x7[9],x7[10],x7[11],x7[12],x7[13],x7[14],x7[15],x7[16],\
    x7[17],x7[18],x7[19],x7[20],x7[21],x7[22],x7[23],x7[24],x7[25],\
    x7[25],x7[24],x7[23],x7[22],x7[21],x7[20],x7[19],x7[18],x7[17],\
    x7[16],x7[15],x7[14],x7[13],x7[12],x7[11],x7[10],x7[9],x7[8],\
    x7[7],x7[6],x7[5],x7[4],x7[3],x7[2],x7[1],x7[0]],[Upper7[0],\
    Upper7[1],Upper7[2],Upper7[3],Upper7[4],Upper7[5],Upper7[6],\
    Upper7[7],Upper7[8],Upper7[9],Upper7[10],Upper7[11],Upper7[12],\
    Upper7[13],Upper7[14],Upper7[15],Upper7[16],Upper7[17],Upper7[18],\
    Upper7[19],Upper7[20],Upper7[21],Upper7[22],Upper7[23],Upper7[24],\
    Upper7[25],Lower7[25],Lower7[24],Lower7[23],Lower7[22],Lower7[21],\
    Lower7[20],Lower7[19],Lower7[18],Lower7[17],Lower7[16],Lower7[15],\
    Lower7[14],Lower7[13],Lower7[12],Lower7[11],Lower7[10],Lower7[9],\
    Lower7[8],Lower7[7],Lower7[6],Lower7[5],Lower7[4],Lower7[3],Lower7[2],\
    Lower7[1],Lower7[0]]])
cp8    = array([[x8[0],x8[1],x8[2],x8[3],x8[4],x8[5],x8[6],x8[7],x8[8],\
    x8[9],x8[10],x8[11],x8[12],x8[13],x8[14],x8[15],x8[16],x8[17],\
    x8[18],x8[19],x8[20],x8[21],x8[22],x8[23],x8[24],x8[25],x8[25],
    x8[24],x8[23],x8[22],x8[21],x8[20],x8[19],x8[18],x8[17],x8[16],\
    x8[15],x8[14],x8[13],x8[12],x8[11],x8[10],x8[9],x8[8],x8[7],x8[6],\
    x8[5],x8[4],x8[3],x8[2],x8[1],x8[0]],[Upper8[0],Upper8[1],Upper8[2],\
    Upper8[3],Upper8[4],Upper8[5],Upper8[6],Upper8[7],Upper8[8],Upper8[9],\
    Upper8[10],Upper8[11],Upper8[12],Upper8[13],Upper8[14],Upper8[15],\
    Upper8[16],Upper8[17],Upper8[18],Upper8[19],Upper8[20],Upper8[21],\
    Upper8[22],Upper8[23],Upper8[24],Upper8[25],Lower8[25],Lower8[24],\
    Lower8[23],Lower8[22],Lower8[21],Lower8[20],Lower8[19],Lower8[18],\
    Lower8[17],Lower8[16],Lower8[15],Lower8[14],Lower8[13],Lower8[12],\
    Lower8[11],Lower8[10],Lower8[9],Lower8[8],Lower8[7],Lower8[6],\
    Lower8[5],Lower8[4],Lower8[3],Lower8[2],Lower8[1],Lower8[0]]])
cp9    = array([[x9[0],x9[1],x9[2],x9[3],x9[4],x9[5],x9[6],x9[7],\
    x9[8],x9[9],x9[10],x9[11],x9[12],x9[13],x9[14],x9[15],x9[16],\
    x9[17],x9[18],x9[19],x9[20],x9[21],x9[22],x9[23],x9[24],x9[25],\
    x9[25],x9[24],x9[23],x9[22],x9[21],x9[20],x9[19],x9[18],x9[17],\
    x9[16],x9[15],x9[14],x9[13],x9[12],x9[11],x9[10],x9[9],x9[8],\
    x9[7],x9[6],x9[5],x9[4],x9[3],x9[2],x9[1],x9[0]],[Upper9[0],\
```

```
    Upper9[1],Upper9[2],Upper9[3],Upper9[4],Upper9[5],Upper9[6],\
    Upper9[7],Upper9[8],Upper9[9],Upper9[10],Upper9[11],Upper9[12],\
    Upper9[13],Upper9[14],Upper9[15],Upper9[16],Upper9[17],Upper9[18],\
    Upper9[19],Upper9[20],Upper9[21],Upper9[22],Upper9[23],Upper9[24],\
    Upper9[25],Lower9[25],Lower9[24],Lower9[23],Lower9[22],Lower9[21],\
    Lower9[20],Lower9[19],Lower9[18],Lower9[17],Lower9[16],Lower9[15],\
    Lower9[14],Lower9[13],Lower9[12],Lower9[11],Lower9[10],Lower9[9],\
    Lower9[8],Lower9[7],Lower9[6],Lower9[5],Lower9[4],Lower9[3],Lower9[2],\
    Lower9[1],Lower9[0]]])
cp10   = array([[x10[0],x10[1],x10[2],x10[3],x10[4],x10[5],x10[6],\
    x10[7],x10[8],x10[9],x10[10],x10[11],x10[12],x10[13],x10[14],\
    x10[15],x10[16],x10[17],x10[18],x10[19],x10[20],x10[21],x10[22],\
    x10[23],x10[24],x10[25],x10[25],x10[24],x10[23],x10[22],x10[21],\
    x10[20],x10[19],x10[18],x10[17],x10[16],x10[15],x10[14],x10[13],\
    x10[12],x10[11],x10[10],x10[9],x10[8],x10[7],x10[6],x10[5],x10[4],\
    x10[3],x10[2],x10[1],x10[0]],[Upper10[0],Upper10[1],Upper10[2],\
    Upper10[3],Upper10[4],Upper10[5],Upper10[6],Upper10[7],Upper10[8],\
    Upper10[9],Upper10[10],Upper10[11],Upper10[12],Upper10[13],Upper10[14],\
    Upper10[15],Upper10[16],Upper10[17],Upper10[18],Upper10[19],Upper10[20],\
    Upper10[21],Upper10[22],Upper10[23],Upper10[24],Upper10[25],Lower10[25],\
    Lower10[24],Lower10[23],Lower10[22],Lower10[21],Lower10[20],Lower10[19],\
    Lower10[18],Lower10[17],Lower10[16],Lower10[15],Lower10[14],Lower10[13],\
    Lower10[12],Lower10[11],Lower10[10],Lower10[9],Lower10[8],Lower10[7],\
    Lower10[6],Lower10[5],Lower10[4],Lower10[3],Lower10[2],Lower10[1],\
    Lower10[0]]])
cp11   = array([[x11[0],x11[1],x11[2],x11[3],x11[4],x11[5],x11[6],\
    x11[7],x11[8],x11[9],x11[10],x11[11],x11[12],x11[13],x11[14],\
    x11[15],x11[16],x11[17],x11[18],x11[19],x11[20],x11[21],x11[22],\
    x11[23],x11[24],x11[25],x11[25],x11[24],x11[23],x11[22],x11[21],\
    x11[20],x11[19],x11[18],x11[17],x11[16],x11[15],x11[14],x11[13],\
    x11[12],x11[11],x11[10],x11[9],x11[8],x11[7],x11[6],x11[5],x11[4],\
    x11[3],x11[2],x11[1],x11[0]],[Upper11[0],Upper11[1],Upper11[2],\
    Upper11[3],Upper11[4],Upper11[5],Upper11[6],Upper11[7],Upper11[8],\
    Upper11[9],Upper11[10],Upper11[11],Upper11[12],Upper11[13],\
    Upper11[14],Upper11[15],Upper11[16],Upper11[17],Upper11[18],\
    Upper11[19],Upper11[20],Upper11[21],Upper11[22],Upper11[23],\
    Upper11[24],Upper11[25],Lower11[25],Lower11[24],Lower11[23],\
    Lower11[22],Lower11[21],Lower11[20],Lower11[19],Lower11[18],\
    Lower11[17],Lower11[16],Lower11[15],Lower11[14],Lower11[13],\
    Lower11[12],Lower11[11],Lower11[10],Lower11[9],Lower11[8],Lower11[7],\
    Lower11[6],Lower11[5],Lower11[4],Lower11[3],Lower11[2],Lower11[1],\
    Lower11[0]]])
cp12   = array([[x12[0],x12[1],x12[2],x12[3],x12[4],x12[5],x12[6],\
    x12[7],x12[8],x12[9],x12[10],x12[11],x12[12],x12[13],x12[14],\
    x12[15],x12[16],x12[17],x12[18],x12[19],x12[20],x12[21],x12[22],\
```

```
    x12[23],x12[24],x12[25],x12[25],x12[24],x12[23],x12[22],x12[21],\
    x12[20],x12[19],x12[18],x12[17],x12[16],x12[15],x12[14],x12[13],\
    x12[12],x12[11],x12[10],x12[9],x12[8],x12[7],x12[6],x12[5],x12[4],\
    x12[3],x12[2],x12[1],x12[0]],[Upper12[0],Upper12[1],Upper12[2],\
    Upper12[3],Upper12[4],Upper12[5],Upper12[6],Upper12[7],Upper12[8],\
    Upper12[9],Upper12[10],Upper12[11],Upper12[12],Upper12[13],Upper12[14],\
    Upper12[15],Upper12[16],Upper12[17],Upper12[18],Upper12[19],Upper12[20],\
    Upper12[21],Upper12[22],Upper12[23],Upper12[24],Upper12[25],Lower12[25],\
    Lower12[24],Lower12[23],Lower12[22],Lower12[21],Lower12[20],Lower12[19],\
    Lower12[18],Lower12[17],Lower12[16],Lower12[15],Lower12[14],Lower12[13],\
    Lower12[12],Lower12[11],Lower12[10],Lower12[9],Lower12[8],Lower12[7],\
    Lower12[6],Lower12[5],Lower12[4],Lower12[3],Lower12[2],Lower12[1],\
    Lower12[0]]])
cp13   = array([[x13[0],x13[1],x13[2],x13[3],x13[4],x13[5],x13[6],x13[7],\
    x13[8],x13[9],x13[10],x13[11],x13[12],x13[13],x13[14],x13[15],x13[16],\
    x13[17],x13[18],x13[19],x13[20],x13[21],x13[22],x13[23],x13[24],x13[25],\
    x13[25],x13[24],x13[23],x13[22],x13[21],x13[20],x13[19],x13[18],x13[17],\
    x13[16],x13[15],x13[14],x13[13],x13[12],x13[11],x13[10],x13[9],x13[8],\
    x13[7],x13[6],x13[5],x13[4],x13[3],x13[2],x13[1],x13[0]],[Upper13[0],\
    Upper13[1],Upper13[2],Upper13[3],Upper13[4],Upper13[5],Upper13[6],\
    Upper13[7],Upper13[8],Upper13[9],Upper13[10],Upper13[11],Upper13[12],\
    Upper13[13],Upper13[14],Upper13[15],Upper13[16],Upper13[17],Upper13[18],\
    Upper13[19],Upper13[20],Upper13[21],Upper13[22],Upper13[23],Upper13[24],\
    Upper13[25],Lower13[25],Lower13[24],Lower13[23],Lower13[22],Lower13[21],\
    Lower13[20],Lower13[19],Lower13[18],Lower13[17],Lower13[16],Lower13[15],\
    Lower13[14],Lower13[13],Lower13[12],Lower13[11],Lower13[10],Lower13[9],\
    Lower13[8],Lower13[7],Lower13[6],Lower13[5],Lower13[4],Lower13[3],\
    Lower13[2],Lower13[1],Lower13[0]]])
cp14   = array([[x14[0],x14[1],x14[2],x14[3],x14[4],x14[5],x14[6],\
    x14[7],x14[8],x14[9],x14[10],x14[11],x14[12],x14[13],x14[14],\
    x14[15],x14[16],x14[17],x14[18],x14[19],x14[20],x14[21],x14[22],\
    x14[23],x14[24],x14[25],x14[25],x14[24],x14[23],x14[22],x14[21],\
    x14[20],x14[19],x14[18],x14[17],x14[16],x14[15],x14[14],x14[13],\
    x14[12],x14[11],x14[10],x14[9],x14[8],x14[7],x14[6],x14[5],x14[4],\
    x14[3],x14[2],x14[1],x14[0]],[Upper14[0],Upper14[1],Upper14[2],\
    Upper14[3],Upper14[4],Upper14[5],Upper14[6],Upper14[7],Upper14[8],\
    Upper14[9],Upper14[10],Upper14[11],Upper14[12],Upper14[13],Upper14[14],\
    Upper14[15],Upper14[16],Upper14[17],Upper14[18],Upper14[19],Upper14[20],\
    Upper14[21],Upper14[22],Upper14[23],Upper14[24],Upper14[25],Lower14[25],\
    Lower14[24],Lower14[23],Lower14[22],Lower14[21],Lower14[20],Lower14[19],\
    Lower14[18],Lower14[17],Lower14[16],Lower14[15],Lower14[14],Lower14[13],\
    Lower14[12],Lower14[11],Lower14[10],Lower14[9],Lower14[8],Lower14[7],\
    Lower14[6],Lower14[5],Lower14[4],Lower14[3],Lower14[2],Lower14[1],\
    Lower14[0]]])
cp15   = array([[x15[0],x15[1],x15[2],x15[3],x15[4],x15[5],x15[6],\
```

```
    x15[7],x15[8],x15[9],x15[10],x15[11],x15[12],x15[13],x15[14],x15[15],\
    x15[16],x15[17],x15[18],x15[19],x15[20],x15[21],x15[22],x15[23],\
    x15[24],x15[25],x15[25],x15[24],x15[23],x15[22],x15[21],x15[20],\
    x15[19],x15[18],x15[17],x15[16],x15[15],x15[14],x15[13],x15[12],\
    x15[11],x15[10],x15[9],x15[8],x15[7],x15[6],x15[5],x15[4],x15[3],\
    x15[2],x15[1],x15[0]],[Upper15[0],Upper15[1],Upper15[2],Upper15[3],\
    Upper15[4],Upper15[5],Upper15[6],Upper15[7],Upper15[8],Upper15[9],\
    Upper15[10],Upper15[11],Upper15[12],Upper15[13],Upper15[14],\
    Upper15[15],Upper15[16],Upper15[17],Upper15[18],Upper15[19],\
    Upper15[20],Upper15[21],Upper15[22],Upper15[23],Upper15[24],\
    Upper15[25],Lower15[25],Lower15[24],Lower15[23],Lower15[22],\
    Lower15[21],Lower15[20],Lower15[19],Lower15[18],Lower15[17],\
    Lower15[16],Lower15[15],Lower15[14],Lower15[13],Lower15[12],\
    Lower15[11],Lower15[10],Lower15[9],Lower15[8],Lower15[7],Lower15[6],\
    Lower15[5],Lower15[4],Lower15[3],Lower15[2],Lower15[1],Lower15[0]]])

Area0, xs0, ys0, Mxp0, My0, Ix0, Iy0, Ixy0 = plasi(cp0)
Area1, xs1, ys1, Mxp1, My1, Ix1, Iy1, Ixy1 = plasi(cp1)
Area2, xs2, ys2, Mxp2, My2, Ix2, Iy2, Ixy2 = plasi(cp2)
Area3, xs3, ys3, Mxp3, My3, Ix3, Iy3, Ixy3 = plasi(cp3)
Area4, xs4, ys4, Mxp4, My4, Ix4, Iy4, Ixy4 = plasi(cp4)
Area5, xs5, ys5, Mxp5, My5, Ix5, Iy5, Ixy5 = plasi(cp5)
Area6, xs6, ys6, Mxp6, My6, Ix6, Iy6, Ixy6 = plasi(cp6)
Area7, xs7, ys7, Mxp7, My7, Ix7, Iy7, Ixy7 = plasi(cp7)
Area8, xs8, ys8, Mxp8, My8, Ix8, Iy8, Ixy8 = plasi(cp8)
Area9, xs9, ys9, Mxp9, My9, Ix9, Iy9, Ixy9 = plasi(cp9)
Area10, xs10, ys10, Mxp10, My10, Ix10, Iy10, Ixy10 = plasi(cp10)
Area11, xs11, ys11, Mxp11, My11, Ix11, Iy11, Ixy11 = plasi(cp11)
Area12, xs12, ys12, Mxp12, My12, Ix12, Iy12, Ixy12 = plasi(cp12)
Area13, xs13, ys13, Mxp13, My13, Ix13, Iy13, Ixy13 = plasi(cp13)
Area14, xs14, ys14, Mxp14, My14, Ix14, Iy14, Ixy14 = plasi(cp14)
Area15, xs15, ys15, Mxp15, My15, Ix15, Iy15, Ixy15 = plasi(cp15)

# Moment Calculations

Mx0  = (0.7*(D/2)-((0./16.)*(D/2)))*THST
Mx1  = (0.7*(D/2)-((1./16.)*(D/2)))*THST
Mx2  = (0.7*(D/2)-((2./16.)*(D/2)))*THST
Mx3  = (0.7*(D/2)-((3./16.)*(D/2)))*THST
Mx4  = (0.7*(D/2)-((4./16.)*(D/2)))*THST
Mx5  = (0.7*(D/2)-((5./16.)*(D/2)))*THST
Mx6  = (0.7*(D/2)-((6./16.)*(D/2)))*THST
Mx7  = (0.7*(D/2)-((7./16.)*(D/2)))*THST
Mx8  = (0.7*(D/2)-((8./16.)*(D/2)))*THST
Mx9  = (0.7*(D/2)-((9./16.)*(D/2)))*THST
```

```
Mx10 = (0.7*(D/2)-((10./16.)*(D/2)))*THST
Mx11 = (0.7*(D/2)-((11./16.)*(D/2)))*THST
Mx12 = 0*(D/2)*THST
Mx13 = 0.*(D/2)*THST
Mx14 = 0.*(D/2)*THST
Mx15 = 0.*(D/2)*THST

# Moment of Inertia

Ixc0     = Ix0 + Area0*ys0
Ixc1     = Ix1 + Area1*ys1
Ixc2     = Ix2 + Area2*ys2
Ixc3     = Ix3 + Area3*ys3
Ixc4     = Ix4 + Area4*ys4
Ixc5     = Ix5 + Area5*ys5
Ixc6     = Ix6 + Area6*ys6
Ixc7     = Ix7 + Area7*ys7
Ixc8     = Ix8 + Area8*ys8
Ixc9     = Ix9 + Area9*ys9
Ixc10    = Ix10 + Area10*ys10
Ixc11    = Ix11 + Area11*ys11
Ixc12    = Ix12 + Area12*ys12
Ixc13    = Ix13 + Area13*ys13
Ixc14    = Ix14 + Area14*ys14
Ixc15    = Ix15 + Area15*ys15

# Section Modulus - may need - Parallel Axis Theorem
SM0     = Ixc0/ys0
SM1     = Ixc1/ys1
SM2     = Ixc2/ys2
SM3     = Ixc3/ys3
SM4     = Ixc4/ys4
SM5     = Ixc5/ys5
SM6     = Ixc6/ys6
SM7     = Ixc7/ys7
SM8     = Ixc8/ys8
SM9     = Ixc9/ys9
SM10    = Ixc10/ys10
SM11    = Ixc11/ys11
SM12    = Ixc12/ys12
SM13    = Ixc13/ys13
SM14    = Ixc14/ys14
SM15    = Ixc15/ys15

sigmamin0  = Mx0/(SM0*144.)
```

```
            sigmamin1  = Mx1/(SM1*144.)
            sigmamin2  = Mx2/(SM2*144.)
            sigmamin3  = Mx3/(SM3*144.)
            sigmamin4  = Mx4/(SM4*144.)
            sigmamin5  = Mx5/(SM5*144.)
            sigmamin6  = Mx6/(SM6*144.)
            sigmamin7  = Mx7/(SM7*144.)
            sigmamin8  = Mx8/(SM8*144.)
            sigmamin9  = Mx9/(SM9*144.)
            sigmamin10 = Mx10/(SM10*144.)
            sigmamin11 = Mx11/(SM11*144.)
            sigmamin12 = Mx12/(SM12*144.)
            sigmamin13 = Mx13/(SM13*144.)
            sigmamin14 = Mx14/(SM14*144.)
            sigmamin15 = Mx15/(SM15*144.)

            sigma = ([sigmamin0,sigmamin1,sigmamin2,sigmamin3,sigmamin4,\
                      sigmamin5,sigmamin6,sigmamin7,sigmamin8,sigmamin9,\
                      sigmamin10,sigmamin11,sigmamin12,sigmamin13,sigmamin14\
                      ,sigmamin15])

            if self.sigmamax < any(sigma):
                print "fail"
            elif self.sigmamax > any(sigma):
                print "Pass"

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

        #OPTIMIZTION METHOD

            #### Set up Constraints #######
            constrvalue = []

            g1 = DHP - PWR
            constrvalue.append(g1)

            g2 = THST
            constrvalue.append(g2)

            g3 = PWR/DHP - 1.00000
            constrvalue.append(g3)

            g4 = THST/T - 1.00000
            constrvalue.append(g4)
```

```
            g5 = (all(TBC))-0.0200000
            constrvalue.append(g5)

            #g6 = 0.22000 - (all(TBC))
            #constrvalue.append(g6)

            g6 = 1.3000 - PA
            constrvalue.append(g6)


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

            MTPROPOPT1 = -(EF)

            print xfree[0], xfree[1], PWR, THST, T, DHP

            for constr in constrvalue:
                if constr<0:
                    MTPROPOPT1 = MTPROPOPT1 + 100.0 * ((constr**2))

            return MTPROPOPT1

        if __name__ == "__main__":

            x0 = [ 20.0, 0.9]
            xsolution, fopt, niter, ncalls, error \
                = fmin(MTPROPOPT, x0, xtol=1e-8, full_output=1, disp=0)
        xs = abs(fopt)
        #print xs

        self.Uopt    = xsolution[0]
        self.PAopt   = xsolution[1]

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

        f = open("MTpropIN.txt")
        lines = f.readlines()
        f.close()

        # Make data from input file into usable data
        for line in lines:
            strlist = line.split()
            erpm    = float(strlist[0])       # Engine Revolutions per Minute
            D       = float(strlist[1])       # Diameter
```

```
    RHOR    = float(strlist[2])       # Hub Radius Ratio
    rn      = float(strlist[3])       # Radial Sections, at least 16 \
                  should be used for serious calculations (Vorus, 2007)
    Z       = float(strlist[4])       # Number of Blades
    ZH      = float(strlist[5])       # Depth of Hub Centerline
    BHP     = float(strlist[6])       # Brake Horsepower - Used as a \
                                        Constraint
    GR      = float(strlist[7])       # Gear Ratio
    td      = float(strlist[8])       # Thrust deduction
    wf      = float(strlist[9])       # Wave fraction
    EFrot   = float(strlist[10])      # Relative Rotative Efficieny
    EFtrans = float(strlist[11])      # Transmission Efficiency
    screws  = float(strlist[12])      # Propulsors - Single or Double screw

# Density [lbs-sec^2/ft^4 - salt water, fresh = 1.94]
rho = 1.9900
# Kinematic viscosity [ft^2/s]
kv  = 0.000011

prpm = erpm/GR

U1 = xsolution[0]
PA = xsolution[1]

# Computed values from givens:
U           = xsolution[0] * 1.689      # Convert speed from knots to ft/s
R           = D/2.                      # Radius calculated from inputs
Rh          = RHOR*R                    # Radius of Hub
AD          = pi*((R)**2-(Rh)**2)       # Propeller disk area
Omega       = (2*pi*prpm)/60.           # Rate of Revolutions converted rad/s
DR          = (1-RHOR)/ rn              # Sectional separation
TBR         = U/(Omega*R)               # TBR is equal to tan Beta at R
RR          = RHOR - 0.5*DR
RR1         = RR + DR
self.DHP    = BHP*EFtrans*(EFrot**0.5)  # Delivered Horsepower - Constraint

# List of all the the arrays created
i           = 0
j           = 0
k           = 0
COR         = []                # Chord/ R distribution
VAOU        = []                # Wake Velocity distribution estimate
RHOR1       = []
rb          = []                # Rbar = ROR in Vorus code
PrDS        = []                # Scaled Pitch distribution [VaU * PA]
```

```
PIOD        = []                # Hydrodynamic Pitch Distribution
FRAC        = []                # Cavitation Pressure [psia] - cavity gas pressure
CDBr        = []
TBRR        = []
C           = []
C2          = []
TB2         = []
P1          = []
P2          = []
CTII        = []                # This array and equation makes CTII \
                                    nondimensional on VaU
CTI         = []                # Nondimensional on U
UI          = []                # Induced velocity
VI          = []                # Induced tangential velocity
WI          = []
EFFr        = []                # radial efficiency
CPI         = []
TBI         = []
CTTI1       = []                # Variable made for CTTI calculation
CPPI1       = []                # Variable made for CPPI calculation
CC          = []                # Variable made for CTTI calculation
PP          = []                # Variable made for CPPI calculation

## Set-up resistance data

RF = self.EQU1 + (self.EQU2*xsolution[0]) + (self.EQU3*(xsolution[0]**2.))
self.T = ((1/screws)*RF)/((1-td)*(EFrot**0.5))

####### IDEAL CALCULATIONS  #########
# Specified chord radius ratio
#CRR  = [0.4937, 0.5148, 0.5360, 0.5573, 0.5783, 0.5985, 0.6174, 0.6338, \
0.6464, 0.6536, 0.6527, 0.6387, 0.6055, 0.5424, 0.4358, 0.2648, 0.0500]
#VaU  = [0.7500, 0.7625, 0.7750, 0.7875, 0.8000, 0.8125, 0.8250, 0.8375, \
0.8500, 0.8625, 0.8750, 0.8875, 0.9000, 0.9125, 0.9250, 0.9375, 0.9500]
CRR  = self.CRR
VaU  = self.VaU
FRAC = self.FRACV
CDBr = self.CDBR

while k <= rn:
    PrDS.append(VaU[k]*xsolution[1])                          # S stands for SCALED
    k += 1

# Array set up to find midpoint location for COR
CRR1  = [CRR[1],  CRR[2],  CRR[3],  CRR[4],  CRR[5],  CRR[6],  CRR[7],  \
```

```
         CRR[8],  CRR[9],  CRR[10],  CRR[11],  CRR[12],  CRR[13],  CRR[14],  \
         CRR[15],  CRR[16],  0.0000]
# Array set up to find midpoint location for PIOD
PrDS1 = [PrDS[1], PrDS[2], PrDS[3], PrDS[4], PrDS[5], PrDS[6], PrDS[7], \
         PrDS[8], PrDS[9], PrDS[10], PrDS[11], PrDS[12], PrDS[13], PrDS[14], \
         PrDS[15], PrDS[16], 0.0000]
# Array set up to find midpoint location for Hydrodynamic wake
VaU1  = [VaU[1],  VaU[2],  VaU[3],  VaU[4],  VaU[5],  VaU[6],  VaU[7],  \
         VaU[8],  VaU[9],  VaU[10],  VaU[11],  VaU[12],  VaU[13],  VaU[14],  \
         VaU[15],  VaU[16],  0.0000]

while j <= rn:
    COR.append(0.5*(CRR[j]+CRR1[j]))
    PIOD.append(0.5*(PrDS[j]+PrDS1[j]))
    VAOU.append(0.5*(VaU[j]+VaU1[j]))
    RHOR1.append(RHOR + DR*j)
    rb.append(RR1+DR*j)
    TBRR.append(TBR*VAOU[j])
    C.append(PIOD[j]/(pi*TBRR[j]))
    C2.append(C[j]*C[j])
    TB2.append((TBRR[j]/rb[j])**2)
    P1.append(((1.-C2[j]*TB2[j])**2)/(4.*C2[j]))
    P2.append(rb[j]/TBR)
    CTII.append((1.-1./C[j]+(C[j]-1.)*TB2[j])/(TB2[j]+P1[j]))
    CTI.append(CTII[j]*(VAOU[j]**2))
    UI.append(0.5*VAOU[j]*(-1.+(sqrt(1.+CTII[j]))))
    VI.append(0.5*(P2[j]-sqrt((P2[j]**2)-CTI[j])))
    WI.append(VI[j]/P2[j])
    EFFr.append((1. - WI[j])/(1.+UI[j]))
    TBI.append((PIOD[j]/(pi*rb[j])))
    CPI.append((CTI[j]*TBI[j]*rb[j])/TBR)
    CC.append(CTI[j]*rb[j])
    PP.append(CPI[j]*rb[j])
    j+=1

CTTI1 = CC[0]+CC[1]+CC[2]+CC[3]+CC[4]+CC[5]+CC[6]+CC[7]+CC[8]+CC[9]+\
CC[10]+CC[11]+CC[12]+CC[13]+CC[14]+CC[15]
self.CTTI  = 2.*CTTI1*DR/(1.-RHOR**2)
CPPI1 = PP[0]+PP[1]+PP[2]+PP[3]+PP[4]+PP[5]+PP[6]+PP[7]+PP[8]+PP[9]+\
PP[10]+PP[11]+PP[12]+PP[13]+PP[14]+PP[15]
self.CPPI  = 2.*CPPI1*DR/(1.-RHOR**2)
self.EFI = self.CTTI/self.CPPI

self.THSTI = self.CTTI*0.5*rho*(U**2)*AD
self.PWRI  = self.CPPI*0.5*rho*(U**3)*AD/550
```

```
####### REAL PERFORMANCE CALCULATIONS  #########

XJD = U*60. / (prpm*D)                  # J - Advance Coefficient

#Add viscous forces
TB          = []
BI          = []
SIBI        = []
COBI        = []
COTBI       = []
VSTAR2      = []                        # Vstar squared
VSTAR       = []                        # Vr/U - Hypotenuse of Blade \
                                        Element Velocity Diagram
REYN        = []                        # Reynolds Number
CDRAG       = []                        # Coefficient of Drag
CTV         = []                        # Viscous Thrust Coefficient
CT          = []                        # Total Thrust Coefficient
CPV         = []                        # Viscous Power Coefficient
CP          = []                        # Total Power Coefficient
CL          = []                        # Coefficient of lift
EFF         = []                        # Efficiency
SIGMA       = []                        # Cavitation Number
CMBR        = []
TBC         = []
CCC         = []                        # Variable made for CTT calculation
PPP         = []                        # Variable made for CPP calculation
EARC        = []                        # Variable made for EAR calculation

while i <= rn:
    TB.append(TBR/rb[i])
    BI.append(arctan(TBI[i]))
    SIBI.append(sin(BI[i]))
    COBI.append(cos(BI[i]))
    COTBI.append(1./TBI[i])
    VSTAR2.append(((VAOU[i]+UI[i])**2)+(1./TB[i]-VI[i])**2)
    VSTAR.append(sqrt(VSTAR2[i]))
    REYN.append(R*COR[i]*U*VSTAR[i]/kv)
    CDRAG.append(CDBr[i]+0.15/(((log10(REYN[i]))-2.)**2)+0.003)
    CTV.append(-Z*CDRAG[i]*COR[i]*VSTAR2[i]*SIBI[i]/(2.*pi*rb[i]))
    CPV.append(Z*CDRAG[i]*COR[i]*VSTAR2[i]*COBI[i]/(2.*pi*rb[i]*TB[i]))
    CL.append((2.*pi*rb[i]*CTI[i])/((VSTAR2[i]**2)*COR[i]*Z*COBI[i]))
    CT.append(CTI[i]+CTV[i])
    CP.append(CPI[i]+CPV[i])
    EFF.append(CT[i]/CP[i])
```

```
    SIGMA.append((2117. + 64.*ZH-FRAC[i]*144.)/(0.5*rho*(U*VSTAR[i])**2))
    CMBR.append(0.0679*CL[i])
    TBC.append(-3.008*CMBR[i]+(0.334-0.425*CMBR[i])*SIGMA[i])
    CCC.append(CT[i]*rb[i])
    PPP.append(CP[i]*rb[i])
    EARC.append(COR[i])
    i+=1

CTT1 = CCC[0]+CCC[1]+CCC[2]+CCC[3]+CCC[4]+CCC[5]+CCC[6]+CCC[7]+CCC[8]\
+CCC[9]+CCC[10]+CCC[11]+CCC[12]+CCC[13]+CCC[14]+CCC[15]
CTT  = 2.*CTT1*DR/(1.-RHOR**2)
CPP1 = PPP[0]+PPP[1]+PPP[2]+PPP[3]+PPP[4]+PPP[5]+PPP[6]+PPP[7]+PPP[8]\
+PPP[9]+PPP[10]+PPP[11]+PPP[12]+PPP[13]+PPP[14]+PPP[15]
CPP  = 2.*CPP1*DR/(1.-RHOR**2)
EAR1 = EARC[0]+EARC[1]+EARC[2]+EARC[3]+EARC[4]+EARC[5]+EARC[6]+EARC[7]\
+EARC[8]+EARC[9]+EARC[10]+EARC[11]+EARC[12]+EARC[13]+EARC[14]+EARC[15]
EAR  = Z * EAR1 * DR / pi
EF   = CTT/CPP

XKT = CTT * XJD**2 * pi * (1.-RHOR**2) / 8.
XKQ = CPP * XJD**3 * (1.-RHOR**2) / 16.

self.THST = CTT*0.5*rho*(U**2)*AD
self.PWR  = CPP*0.5*rho*(U**3)*AD/550.

##Any variables with rp stands for global variable used in output file.
self.PrDSrp  = PrDS
self.rnrp    = rn
self.rbrp    = rb
self.CORrp    = COR
self.CTIrp   = CTI
self.CPIrp   = CPI
self.EFFrrp  = EFFr
self.PIODrp  = PIOD
self.UIrp    = UI
self.VIrp    = UI
self.CTrp    = CT
self.CPrp    = CP
self.EFFrp   = EFF
self.CDRAGrp = CDRAG
self.CLrp    = CL
self.REYNrp  = REYN
self.VSTARrp = VSTAR
self.SIGMArp = SIGMA
self.TBCrp   = TBC
```

```
        self.EARrp   = EAR
        self.CTTrp   = CTT
        self.CPPrp   = CPP
        self.EFrp    = EF
        self.XJDrp   = XJD
        self.XKTrp   = XKT
        self.XKQrp   = XKQ

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

 # Display results in an onscrene dialog box
    def OnResults(self, event):
            results ="%6s = %6.2f\n\
%2s           = %6.2f\n\
%15s = %6.2f\n\
%15s                = %6.2f\n\
%15s = %6.2f\n\
%15s         = %6.2f\n\
%15s                      = %6.4f"\
            % ('U [knots]', self.Uopt, 'PA', self.PAopt, 'Calculated Real Thrust\
                [lbs]', self.THST, 'Given Thrust [lbs]', self.T, 'Calculated Real Power [lbs]',\
                 self.PWR, 'Given Real Power [lbs]', self.DHP, 'Real Efficiency', self.EFrp)
            dialog = wx.MessageBox (results, 'Results', style = wx.OK )

# Saves output final data to a "txt" file for printing or viewing.
    def OnOutput(self, event):
        f = open("MTPresults.txt", "w")
        f.write("%s    %s    %s    %s   %s\n" %\
                ('CRR', 'VaU', 'PrD', 'FRAC', 'CDBr'))
        for i in range(int(self.rnrp)+1):
            f.write("%5.4f %5.4f %5.4f %5.4f %5.4f\n" %\
                     (self.CRR[i], self.VaU[i], self.PrDSrp[i], self.FRACV[i], self.CDBR[i]))

        f.write("\n 'Ideal Performance'")
        f.write("\n%s    %s    %s   %s   %s %s   %s    %s\n" %\
                ('rb','CTi','Cpdi', 'PrD', 'C(r)/R', 'EFFr', 'u/U', 'v/U'))
        for k in range(int(self.rnrp)):
            f.write("%5.3f %5.4f %5.4f %5.3f %5.4f %5.4f %5.4f %5.4f\n" %\
                     (self.rbrp[k], self.CTIrp[k], self.CPIrp[k], self.PIODrp[k],\
                       self.CORrp[k], self.EFFrrp[k], self.UIrp[k], self.VIrp[k]))

        f.write("\n%s     %s     %s\n" %('CTTI','CPPI', 'EFI'))
        f.write("%1.6f %1.6f %1.6f\n" % (self.CTTI, self.CPPI, self.EFI))

        f.write("\n%s     %s\n" %('Ideal Thrust','Ideal Power'))
```

```
        f.write("%6.2f         %6.3f\n" % (self.THSTI, self.PWRI))

        f.write("\n 'Design Calculations'")

        f.write("\n 'Real Performance'")
        f.write("\n%s    %s    %s   %s    %s         %s     %s        %s\n" %\
                ('rb','CTi','Cpdi', 'EFF', 'CD', 'CL', 'REYN', 'V*'))
        for m in range(int(self.rnrp)):
            f.write("%5.3f %5.4f %5.4f %5.4f %1.3e %5.4f %1.4e %5.4f\n" % \
                    (self.rbrp[m], self.CTrp[m], self.CPrp[m], self.EFFrp[m], \
                     self.CDRAGrp[m], self.CLrp[m], self.REYNrp[m], self.VSTARrp[m]))

        f.write("\n 'Optimum Thickness/C for Cavitation'")

        f.write("\n '(< To Avoid BB Inception)'")
        f.write("\n%s    %s  %s\n" %('rb', 'SIGMA', 'TBC'))
        for n in range(int(self.rnrp)):
            f.write("%5.3f %5.4f %5.4f\n" % (self.rbrp[n], self.SIGMArp[n], \
                                             self.TBCrp[n]))

        f.write("\n%s " %('EAR'))
        f.write("%1.6f\n" % (self.EARrp))

        f.write("\n%s      %s      %s\n" %('CTT','CPP', 'EF'))
        f.write("%1.6f %1.6f %1.6f\n" % (self.CTTrp, self.CPPrp, self.EFrp))

        f.write("\n%s        %s       %s       %s\n" %('J','KT', 'KQ', 'EF'))
        f.write("%1.6f %1.6f %1.6f %1.6f\n" % (self.XJDrp, self.XKTrp, self.XKQrp, self.EFrp))

        f.write("\n%s     %s\n" %('Real Thrust','Real Power'))
        f.write("%6.2f        %6.3f\n" % (self.THST, self.PWR))

        f.close()


if __name__ == '__main__':
    app = wx.PySimpleApp()
    TextFrame().Show()
    app.MainLoop()
```

Bibliography

[1] G. Kuiper. *The Wageningen Propeller Series*. MARIN Publication 92-001, 1992.

[2] M.W.C. Oosterveld and P. Van Oossanen. Further computer-analyzed data of the wageningen b-screw series. Technical report, Netherlands Ship Model Basin, Wageningen, the Netherlands, 1975.

[3] Ira H. Abbott and Albert E. Von Doenhoff. *Theory of Wing Sections*. Dover Publications, Inc., 1959.

[4] Guido van Rossum and Jr. Fred L. Drake. Python tutorial, 2009.

[5] February 9, 2005. February, 2009. URL http://www.numpy.org/.

[6] August 11, 2009. February, 2009. URL http://www.scipy.org/.

[7] Noel Rappin and Robin Dunn. *wxPython in Action*. Manning, , 2006.

[8] Sebastion Wolff. A local and globalized, constrained and simple bounded nelder-mead method. , 2004.

[9] February 2009. URL http://en.wikipedia.org/wiki/simplex.

[10] Alice E. Smith and David W. Coit. *Handbook of Evolutionary Computations*. Oxford University Press and Institute of Physics Publishing, 1995.

[11] G.S. Baker and A. W. Riddle. Screw propellers of varying blade section in open water. Technical report, Royal Institute of Naval Architects, 1934.

[12] *Recent Developments in Marine Propeller Hydrodynamics*, 1972. International Jubilee Meeting 40th anniversary of the N.S.M.B.

[13] *Representation of Propeller Characteristics suitable for Preliminary Design Studies*, 1973. International Conference on computer Applications in the Automation of Shipyard Operation and Ship Design.

[14] J. D. van Manen and P. van Oossanen. *Principles of Naval Architecture*. The Society of Naval Architects and Marine Engineers, 1988.

[15] L.C. Burrill. *Developments in Propeller Design and Manufacture for Merchant Ships*. IMarEST, 1943.

[16] W.H. Keller. *Engine Aspecten bij het Ontwepen van Scheepsschroeven*. Schip en Werf, 1966.

[17] William S. Vorus. Marine hydrodynamics II. Technical report, University of New Orleans, 2007.

[18] M.M Bernitsas and D. Ray. Optimal diameter b-series propellers. Technical report, University of Michigan, August 1982.

[19] M.M. Bernitsas and D. Ray. Optimal revolution b-series propeller. Technical report, University of Michigan, August 1982.

[20] D. Radojčić. Optimal preliminary propeller design using nonlinear constrained mathematical programming technique. Technical report, University of Southampton, 1985.

Vita

Ricky Biven was born in New Orleans, Louisiana and received his B.S. in Mechanical Engineering from Christian Brothers University in Memphis, Tennessee.