Utah State University
DigitalCommons@USU

All Graduate Plan B and other Reports | Graduate Studies, School of

9-1-2011

Nonlinear Aerodynamic Corrections to Blade Element Momentum Modul with Validation Experiments

Robert S. Merrill
Utah State University

Recommended Citation
Merrill, Robert S., "Nonlinear Aerodynamic Corrections to Blade Element Momentum Modul with Validation Experiments" (2011). *All Graduate Plan B and other Reports.* Paper 67.
http://digitalcommons.usu.edu/gradreports/67

This Creative Project is brought to you for free and open access by the Graduate Studies, School of at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

NONLINEAR AERODYNAMIC CORRECTIONS TO BLADE ELEMENT MOMENTUM MODEL WITH VALIDATION EXPERIMENTS

by

Robert S. Merrill

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Aerospace Engineering

Approved:

Dr. Stephen Whitmore Major Professor	Dr. Warren Phillips Committee Member

Dr. Barton Smith
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011

Copyright © Robert S. Merrill 2011

All Rights Reserved

Abstract

Nonlinear Aerodynamic Corrections to Blade Element Momentum Model with Validation Experiments

by

Robert S. Merrill, Master of Science

Utah State University, 2011

Major Professor: Dr. Stephen Whitmore
Department: Mechanical and Aerospace Engineering

Blade element momentum theory is well suited for propeller analysis during the early stages of design. The analytic blade element momentum model is presented along with a proposed nonlinear improvement. The analytical model makes small angle assumptions which are known to be inaccurate under some conditions. The nonlinear model avoids these assumptions. The results of the analytical and nonlinear models are compared against each other. The differences between these are most prevalent on lower pitch propellers at high advance ratios. A wind tunnel validation test is outlined. Results of the validation test and other published data from the University of Illinois at Urbana-Champaign are compared to the analytical and nonlinear blade element models. The test data matches the nonlinear data with reasonable accuracy at high advance ratios.

(54 pages)

Public Abstract

Nonlinear Aerodynamic Corrections to Blade Element Momentum Model with Validation Experiments

by

Robert S. Merrill, Master of Science

Utah State University, 2011

Major Professor: Dr. Stephen Whitmore
Department: Mechanical and Aerospace Engineering

Many different mathematical models have proved to estimate propeller performance. This study looks at a common method called blade element momentum theory. Some inaccurate assumptions are made to complete the calculations. A proposed model without these assumptions is presented. Propeller performance is tested in a wind tunnel and compared to the predictions made by both models. Published test data is also compared to both models. The test data matches the proposed model with reasonable accuracy.

(54 pages)

Acknowledgments

I would like to acknowledge the help and support I received from my graduate committee by supplying me with the knowledge that I needed to complete this work. It was such an honor to work with each of them. I would especially like to thank Dr. Whitmore for allowing me to complete this work under his guidance and for his patience.

I would like to thank fellow students, Casey Talbot and Shannon Eilers, who spared their own time in helping me through the course of this research.

Most importantly, I would like to thank my wife Jen for supporting me in every imaginable way. This work would not have been possible without her help.

Contents

Page

Abstract . . . . . **iii**

Public Abstract . . . . . **iv**

Acknowledgments . . . . . **v**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

Nomenclature . . . . . **ix**

Introduction **1**
Background 3

Blade Element Momentum Theory **5**
Blade Element Theory 5
Momentum Theory 8
Analytical BEM 11
Nonlinear BEM 12
Programming Implementation 13
Analytical and Nonlinear Comparisons 14

Experimental Procedures and Considerations **17**
Airspeed Measurements 18
Thrust Measurements 19
Power Measurements 20
Data Acquisition 20
Test Procedure 20
Data Reduction 21
Uncertainty 22

Results and Comparisons **24**

Conclusion **28**

References **30**

Appendix **31**
Appendix A: Manufacturer Supplied Data Files for APC 8x8 and 11x5.5 Thin Electric Propellers 32
Appendix B: Analysis Code Written in Java 36

List of Tables

Table Page

1 List of Transducers and Their Respective Function and Specifications 19

2 Aerodynamic Constants Used in the BEM Code 24

List of Figures

Figure Page

1 Illustration of propeller pitch and its effect on the angle of attack. 3

2 Illustration of propeller pitch length. 3

3 Velocities experienced by a blade element and their effect on the angle of attack. 5

4 Stream tube encompassing concentric stream surfaces. 8

5 Velocities and pressure along the slip stream of the propeller. 8

6 User interface of the BEM code. 13

7 Comparison of thrust coefficients between analytical and nonlinear BEM models for propellers with different pitch lengths. 15

8 Comparison of power coefficients between analytical and nonlinear BEM models for propellers with different pitch lengths. 15

9 Photo of propeller installation, pivot arm and load cell configuration. 18

10 Block diagram of mechanical measurements. 18

11 In situ calibration of load cell using calibrated weights. 20

12 Averaged wake profiles highlighting sources of wake survey shortcomings. . . 25

13 Comparison of load cell derived thrust and power coefficients to BEM models (APC 8x8 Thin Electric propeller). 25

14 Angle of attack at each blade element at advance ratios $J = 0$ and $J = 0.45$ (APC 8x8 Thin Electric propeller). 26

15 Comparison of thrust and power coefficients (UIUC data) to BEM models (APC 11x5.5 Thin Electric propeller). 27

Nomenclature

c	Chord length of blade, cm (in)
C_D	Drag coefficient
C_{D0}	Parasitic drag coefficient
$C_{D,L}$	Drag coefficient contribution proportional to lift
C_{D,L^2}	Drag coefficent contribution proportional to lift squared
C_L	Lift coefficient
C_{L0}	Lift coefficient at zero angle of attack
$C_{L,\alpha}$	Lift slope
C_P	Power coefficient
C_T	Thrust coefficient
dC_P	Differential power coefficient
dC_T	Differential thrust coefficient
dQ	Differential torque, N (lbf)
dr	Differential radius of blade, cm (in)
dT	Differential thrust, N (lbf)
dx	Differential nondimensional radius of blade
E	Supplied motor voltage, volts
I	Supplied motor current, amps
I_0	No-load current, amps
J	Advance ratio
J_i	Nondimensional induced velocity
$\dot{m}$	Mass flow rate, kg/sec (slug/sec)
N	Number of blades in propeller
P	Power, watts
p	Pressure before propeller disk, Pa (psi)
p_∞	Free stream atmospheric pressure, Pa (psi)

Q	Propeller torque, N-m (ft-lbf)
r	Radial location along propeller blade, cm (in)
R	Tip radius of blade, cm (in)
R_a	Armature resistance, ohms
R_w	Integration limit of the wake defect, cm (in)
S_t	Wind tunnel test section cross sectional area, m^2 (ft^2)
T	Propeller thrust, N (lbf)
U_I	Standard uncertainty of current measurement, amps
U_{I_0}	Standard uncertainty of no-load current, amps
U_P	Standard uncertainty of power measurement, watts
U_{R_a}	Standard uncertainty of armature resistance, ohms
V_e	Stream tube exit velocity, m/sec (ft/sec)
V_i	Induced velocity, m/sec (ft/sec)
V_∞	Free stream velocity parallel to propeller axis, m/sec (ft/sec)
V'_∞	Corrected free stream velocity
$V(r)$	Velocity at radial location r, m/sec (ft/sec)
x	Nondimensional blade radius
x_r	Nondimensional blade root location
x_t	Nondimensional blade tip location
α	Angle of attack, deg.
α_i	Induced angle of attack, deg.
β	Propeller blade pitch angle, deg.
Δp	Pressure jump across propeller disk, Pa (psi)
γ	Ratio of propeller disk area to tunnel area
λ	Propeller pitch length, cm (in)
Φ	Advance angle, deg.
ρ	Air density, kg/m^2 ($slug/ft^2$)
τ	Nondimensional thrust used for veloctiy correction
θ	Circumferential location around the propeller axis
ω	Rotation rate of propeller, rad/sec

Introduction

Predicting the characteristics of high speed rotating flow fields, because of the inherently unsteady flow properties, is one of the most complex analytical problems in modern fluid mechanics. Advances in computer execution speed, memory capacity and user interactivity have allowed numerical techniques in computational fluid dynamics (CFD) to grow rapidly in the last decade. Unfortunately, even with the growing capabilities of CFD, solutions for rotating propellers and turbine blades are very difficult to achieve. Generating CFD grids that allow for accurate, converged solutions is an extremely difficult problem. Small changes in the grid layout or boundary conditions can dramatically effect the end results. Thus, the sheer cost, volume of labor required for grid generation and long computational times preclude the use of CFD during the early design stages of flight vehicles that include propeller systems.

For conceptual design, low-order engineering codes are still the preferred method for designers. Engineering codes are powerful tools when applied in the conceptual design stage. Their use in the early stages of design enables higher fidelity CFD calculations or wind tunnel tests to be performed on more mature vehicle concepts and can trim many months off the design process. Traditionally, one of the following low-order engineering design methods have been used to calculate the steady state flow properties behind rotating propellers and turbine blades. These are momentum theory, Goldstein's vortex theory and blade element theory. Often momentum theory is combined with blade element theory to produce a single low-order prediction tool referred to as the combined blade element momentum (BEM) theory.

Goldstein [1] developed a potential flow vortex theory for propellers with a finite number of propellers in axial flow. For moderate inflow conditions, vortex theory has successfully been used to derive the performance of propellers and wind turbines [2]. In the model, the wake is considered to be a helical vortex sheet trailing the propeller at a constant pitch.

Goldstein solves the problem of inter-meshed helical surfaces of infinite axial extension and finite radius. The solution takes the form of a tip loading factor, a function of the inflow, the number of blades and the radial blade station [3].

The single biggest drawback of Goldstein's method is the requirement of precise knowledge of the blade geometry including the chord profile and blade twist profile. Goldstein's vortex theory, especially for screw propellers is very sensitive to errors in the input geometry. Consequently, it is unclear whether vortex theory modeling error reported in technical literature is a result of error in the geometric model of the propeller, error in the airfoil section aerodynamic performance modeling or errors in the assumptions associated with Goldstein's vortex theory.

While the vortex theory represents an important tool for analysis, the geometry sensitivity makes a difficult tool to use for initial design. This process can be quite cumbersome when a designer desires to embark without a preconceived geometry. As an alternative, this research investigates the BEM theory and introduces several modifications to enhance the predictive accuracy of the model. Momentum theory provides a simple momentum analysis across the propeller disk. It gives a general description of the flow through the propeller. Blade element theory discretizes a propeller blade and each element is analyzed individually. However, blade element theory alone lacks the ability of predicting the propeller induced velocity needed to complete the flow field description. The BEM theory uses concepts of momentum theory to complete the blade element model.

The traditional method presented by McCormick [4] includes small angle assumptions to obtain an analytical solution to the BEM equations. These assumptions are known to be inaccurate, especially for low advance ratios and high advance angles. This paper presents a nonlinear solution method that avoids these inaccurate small angle assumptions and as such provides an enhancement to the well known BEM model. This paper compares the two BEM solution methods to each other and to test data collected from propellers sized 8x8 and 11x5.5.

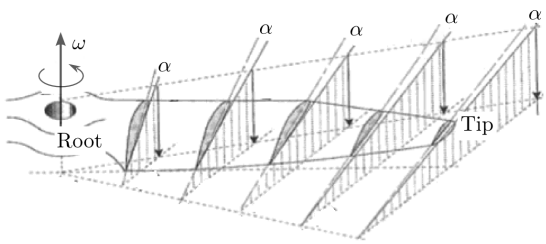


Fig. 1: Illustration of propeller pitch and its effect on the angle of attack.

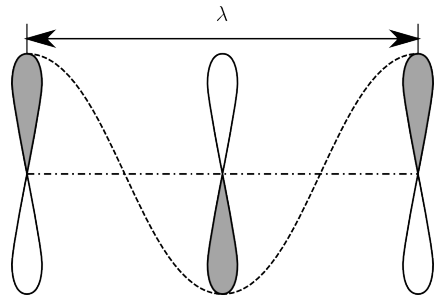


Fig. 2: Illustration of propeller pitch length.

Background

Propellers add momentum to the surrounding fluid to propel an object through the fluid. Early airplane propellers were known as "airscrews" and act appropriately. A propeller forces fluid in one direction, pushing itself in the other. It can also be thought of a rotating wing such that the experienced velocity is the vector sum of the rotational velocity and the forward velocity. As a propeller spins in air, it experiences a drag resistance to the rotational motion. The shaft power required to overcome drag and keep the propeller spinning at a constant rotational rate is known as the braking power. The shape of the propeller blade determines its effectiveness at producing thrust. Propeller blades are inclined to the flow to allow better air capture and greater mechanical efficiency. This inclination is referred to as pitch. Figure 1 shows how the pitch angle is often varied along the length of the blade to keep the angle of attack more constant. Pitch is measured as the distance the propeller would travel in one revolution through a solid medium like a screw through wood. Figure 2 shows an example of pitch length. The local pitch angle is equal to

$$\beta = \tan^{-1}\left(\frac{\lambda}{2\pi r}\right) \tag{1}$$

where λ is the pitch length and r is the local radius of the blade. Often a propeller may not have the pitch profile as described in Eq. 1. In these cases, the pitch is measured by the pitch length or angle at 75% of the radius of the blade. Propellers in this study follow the profile of Eq. 1. Typically, performance of a propeller is quoted at a nondimensional velocity

known as the advance ratio. The advance ratio is the ratio of the distance a propeller moves forward in the working fluid in one revolution to the diameter of the propeller itself. The advance ratio is equal to

$$J = \frac{V_\infty}{\frac{\omega}{\pi} R} \tag{2}$$

The root of the blade is defined here as the innermost portion of the propeller that is not intended to produce lift. The tip of the blade is the outermost portion of the blade. Propellers are sized by their diameter and pitch quoted in inches. For example, a 10x5 propeller has a 25.4 cm (10 in) diameter propeller with a 12.7 cm (5 in) pitch length. The propeller nomenclature gives no indication of its pitch profile.

Propellers are the main method of propulsion for small UAV's and hobby remote controlled aircraft. They are commonly paired with electric DC motors. Direct current motors are employed in this study as the means of providing the shaft power to spin the propeller. A DC motor can be characterized by its no-load current and armature resistance. The no-load current is the current that is drawn from a power source without any torque opposing the motion of the motor. The torque of a motor is proportional to the current that is being supplied to the motor, thus any current drawn by a motor over the no-load current contributes directly to the torque production of the motor. The armature resistance is the electrical resistance that is seen through the armature of the motor. The voltage being supplied to the motor is proportional to the motor speed. The excess voltage above the voltage drop due to the armature resistance contributes directly to the speed of the motor.

Blade Element Momentum Theory

Blade Element Theory

Blade element theory estimates the performance of propellers by analyzing the aerodynamic forces on discrete elements along the radius of the blade. Airfoil section properties are used to find these discrete forces. They are integrated along the propeller to estimate a total resultant thrust force and opposing torque. This torque multiplied by the angular rotation rate is the braking power. The velocity that the blade elements experience is the vector combination of the free stream axial velocity, induced velocity and rotational velocity. The induced velocity is the increased inflow created by the propeller. Figure 3 shows a propeller blade section. The local pitch angle of the blade, β, is a function of the pitch length and the radial distance from the axis. The advance angle, Φ, is the reduction in angle of attack that results from the free stream velocity and is equal to

$$\Phi = \tan^{-1}\left(\frac{V_\infty}{\omega r}\right) \tag{3}$$

The induced angle of attack is the reduction of the angle of attack due to the induced velocity. As the free stream and the induced velocities increase, the angle of attack decreases. The net angle of attack is equal to

$$\alpha = \beta - \alpha_i - \Phi \tag{4}$$

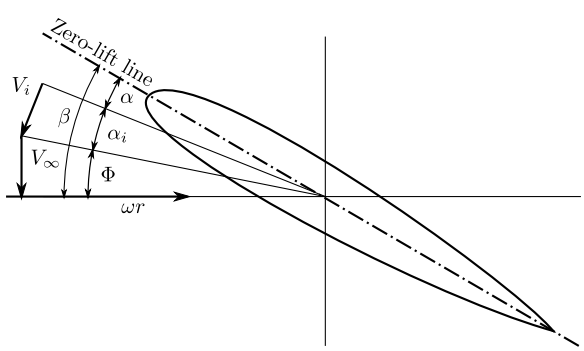


Fig. 3: Velocities experienced by a blade element and their effect on the angle of attack.

The lift and drag coefficients for a single blade element are the same as those of an airfoil cross section at the resultant angle of attack. The linear lift model and quadratic drag model are

$$C_L = C_{L0} + C_{L,\alpha}(\beta - \alpha_i - \Phi) \tag{5}$$

$$C_D = C_{D0} + C_{D,L}C_L + C_{D,L^2}C_L^2 \tag{6}$$

These models predict the thrust and drag well for moderate angles of attack. At angles of attack above 10-15 degrees, the airfoil becomes stalled and these models are no longer accurate. The thrust and torque contributions of each element are

$$dT = \frac{1}{2}\rho\left(V_\infty^2 + (\omega r)^2\right)\left(C_L\cos(\alpha_i + \Phi) - C_D\sin(\alpha_i + \Phi)\right) N\, c\, dr \tag{7}$$

$$dQ = \frac{1}{2}\rho\left(V_\infty^2 + (\omega r)^2\right)\left(C_L\cos(\alpha_i + \Phi) - C_D\sin(\alpha_i + \Phi)\right) N\, c\, r\, dr \tag{8}$$

It is important to note that the lift and drag coefficients are nondimensionalized with respect to the combined velocity $V_\infty^2 + \omega^2 r^2$.

The thrust coefficient is defined by

$$C_T = \frac{T}{\rho(\frac{\omega}{2\pi})^2\,(2R)^4} = \frac{T}{\rho R^4 \frac{4\omega^2}{\pi^2}} \tag{9}$$

The differential thrust coefficient, found by combining Eq. 7 and Eq. 9, is

$$dC_T = \frac{V_\infty^2 + (r\omega)^2}{R^4\frac{4\omega^2}{\pi^2}}\left(C_L\cos(\alpha_i + \Phi) - C_D\sin(\alpha_i + \Phi)\right) N\, c\, dr \tag{10}$$

and in dimensionless form

$$dC_T = \frac{J^2 + \pi^2 x^2}{8}\left(C_L\cos(\alpha_i + \Phi) - C_D\sin(\alpha_i + \Phi)\right)\sigma\, dx \tag{11}$$

where

$$x = \frac{r}{R}$$

$$\sigma = \frac{N\,c}{R}$$

The power coefficient is defined by

$$C_P = \frac{Q\omega}{\rho(\frac{\omega}{2\pi})^3\,(2R)^5} = \frac{Q\omega}{\rho R^5 \frac{4\omega^3}{\pi^3}} \tag{12}$$

The differential power coefficient, found by combining Eq. 8 and Eq. 12, is

$$dC_P = \frac{V_\infty^2 + r^2\omega^2}{R^5 \frac{4\omega^3}{\pi^3}} \left(C_L \sin(\alpha_i + \Phi) + C_D \cos(\alpha_i + \Phi)\right) N\,c\,r\,dr \tag{13}$$

and in dimensionless form

$$dC_P = \pi \frac{J^2 + \pi^2 x^2}{8} \left(C_L \sin(\alpha_i + \Phi) + C_D \cos(\alpha_i + \Phi)\right) \sigma\,x\,dx \tag{14}$$

Differential thrust and power are solved for at each blade element. Integrating these differentials give the total thrust and power coefficient for the propeller.

$$C_T = \int_{x_r}^{x_t} \frac{J^2 + \pi^2 x^2}{8} \left(C_L \cos(\alpha_i + \Phi) - C_D \sin(\alpha_i + \Phi)\right) \sigma\,dx \tag{15}$$

$$C_P = \int_{x_r}^{x_t} \pi \frac{J^2 + \pi^2 x^2}{8} \left(C_L \sin(\alpha_i + \Phi) + C_D \cos(\alpha_i + \Phi)\right) \sigma\,x\,dx \tag{16}$$

The integration limits are from the propeller root to the propeller tip and will be discussed in more detail in the programming implementation section.

The required propeller geometry for the blade element theory is the diameter, number of blades, pitch and chord variation along the radius. The propeller blade cross section is required to solve for the lift and drag coefficients in Eqs. 5 and 6. The last piece of

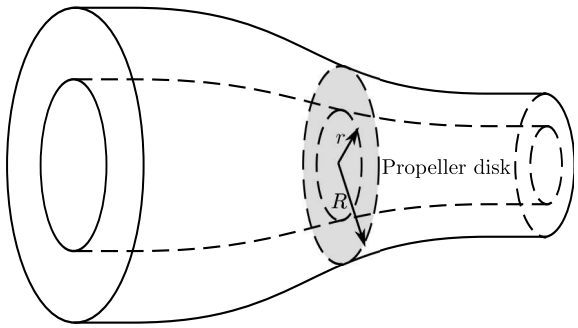


Fig. 4: Stream tube encompassing concentric stream surfaces.

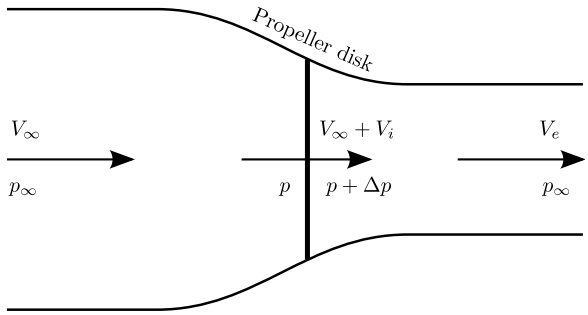


Fig. 5: Velocities and pressure along the slip stream of the propeller.

information required for closure of the differential thrust and power coefficients is the induced angle of attack on each element. The induced angle of attack is obtained by momentum theory.

Momentum Theory

Momentum theory uses a simple flow analysis to calculate the velocity distribution of a slip stream by analyzing the flow through axis-symmetric differential stream tubes. The differential stream tube is made of stream surfaces which are comprised of all streamlines that occupy a common radial location, thus the velocity is radius dependent. Figure 4 shows the concentric stream surfaces. The flow is assumed to be incompressible, inviscid and irrotational, even across the propeller disk. Momentum is balanced from far upstream of the propeller to far downstream. The propeller is treated as an infinitely thin disk with an abrupt pressure increase across it. Figure 5 shows the the momentum flow model. Each stream surface inlet starts at the free stream velocity. The velocity continuously increases

along the stream surfaces as it passes through the propeller disk to the exit of the stream tube. The pressure decreases from the free stream condition to p just before the propeller disk. The velocity at the propeller disk is $V_\infty + V_i(r)$. The pressure increases by Δp across the propeller disk and decreases to the free stream condition at the exit. The velocity at the exit of the stream tube increases to V_e. Bernoulli's equation is applied to the stream surface from the free stream to the propeller and from the propeller to the exit conditions which yields the following:

$$p_\infty + \frac{\rho V_\infty^2}{2} = p + \frac{\rho \left(V_\infty + V_i(r)\right)^2}{2} \tag{17}$$

$$p + \Delta p + \frac{\rho \left(V_\infty + V_i(r)\right)^2}{2} = p_\infty + \frac{\rho V_e^2}{2} \tag{18}$$

Eqs. 17 and 18 are combined and the pressure change across the propeller disk is solved for

$$\Delta p = \frac{\rho \left(V_e^2 - V_\infty^2\right)}{2} \tag{19}$$

Knowing the pressure difference across the propeller, a differential thrust from the propeller is found by multiplying it by the differential area at the propeller disk

$$dT = \rho \left(V_e^2 - V_\infty^2\right) \pi r \, dr = \rho \left(V_e - V_\infty\right) \left(V_e + V_\infty\right) \pi r \, dr \tag{20}$$

The differential thrust can also be found by a momentum balance through a stream surface. The difference between the upstream and downstream momentum is that which is added to the flow by the propeller. The momentum balance is

$$dT + \dot{m}(r) V_\infty = \dot{m}(r) V_e \tag{21}$$

The differential mass flow through the stream surface, calculated at the propeller disk, is

$$\dot{m}(r) = 2\rho \left(V_\infty + V_i(r)\right) \pi r \, dr \tag{22}$$

The momentum balance in Eq. 21 can then be written as

$$dT = 2\rho \left(V_\infty + V_i(r)\right)\left(V_e - V_\infty\right)\pi r\, dr \tag{23}$$

The differential thrust, Eqs. 20 and 23, are equated to produce the following relationship between the free stream, induced and exit velocities

$$V_e(r) = V_\infty + 2V_i(r) \tag{24}$$

Using the exit velocity in Eq. 23 gives a definition of the differential thrust

$$dT = 4\rho V_i(r)\left(V_\infty + V_i(r)\right)\pi r\, dr \tag{25}$$

which can be solved for the induced velocity. The induced velocity at a radial location is

$$V_i(r) = \pm\sqrt{\frac{V_\infty^2}{4} + \frac{dT}{4\rho\pi r\, dr}} - \frac{V_\infty}{2} \tag{26}$$

The positive root is kept meaning that $V_i(r) > 0$ which is expected when thrust is produced opposite the direction of the free stream. The nondimensional induced velocity at a location is

$$J_i(r) = \sqrt{\frac{J^2}{4} + \frac{1}{\pi x}\frac{dC_T}{dx}} - \frac{J}{2} \tag{27}$$

Equations 25 and 26 can be combined to give a relationship between thrust and power. In a nondifferential and nondimensionalized form, the ideal power coefficient is

$$C_P = C_T\left(\frac{J}{2} + \sqrt{\frac{J^2}{4} + \frac{2C_T}{\pi}}\right) \tag{28}$$

Equation 28 is considered to be an idealized relationship and is the upper limit of propeller efficiency.

Momentum theory has come under heavy scrutiny due to its neglect of obvious rotation [5]. A propeller has been likened to a rotating wing and as such, creates a trailing vortices

as the individual blade elements create lift. The vortices trail the propeller in a helical fashion about the propeller axis as it moves through the working fluid. The actual source of the induced velocity is from this helical vortex system. The induced flow actually originates from behind the propeller as opposed to in front as momentum theory assumes. As the focus of this paper is on an improvement on combined blade element and momentum theory, the shortcomings of momentum theory are accepted while warning the reader of its deficiencies.

Analytical BEM

The momentum analysis provides the necessary flow information for the blade element theory to calculate thrust and power. The induced angle of attack from Figure 3 is

$$\alpha_i = tan^{-1}\left(\frac{V_i(r)}{\sqrt{V_\infty^2 + (\omega r)^2}}\right) \tag{29}$$

and defined in nondimensional terms

$$\alpha_i = tan^{-1}\left(\frac{J_i(r)}{\sqrt{J^2 + \pi^2 x^2}}\right) \tag{30}$$

McCormick assumes the thrust is much greater than the drag such that it has little effect on the induced angle of attack, induced angle of attack is small and the advance angle is small. The differential thrust on a blade element, Eq. 7, then becomes

$$dT = \frac{1}{2}\rho\left(V_\infty^2 + (\omega r)^2\right) C_L N\, c\, dr \tag{31}$$

Small angle approximation of the induced angle of attack, Eq. 29 produces

$$\alpha_i = \frac{V_i(r)}{\sqrt{V_\infty^2 + (\omega r)^2}} \tag{32}$$

The differential thrust from momentum theory, Eq. 25, can be equated to the approximate differential thrust, Eq. 31. After some manipulation, it becomes

$$\alpha_i^2 + \alpha_i \left(\frac{V_\infty}{\omega r} + \frac{C_{L,\alpha} N c}{8\pi r} \sqrt{1 + \left(\frac{V_\infty}{\omega r} \right)^2} \right) - \frac{C_{L,\alpha} N c}{8\pi r} \sqrt{1 + \left(\frac{V_\infty}{\omega r} \right)^2} (\beta - \Phi) = 0 \tag{33}$$

The solution of this quadratic equation is the analytic induced angle of attack

$$\begin{aligned} \alpha_i \quad = \quad & -\frac{1}{2} \left(\frac{V_\infty}{\omega r} + \frac{C_{L,\alpha} N c}{8\pi r} \sqrt{1 + \left(\frac{V_\infty}{\omega r} \right)^2} \right) \\ & + \frac{1}{2} \sqrt{\left(\frac{V_\infty}{\omega r} + \frac{C_{L,\alpha} N c}{8\pi r} \sqrt{1 + \left(\frac{V_\infty}{\omega r} \right)^2} \right)^2 + \frac{C_{L,\alpha} N c}{8\pi r} \sqrt{1 + \left(\frac{V_\infty}{\omega r} \right)^2} (\beta - \Phi)} \end{aligned} \tag{34}$$

The induced angle of attack is solved at each blade element and then used in Eq. 15 and Eq. 16 to determine the thrust and power coefficients.

Nonlinear BEM

The purpose of the nonlinear BEM model is to provide a more accurate representation of what the propeller blade experiences by challenging the assumptions that the drag has little effect on the induced angle of attack, the induced angle of attack is small and the advance angle is small. The inaccuracies of these assumptions are clearly seen with high pitch propellers at low advance ratios. The nonlinear modification of blade element momentum model iterates through a series of equations from both blade element and momentum theory until a convergence on the induced angle of attack is reached on each blade element. The induced angle of attack analytical approximation, Eq. 34, is the starting point for the nonlinear iteration. The thrust coefficient derivative with respect to x is then solved for. This derivative is used to find the induced velocity. Last, the induced velocity is used to calculate a refined estimation of the induced angle of attack. The iterated equations are presented in Eqs. 35, 36 and 37.

$$\frac{dC_T}{dx} = \frac{J^2 + \pi^2 x^2}{8} \left(C_L \cos(\alpha_i + \Phi) - C_D \sin(\alpha_i + \Phi) \right) \sigma \tag{35}$$

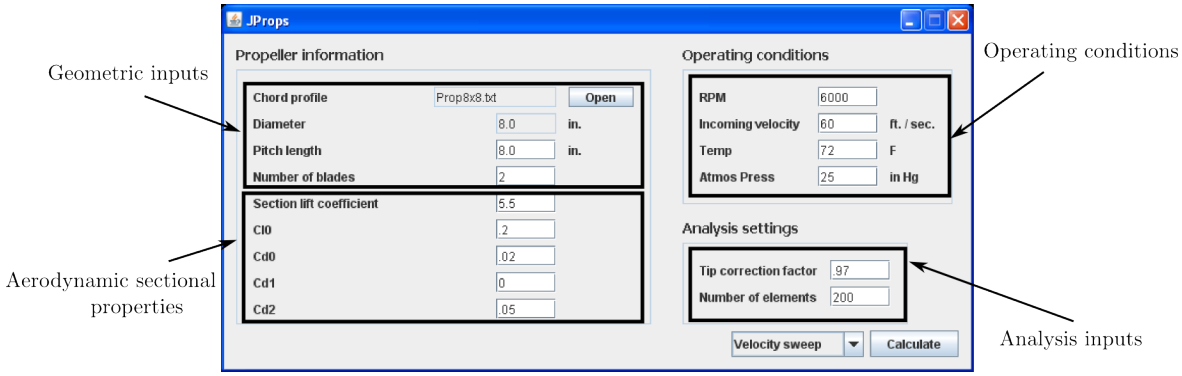


Fig. 6: User interface of the BEM code.

$$J_i(r) = \sqrt{\frac{J^2}{4} + \frac{1}{\pi x}\frac{dC_T}{dx}} - \frac{J}{2} \tag{36}$$

$$\alpha_i = tan^{-1}\left(\frac{J_i(r)}{\sqrt{J^2 + \pi^2 x^2}}\right) \tag{37}$$

with

$$\Phi = \tan^{-1}\left(\frac{V_\infty}{\omega r}\right) \tag{38}$$

$$dC_T = \frac{J^2 + \pi^2 x^2}{8}\left(C_L \cos(\alpha_i + \Phi) - C_D \sin(\alpha_i + \Phi)\right)\sigma\, dx \tag{39}$$

$$dC_P = \pi\frac{J^2 + \pi^2 x^2}{8}\left(C_L \sin(\alpha_i + \Phi) + C_D \cos(\alpha_i + \Phi)\right)\sigma\, x\, dx \tag{40}$$

for closure. The converged induced angle of attack is solved at each blade element and then used in Eq. 15 and Eq. 16 to determine the thrust and power coefficients.

Programming Implementation

The computer code for the current study was written in Java for is ease in GUI programming, ability to run independent of platform and as a learning experience for the author. The program interface is shown in Figure 6. The geometric inputs to the code are the chord/pitch profile file, optionally pitch length, number of blades, lift slope, zero angle lift coefficient, and the drag coefficients in the parabolic relation of drag to lift. The chord/pitch profile file is formatted in plain text. Each line contains a radial location, local chord length

and optionally, the local pitch length. The radius of the propeller root and tip are defined as the first and last radius value found in the file. The diameter is twice the tip radius. The operating condition inputs are rotational velocity, incoming velocity, temperature and atmospheric pressure. The purpose of temperature and pressure is to find the air density for dimensional results. Analysis inputs are a tip correction factor and number of blade elements. For a finite wing or lifting surface, the local sectional lift coefficient must approach zero near the wing tip. Many different approaches have been proposed to implement this condition [6]. This analysis uses most simple accepted model for initial estimations by assuming no lift is produced beyond $x = 0.97$ [7]. The blade element chord length and pitch are found at the center of each blade element. The chord and pitch is linearly interpolated from the provided blade profile geometry. If the center of the blade element is at a location beyond the tip correction, the lift in the analytical BEM is set to zero. However, in the nonlinear implementation, by setting the lift to zero, the derivative dC_T becomes less than zero. To avoid taking the root of a negative number, in the nonlinear BEM the differential thrust coefficient is alternatively set to zero. The thrust of a propeller is primarily a product of the lift of the blade elements and this modification is a reasonable approximation to the tip effect correction developed by McCormick.

With the required information, the code solves for the differential thrust and power coefficients for each blade element and numerically integrates along the length of the blade.

Analytical and Nonlinear Comparisons

For the sake of comparison, the geometry of an APC 8x4, 8x6 and 8x8 Thin Electric propellers are used for the following calculations. The chord profiles remain equal between the different pitch lengths. It was hypothesized that the inaccuracies of the the analytical BEM model would occur at high advance ratios where the advance angle is large and with high pitch propellers. The high advance ratios would made the advance angle large, rejecting any small angle approximations. The higher pitch propellers would also create a greater induced angle of attack, again being less suited for a small angle approximation. Figures 7 and 8 compare the power and thrust coefficients for each of the propellers. It is clear that

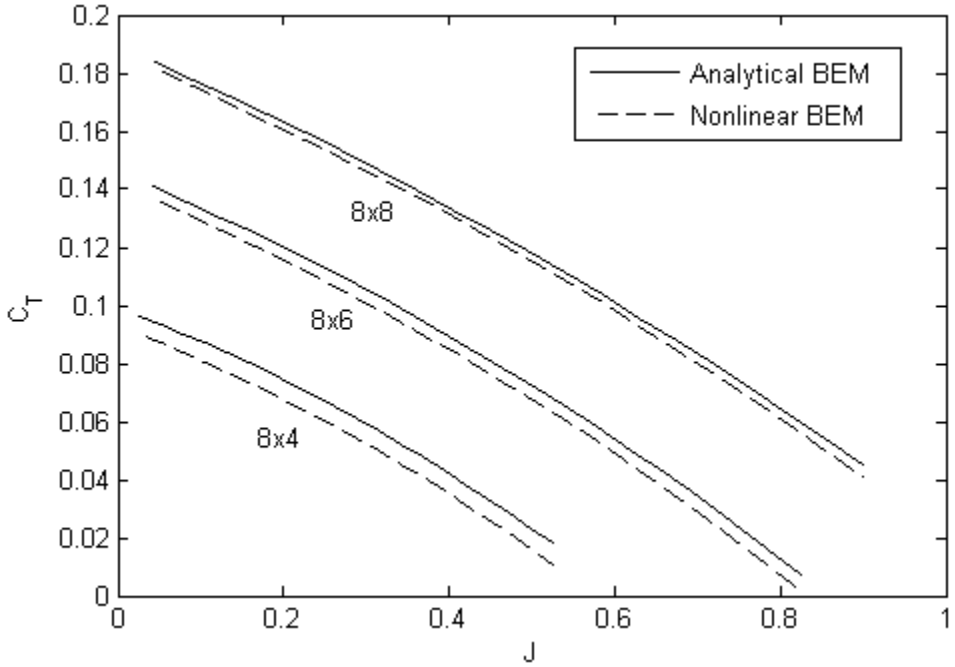


Fig. 7: Comparison of thrust coefficients between analytical and nonlinear BEM models for propellers with different pitch lengths.

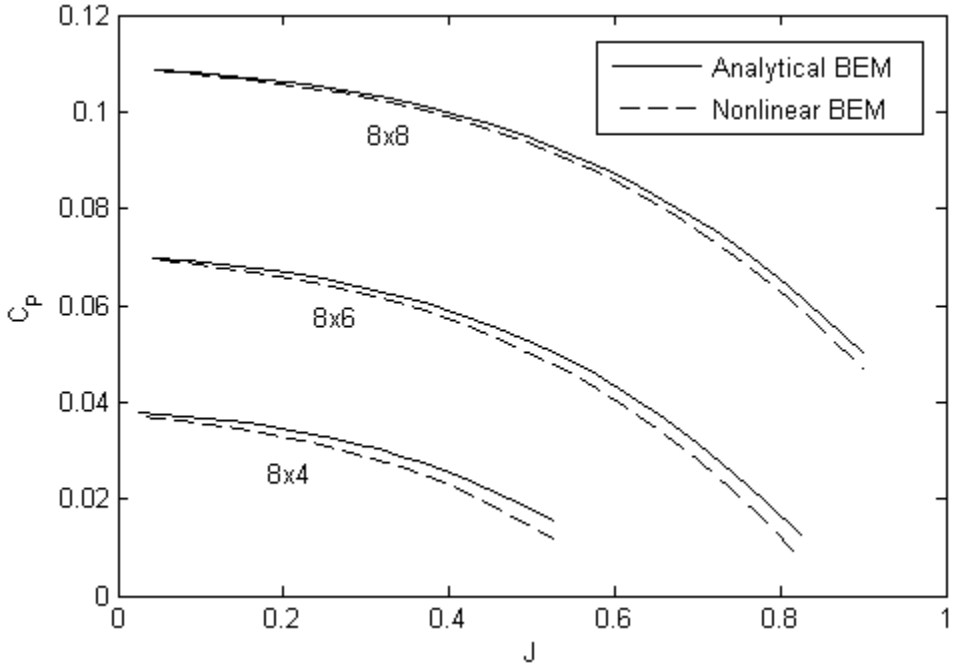


Fig. 8: Comparison of power coefficients between analytical and nonlinear BEM models for propellers with different pitch lengths.

the impact that the high advance angle has on performance is more pronounced in the power coefficient than in the thrust coefficient. Interestingly, the thrust coefficient differs more with the lower pitch propeller than the higher pitch propeller. The most reasonable cause for this is the analytical assumption that lift is much greater than the drag. The assumption over predicts the lift and therefore, thrust of the analytical model. The higher pitch a propeller is, the more lift is produced on the blade, thus more thrust. This assumption is more valid on higher pitch propellers than it is on lower as is evident in the thrust curves. This difference outweighs any effect the induced angle of attack has on the higher pitch propellers as originally thought.

Experimental Procedures

Three types of experiments were performed. The sole purpose for two tests was to characterize the DC motor used to power the propeller. The first, to measure the no-load current by measuring the current drawn by the motor without a resistive load on the shaft. The second measured the armature resistance by measuring the voltage and current supplied to the motor while preventing the motor from spinning. The last, and most intensive experiment was the validation experiment in which the thrust and braking power was measured inside the wind tunnel at different tunnel velocities. The 8x8 APC Thin Electric propeller was tested in the wind tunnel, measuring thrust using both a load cell an calculating a momentum integral from the wake. The wake was measured using a sweeping pitot probe. Power being supplied to the motor was calculated through voltage and current measurements.

Validation experiments were performed inside of a low speed nonrecirculating wind tunnel owned by the Mechanical and Aerospace Department at Utah State University. The wind tunnel test section dimensions are 40.6×40.6×121.9 cm (16×16×48 in.). The 8x8 propeller was installed to a motor and gear box assembly[1]. The motor assembly was situated in the center of the test section and mounted to one end of a pivoting arm. The opposite end of the lever arm extends out the bottom of the test section with the pivot just outside of the wall of the tunnel. A load cell was attached to the outside end of the lever arm. The pivot allows the forces to be isolated in the axial direction and allows a mechanical advantage to use more of the load cell range. Figure 9 is a photo of the propeller installed in the wind tunnel test section. Airspeed measurements consist of total pressure ports near the side of the tunnel fore and aft of the propeller, a static port aft of the propeller and a sweeping pitot probe aft of the propeller. The pitot probe is mounted to a traversing track. The traversing track slides along a linear potentiometer for a position measurement

[1]The motor and gearbox are parts of a remote controlled airplane, a Hobby-Zone Super Cub. The motor part number is HBZ7134 and the gear box, HBZ7129. http://secure.hobbyzone.com/index/index_park_flyers_rtf/HBZ7300.html

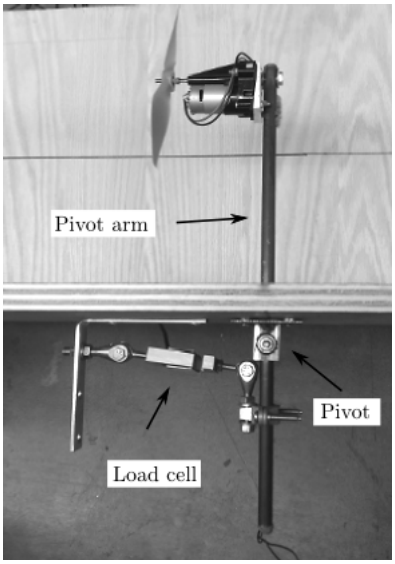


Fig. 9: Photo of propeller installation, pivot arm and load cell configuration.

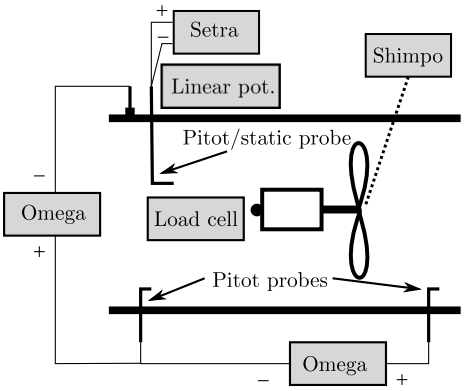


Fig. 10: Block diagram of mechanical measurements.

of the pitot probe. The voltage being supplied to the motor is measured through a voltage divider circuit. The current being supplied to the motor is measured through an inductive ammeter. Figure 10 shows a block diagram of the mechanical measurements. Table 1 lists each transducer, their function, ranges and accuracies. The transducers will be discussed individually in the following sections.

Airspeed Measurements

The total air pressure is measured by pitot probes 44.5 cm (17.5 in) upstream and 54.6 cm (21.5 in) downstream of the propeller. The difference between these measurements indicates losses in the test section. A static pressure port is also located 54.6 cm (21.5 in) downstream of the propeller. The difference between the downstream total pressure and the downstream static pressure is measured. Both measurements are performed by two Omega PX143-01BD differential pressure transducers. These two measurements give both an upstream and downstream velocity outside of the propeller wake.

The sweeping pitot probe is oriented at the same height and 50.8 cm (20 in) downstream of the propeller. The probe sweeps horizontally from the tunnel wall to approximately 3

Table 1: List of Transducers and Their Respective Function and Specifications

Make/Model Number	Measurement	Function	Range	Accuracy
Omega PX143-01BD	Differential pressure	Tunnel loss	±6.9 kPa (1 psi)	0.03% FS*
Omega PX143-01BD	Differential pressure	Free stream velocity	±6.9 kPa (1 psi)	0.03% FS*
Setra Datum 2000 model 2239	Differential pressure	Velocity profile	0 - 3.7 kPa (0.54 psi)	0.14% FS*
Omega LCCD-25	Force	Propeller thrust	±111 N (25 lbf)	0.2% FS*
Fluke i30	Current	Motor current	0.03 - 30A	1% of reading
Shimpo DT-209X	Rotation rate	Motor RPM	6.0 - 99,999 RPM	±1 RPM

*As measured from calibration data

cm (1.2 in) past the center of the tunnel. The total and static pressure feed into a Setra Datum 2000 model 2239 differential pressure transducer. The position of the pitot probe is measured by a linear, pressure sensitive potentiometer. The combined measurement of the position and airspeed allow for the measurement of the radial velocity profile from the propeller. Each pressure sensor was calibrated using a wall mounted Meriam GP-6 model 40GE4 manometer.

Thrust Measurements

The propeller thrust is measured directly by the load cell mounted opposite of the propeller on the pivot arm. One end of the load cell is attached to the pivot arm, and the other to a rigid mount attached to the outer wall of the test section. The load cell is an Omega LCCD-25. The calibration was performed in situ with known weights hanging on a cord using a pulley to allow the cord to pull straight along the propeller axis. Figure 11 shows this arrangement.

Wake surveys were also used to indirectly measure the thrust of the propeller. The wake profile is an indication of how much momentum was added to the flow from the propeller.

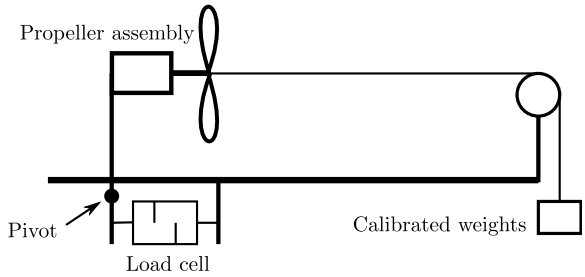


Fig. 11: In situ calibration of load cell using calibrated weights.

Power Measurements

The current being supplied to the motor is measured by a Fluke i30 inductive current clamp. The speed of the propeller is measured by a Shimpo DT-209X laser tachometer. The voltage supplied to the motor is measured directly through a voltage divider circuit into the data acquisition unit.

Data Acquisition

The measurements were taken by a National Instruments DAQCard-6024 PCMCIA card with a 12 bit resolution. The data acquisition card was connected to a National Instruments BNC-2110 connector block. These make up the data acquisition unit (DAQ). Samples were taken at 500 Hz for all the measurements apart from the RPM measurement which was taken at 2 Hz.

Test Procedure

The motor no-load current, I_0, was measured by simply allowing the motor to spin freely without any opposition. The armature resistance is calculated by measuring the voltage and current supplied to the motor while preventing the motor from spinning. Minutes prior to propeller testing, atmospheric pressure and temperature were measured and logged. At each tunnel velocity, a set of measurements was taken with and without powering the propeller. The amount of drag created by the motor assembly and pivot arm was measured. The propeller was allowed to spin during the drag measurements. This measured drag was later subtracted from the thrust measurement. The motor was then powered and another set of

measurements are taken. The pitot swept forward, from the tunnel wall to just past the propeller axis and back to the wall. In total, 3 forward and back sweeps were made as part of each test. The duration of a test was approximately 55 to 65 seconds.

Data Reduction

Momentum and mass are balanced upstream and downstream of the propeller to yield the thrust momentum integral

$$T = \int_0^{2\pi} \int_0^{R_w} \rho V_\infty^2 \left(\frac{V(r)^2}{V_\infty^2} - 1 \right) r \, dr \, d\theta \tag{41}$$

The R_w limit of integration is the outer location of the wake and is observed from the data where wake remains equal with the free stream velocity. Conservation of mass through the test section dictates that

$$\frac{R_w^2}{2} = \int_0^{R_w} \frac{V(r)}{V_\infty} r \, dr \tag{42}$$

Equation 41 is integrated with respect to circumferential dimension, θ, and combined with Eq. 42 such that the gross thrust is equal to

$$T = 2\pi\rho V_\infty \int_0^{R_w} \left(V(r) - \frac{V(r)^2}{V_\infty} \right) r \, dr \tag{43}$$

The measured velocity profiles are biased in the position data. It is reasonable to assume that the velocity profiles are symmetric about the axis, therefore, in some cases, the profile position data was biased such that the majority of any measured asymmetry was removed. This asymmetry was an artifact of the measurement system. The velocity data was averaged at each discrete measured position created by the bit resolution of the DAQ. A trapezoidal integration was performed using the averaged velocity points to compute the wake integral in Eq. 43. The drag and thrust profile were integrated individually. The drag was subtracted from gross thrust to calculate the net amount of thrust.

The free stream velocity used in the advance ratio is adjusted to reflect a more accurate approximation of velocity where measured thrust and torque would be produced in open

air. The propeller flow is constrained by the tunnel. Therefore, the measured velocity is different than what would occur in an unconstrained flow at the same level of performance. This free stream velocity correction is outlined in Glauert [8] and is equal to

$$\frac{V'_\infty}{V_\infty} = 1 - \frac{\gamma}{2}\frac{\tau}{\sqrt{1+2\tau}} \tag{44}$$

where

$$\tau = \frac{T}{\pi R^2 \rho V_\infty^2}$$

$$\gamma = \frac{\pi R^2}{S_t}$$

and S_t is the cross sectional area of the wind tunnel test section. The free stream velocity was calculated by the fore total pressure probe and the aft static pressure port.

Using a measured no-load current I_0, and armature resistance R_a, the braking power of the motor is calculated by

$$P = (I - I_0)\,(E - I\,R_a) \tag{45}$$

The measured pressures, the load cell thrust measurement, propeller speed and motor current were averaged through the duration of the test with exception to the sweeping pitot measurements.

Uncertainty

An uncertainty analysis was performed on each of the measured values. It was found that the bit resolution of the Omega pressure sensors is primary cause of the measured uncertainty. It was, at least, an order of magnitude greater than the random uncertainty, instrument specifications or propagated uncertainties from atmospheric pressure or temperature.

The load cell experienced an additional calibration uncertainty that could not be removed through subsequent calibrations. The measured points from the calibration and the

linear calibrated curve fit were as much as 0.044 lbf. apart. This quantity was assumed as a standard error and root mean squared with the bit resolution to find the total uncertainty of the load cell measurements.

The sources of uncertainty for the power calculations come from the bit resolution of current measurement, ammeter instrument specification and uncertainty of the motor constants. The current bit resolution and 1% of reading specification were root mean squared to total a current standard uncertainty. The variation of the no-load current and armature resistance are approximated to be $U_{I_0} = .05$ and $U_{R_a} = .03$ as found by multiple tests. The uncertainty of the power measurement was root mean squared as

$$U_P = \sqrt{\left(\frac{\partial P}{\partial I_0} U_{I_0}\right)^2 + \left(\frac{\partial P}{\partial I} U_I\right)^2 + \left(\frac{\partial P}{\partial R_a} U_{R_a}\right)^2} \tag{46}$$

The uncertainty of the measured voltage was insignificant compared to the other uncertainties.

The uncertainty of the thrust measurement calculated by the wake surveys was estimated using a Monte Carlo simulation. The measured velocity at each discrete location, the position biases and the upper integration limit of the wake integral were randomly varied in the simulation. All quantities are varied such that they have a normal distribution with a specified standard deviation. The standard deviation of the discrete velocities was calculated from the measurement. The position bias, used to center the wake, and the outward integration limit was varied such that they had a standard deviation of 0.25 in. The simulation was run 1000 times and the variation in the wake integral results were calculated.

Results and Discussion

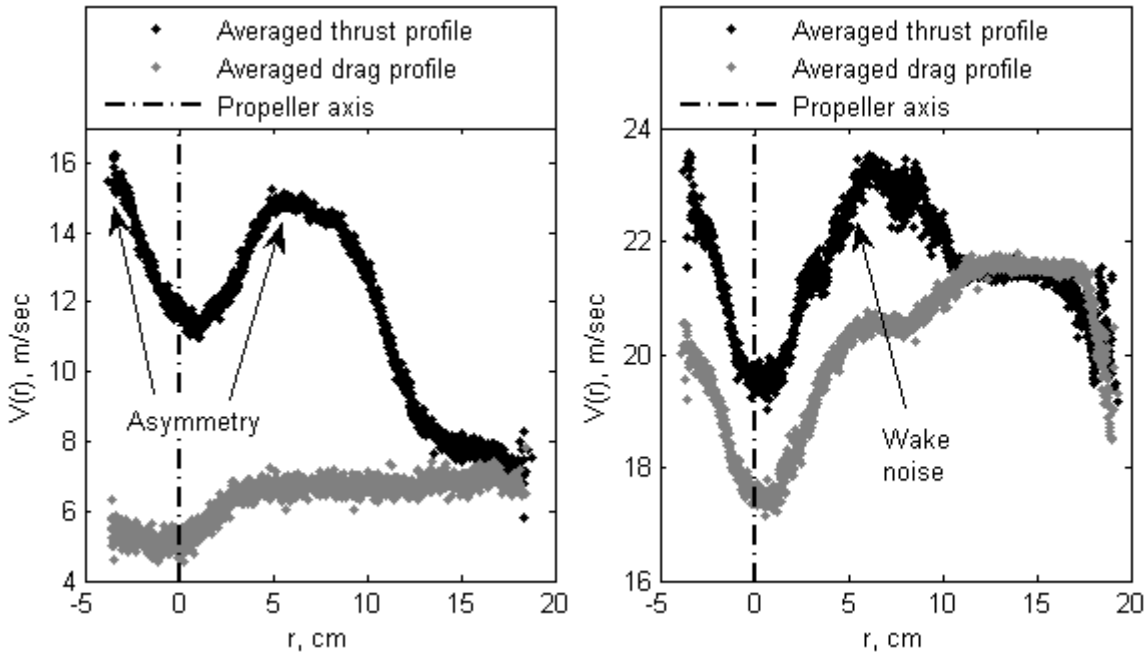
An APC 8x8 Thin Electric propeller was tested in the wind tunnel and compared to the BEM code calculations. The chord and pitch profile of the propeller was supplied by the manufacturer upon request. Airfoil geometry was not given therefore the aerodynamic lift and drag constants were chosen such that they sufficiently matched the load cell measurement data. These constants are in Table 2 and were used to characterize other propellers of the same manufacturer and series.

Figure 12 shows averaged wake profiles and are representative of the issues encountered. Each profile shows some form of off-centered asymmetry. Figure 12 (a) exhibits an additional asymmetry such that the velocity is higher on the negative side of the r axis than on the positive side. The source of this asymmetry is unknown and inconsistent with the symmetric wake assumption. Figure 12 (b) shows a noisy location in the wake. This was typical of each wake at higher tunnel velocities. Upon closer inspection, the measured wakes along individual pitot sweeps varied. This effect was unpredictable and localized to the 6 cm (2.5 in) to 10 cm (4 in) region. The uncertainty was also large, and therefore will not be considered in the remaining comparisons.

Figure 13 shows the thrust and power coefficients with 95% confidence interval error bars. These measurements are compared against the BEM models predictions. The test data matches the models reasonably well, especially at higher advance ratios. At low advance ratios, the thrust coefficient of the BEM overestimates the thrust coefficient. The reason

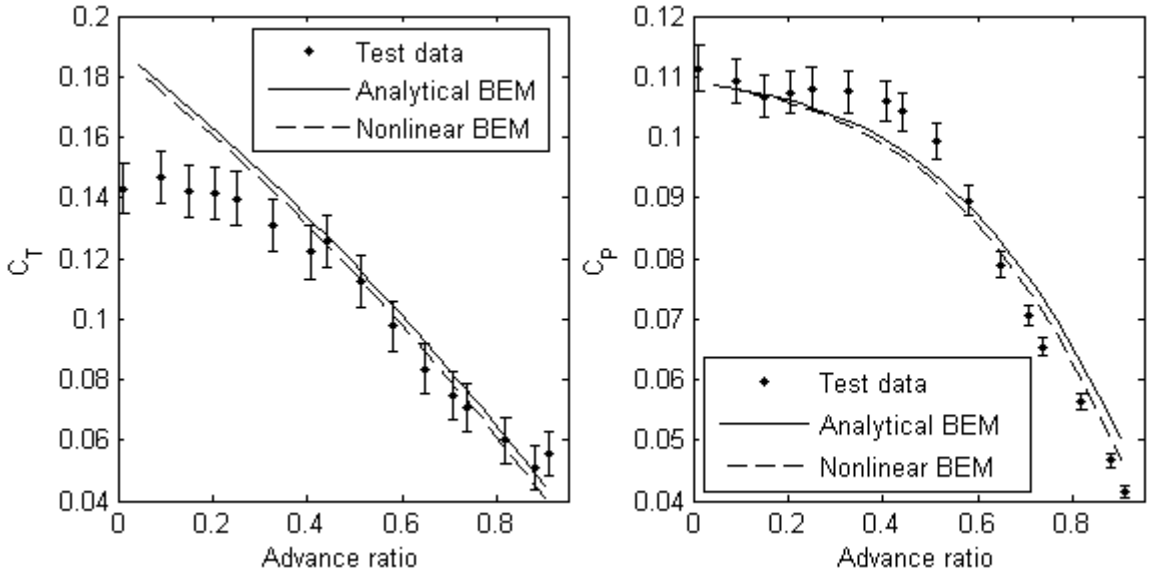
Table 2: Aerodynamic Constants Used in the BEM Code

Quantity	Value
$C_{L,\alpha}$	5.5
C_{L_0}	0.2
C_{D_0}	0.02
$C_{D,L}$	0.0
C_{D,L^2}	0.05



(a) Thrust coefficient (b) Power coefficient

Fig. 12: Averaged wake profiles highlighting sources of wake survey shortcomings.



(a) Thrust coefficient (b) Power coefficient

Fig. 13: Comparison of load cell derived thrust and power coefficients to BEM models (APC 8x8 Thin Electric propeller).

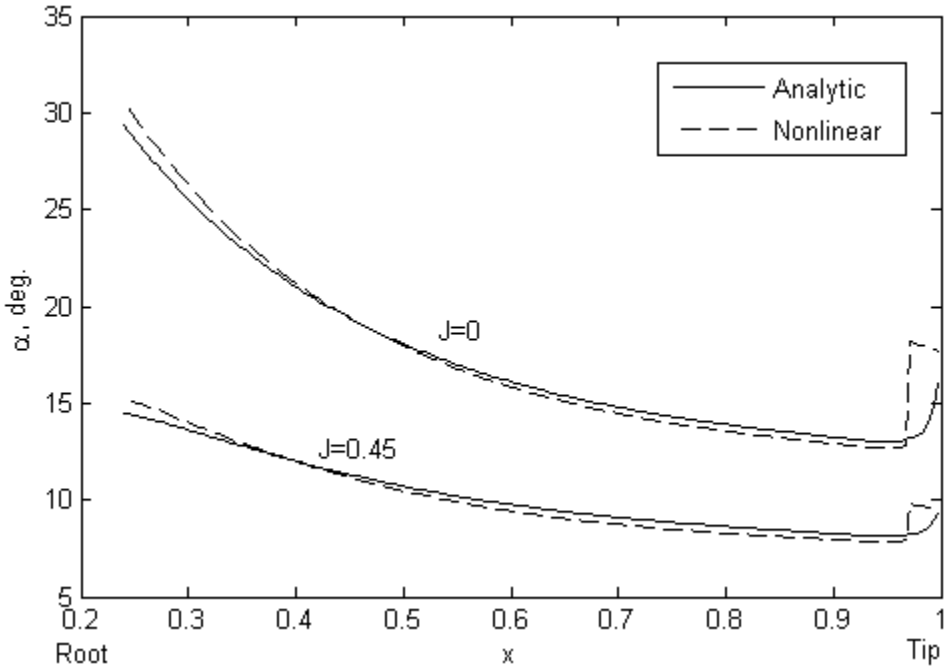
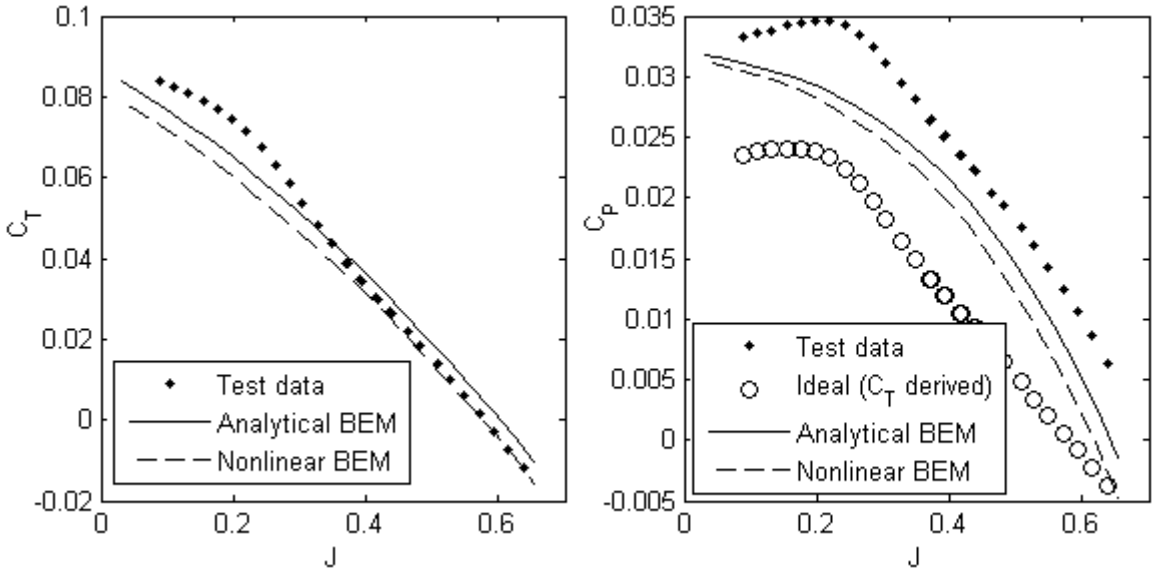


Fig. 14: Angle of attack at each blade element at advance ratios $J = 0$ and $J = 0.45$ (APC 8x8 Thin Electric propeller).

for this discrepancy is the linear lift model that is used in the BEM models. The propeller blade is stalled close to the root where the local pitch angle, β, is large. Figure 14 shows the BEM-calculated angle of attack at two different advance ratios. As the advance ratio approaches zero, the majority of the blade experiences an angle of attack that is likely to be out of its linear lift range. It is difficult to determine exactly how wide the linear lift range is without detailed airfoil information however, above 15° it is safely assumed that the blade is stalled. At the higher advance ratio the angle of attack is closer to the linear lift range for most of the blade span.

The BEM models were also compared to test data taken on APC propellers by Brandt and Selig [9] of the University of Illinois at Urbana-Champaign. Figure 15 compares the thrust an power coefficients from the UIUC database against the model predictions for the 11x5.5 propeller. For a lower pitch propeller the thrust coefficients show very good agreements, especially at the higher advance ratios. Clearly at the higher advance ratios where Φ is large the nonlinear BEM model exhibits greater accuracy. Interestingly at the



(a) Thrust coefficient

(b) Power coefficient

Fig. 15: Comparison of thrust and power coefficients (UIUC data) to BEM models (APC 11x5.5 Thin Electric propeller).

lower advance ratios, both BEM models under predict the thrust coefficient. This result is in direct contrast to the data presented in Figure 13 (a). For this case the blade pitch is significantly lower thus the blade remains unstalled for a significantly larger segment of the blade span. The UIUC power coefficient data appears to be biased when compared to the thrust coefficient data. Insufficient information is provided by the UIUC authors to understand the source of this bias. The power coefficient data was recalculated using Eq. 28 from the thrust coefficient. These recalculated coefficients are also plotted on Figure 15 (b). The recalculated data exhibit a closer relation to the nonlinear model.

Conclusion

For conceptual design, low-order engineering codes are still the preferred method for designers. Engineering codes are powerful tools when applied in the conceptual design stage. Their use in the early stages of design enables higher fidelity CFD calculations or wind tunnel tests to be performed on more mature vehicle concepts and can trim many months off the design process. Traditionally, one of the following low-order engineering design methods have been used to calculate the steady state flow properties behind rotating propellers and turbine blades. These are momentum theory, Goldstein's vortex theory and blade element theory. Often momentum theory is combined with blade element theory to produce a single low-order prediction tool referred to as the combined blade element momentum theory.

Goldstein's vortex method, although it has been successfully used to calculate the performance of propellers and wind turbines, is very sensitive to errors in the input geometry of the propeller, and as such is of limited use for the early stage design problems where the propeller characteristics may not be well known. This paper revisits the combined blade element momentum theory as an alternative. Unfortunately, the traditional analytical method used by McCormick to solve for the thrust and power coefficients using momentum theory, assumes that the advance angle of the blade is small for all radial locations and at all advance ratios. This assumption is clearly inaccurate. At even moderate advance ratios, the advance angle of the blade is significantly larger than can be allowed in the small angle approximation. The blade element thrust and power coefficients are numerically calculated without the inaccurate small advance angle assumption. The revised algorithm is derived and example calculation using a Java based graphical user interface are presented. The effect of this revised model is investigated and presented. As expected the modifications show the greatest effect at higher advance ratios. However, effects are shown to be greater for lower pitch propellers. McCormick assumes in the analytical model that the thrust is much greater than the drag which fails with lower pitch propellers at high advance ratios. The traditional

analytical model, thrust and power coefficient predictions are typically overestimated by 1-5% when compared to the revised model.

The analytical models are compared against experimental measurements collected for an APC 8x8 Thin Electric propeller blade, and against additional propeller data retrieved from a database at the the University of Illinois at Urbana-Champaign. Generally the models exhibit reasonable comparisons with the nonlinear (improved) BEM model showing the best accuracy at higher advance ratios. At lower advance ratios, the models predictions generally depart from the experimental measurements. This discrepancy is a result of poorly known lift coefficient distributions along the blade profiles. The models as are currently implemented rely on linear airfoil theory to predict the blade element lift coefficients, and for lower advance ratios the real blades are stalled along the lower sections of the blade near the root. Simple code modification to allow actual airfoil sectional lift coefficients to be input should greatly improve the accuracy of the model.

The greatest significance of the revised model is the elimination of the theoretically incorrect and philosophically unsatisfying assumptions used by McCormick to solve the blade element momentum equations. The resulting algorithm is easily implemented with only a small increase in complexity and computational time. The revised algorithm can easily be programmed on any computer using readily available spreadsheet analysis tools.

Momentum theory makes simplistic assumptions on the source of the induced flow. The induced flow is created by vorticity shed from the propeller. A point of future research would be to continue with blade element theory and account for vorticity in the flow. Goldstein's vortex theory approximates a vortices trailing from the whole span of the propeller creating a vortex sheet. A simplifying assumption would be that the propeller creates a single line vortex from the tip of the propeller. This is supported by propellers typically having high aspect ratios and minimal sweep creating a comparatively strong vortex off the tip of the blade greatly simplifying the helical vortex analysis.

References

[1] Goldstein, S., "On the Vortex Theory of Screw Propellers," *Porceeding of the Royal Society of London*, Vol. 123, The Royal Society, 1929, pp. 440–465.

[2] Kelly, Q. J., *Validation and Implementation of Goldstein's Vortex Theory of Screw Propellers*, Master's thesis, Utah State University, 2002.

[3] Tibery, C. L. and Jr., J. W. W., "Tables of the Goldstein Factor," Tech. Rep. 1534, David Taylor Model Basin, Washington, 1964.

[4] McCormick, B. W., *Aerodynamics, Aeronautics, and Flight Mechanics*, Wiley, Hoboken, NJ, 1994.

[5] Phillips, W. F., *Mechanics of Flight*, Wiley, Hoboken, NJ, 1st ed., 2004.

[6] Shen, W. Z., Mikkelsen, R., Sørensen, J. N., and Bak, C., "Tip Loss Corrections for Wind Turbine Computations," *Wind Energy*, Vol. 8, 2005, pp. 457–475.

[7] McCormick, B. W., *Aerodynamics of V/STOL Flight*, Dover Publications, Mineola, NY, 1999.

[8] Glauert, H., "Airplane Propellers," *Aerodynamic Theory Volume IV*, edited by W. F. Durand, Julius Springer, Berlin, 1935.

[9] Brandt, J. B. and Selig, M. S., "Propeller Performance Data at Low Reynolds Numbers," *49th AIAA Aerospace Sciences Meeting*, January 2011.

Appendix

Appendix A: Manufacturer Supplied Data Files for APC 8x8 and 11x5.5 Thin Electric Propellers

```
8x8  REV1     1/14/00        ELECTRIC GRAUPNER COMPETITION                12-11-

    ORIGINAL FILE DATE BEFORE SURFDOC MODIFICATION: 12/18/2008
PROPDATA
RPM       16000.0000
RPS (n)     266.6667
DIA (D)       8.0000
NRJPTS            31

 Airfoil section Cl & Cd coefficients ==> HAVE BEEN <== altered
 to reflect stalling effects using Beta Angle, Advance Ratio,
 and (empirical, P/D correlated) static thrust data as
 correlating parameters.

 ======== NORMALIZED PERFORMANCE DATA (PLUS V) =========

         DEFINITIONS:
         J=V/nD (advance ratio)
         Ct=T/(rho * n**2 * D**4) (thrust coef.)
         Cp=P/(rho * n**3 * D**5) (power coef.)
         Pe=Ct*J/Cp (efficiency)
         V  (model speed in MPH)
```

J	Ct	Cp	Pe	V
0.0500	0.0964	0.0703	0.0686	6.
0.0898	0.0982	0.0732	0.1205	11.
0.1296	0.1000	0.0762	0.1701	16.
0.1694	0.1016	0.0791	0.2175	21.
0.2092	0.1030	0.0820	0.2628	25.
0.2490	0.1041	0.0847	0.3061	30.
0.2888	0.1048	0.0871	0.3477	35.
0.3286	0.1052	0.0892	0.3876	40.
0.3684	0.1051	0.0909	0.4258	45.
0.4082	0.1046	0.0924	0.4623	49.
0.4480	0.1026	0.0924	0.4973	54.
0.4878	0.0994	0.0914	0.5308	59.
0.5276	0.0960	0.0899	0.5629	64.
0.5673	0.0921	0.0880	0.5937	69.
0.6071	0.0879	0.0856	0.6233	74.
0.6469	0.0833	0.0827	0.6517	78.
0.6867	0.0786	0.0796	0.6787	83.
0.7265	0.0738	0.0762	0.7042	88.
0.7663	0.0689	0.0726	0.7282	93.
0.8061	0.0639	0.0686	0.7509	98.
0.8459	0.0588	0.0644	0.7720	103.
0.8857	0.0536	0.0600	0.7912	107.
0.9255	0.0482	0.0552	0.8084	112.
0.9653	0.0427	0.0501	0.8236	117.
1.0051	0.0371	0.0446	0.8364	122.
1.0449	0.0314	0.0388	0.8450	127.
1.0847	0.0256	0.0327	0.8469	131.
1.1245	0.0196	0.0263	0.8388	136.
1.1643	0.0136	0.0195	0.8109	141.
1.2041	0.0074	0.0123	0.7235	146.
1.2439	0.0011	0.0051	0.2738	151.

```
 ========= GEOMETRY DATA ==============

    DEFINITIONS:
        THE QUOTED PITCH REFLECTS, IN GENERAL, ANGULAR MEASURE
        AS DEFINED WITH A FLAT BOTTOM SURFACE.  THIS WILL
        AGREE WITH A PRATHER GAGE MEASUREMENT OVER MOST OF THE
        EFFECTIVE PORTION OF THE BLADE.
        THE LE-TE MEASURE IS DEFINED IN TERMS OF LEADING EDGE
        AND TRAILING EDGE (MOLD) PARTING LINE DATUMS.
        THE PRATHER MEASURE REFLECTS THE MOST LIKELY PITCH
        INTERPRETATION FROM A PITCH MEASUREMENT DEVICE
        THAT RESTS AGAINST THE LOWER SURFACE.
        SWEEP IS DEFINED WITH L.E. POSITION.
```

STATION (IN)	CHORD (IN)	PITCH (QUOTED)	PITCH (LE-TE)	PITCH (PRATHER)	SWEEP (IN)	THICKNESS RATIO	TWIST (DEG)	MAX THICK (IN)
0.9536	0.7216	8.0000	8.0000	7.0552	0.3978	0.1794	53.1685	0.1294
1.0038	0.7503	8.0000	8.0000	7.1322	0.4039	0.1715	51.7475	0.1287
1.0541	0.7767	8.0000	8.0000	7.1983	0.4094	0.1642	50.3800	0.1275
1.1043	0.8009	8.0000	8.0000	7.2561	0.4146	0.1574	49.0644	0.1261
1.1545	0.8229	8.0000	8.0000	7.3077	0.4193	0.1513	47.7993	0.1245
1.2048	0.8429	8.0000	8.0000	7.3544	0.4235	0.1456	46.5828	0.1227
1.2602	0.8625	8.0000	8.0000	7.4011	0.4277	0.1400	45.2958	0.1208
1.3547	0.8904	8.0000	8.0000	7.4708	0.4337	0.1321	43.2237	0.1176
1.4544	0.9126	8.0000	8.0000	7.5312	0.4384	0.1258	41.1997	0.1148

1.5541	0.9278	8.0000	8.0000	7.5772	0.4417	0.1218	39.3264	0.1130
1.6538	0.9364	8.0000	8.0000	7.6038	0.4434	0.1199	37.5914	0.1123
1.7536	0.9390	8.0000	8.0000	7.6178	0.4437	0.1191	35.9830	0.1118
1.8533	0.9359	8.0000	8.0000	7.6290	0.4426	0.1182	34.4900	0.1106
1.9530	0.9275	8.0000	8.0000	7.6377	0.4402	0.1174	33.1022	0.1089
2.0527	0.9143	8.0000	8.0000	7.6449	0.4366	0.1165	31.8104	0.1065
2.1524	0.8968	8.0000	8.0000	7.6541	0.4317	0.1157	30.6061	0.1037
2.2521	0.8753	8.0000	8.0000	7.6647	0.4258	0.1148	29.4818	0.1005
2.3518	0.8503	8.0000	8.0000	7.6750	0.4187	0.1140	28.4304	0.0969
2.4515	0.8223	8.0000	8.0000	7.6839	0.4106	0.1131	27.4457	0.0930
2.5512	0.7917	8.0000	8.0000	7.6914	0.4015	0.1123	26.5222	0.0889
2.6510	0.7588	8.0000	8.0000	7.6937	0.3915	0.1114	25.6547	0.0846
2.7507	0.7242	8.0000	8.0000	7.6960	0.3807	0.1106	24.8386	0.0801
2.8504	0.6883	8.0000	8.0000	7.6990	0.3691	0.1097	24.0699	0.0755
2.9501	0.6515	8.0000	8.0000	7.7030	0.3567	0.1089	23.3447	0.0709
3.0498	0.6143	8.0000	8.0000	7.7082	0.3436	0.1080	22.6597	0.0664
3.1495	0.5771	8.0000	8.0000	7.7143	0.3299	0.1072	22.0117	0.0619
3.2492	0.5403	8.0000	8.0000	7.7218	0.3156	0.1063	21.3982	0.0575
3.3489	0.5043	8.0000	8.0000	7.7265	0.3009	0.1055	20.8164	0.0532
3.4487	0.4697	8.0000	8.0000	7.7244	0.2856	0.1046	20.2641	0.0492
3.5484	0.4368	8.0000	8.0000	7.7153	0.2700	0.1038	19.7392	0.0453
3.6481	0.4060	8.0000	8.0000	7.6972	0.2540	0.1030	19.2398	0.0418
3.7478	0.3779	8.0000	8.0000	7.6679	0.2377	0.1021	18.7642	0.0386
3.8466	0.3333	8.0000	8.0000	7.5050	0.2027	0.1013	18.3150	0.0338
3.9404	0.2245	8.0000	8.0000	7.4682	0.1029	0.1005	17.9070	0.0226

---- EFFICIENCY, POWER, TORQUE & THRUST DISTRIBUTION ----

MPH	EFF	POWER (Hp)	TORQUE (in-lbf)	THRUST (lbf)
6.061	0.6856E-01	0.7593	2.991	3.221
10.88	0.1205	0.7906	3.114	3.282
15.71	0.1701	0.8223	3.239	3.339
20.53	0.2175	0.8541	3.364	3.393
25.36	0.2628	0.8854	3.488	3.441
30.18	0.3061	0.9143	3.602	3.478
35.00	0.3477	0.9402	3.703	3.502
39.83	0.3876	0.9627	3.792	3.513
44.65	0.4258	0.9818	3.868	3.511
49.47	0.4623	0.9973	3.928	3.495
54.30	0.4973	0.9976	3.930	3.426
59.12	0.5308	0.9865	3.886	3.322
63.95	0.5629	0.9711	3.825	3.206
68.77	0.5937	0.9506	3.745	3.078
73.59	0.6233	0.9245	3.642	2.936
78.42	0.6517	0.8930	3.518	2.783
83.24	0.6787	0.8592	3.385	2.627
88.06	0.7042	0.8226	3.240	2.467
92.89	0.7282	0.7833	3.086	2.303
97.71	0.7509	0.7411	2.919	2.136
102.5	0.7720	0.6957	2.740	1.964
107.4	0.7912	0.6473	2.550	1.789
112.2	0.8084	0.5958	2.347	1.610
117.0	0.8236	0.5407	2.130	1.427
121.8	0.8364	0.4816	1.897	1.240
126.7	0.8450	0.4191	1.651	1.049
131.5	0.8469	0.3535	1.392	0.8539
136.3	0.8388	0.2841	1.119	0.6557
141.1	0.8109	0.2103	0.8286	0.4532
145.9	0.7235	0.1328	0.5231	0.2469
150.8	0.2738	0.5483E-01	0.2160	0.3734E-01

11x5.5 REV1 ELECTRIC GRAUPNER COMPETITION 12-11-99

ORIGINAL FILE DATE BEFORE SURFDOC MODIFICATION: 12/18/2008
PROPDATA
RPM 12000.0000
RPS (n) 200.0000
DIA (D) 11.0000
NRJPTS 17

Airfoil section Cl & Cd coefficients ==> HAVE BEEN <== altered
to reflect stalling effects using Beta Angle, Advance Ratio,
and (empirical, P/D correlated) static thrust data as
correlating parameters.

======== NORMALIZED PERFORMANCE DATA (PLUS V) =========

DEFINITIONS:
J=V/nD (advance ratio)
Ct=T/(rho * n**2 * D**4) (thrust coef.)
Cp=P/(rho * n**3 * D**5) (power coef.)
Pe=Ct*J/Cp (efficiency)
V (model speed in MPH)

J	Ct	Cp	Pe	V
0.0500	0.0735	0.0394	0.0932	6.

0.0898	0.0641	0.0351	0.1638	11.
0.1296	0.0551	0.0309	0.2311	16.
0.1694	0.0495	0.0284	0.2951	21.
0.2092	0.0465	0.0273	0.3557	26.
0.2490	0.0433	0.0261	0.4126	31.
0.2888	0.0401	0.0249	0.4656	36.
0.3286	0.0367	0.0234	0.5147	41.
0.3684	0.0332	0.0219	0.5594	46.
0.4082	0.0296	0.0201	0.5992	51.
0.4480	0.0258	0.0183	0.6325	56.
0.4878	0.0219	0.0163	0.6560	61.
0.5276	0.0179	0.0142	0.6669	66.
0.5673	0.0138	0.0119	0.6584	71.
0.6071	0.0096	0.0094	0.6159	76.
0.6469	0.0052	0.0068	0.4896	81.
0.6867	0.0006	0.0042	0.1035	86.

========= GEOMETRY DATA ==========

DEFINITIONS:
THE QUOTED PITCH REFLECTS, IN GENERAL, ANGULAR MEASURE AS DEFINED WITH A FLAT BOTTOM SURFACE. THIS WILL AGREE WITH A PRATHER GAGE MEASUREMENT OVER MOST OF THE EFFECTIVE PORTION OF THE BLADE.
THE LE-TE MEASURE IS DEFINED IN TERMS OF LEADING EDGE AND TRAILING EDGE (MOLD) PARTING LINE DATUMS.
THE PRATHER MEASURE REFLECTS THE MOST LIKELY PITCH INTERPRETATION FROM A PITCH MEASUREMENT DEVICE THAT RESTS AGAINST THE LOWER SURFACE.
SWEEP IS DEFINED WITH L.E. POSITION.

STATION (IN)	CHORD (IN)	PITCH (QUOTED)	PITCH (LE-TE)	PITCH (PRATHER)	SWEEP (IN)	THICKNESS RATIO	TWIST (DEG)	MAX THICK (IN)
0.9632	0.7941	5.5000	5.5000	4.2163	0.3922	0.2822	42.2645	0.2241
1.0234	0.8260	5.5000	5.5000	4.3014	0.3999	0.2662	40.5404	0.2199
1.0837	0.8555	5.5000	5.5000	4.3833	0.4072	0.2510	38.9296	0.2148
1.1439	0.8825	5.5000	5.5000	4.4616	0.4139	0.2369	37.4236	0.2090
1.2042	0.9070	5.5000	5.5000	4.5348	0.4202	0.2236	36.0144	0.2028
1.2644	0.9293	5.5000	5.5000	4.6028	0.4260	0.2112	34.6945	0.1963
1.3247	0.9491	5.5000	5.5000	4.6674	0.4314	0.1998	33.4570	0.1896
1.3851	0.9668	5.5000	5.5000	4.7287	0.4363	0.1893	32.2925	0.1830
1.4844	0.9927	5.5000	5.5000	4.8210	0.4433	0.1740	30.5276	0.1727
1.6033	1.0185	5.5000	5.5000	4.9146	0.4502	0.1590	28.6327	0.1619
1.7223	1.0389	5.5000	5.5000	4.9868	0.4554	0.1476	26.9413	0.1533
1.8413	1.0544	5.5000	5.5000	5.0342	0.4591	0.1398	25.4260	0.1474
1.9603	1.0649	5.5000	5.5000	5.0509	0.4611	0.1356	24.0623	0.1444
2.0793	1.0709	5.5000	5.5000	5.0466	0.4618	0.1342	22.8300	0.1437
2.1983	1.0725	5.5000	5.5000	5.0408	0.4609	0.1330	21.7119	0.1427
2.3173	1.0700	5.5000	5.5000	5.0374	0.4587	0.1318	20.6936	0.1410
2.4363	1.0636	5.5000	5.5000	5.0357	0.4553	0.1306	19.7629	0.1389
2.5553	1.0536	5.5000	5.5000	5.0356	0.4505	0.1294	18.9094	0.1364
2.6743	1.0401	5.5000	5.5000	5.0367	0.4446	0.1283	18.1241	0.1334
2.7933	1.0236	5.5000	5.5000	5.0394	0.4375	0.1271	17.3995	0.1301
2.9123	1.0041	5.5000	5.5000	5.0437	0.4294	0.1259	16.7291	0.1264
3.0313	0.9820	5.5000	5.5000	5.0494	0.4203	0.1247	16.1070	0.1224
3.1503	0.9575	5.5000	5.5000	5.0562	0.4102	0.1235	15.5285	0.1182
3.2693	0.9308	5.5000	5.5000	5.0640	0.3992	0.1223	14.9892	0.1138
3.3883	0.9022	5.5000	5.5000	5.0723	0.3874	0.1211	14.4853	0.1093
3.5073	0.8718	5.5000	5.5000	5.0794	0.3748	0.1199	14.0135	0.1046
3.6263	0.8401	5.5000	5.5000	5.0783	0.3614	0.1187	13.5709	0.0997
3.7453	0.8071	5.5000	5.5000	5.0749	0.3474	0.1175	13.1549	0.0949
3.8643	0.7732	5.5000	5.5000	5.0716	0.3329	0.1164	12.7633	0.0900
3.9833	0.7385	5.5000	5.5000	5.0683	0.3177	0.1152	12.3940	0.0851
4.1023	0.7034	5.5000	5.5000	5.0653	0.3021	0.1140	12.0451	0.0802
4.2213	0.6680	5.5000	5.5000	5.0625	0.2861	0.1128	11.7150	0.0753
4.3403	0.6326	5.5000	5.5000	5.0592	0.2696	0.1116	11.4024	0.0706
4.4593	0.5975	5.5000	5.5000	5.0557	0.2529	0.1104	11.1058	0.0660
4.5783	0.5629	5.5000	5.5000	5.0528	0.2360	0.1092	10.8240	0.0615
4.6973	0.5291	5.5000	5.5000	5.0501	0.2188	0.1080	10.5560	0.0572
4.8163	0.4962	5.5000	5.5000	5.0452	0.2015	0.1068	10.3009	0.0530
4.9353	0.4645	5.5000	5.5000	5.0252	0.1841	0.1056	10.0576	0.0491
5.0543	0.4343	5.5000	5.5000	4.9872	0.1667	0.1045	9.8255	0.0454
5.1733	0.4058	5.5000	5.5000	4.9281	0.1494	0.1033	9.6037	0.0419
5.2920	0.3723	5.5000	5.5000	4.6965	0.1253	0.1021	9.3922	0.0380
5.4081	0.2791	5.5000	5.5000	4.4124	0.0401	0.1009	9.1942	0.0282

---- EFFICIENCY, POWER, TORQUE & THRUST DISTRIBUTION ----

MPH	EFF	POWER (Hp)	TORQUE (in-lbf)	THRUST (lbf)
6.250	0.9321E-01	0.8822	4.634	4.934
11.22	0.1638	0.7865	4.131	4.303
16.20	0.2311	0.6918	3.633	3.701
21.17	0.2951	0.6357	3.339	3.323
26.15	0.3557	0.6118	3.213	3.121
31.12	0.4126	0.5853	3.074	2.910

36.10	0.4656	0.5563	2.922	2.691
41.07	0.5147	0.5244	2.754	2.464
46.05	0.5594	0.4894	2.570	2.229
51.02	0.5992	0.4509	2.368	1.986
55.99	0.6325	0.4094	2.150	1.734
60.97	0.6560	0.3652	1.918	1.474
65.94	0.6669	0.3178	1.669	1.205
70.92	0.6584	0.2665	1.400	0.9280
75.89	0.6159	0.2108	1.107	0.6415
80.87	0.4896	0.1526	0.8013	0.3464
85.84	0.1035	0.9448E-01	0.4962	0.4273E-01

Appendix B: Analysis Code Written in Java

An attempt has been made to include only the portions of code applicable to the blade element momentum model and not to the portions pertaining to the graphical user interface.

Main class

```
private void calcButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Create settings object and check for empty fields
    try {
        settings = new Settings();
        initSettings();
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(null, "Please make sure all fields are entered correctly");
        return;
    }
    // get results type from dropbox
    String resType;
    resType = resultsCombo.getSelectedItem().toString();

    // determine with method to use based off of the results type
    if (resType.equals("Operating conditions")) {
        calcOpCond();
    }
    if (resType.equals("RPM sweep")) {
        calcRpmSweep();
    }
    if (resType.equals("Velocity sweep")) {
        calcVinfSweep();
    }
    if (resType.equals("Angle of attack")) {
        calcAoa();
    }
    if (resType.equals("Velocity profile")) {
        calcVelProf();
    }
    if (resType.equals("Thrust distribution")) {
        calcTDistb();
    }

}
private void initSettings() {
    // initialize settings object to store all parameters
    settings.diameter = Double.parseDouble(textDiameter.getText()) / 12.0;
    settings.radius = settings.diameter / 2.0;
    if (!userDefPitch) {
        settings.pitch = Double.parseDouble(textPitch.getText()) / 12.0;
    }
    settings.cd0 = Double.parseDouble(textCd0.getText());
    settings.cd1 = Double.parseDouble(textCd1.getText());
    settings.cd2 = Double.parseDouble(textCd2.getText());
    settings.cl0 = Double.parseDouble(textCl0.getText());
    settings.cla = Double.parseDouble(textCla.getText());
    double RPM = Double.parseDouble(textRPM.getText());
    settings.omega = RPM / 60.0 * 2.0 * Math.PI;
    settings.vInf = Double.parseDouble(textVInf.getText());

    double T = Double.parseDouble(textTemp.getText()) + 459.67; // Rankine
    double P = Double.parseDouble(textPress.getText()) * .4911542 * 144; // lbs/ft^2
    settings.rho = P / (1716 * T); // slug/ft^3
    settings.tipCorr = Double.parseDouble(textTipCorr.getText());
    settings.numOfElem = Integer.parseInt(textNumOfElem.getText());
    settings.lambda = settings.vInf / (settings.omega * settings.radius);
    settings.thetaR = settings.omega * settings.radius;
    settings.numOfBlades = Integer.parseInt(textNumOfBlades.getText());

}

public Settings getSettings() {
    return (settings);
}

private void calcOpCond() {
    // Calculate performance at RPM and Vinf
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);
    Element[] elems = new Element[num];
    Element tempElem;
    double locR = rRoot + intv / 2.0;
```

```
    // Initialize each element
    for (int i = 0; i <= num - 1; i++) {
        tempElem = new Element(locR, this);
        elems[i] = tempElem;
        locR += intv;
    }
    double ct = 0.0;
    double cp = 0.0;

    // Integrate (sum) along the blade
    for (int i = 0; i <= num - 1; i++) {
        tempElem = elems[i];
        ct += tempElem.getDCt_dx() * intv / settings.radius;
        cp += tempElem.getDCp_dx() * intv / settings.radius;
    }
    //System.out.println("CT = " + Double.toString(ct));
    //System.out.println("CP = " + Double.toString(cp));

    double nonLinCt = 0.0;
    double nonLinCp = 0.0;
    for (int i = 0; i <= num - 1; i++) {
        tempElem = elems[i];
        nonLinCt += tempElem.getNonLinDCt_dx() * intv / settings.radius;
        nonLinCp += tempElem.getNonLinDCp_dx() * intv / settings.radius;
    }

    //System.out.println("NLCT = " + Double.toString(nonLinCt));
    //System.out.println("NLCP = " + Double.toString(nonLinCp));

    // Calculate thrust and power from coefficients
    double T = ct * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 4.0) * 8.0
            * Math.pow(settings.omega, 2.0) / Math.pow(Math.PI, 2.0);
    double P = cp * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 5.0) * 8.0
            * Math.pow(settings.omega, 3.0) / Math.pow(Math.PI, 3.0);
    //System.out.println("T = " + Double.toString(T) + " lbf");
    //System.out.println("P = " + Double.toString(P) + " Watts");

    // Create results frame
    ResultsFrame res = new ResultsFrame();
    double J = settings.vInf / (settings.omega / (2.0 * Math.PI) * settings.diameter);
    Double[] results = {ct, cp, nonLinCt, nonLinCp, T, P, J};
    res.printResults(results);
}

private void calcRpmSweep() {
    // Calculate performance at multiple RPMs
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);
    double[][] results = new double[20][];
    double dOmega = settings.omega / 20;

    // Loop through RPM and calculate performance
    for (int j = 1; j <= 20; j++) {
        Element[] elems = new Element[num];
        Element tempElem;
        settings.omega = j * dOmega;
        settings.lambda = settings.vInf / (settings.omega * settings.radius);
        settings.thetaR = settings.omega * settings.radius;
        double locR = rRoot + intv / 2.0;
        // Initialize each element
        for (int i = 0; i <= num - 1; i++) {
            tempElem = new Element(locR, this);
            elems[i] = tempElem;
            locR += intv;
        }
        double ct = 0.0;
        double cp = 0.0;

        // Integrate (sum) along the blade
        for (int i = 0; i <= num - 1; i++) {
            tempElem = elems[i];
            ct += tempElem.getDCt_dx() * intv / settings.radius;
            cp += tempElem.getDCp_dx() * intv / settings.radius;
        }
        //System.out.println("CT = " + Double.toString(ct));
        //System.out.println("CP = " + Double.toString(cp));

        double nonLinCt = 0.0;
        double nonLinCp = 0.0;
        for (int i = 0; i <= num - 1; i++) {
            tempElem = elems[i];
            nonLinCt += tempElem.getNonLinDCt_dx() * intv / settings.radius;
            nonLinCp += tempElem.getNonLinDCp_dx() * intv / settings.radius;
        }
```

```
        //System.out.println("NLCT = " + Double.toString(nonLinCt));
        //System.out.println("NLCP = " + Double.toString(nonLinCp));

        double T = ct * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 4.0) * 8.0
                * Math.pow(settings.omega, 2.0) / Math.pow(Math.PI, 2.0);
        double P = cp * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 5.0) * 8.0
                * Math.pow(settings.omega, 3.0) / Math.pow(Math.PI, 3.0);
        double nlT = nonLinCt * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 4.0)
                * 8.0 * Math.pow(settings.omega, 2.0) / Math.pow(Math.PI, 2.0);
        double nlP = nonLinCp * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 5.0)
                * 8.0 * Math.pow(settings.omega, 3.0) / Math.pow(Math.PI, 3.0);
        //System.out.println("T = " + Double.toString(T) + " lbf");
        //System.out.println("P = " + Double.toString(P) + " Watts");

        double J = settings.vInf / (settings.omega / (2.0 * Math.PI) * settings.diameter);
        double[] temp = {j * dOmega * 60 / (2.0 * Math.PI), ct, cp, nonLinCt, nonLinCp, T, P, nlT, nlP};
        System.out.println(temp);
        results[j - 1] = temp;
    }

    // Create results frame
    String[] labels = {"RPM", "Ct", "Cp", "Nonlinear Ct", "Nonlinear Cp",
                "Thrust", "Power", "Nonlinear Thrust", "Nonlinear Power"};
    multiResultsFrame res = new multiResultsFrame();
    res.printResults(results, labels);
}

private void calcVinfSweep() {
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);

    double[][] results = new double[20][];
    double dvInf = settings.vInf / 20;

    for (int j = 1; j <= 20; j++) {

        settings.vInf = j * dvInf;
        settings.lambda = settings.vInf / (settings.omega * settings.radius);

        Element[] elems = new Element[num];
        Element tempElem;

        double locR = rRoot + intv / 2.0;
        // Initialize each element
        for (int i = 0; i <= num - 1; i++) {
            tempElem = new Element(locR, this);
            elems[i] = tempElem;
            locR += intv;
        }
        double ct = 0.0;
        double cp = 0.0;

        // Integrate (sum) along the blade
        for (int i = 0; i <= num - 1; i++) {
            tempElem = elems[i];
            ct += tempElem.getDCt_dx() * intv / settings.radius;
            cp += tempElem.getDCp_dx() * intv / settings.radius;
        }
        //System.out.println("CT = " + Double.toString(ct));
        //System.out.println("CP = " + Double.toString(cp));

        double nonLinCt = 0.0;
        double nonLinCp = 0.0;
        for (int i = 0; i <= num - 1; i++) {
            tempElem = elems[i];
            nonLinCt += tempElem.getNonLinDCt_dx() * intv / settings.radius;
            nonLinCp += tempElem.getNonLinDCp_dx() * intv / settings.radius;
        }

        //System.out.println("NLCT = " + Double.toString(nonLinCt));
        //System.out.println("NLCP = " + Double.toString(nonLinCp));

        double T = ct * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 4.0) * 8.0
                * Math.pow(settings.omega, 2.0) / Math.pow(Math.PI, 2.0);
        double P = cp * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 5.0) * 8.0
                * Math.pow(settings.omega, 3.0) / Math.pow(Math.PI, 3.0);
        double nlT = nonLinCt * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 4.0)
                * 8.0 * Math.pow(settings.omega, 2.0) / Math.pow(Math.PI, 2.0);
        double nlP = nonLinCp * 1.0 / 2.0 * settings.rho * Math.pow(rTip, 5.0)
                * 8.0 * Math.pow(settings.omega, 3.0) / Math.pow(Math.PI, 3.0);
        //System.out.println("T = " + Double.toString(T) + " lbf");
        //System.out.println("P = " + Double.toString(P) + " Watts");

        double J = settings.vInf / (settings.omega / (2.0 * Math.PI) * settings.diameter);
        double[] temp = {J, ct, cp, nonLinCt, nonLinCp, T, P, nlT, nlP};
```

```
        results[j - 1] = temp;
    }
    String[] labels = {"J", "Ct", "Cp", "Nonlinear Ct", "Nonlinear Cp",
                "Thrust", "Power", "Nonlinear Thrust", "Nonlinear Power"};
    multiResultsFrame res = new multiResultsFrame();
    res.printResults(results, labels);
    System.out.println(results[1].length);
    System.out.println(results.length);
}

private void calcAoa() {
    // Calculate the angle of attack all along the blade
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);

    double[][] results = new double[num][];
    Element[] elems = new Element[num];
    Element tempElem;
    double c = 180.0 / Math.PI;
    double locR = rRoot + intv / 2.0;

    // Initialize each element
    for (int i = 0; i <= num - 1; i++) {
        tempElem = new Element(locR, this);
        elems[i] = tempElem;

        double[] temp = {locR, locR / settings.radius, tempElem.getAoa()
                * c, tempElem.getNonLinAoa() * c, tempElem.getAlphaI()
                * c, tempElem.getNonLinAlphaI() * c};

        results[i] = temp;
        locR += intv;

    }
    // Create results frame
    String[] labels = {"r", "x", "Linear aoa", "Nonlinear aoa",
                "Linear induced aoa", "Nonlinear induced aoa"};
    multiResultsFrame res = new multiResultsFrame();
    res.printResults(results, labels);

}

private void calcVelProf() {
    // Calculate velocity profile created at the blade, Vinf + Vi
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);

    double[][] results = new double[num][];
    Element[] elems = new Element[num];
    Element tempElem;
    double c = 180.0 / Math.PI;
    double locR = rRoot + intv / 2.0;
    // Initialize each element
    for (int i = 0; i <= num - 1; i++) {
        tempElem = new Element(locR, this);
        elems[i] = tempElem;

        double[] temp = {locR, locR / settings.radius, settings.vInf
                + tempElem.getInducedVel(), settings.vInf + tempElem.getNonLinInducedVel()};

        results[i] = temp;
        locR += intv;

    }

    String[] labels = {"r", "x", "Linear velocity", "Nonlinear velocity"};
    multiResultsFrame res = new multiResultsFrame();
    res.printResults(results, labels);

}

private void calcTDistb() {
    // Calculate the thrust distribution along the blade
    int num = settings.numOfElem;
    double rRoot = chord.chordData[0][0];
    double rTip = settings.radius;
    double intv = (rTip - rRoot) / ((double) num);

    double[][] results = new double[num][];
    Element[] elems = new Element[num];
    Element tempElem;
    double c = 180.0 / Math.PI;
    double locR = rRoot + intv / 2.0;
    // Initialize each element
```

```
        for (int i = 0; i <= num - 1; i++) {
            tempElem = new Element(locR, this);
            elems[i] = tempElem;

            double[] temp = {locR, locR / settings.radius, tempElem.getDCt_dx()
                    * intv / settings.radius, tempElem.getNonLinDCt_dx() * intv / settings.radius};

            results[i] = temp;
            locR += intv;

        }
        // Create results frame
        String[] labels = {"r", "x", "Linear Ct", "Nonlinear Ct"};
        multiResultsFrame res = new multiResultsFrame();
        res.printResults(results, labels);

    }
```

Settings class

```
public class Settings {

    public double diameter;
    public double radius;
    public double thetaR;
    public double pitch;
    public double cl0;
    public double cd0;
    public double cd1;
    public double cd2;
    public double cla;
    public double omega;
    public double vInf;
    public double rho;
    public double tipCorr;
    public int numOfElem;
    public double lambda;
    public int numOfBlades;
    public double eps;
    public double aspRatio;
    public double oswEffFactor;
}
```

Chord class

```
public class Chord {

    public double[][] chordData;
    public double[][] pitchData;
    private JProps jProps;

    public void openAndParse(File file) {
        try {
            // Open file
            BufferedReader bReader = new BufferedReader(new FileReader(file));
            try {
                /*
                 * Declare a temporary string buffer and a temporary dynamic
                 * array to store the data in a string format
                 */
                String buffString;
                ArrayList stringChordData = new ArrayList();
                // Build array
                while ((buffString = bReader.readLine()) != null) {
                    stringChordData.add(buffString);
                }
                // Close file
                bReader.close();

                double area = 0.0;

                String line = (String) stringChordData.get(0);
                int dims = line.split("\t").length;
                // Put string data into a workable double array

                if (dims == 2) {
                    chordData = new double[stringChordData.size()][2];
                    jProps.userDefPitch = false;
                    for (int i = 0; i < stringChordData.size(); i++) {
                        line = (String) stringChordData.get(i);
```

```
                    String[] arrLine = line.split("\t");
                    chordData[i][0] = Double.parseDouble(arrLine[0]) / 12.0;
                    chordData[i][1] = Double.parseDouble(arrLine[1]) / 12.0;
                }
            } else if (dims == 3) {
                chordData = new double[stringChordData.size()][2];
                pitchData = new double[stringChordData.size()][2];
                jProps.userDefPitch = true;
                for (int i = 0; i < stringChordData.size(); i++) {
                    line = (String) stringChordData.get(i);
                    String[] arrLine = line.split("\t");
                    chordData[i][0] = Double.parseDouble(arrLine[0]) / 12.0;
                    chordData[i][1] = Double.parseDouble(arrLine[1]) / 12.0;
                    pitchData[i][0] = Double.parseDouble(arrLine[0]) / 12.0;
                    pitchData[i][1] = Double.parseDouble(arrLine[2]);

                }
            }

        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    } catch (FileNotFoundException ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}

/**
 * Method to find the local chord length
 * @param radius
 * @return
 */
public double findChord(double radius) {
    // Default value of chord length in
    double chord = 0.0;

    // Iterate to find where the radius is from the given chord profile info
    for (int i = 0; i < chordData.length - 1; i++) {

        if (radius > chordData[i][0] && radius <= chordData[i + 1][0]) {
            // Linear interpolation between the known chord points
            chord = chordData[i][1] + (chordData[i + 1][1] - chordData[i][1])
                    / (chordData[i + 1][0] - chordData[i][0])
                    * (radius - chordData[i][0]);
            break;
        }
    }
    return chord;
}

public double findPitch(double radius) {
    // Default value of chord length in
    double pitch = 0.0;

    // Iterate to find where the radius is from the given chord profile info
    for (int i = 0; i < pitchData.length - 1; i++) {

        if (radius > pitchData[i][0] && radius <= pitchData[i + 1][0]) {
            // Linear interpolation between the known chord points
            pitch = pitchData[i][1] + (pitchData[i + 1][1] - pitchData[i][1])
                    / (pitchData[i + 1][0] - pitchData[i][0])
                    * (radius - pitchData[i][0]);
            break;
        }
    }
    return pitch;
}

/**
 * Method to find the aspect ratio of the propeller blade
 * @param jProps
 */
public void initAspRatio(JProps jProps) {
    double area = 0.0;
    for (int i = 0; i < chordData.length - 1; i++) {
        area = (chordData[i + 1][0] - chordData[i][0]) * (chordData[i + 1][1]
                + chordData[i][1]) / 2.0;
    }

    jProps.settings.aspRatio = Math.pow(jProps.settings.radius, 2.0) / area;
}

public Chord(JProps temp) {
    jProps = temp;
}
}
```

Element class

```
public class Element {

    private double locR;
    private double beta;
    private double x;
    private double thetaT;
    private double sigma;
    private double phi;
    private double chord;
    private double alphaI;
    private double cl;
    private double cd;
    private double dCt_dx;
    private double dCp_dx;
    private double nonLinDCt_dx;
    private double nonLinDCp_dx;
    private double nonLinAlphaI;
    private double nonLinCl;
    private double nonLinCd;
    /**
     * Settings from JProps
     */
    private Settings settings;

    Element(double rad, JProps jProps) {
        locR = rad;
        settings = jProps.settings;
        chord = jProps.chord.findChord(locR);
        findBeta(jProps);
        nonDimen();
        findLinInducedAOA();
        findLandD();
        findDCt_dx();
        findDcp_dx();

        findNonLinDCt_dxDCp_dx();
    }

    private void findBeta(JProps jProps) {
        // Find geometric pitch
        if (jProps.userDefPitch) {
            beta = jProps.chord.findPitch(locR);
        } else {
            beta = Math.atan2(settings.pitch, (2.0 * Math.PI * locR));
        }

    }

    private void nonDimen() {
        // Calculate nondimensional parameters
        x = locR / (settings.radius);
        thetaT = Math.sqrt(Math.pow(settings.lambda, 2.) + Math.pow(x, 2.)) * settings.thetaR;

        sigma = (settings.numOfBlades * chord) / (Math.PI * settings.radius);
        phi = Math.atan2(settings.lambda, x);

    }

    private void findLinInducedAOA() {
        // Find the linear induced angle of attack
        double a = settings.lambda / x + sigma * settings.cla / (8.0 * Math.pow(x, 2.0)) * thetaT / settings.thetaR;
        double b = sigma * settings.cla / (2.0 * Math.pow(x, 2.0)) * thetaT / settings.thetaR * (beta - phi);
        alphaI = 1.0 / 2.0 * (-a + Math.sqrt(Math.pow(a, 2.0) + b));

    }

    private void findLandD() {
        // Find lift and drag
        cl = settings.cl0 + settings.cla * (beta - (alphaI + phi));
        cd = settings.cd0 + settings.cd1 * cl + settings.cd2 * Math.pow(cl, 2.0);
    }

    private void findDCt_dx() {
        // Find differential thrust
        if (x < settings.tipCorr) {
            dCt_dx = Math.pow(Math.PI, 3.0) / 8.0 * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                    * (cl * Math.cos(phi + alphaI) - cd * Math.sin(phi + alphaI)) * sigma;

        } else {
            dCt_dx = Math.pow(Math.PI, 3.0) / 8.0 * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                    * (-settings.cd0 * Math.sin(phi + alphaI)) * sigma;
        }
    }

    public double getDCt_dx() {
        return dCt_dx;
```

```
}

private void findDcp_dx() {
    // find differential power
    if (x < settings.tipCorr) {
        dCp_dx = Math.pow(Math.PI, 4.0) / 8.0 * x * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                * (cd * Math.cos(phi + alphaI) + cl * Math.sin(phi + alphaI)) * sigma;

    } else {
        dCp_dx = Math.pow(Math.PI, 4.0) / 8.0 * x * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                * (settings.cd0 * Math.cos(phi + alphaI)) * sigma;
    }
}

public double getDCp_dx() {
    return dCp_dx;
}

private double[] findLandD(double ai) {
    // find lift and drag of nonlinear induced angle of attack
    double cl_ = settings.cl0 + settings.cla * (beta - (ai + phi));
    double cd_ = settings.cd0 + settings.cd1 * cl_ + settings.cd2 * Math.pow(cl_, 2.0);
    double[] data = {cl_, cd_};
    return data;

}

private void findNonLinDCt_dxDCp_dx() {
    // find nonlinear differential thrust and power
    double err = 1.0;
    double iterAlphaI1 = alphaI;
    double iterAlphaI2 = alphaI;
    double dLambda;
    double iterDCt_dx = dCt_dx;
    if (x > settings.tipCorr) {
        iterDCt_dx = 0.0;
    }

    //double iterDCt_dx = dCt_dx;
    double[] data;
    double iterCl = cl;
    double iterCd = cd;
    int i;

    for (i = 0; i <= 50 && (err > .000000001); i++) {
        // Errors are produced in the dLambda calculation when dCt_Dx is less than zero.
        // The only time this is zero is when x is greater than the tip correction
        dLambda = -settings.lambda / 2.0 + Math.sqrt(Math.pow(settings.lambda / 2.0, 2.0)
                + 1.0 / (Math.pow(Math.PI, 3.0) * x) * iterDCt_dx);
        iterAlphaI2 = Math.atan2(dLambda, Math.sqrt(Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0)));

        data = findLandD(iterAlphaI2);
        iterCl = data[0];
        iterCd = data[1];
        if (x < settings.tipCorr) {
            iterDCt_dx = Math.pow(Math.PI, 3.0) / 8.0 * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                    * (iterCl * Math.cos(phi + iterAlphaI2) - iterCd * Math.sin(phi + iterAlphaI2)) * sigma;

        } else {
            iterDCt_dx = Math.pow(Math.PI, 3.0) / 8.0 * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                    * (-settings.cd0 * Math.sin(phi + iterAlphaI2)) * sigma;
        }
        err = Math.abs(iterAlphaI2 - iterAlphaI1);
        iterAlphaI1 = iterAlphaI2;
    }

    nonLinDCt_dx = iterDCt_dx;

    if (x < settings.tipCorr) {
        nonLinDCp_dx = Math.pow(Math.PI, 4.0) / 8.0 * x * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                * (iterCd * Math.cos(phi + iterAlphaI1) + iterCl * Math.sin(phi + iterAlphaI1)) * sigma;

    } else {
        nonLinDCp_dx = Math.pow(Math.PI, 4.0) / 8.0 * x * (Math.pow(settings.lambda, 2.0) + Math.pow(x, 2.0))
                * (settings.cd0 * Math.cos(phi + iterAlphaI1)) * sigma;
    }

    nonLinAlphaI = iterAlphaI1;

    nonLinCl = iterCl;
    nonLinCd = iterCd;

}

double getNonLinDCt_dx() {
    return nonLinDCt_dx;
}
```

```
    double getNonLinDCp_dx() {
        return nonLinDCp_dx;
    }

    double getAlphaI() {
        return alphaI;
    }

    double getNonLinAlphaI() {
        return nonLinAlphaI;
    }

    double getAoa() {
        return beta - (phi + alphaI);
    }

    double getNonLinAoa() {
        return beta - (phi + nonLinAlphaI);
    }

    double getInducedVel() {
        double dLambda = (-settings.lambda / 2.0 + Math.sqrt(Math.pow(settings.lambda / 2.0, 2.0)
                + 1.0 / (Math.pow(Math.PI, 3.0) * x) * dCt_dx));
        return (dLambda * settings.omega * settings.radius);
    }

    double getNonLinInducedVel() {
        double dLambda = (-settings.lambda / 2.0 + Math.sqrt(Math.pow(settings.lambda / 2.0, 2.0)
                + 1.0 / (Math.pow(Math.PI, 3.0) * x) * nonLinDCt_dx));
        return (dLambda * settings.omega * settings.radius);
    }
}
```

Results frame class

```
    public void printResults(Double[] results) {
        String stringResults = "Ct = \t" + Double.toString(results[0]) + "\n";
        this.textResults.append(stringResults );
        stringResults = "Cp = \t" + Double.toString(results[1]) + "\n";
        this.textResults.append(stringResults);
        stringResults = "Nonlinear Ct = \t" + Double.toString(results[2]) + "\n";
        this.textResults.append(stringResults);
        stringResults = "Nonlinear Cp = \t" + Double.toString(results[3]) + "\n";
        this.textResults.append(stringResults);
        stringResults = "Thrust = \t" + Double.toString(results[4]) + "\n";
        this.textResults.append(stringResults);
        stringResults = "Power = \t" + Double.toString(results[5]) + "\n";
        this.textResults.append(stringResults);
        stringResults = "Advance Ratio = \t" + Double.toString(results[6]) + "\n";
        this.textResults.append(stringResults);

    }
```

Multi-results frame class

```
    public void printResults(double[][] results, String[] labels) {
        int n = labels.length;
        int m = results.length;
        String stringResults;
        for (int i = 0; i < n; i++) {
            stringResults = labels[i] + "\t";
            this.textResults.append(stringResults);
        }

        stringResults = "\n";
        this.textResults.append(stringResults);


        for (int j = 0; j < m; j++) {
            for (int i = 0; i < n; i++) {
                stringResults = Double.toString(results[j][i]) + "\t";
                this.textResults.append(stringResults);
            }
            stringResults = "\n";
            this.textResults.append(stringResults);
        }


    }
```