JAVAPROP – USERS GUIDE

June 2009
November 2010
Martin Hepperle

Table of Contents

Introduction

JAVAPROP is a simple tool for the design and the analysis of propellers and wind turbines. Within its limits, it is applicable to aeronautical as well as to marine applications. The implemented classical blade element methods are based on a coupling of momentum considerations with two-dimensional airfoil characteristics. It is therefore possible to consider different airfoil sections and the impact of their characteristics on rotor performance.
This document describes the pure application of JAVAPROP and is not a complete description of the underlying theory. For a description of the theoretical background, the reader is referred to the references cited on page 29.

As a relatively simple tool the capabilities of JAVAPROP should not be exploited beyond reasonable limits. The underlying blade-element-momentum theory is valid for the design and analysis of many propellers as long as

- the disc loading of the propeller is not too high (thrust coefficient $T_C \lessapprox 2$), which excludes static operation conditions,
- the number of blades is small (say below 15) so that no strong interaction due to thickness occurs,
- three-dimensional effects are small (no winglets, no highly curved blades), and
- compressible flow effects are small and mostly two-dimensional ($M_{tip} < 1.0$).

Symbols and Coefficients

Symbols

In the field of propellers and windmills a variety of definitions are used to describe operating points and performance. JAVAPROP follows mostly the traditional American notation as described in the following tables.
Note that in some publications coefficients of the same name (e.g. T_C) are used, which follow their own definitions – so be careful when comparing results.

Symbol	Description	Unit
D	diameter	m
D_{sp}	spinner or hub diameter	m
$R = \frac{D}{2}$	radius	m
RPM	revolutions per minute	1/min
$n = \frac{RPM}{60}$	revolutions per second	1/s
P	power	W
T	thrust	N
v_∞	axial inflow speed (flight speed, wind speed)	m/s
$S = \pi \cdot R^2 = \pi \cdot \frac{D^2}{4}$	disc area	m^2
ρ	density of medium	kg/m^3
$\Omega = 2 \cdot \pi \cdot n$	angular speed	1/s

Coefficients

Description	Definition	Conversions
thrust coefficient (propeller)	$C_T = \frac{T}{\rho \cdot n^2 \cdot D^4}$	$= \frac{\pi}{8} \cdot T_C \cdot \left(\frac{v_\infty}{n \cdot D}\right)^2$ $= \frac{\pi}{8} \cdot \eta \cdot P_C \cdot \left(\frac{v_\infty}{n \cdot D}\right)^2$ $= \eta \cdot \frac{C_P}{\left(\frac{v_\infty}{n \cdot D}\right)}$
thrust coefficient (propeller)	$T_C = \frac{T}{\frac{\rho}{2} \cdot {v_\infty}^2 \cdot S}$	$= \frac{T}{\frac{\rho}{2} \cdot {v_\infty}^2 \cdot \pi \cdot R^2}$ $= \frac{8}{\pi} \cdot \frac{C_T}{\left(\frac{v_\infty}{n \cdot D}\right)^2}$ $= \frac{8}{\pi} \cdot \eta \cdot \frac{C_P}{\left(\frac{v_\infty}{n \cdot D}\right)^3}$

power coefficient (propeller)	$C_P = \frac{P}{\rho \cdot n^3 \cdot D^5}$	$= \frac{\pi}{8} \cdot P_C \cdot \left(\frac{v_\infty}{n \cdot D}\right)^3$ $= \frac{\pi}{8} \cdot \frac{1}{\eta} \cdot T_C \cdot \left(\frac{v_\infty}{n \cdot D}\right)^3$ $= \frac{1}{\eta} \cdot \frac{C_T}{\left(\frac{v_\infty}{n \cdot D}\right)}$ $= \left(\left(\frac{v_\infty}{n \cdot D}\right) \cdot \frac{1}{C_S}\right)^5$
power coefficient (propeller, wind turbine)	$P_C = \frac{P}{\frac{\rho}{2} \cdot {v_\infty}^3 \cdot S}$ $= C_{P\ \text{wind turbine}}$	$= \frac{8 \cdot P}{\rho \cdot v_\infty^3 \cdot \pi \cdot D^2}$ $= \frac{P}{\frac{\rho}{2} \cdot {v_\infty}^3 \cdot \pi \cdot R^2}$ $= \frac{8}{\pi} \cdot \frac{C_P}{\left(\frac{v_\infty}{n \cdot D}\right)^3}$ $= \frac{8}{\pi} \cdot \frac{1}{\eta} \cdot \frac{C_T}{\left(\frac{v_\infty}{n \cdot D}\right)^2}$
efficiency (propeller)	$\eta = \frac{T \cdot v}{P}$	$= \frac{T_C}{P_C}$ $= \frac{C_T}{C_P} \cdot \left(\frac{v_\infty}{n \cdot D}\right)$
advance ratio (propeller)	$J = \frac{v_\infty}{n \cdot D}$	$= \pi \cdot \lambda$
advance ratio (propeller)	$\lambda = \frac{v_\infty}{\Omega \cdot R}$	$= \frac{1}{\pi} \cdot \frac{v_\infty}{n \cdot D}$
tip speed ratio (wind mill)	$X = \frac{\Omega \cdot R}{v_\infty}$	$= \frac{1}{\lambda}$

Propellers

How to design a propeller

JAVAPROP contains a powerful direct inverse design module. Inverse design means that you specify only a few basic parameters and JavaFoil produces a geometry which automatically has the maximum efficiency for the selected design parameters. The beautiful thing is that JAVAPROP creates an optimum propeller with just 5 design parameters plus a selection of airfoil operating points along the radius. You can later modify this design to adapt to additional off-design conditions.

The following cards are relevant for the design of a propeller:

- Design,
- Airfoils,
- Options.

Parameters on the Design card

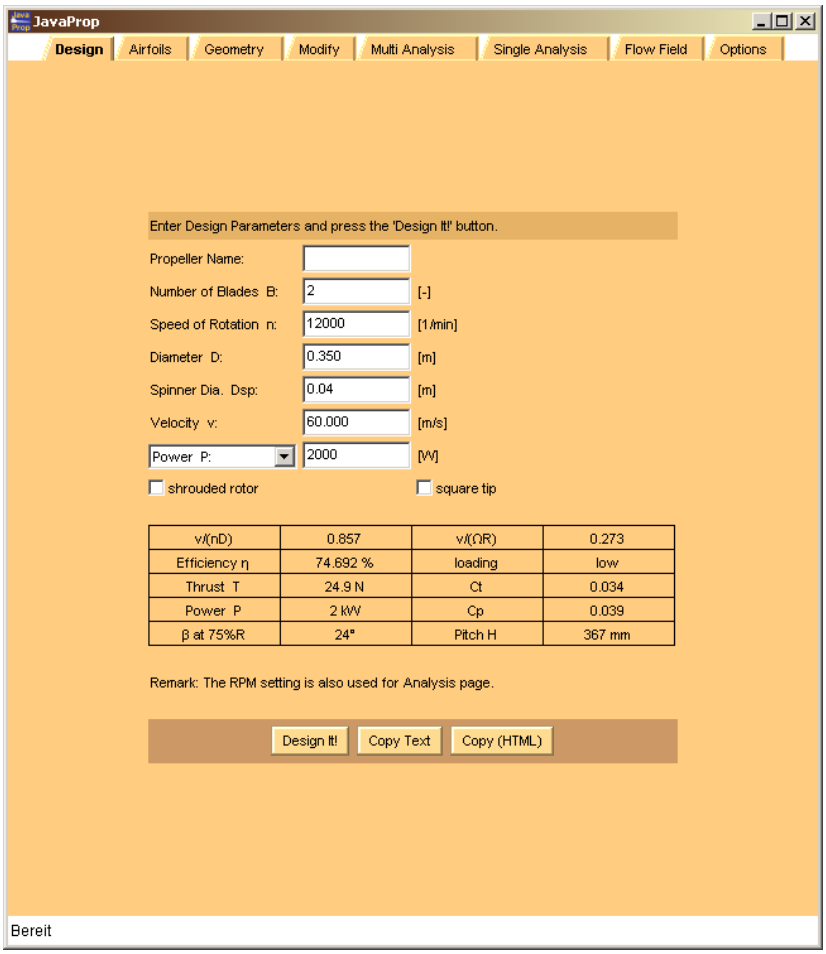


Figure 1: Design card after a design has been performed.

The design card holds most of the parameters which are required for a design.
It is possible to perform a design for either

- power (the propeller will consume the specified power),
- thrust (the propeller will produce the specified thrust), or
- torque (the propeller will consume the specified torque).

Shrouded propeller option

The thrust distribution along a propeller with free tips drops to zero at the tips. If a shroud is added to the propeller, this "tip loss" is suppressed. Note that this is only a crude approximation of the real flow field because the shroud itself affects the flow through the propeller and can create additional thrust, especially at low flight speeds. Also interaction between shroud shape and propeller would require a more complete model of such a system. Note also that the thrust calculated by JAVAPROP is only the propeller thrust; under static conditions; a good shroud design can double this figure. The thrust of the shroud drops with forward speed and will turn into drag at high speeds.
A detailed modeling of such configurations is beyond the targeted capabilities of JAVAPROP.

Square tip option

The optimum design procedure creates blades with rounded tips. As this is not always practical the option "square tip" produces a tip with finite chord length by simple extrapolation of the last section.

Loading

The optimum design procedure works well for lightly or medium loaded propellers. While the term loading is not clearly defined, JavaProp sorts the design into three categories depending on the thrust coefficient T_C:

- $T_C > 1$: highly loaded,
- $T_C > 0.25$: medium load,
- $T_C \leq 0.25$: lightly loaded.

If the loading is very high, the theory will become more and more inaccurate.

Airfoils card

In addition to the basic parameters on the Design card, airfoils have to be selected and their operating point must be specified on the Airfoils card.
JAVAPROP comes with several built-in airfoil sections. For each section tables of lift and drag coefficients versus angle of attack are shown on the Airfoils card. For the design it is necessary to assign airfoil sections to four radial stations – JAVAPROP interpolates linearly between these design sections. You define lift- and drag coefficient by selecting a design angle of attack for each section. Note that the absolute maximum efficiency is obtained when the airfoil sections are operated at the individual maximum L/D. For real world propellers, which must also be useable at low speed off-design conditions, it is usually better to select angles of attack, which are lower than the point of maximum L/D. This is especially true for the inner sections towards the root, which see a large variation of angle of attack with forward speed.

How to use your own airfoil polars

JAVAPROP comes with a set of canned airfoil polars. These are sufficient for first steps and for understanding the main design parameters. Nevertheless, some users may want to add their own airfoil data. This is possible in case of a local installation by copying polar data files into the JAVAPROP installation directory. This option is not available when JAVAPROP is run via WEBSTART or as an APPLET in a browser, because these applications are not allowed to access your computers file system for security reasons.

The polar data files must be named "af_#.EXT", where the # character represents a serial number and the extension EXT is either "afl" or "xml". JAVAPROP will search for files beginning with "af_1", trying the extension ".afl" first. If no matching file is found JAVAPROP

tries the alternative file name ending in “xml”. If either file was found it is read in and the airfoil index is incremented until no corresponding file is found (because no more files exist or the file names have a gap in their numbering).

The polar data files in “xml” format are in my standard XML format .These can be created with JAVAFOIL’s Polar card and by saving the polar with the extension “.xml”. The file must contain at least the three variables “alpha”, “Cl” and “Cd” declared in the <variables> block. JAVAPROP will read only the first <configuration> found in the file, therefore it is recommended to perform the analysis for just one Reynolds number close to the Reynolds numbers occurring during the operation of the propeller. You can have as many data points in the regime from -180° to +180°, but it is usually sufficient to provide polars with a range from -45° to +45° in steps of 2.5 degrees. JAVAPROP adds data points at +/- 90° automatically if not supplied. In order to achieve realistic results it makes sense to select a NACA standard roughness and no perfect surface finish.

Polar data files in “afl” format are primitive text files with exactly 5 header lines, followed by data points describing the airfoil polars. The first line contains the airfoil name and will appear in the dropdown list boxes of the Airfoils card.

```
Tabulated Airfoil 1
This is an airfoil polar file for JavaProp. It can have up to 1000 data triples.
This format will be changed to my airfoil-polar-XML form in a future release.
------------------------------------------------------------
alpha         cl         cd        cm
-180.00000  0.00000  0.49786  -0.13940
-175.00000  0.19970  0.27181  -0.07611
... some polar points omitted...
178.00000  -0.08022  0.00005  -0.00001
179.00000  -0.04013  0.00001  -0.00000
180.00000  -0.00000  0.00000  -0.00000
```

Figure 2: Example of a tabulated airfoil polar data set. Some data points have been omitted for clarity.

It is recommended to use the XML format, the AFL format exists only for backwards compatibility.

Note that the polars must also include the stall delay effect due to 3D effects on the rotating blades. In general the three-dimensional flow past the rotating rotor blade airfoil changes the airfoil characteristics, especially in the stall region. The inertial forces acting on the boundary layer tend to delay stall so that it occurs at higher angles of attack and higher lift coefficients. JAVAPROP does not modify the given polars for this effect because there is no general method to do so. Many stall delay models exist and each fits only a limited class of cases.
In any case, the pitching moment coefficients contained in the airfoil data are not used by JAVAPROP.

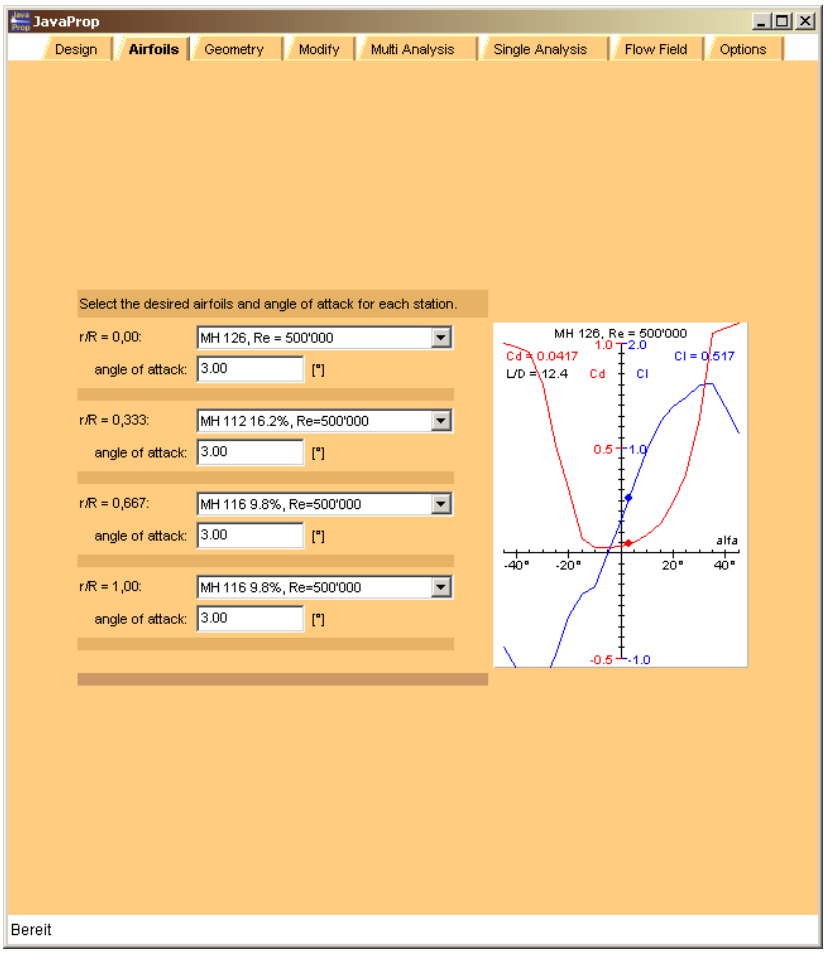


Figure 3: Airfoils card with lift and drag coefficients over the angle of attack.

Design parameters on the Options card

Finally the density of the fluid from the Options card is used for the design. A propeller designed for a low density medium (e.g. high altitude) must have blades of a wider chord length than a design for a high density medium.
This difference is also visible when a hydroprop is designed – the density of water is roughly 1000 higher than the density of air. Therefore a propeller for underwater operation would have blades of only 1/1000 the chord length of an aircraft propeller – if the diameter and the design lift coefficients were the same. Note that the cavitation phenomenon limits the airfoils design lift coefficient to rather lower values than the high values useable for the design of aircraft propellers (close to maximum L/D).

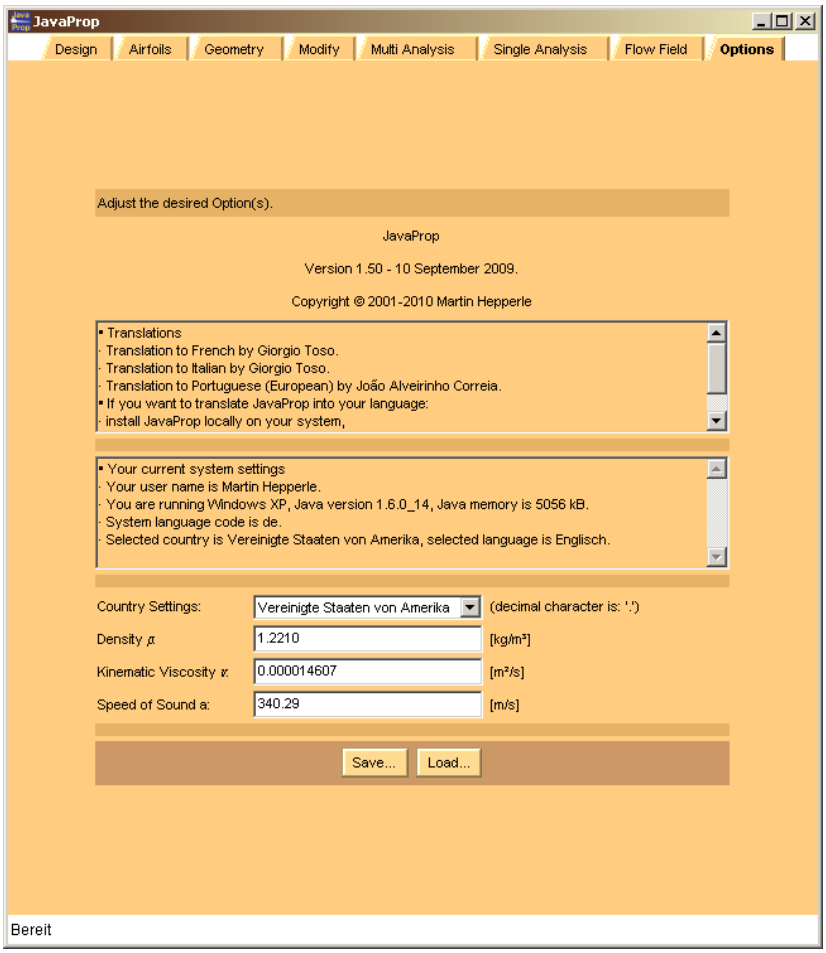


Figure 4: The Options card holds the density of the medium.

The Geometry card

This card (Figure 5) presents the geometry of the current propeller in form of a table and a three view sketch. It also presents the distribution of the pitch to diameter ratio H/D over the radius of the propeller.

The data table presents the following columns:

- "r/R" – the radius station, normalized by propeller radius,
- "c/R" – the corresponding chord length at each station, normalized by propeller radius,
- "β" – the blade angle at the station,
- "H/D" – the local pitch to diameter ratio,
- "r" – the radius of the station in absolute dimensions,
- "c" – the local chord length in absolute dimensions,
- "H" – the local pitch height in absolute dimensions,
- "Airfoil" – the airfoil at each station as selected on the "Airfoils" card.

The geometry card also allows you to copy or export the geometry in form of text files or as a 3D geometry, either in AutoCAD DXF or IGES format. Note that this option is intended mainly for illustration purposes. The geometry is exported accurately, but the resolution is possibly too low for application in high level CAD or for CNC machining. Note that JAVAPROP selects the export format by file name extension so that you have to use one of the proposed extensions.

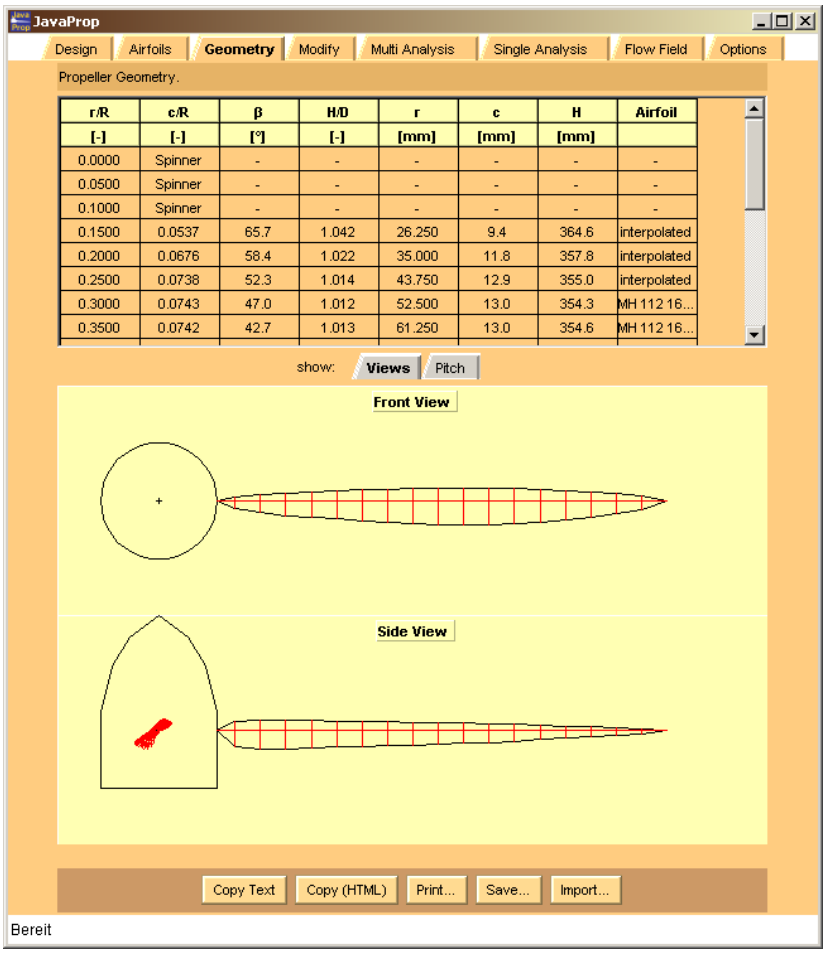


Figure 5: Geometry card with current propeller.

Additionally, this card also offers the option to import a given propeller geometry (Figure 6). The table must contain the planform as well as the blade angle in columns arranged “r/R”, “c/R”, “β”. Note that the blade angle must be specified in degrees. An example data set can be produced by copying the current propeller in text format to the clipboard and then opening the “Geometry Import” form. On opening, the import form automatically pastes the content of the clipboard to its text field. This also allows for manual modifications of the current propeller. You can copy and paste your prepared data via your systems clipboard. During the import process, JAVAPROP tries to be smart and skips non-numeric data, but it is a good idea to keep to the proposed format.

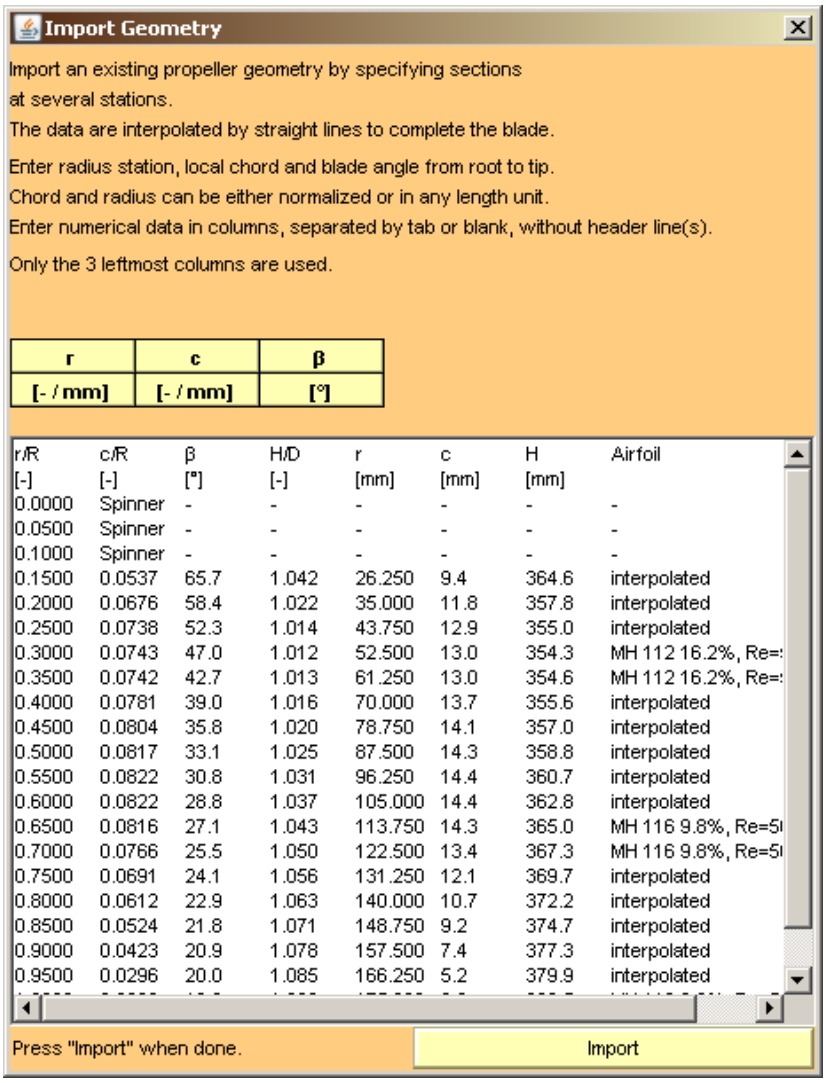


Figure 6: Geometry import form with example data.

The Modify card

This card offers tools to modify the current propeller blade or the inflow conditions as used for design and analysis. All modifications are performed in sequence from top to bottom. Usually you want to use only one type of modification and you should take care to reset it when you apply another, but different type of modification.

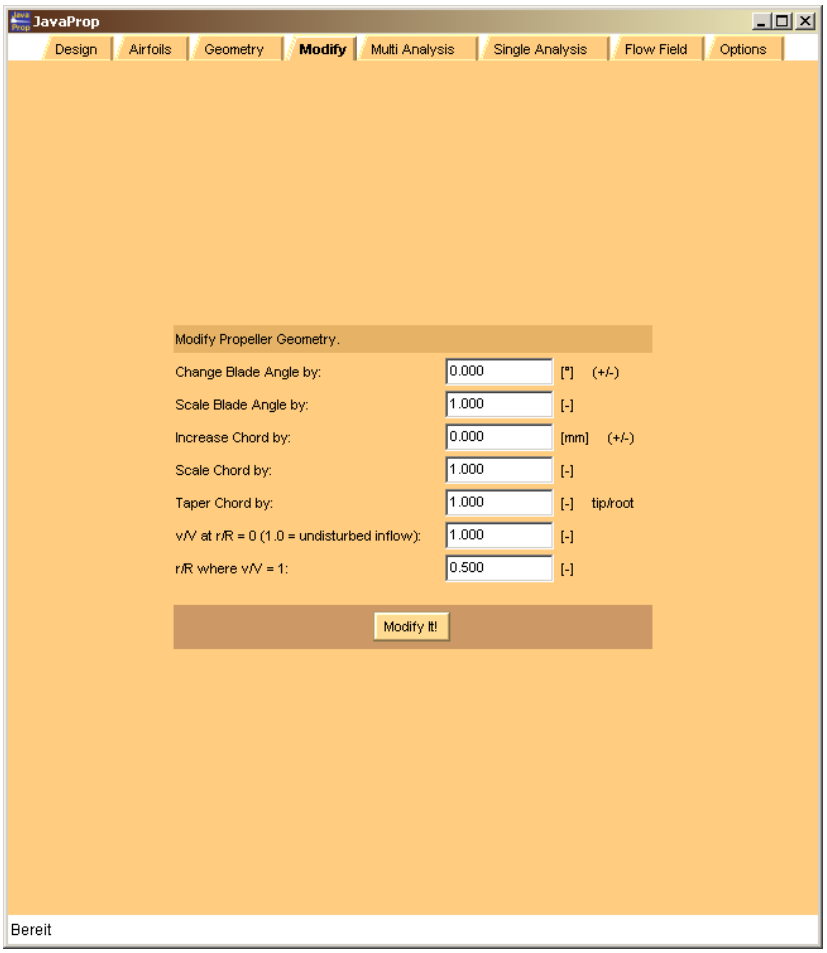


Figure 7: Modifications of the blade geometry can be performed using the Modify card.

The following modifications are available:

- *Change blade angle by a given angle* – this option changes the blade angle like on a variable pitch propeller by rotation of the complete blade. You can enter positive or negative values and check the result on the Geometry card.
- *Scale blade angle by a factor* – this modification multiplies the blade angle at each station by the given factor, this changing the internal twist of the blade.
- *Increase chord by a given distance* – adds or removes the given length from the chord length at each station. You can also specify negative values, but the chord length must not fall below zero.
- *Scale chord by a factor* – applies the given scale factor to the chord length at each station.
- *Taper chord by a taper ratio* – similar to the constant scaling factor option, but applies a variable scaling factor. For a given ratio "tip/root" of 0.5, the blade tip will be narrowed to 50% of its current chord while the root will maintain its current chord length.
- v/v_∞ *at* $r/R = 0$ and r/R *where* $v/v_\infty = 1$ – can be used to define a linear variation of the inflow velocity profile $v/v_\infty = f(r/R)$. Such an inflow profile can be used represent the influence of an axisymmetric body in front of or behind the propeller. The inflow velocity profile is used for design and analysis. The two values define a linear variation of the axial inflow velocity v which is normally equal to v_∞ (then "v/v_∞ at $r/R = 0$" must be set to 1.0 and "r/R where $v/v_\infty = 1$" is arbitrary).

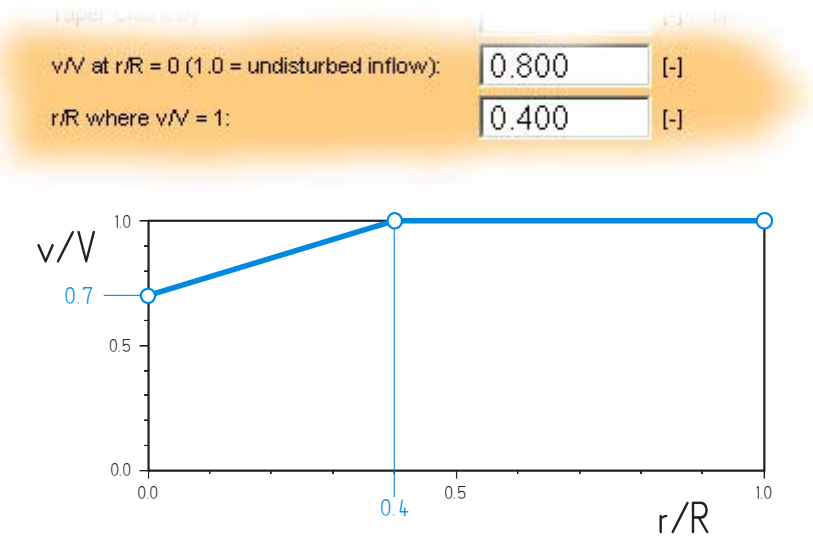


Figure 8: Modified inflow profile with a reduced axial flow speed towards the axis of rotation.

How to analyze a propeller

JAVAPROP can analyze propellers at arbitrary operating points. The propellers can be created by the design module of JAVAPROP or by importing a given geometry. There are two cards available for analysis:

- Multi-Analysis,
- Single-Analysis.

Both cards differ in their analysis range and in the level of detail of their output but apply the same analysis method.

Usage of the Multi-Analysis card

The Multi-Analysis card is used to analyze the propeller over its complete useable operating range from static operation up to the beginning of the windmilling regime at high speeds. The output of this card consists of the global propeller data like thrust, power or efficiency versus advance speed.
The coefficients shown on this card are generally applicable performance parameters. On the other hand the absolute values like thrust or power are calculated using these coefficients plus additional data taken from the

- design card (diameter, and one of n, P, T or Q),
- options card (density).

You can change any of these values and perform an additional analysis to study their effect. As long as you do not modify the geometry, the coefficients will always be the same, but the absolute values will change.

The four different cases for the calculation of the absolute values represent:

- n=given – constant speed propeller, P, T, Q vary with air speed,
- P=given – n is adjusted so that the propeller consumes the given power,
- T=given – n is adjusted so that the propeller produces the given thrust,
- Q=given – n is adjusted so that the propeller consumes the given torque.

In order to analyze for example a constant speed propeller at different speeds of rotation n, you would just change the value of n on the design card and then perform and additional Multi-Analysis. Careful: do not perform a new design on the Design card – this would create a new blade shape.

Note that none of these cases exactly represents a propeller operating on a given engine because for simplicity no engine performance curve model is used in JAVAPROP. While the "constant speed" (n=given) is a realistic operating procedure, the constant P, T or Q methods are somewhat artificial, but can be used to get an overview of the basic characteristics. Note that all these dimensional results are obtained from the single set of thrust and power coefficients versus advance ratio.

The point corresponding to the data on the Design card is marked in the graphical output by a black circle.
The scaling of the graphs is determined by the first set of analysis data. If you want to combine the results of several analyses into the graphs, you must therefore run the case with the maximum data extents first.

The output of the Multi-Analysis card contains a table with the following columns:

Symbol	Description
$v_\infty / (n \cdot D)$	advance ratio
$v_\infty / (\Omega \cdot R)$	advance ratio
C_T	thrust coefficient
C_P	power coefficient
C_S	speed-power coefficient
P_C	power coefficient
η	efficiency
η^*	maximum possible efficiency
stalled	relative disc area swept by stalled airfoils
v	flight speed resp. wind speed
n	speed of rotation
P	power
T	thrust
Q	torque

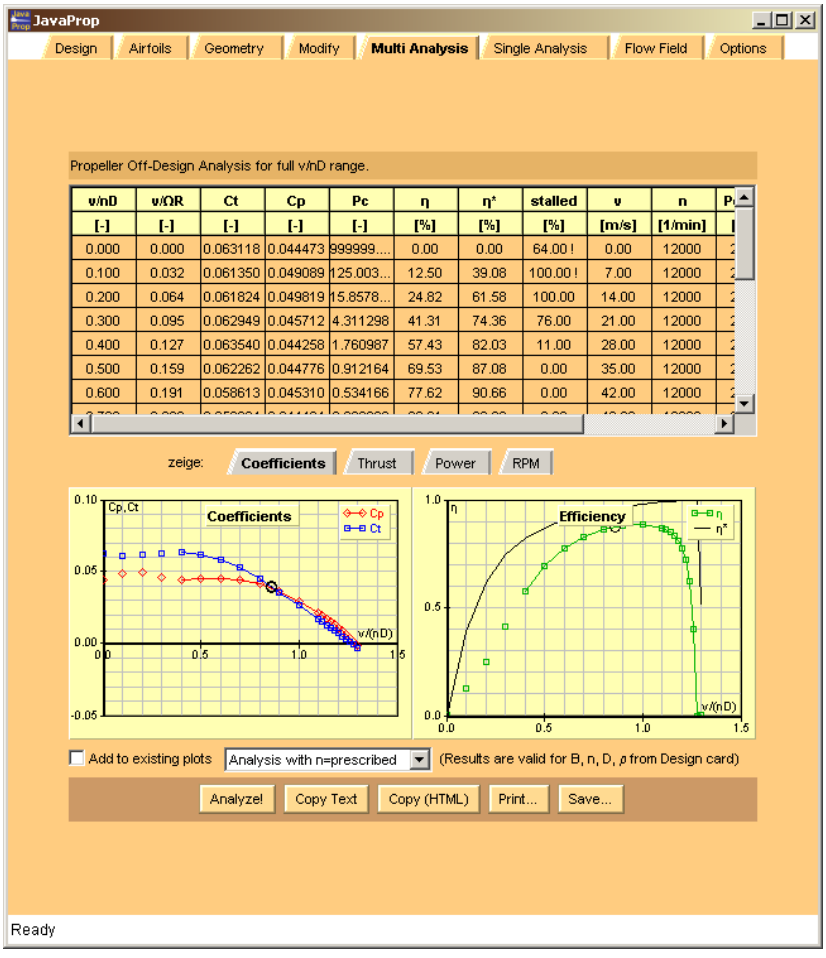


Figure 9: The Multi-Analysis card produces global propeller coefficients over a range of operating conditions.

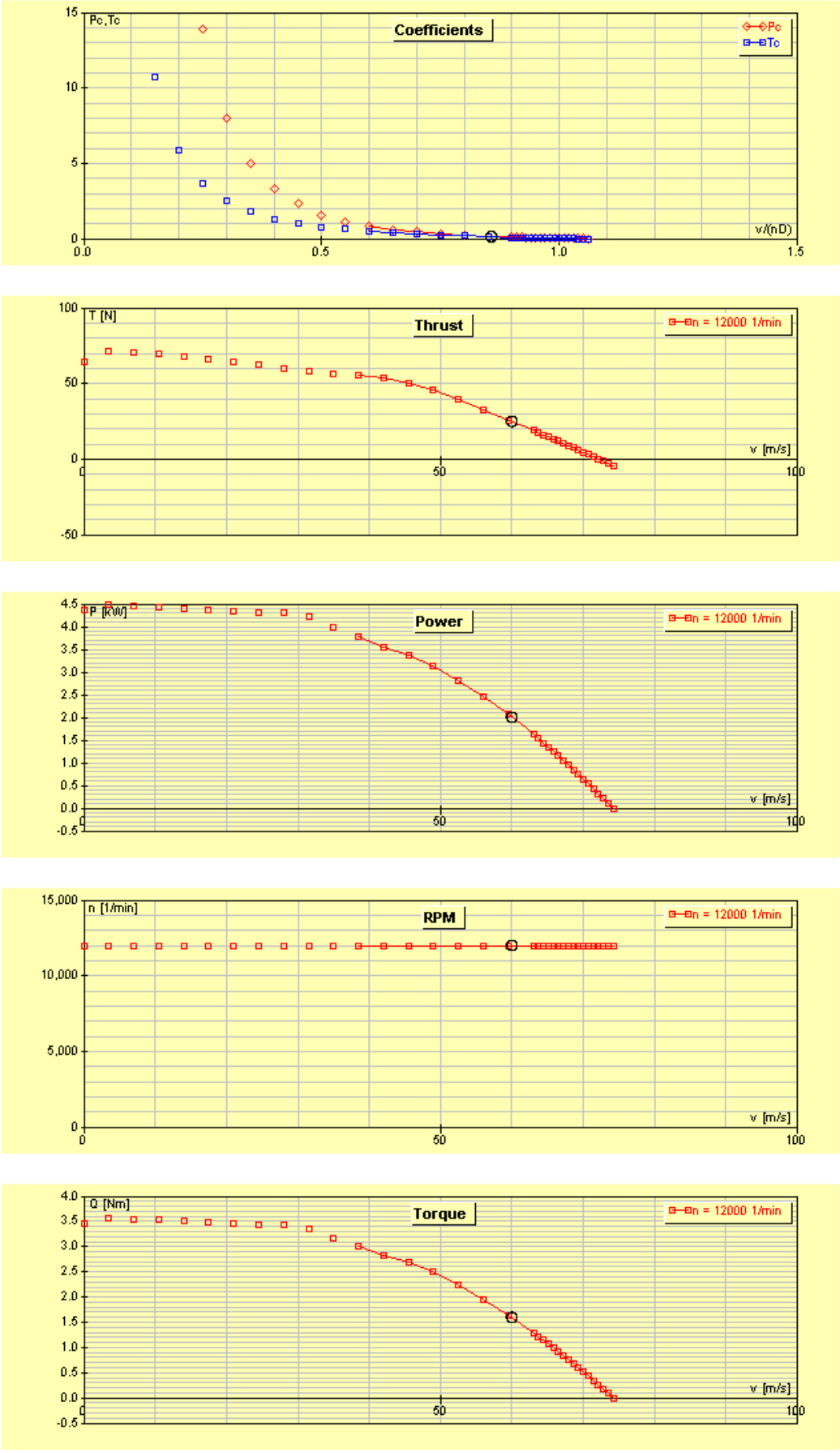


Figure 10: The individual graphs on the Multi-Analysis card present thrust, power, RPM and torque versus flight speed for the selected operating mode (in this case RPM=constant).

Usage of the Single-Analysis card

The Single-Analysis card is used to analyze the propeller at a single, arbitrary operating point. This point is specified by the flight speed v_∞, rotational speed n and diameter D on the Design card, which define an advance ratio

The output of the Single Analysis card is more detailed than that of the Multi-Analysis card. It consists of distribution of local aerodynamic data along the radius of the blade and includes coefficients related to structural loads (shear force and bending moment) as well. All results are available in a table and some specific data is presented in form of individual graphs.

The first page of graphs shows parameters relevant for airfoil aerodynamics, the second page displays the distributions of thrust and power as well as the local efficiency, whereas the third graph page presents the structural loads.

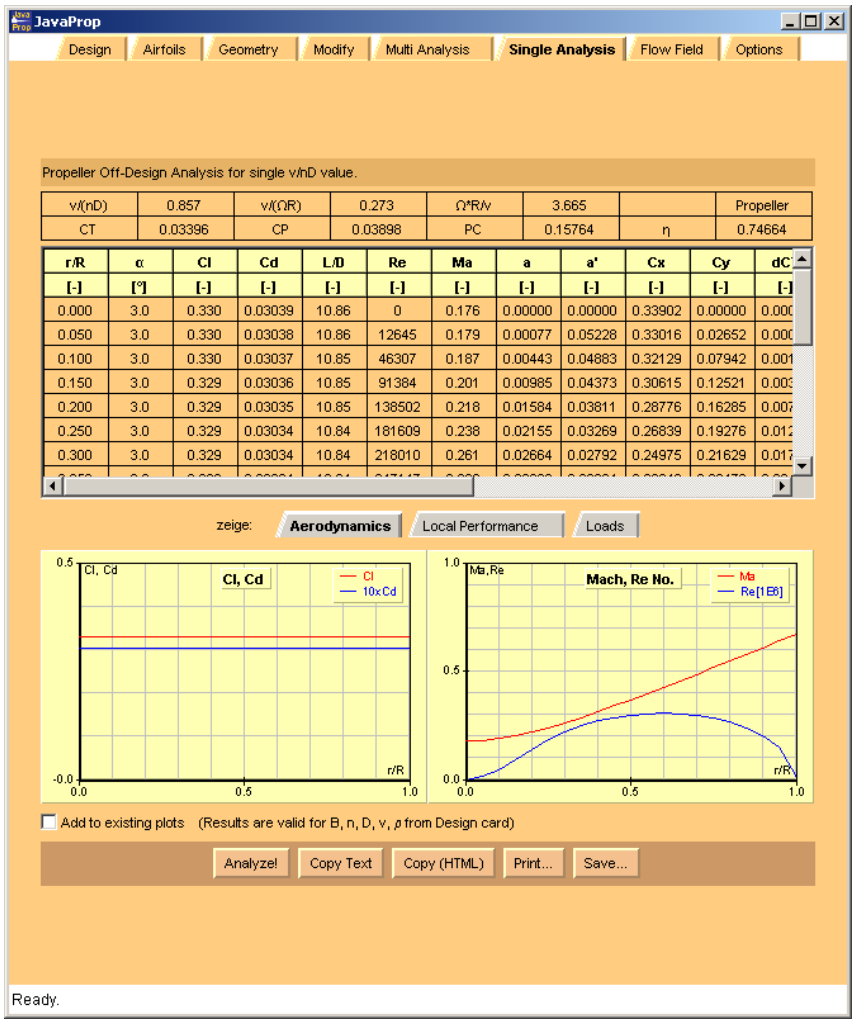


Figure 11: The Single-Analysis card shows detailed results for a single operating condition. The first page of graphs contains parameters related to airfoil section aerodynamics.

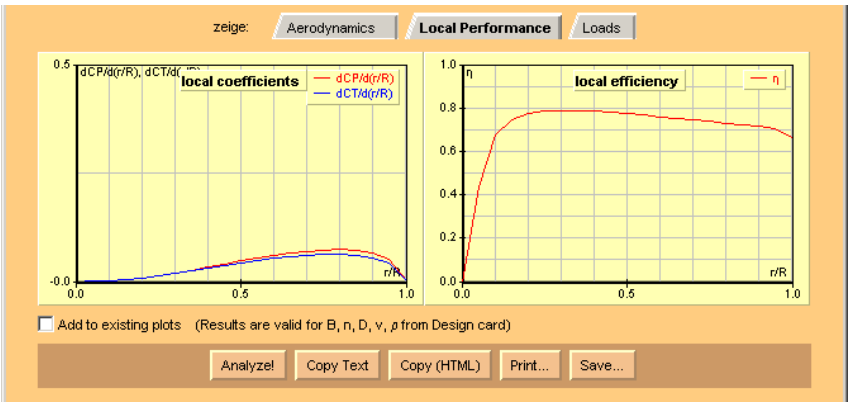


Figure 12: The second page of graphs contains local power and thrust coefficients as well as the local efficiency.

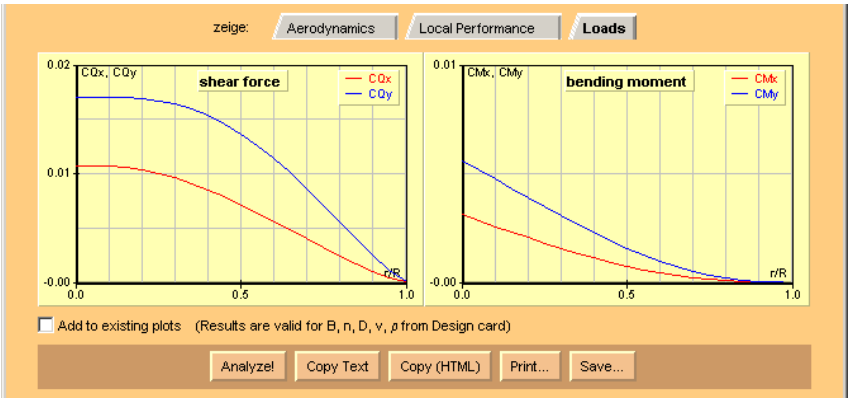


Figure 13: The third set of graphs shows shear force and bending moment.

Symbol	Description
r/R	relative radius
α	angle of attack in degrees
C_ℓ	lift coefficient
C_d	drag coefficient
L/D	lift to drag ratio
Re	Reynolds number
Ma	Mach number
a	axial induction factor in the propeller plane (the axial velocity through the propeller plane is $v_\infty \cdot (1+a)$)
a'	tangential induction factor in the propeller plane (the tangential velocity in the propeller plane is $(\Omega \cdot r \cdot (1-a'))$)
C_x	local tangential (in-plane) force coefficient at a blade station $C_x = \frac{F_x}{\frac{\rho_\infty}{2} \cdot v_{eff}^2 \cdot c \cdot dr}$
C_y	local thrust force coefficient at a blade station $C_y = \frac{F_y}{\frac{\rho_\infty}{2} \cdot v_{eff}^2 \cdot c \cdot dr}$
dC_T	local thrust coefficient of all blades for a ring element, $C_T = d(r/R) \cdot \sum dC_T$
dC_P	local power coefficient of all blades for a ring element, $C_P = d(r/R) \cdot \sum dC_P$
η	local efficiency at a blade station $\frac{v_\infty}{n \cdot D} \cdot \frac{dC_T}{dC_P}$

Symbol	Description
δ	swirl angle in the propeller plane in degrees
δ_{ff}	swirl angle far behind the propeller plane in degrees
$C_{Q\,x}$	tangential (in-plane) shear force coefficient (integrated from tip to root)
$C_{M\,x}$	tangential (in-plane) bending moment coefficient (integrated from tip to root)
$C_{Q\,y}$	normal (out of plane) shear force coefficient (integrated from tip to root)
$C_{M\,y}$	normal (out of plane) bending moment coefficient (integrated from tip to root)

Table 1: Description of the tabular results on the Single Analysis card.

Definition of shear force and bending moment coefficients

In order to determine the loads on the propeller blades the local aerodynamic forces represented by the coefficients C_x and C_y are integrated along the blade from tip to root. These coefficients are defined similar to the thrust and torque coefficients of the propeller.

Shear force due to out-of-plane axial force (thrust)

$$Q_y = C_{Q,\,y} \cdot \rho \cdot n^2 \cdot D^4$$

Shear force due to in-plane tangential force (torque/r)

$$Q_x = C_{Q,\,x} \cdot \rho \cdot n^2 \cdot D^4$$

Bending moment due to out-of-plane axial force (thrust)

$$M_y = C_{M,\,y} \cdot \rho \cdot n^2 \cdot D^5$$

Bending moment due to in-plane tangential force (torque/r)

$$M_x = C_{M,\,x} \cdot \rho \cdot n^2 \cdot D^5$$

Note that besides these aerodynamic forces and moments propellers are also largely affected by inertial loads. Torsional loads due to airfoil pitching moment or local sweep are not calculated by *JavaProp*.

Some simple validation checks

A quick validity check is the fact the axial shear force coefficient at the root must be equal to the thrust coefficient divided by the number of blades;

$$Q_y = \frac{C_T}{n_{blades}}$$

Another plausibility check is the center of thrust of each blade, which the radial position of a single force representing the thrust of the blade. This replacement force is acting at

$$\left(\frac{r}{R}\right)_{\text{center of thrust}} = 2 \cdot \frac{C_{M\,y}(0)}{C_{Q\,y}(0)}.$$

Most propellers have their thrust force located between 60 and 70% of the radius.

The flow field around the propeller

JAVAPROP has no capability to predict the flow around the propeller accurately. For this purpose more elaborate and time consuming methods would be required. However, a simple momentum theory model provides some basic results to show the main features of the flow through the propeller. This model includes the contraction of the stream tube as well as swirl loses. The results are presented on the Flow Field card in form of a contour plot of the axial velocity ratio (upper half of graph) and the stream tube boundary (lower half). It can be seen that half of the acceleration of the flow occurs at the propeller while the remaining half occurs due to the vortices in the slipstream.

If you analyze a wind turbine, you will notice that the stream tube is expanding, i.e. the flow is decelerated.

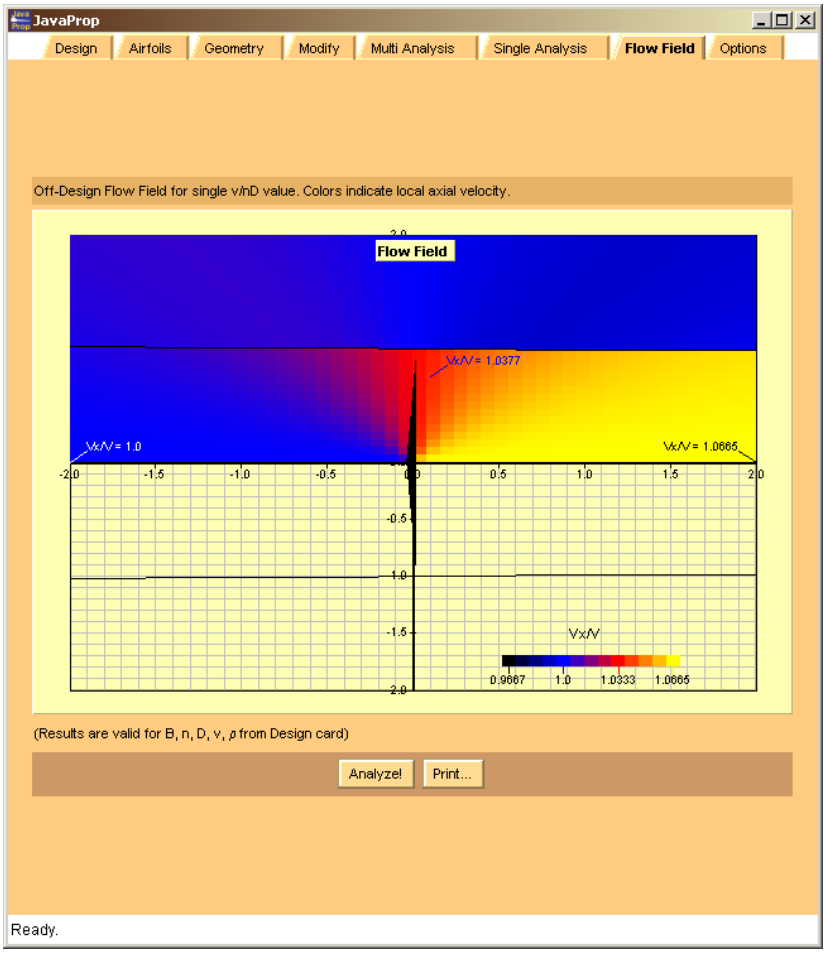


Figure 14: The Flow Field card produces a picture of the flow through the propeller.

Wind Turbines

While originally being designed as a propeller design and analysis tool, JAVAPROP can also be used for wind turbines. Some differences must be considered, though. Figure 15 shows the general power and thrust curves of a rotor, covering a wide speed range. In the case of propellers, only the left hand side of the graph is of interest, for wind turbines it is the right hand side. The transition between propeller and wind turbine state is fluent. Any fixed pitch propeller running at constant speed of rotation will eventually reach the windmilling state. When the air speed is increased further, it will act as a wind turbine, albeit a relatively poor one. This is because the airfoils on a wind turbine operate a negative lift and hence must be applied "upside down". Note also, that between the propeller and the wind turbine regimes there is a small range of advance ratios where the propeller already produces drag but still consumes power. This is a not very useful condition as the propeller merely creates entropy (heat). This effect is cause by friction and induced losses due to the radial lift distribution and cannot be avoided. Luckily this is only a very narrow band.

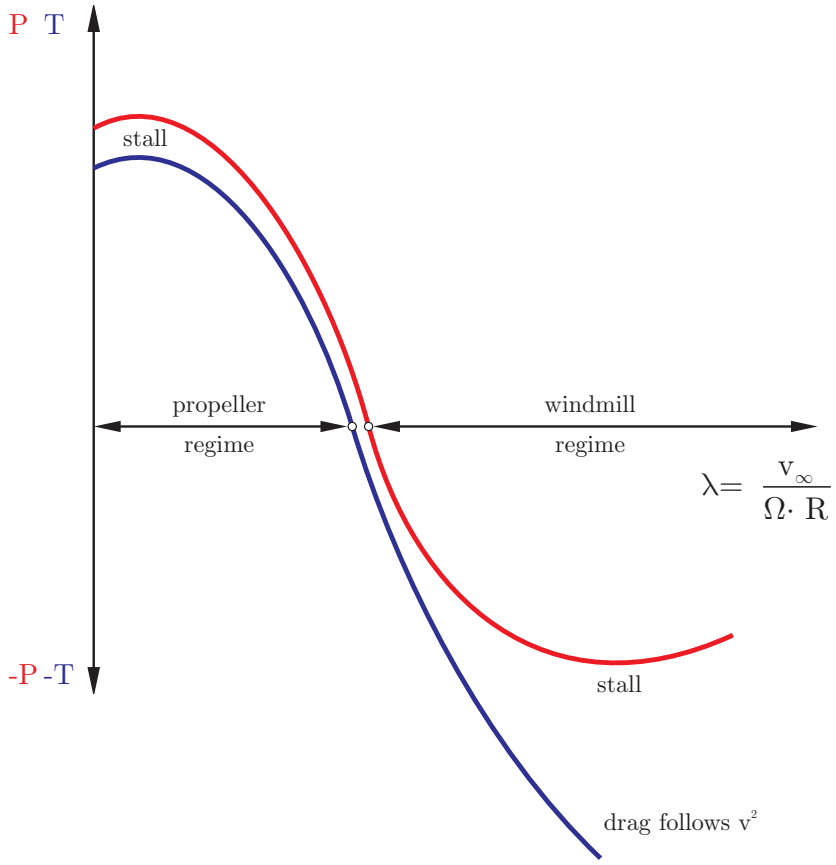


Figure 15: General operating characteristics of propellers and wind turbines, plotted versus advance ratio.

For the aerodynamic design and analysis of wind turbines the methods used for propellers can be applied. The analysis routines are the same, while the optimum design method is different, because the figure of merit of a wind turbine is different from the efficiency of a propeller. For a propeller the figure of merit is how much thrust can be generated for a given input power. The efficiency of a wind turbine can be expressed in how much energy is extracted from the mass of air passing through the rotor disc in relation to the amount of energy contained in this stream of air. The drag (negative thrust) acting on the tower is of no primary interest, only the amount of power extracted.

Because the performance characteristics of a wind turbine start where the operating range of the propeller ends, this windmilling regime does not start at a wind speed of zero. There is a required minimum wind speed at which the wind turbine can start to turn. This has some implications on the design parameters, as a design for a too low advance ratio $v/(n \cdot D)$ would not work.
The design method implemented in JAVAPROP is based on the work of Prandtl, Betz and Glauert in the 1930s. However, it takes friction forces into account, as described in [1], which produces more realistic designs. Nevertheless, the windmill design is rather sensitive to design parameters and a subsequent Multi-Analysis may produce poor results. In this case you should move the design advance ratio to a more reasonable value, e.g. by changing the design speed of rotation n. Figure 16 gives an overview of typical operating parameters depending on the wind turbine size.

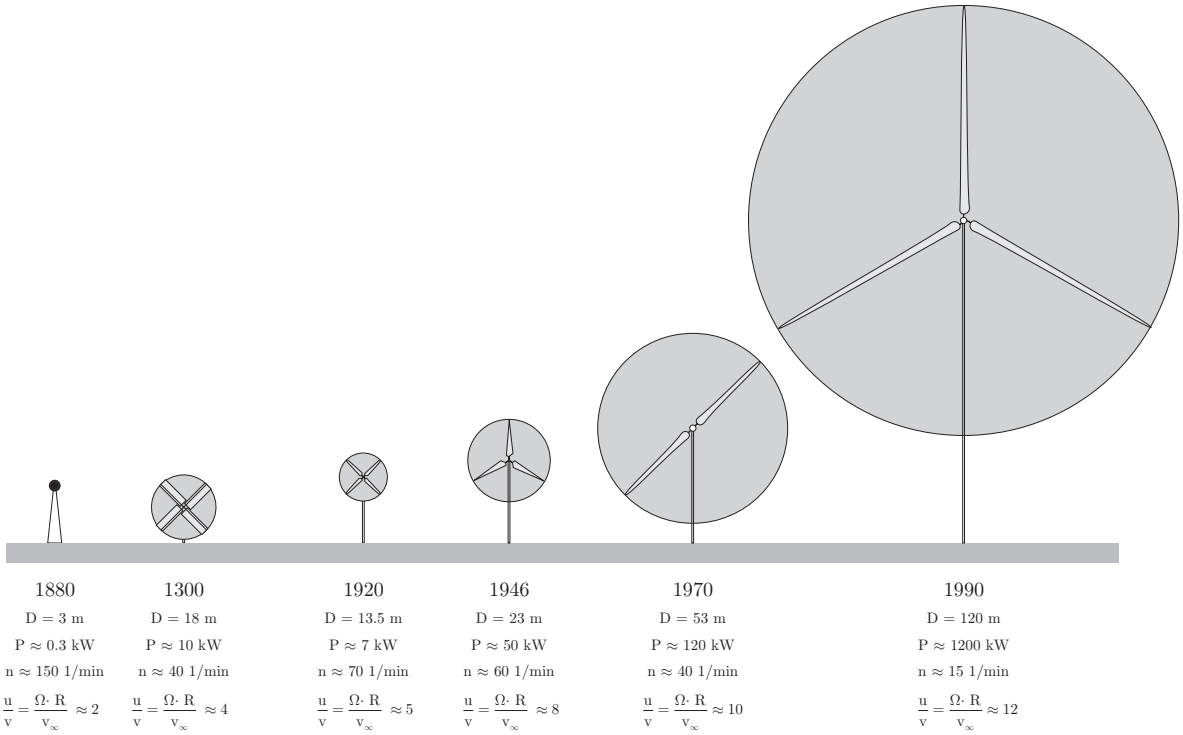


Figure 16: Typical operating parameters of different wind turbine types and sizes.

Parameters and coefficients

Note that JAVAPROP sticks to its propeller roots in maintaining the usual propeller coefficients and plots. This leads to wind turbines having negative values for torque and power as well as thrust. This sign change indicates that power and torque are delivered, not consumed, and that the thrust is actually a drag force, acting on the tower. Also the graphs of coefficients versus advance ratio are different from the common graphing of coefficients versus the tip speed ratio X, which is the reciprocal of the advance ratio λ, i.e. $X = 1/\lambda = \Omega \cdot R / v_\infty$.
Keeping the propeller conventions is not too inconvenient though, as the graphs versus $v_\infty/(n \cdot D)$ still display the behavior versus wind speed for a constant speed of rotation. Note especially, theta the power coefficient C_P as commonly used for wind turbines is not identical to the power coefficient C_P of propellers, but to the coefficient named P_C according to propeller terminology. Also, many parameters, like the axial and circumferential induction factors of a wind turbine are output as a negative value because it delivers power instead of the propeller which absorbs power.

For comparison with wind turbine codes you should compare the coefficient P_C versus the reciprocal of the advance ratio λ, which equals the tip speed ratio X as used for wind turbines.

Figure 17 shows how the starting advance ratio of a wind turbine depends on the airfoil performance. An ideal wind turbine would start at low wind speeds, but due to finite lift over drag ratios a realistic wind turbine requires a minimum advance ratio to start. The required minimum wind speed v_∞ for a given generator speed Ω and average airfoil performance can be estimated from the figure.

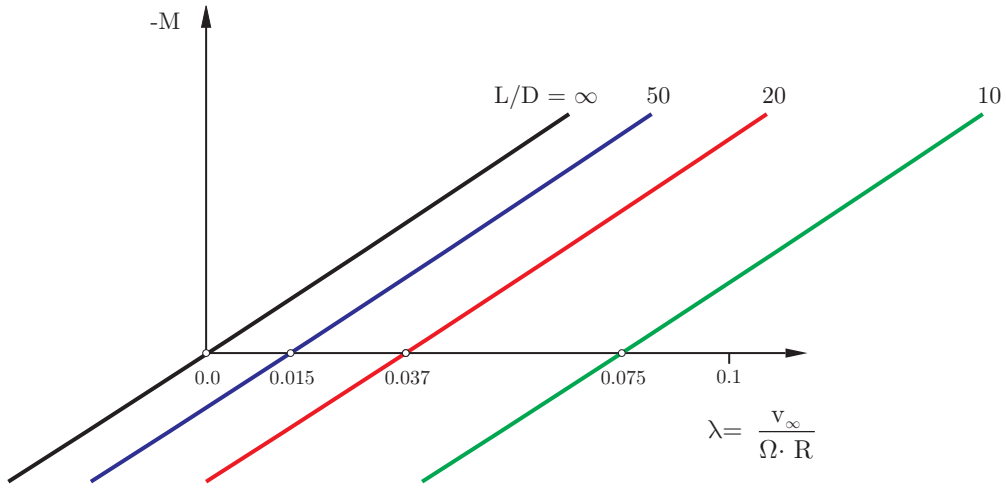


Figure 17: Effect of airfoil L/D ratio on the starting speed of a wind turbine.

Airfoil orientation

One main difference between the geometry of a wind turbine and a propeller is the orientation of the airfoil sections. JAVAPROP automatically turns the airfoils upside down if you specify a negative value for power, thrust or torque in the combo box on the design card. This indicates that a wind turbine is designed or analyzed. JAVAPROP maintains these inverted airfoils for the analysis as long as the negative value is maintained on the Design card. In terms of geometry you will note that for example in the output of the Geometry card all airfoils are arranged with an "upside down" orientation.
The inversion of the airfoils is indicated in the polar graph on Airfoils card by an additional subtitle "(upside down)". When the state of JavaFoil is restored from a saved `.jpdata` configuration file, any negative value for power, thrust or torque will switch the airfoils to wind turbine orientation.

How to design a wind turbine

JAVAPROP can design an optimum wind turbine for a given wind speed, speed of rotation and diameter. In order to design a wind turbine, you have to specify a negative value for the power on the Design card. The value itself is not used, only its sign is checked. Everything else is identical to the propeller design, the data on the Airfoils and the Options cards are used for the design.
The design follows the method of Glauert and therefore takes swirl losses into account but neglects friction losses. JAVAPROP determines the efficiency of a wind turbine as the ratio of the power coefficient P_C for wind turbines) to the power coefficient P_C^* which represents the

maximum power which could be extracted from the stream tube passing through the rotor. Swirl losses become very large when the tip speed ratio X is considerably lower than 1.0, i.e. the wind turbine is turning too slow. At high tip speed ratios the power coefficient approaches the limit derived by Betz for zero swirl, i.e. $P_C^* = 16/27$.

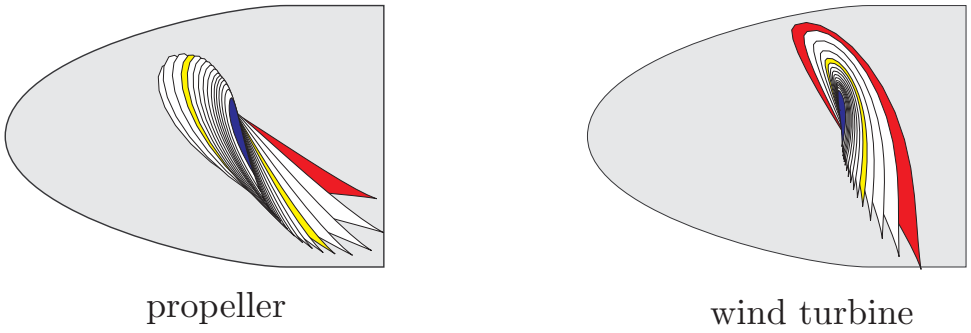


Figure 18: Orientation of airfoil sections on a propeller and on a wind turbine.

Because JAVAFOIL turns the airfoils automatically upside down when a wind turbine design is performed, the design can be performed as usual with a positive lift coefficient, typically close to the maximum L/D of each airfoil section. The inversion of the airfoils is indicated in the polar graph on airfoils card by an additional subtitle "(upside down)".
Note that the wind turbine design procedure is rather sensitive and may produce shapes, which will not lead to realistic results in subsequent analysis. This is usually the case when the advance ratio is too far off. Compared to propeller design, where any magnitude of power can be made available, the useful range of advance ratios for wind turbines is smaller. The wind turbine draws its power only from the incoming wind and must overcome airfoil drag and momentum losses.
Therefore some experimentation may be required to achieve a reasonable design. Typical advance ratios for wind turbines are in the order of $v/(n \cdot D) = 0.5$ to $v/(n \cdot D) = 1.5$. It is also recommended to use more sophisticated airfoils than for example the flat plate. It is also recommended to define a spinner of a not too small diameter to cover the hub region.
The design method can be used to produce a first geometry, which is later refined and modified to suit additional requirements, like a desired maximum chord length.

Large Wind Turbine		
diameter D	120	m
spinner $D_{spinner}$	4	m
speed of rotation n	12	1/min
velocity v	10	m/s
number of blades B	3	
airfoils		
r/R=0.0	MH 126	α=14°
r/R=0.333	MH 112	α=9°
r/R=0.667	MH 116	α=6°
r/R=1.0	MH 116	α=5°

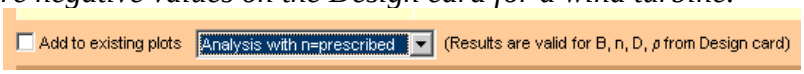
Small Wind Turbine		
diameter D	0.35	m
spinner $D_{spinner}$	0.05	m
speed of rotation n	1600	1/min
velocity v	7	m/s
number of blades B	2	
airfoils		
r/R=0.0	E 193	α=8°
r/R=0.333	E 193	α=7°
r/R=0.667	E 193	α=6°
r/R=1.0	E 193	α=3°

Table 2: Example cases: "Large Wind Turbine" and "Small Wind Turbine".

How to analyze a wind turbine

The single and multiple operating point analyses of JAVAPROP are performed exactly in the same way as the propeller analysis. At times, the analysis of wind turbines can be a bit more sensitive – it is recommended to use a spinner to blank out the innermost sections and to use reasonable airfoil sections, not the flat plate.
If you want to analyze an imported geometry, you should make sure that the airfoil polars are switched upside down by specifying a negative value in the power/thrust/torque combo box on the Design card.

> *Note that the analysis of rotors in JAVAPROP can be performed for constant values of n or V as well as constant power, thrust or torque –the last three options (constant P, T, Q) require negative values on the Design card for a wind turbine.*



A wind turbine should always produce the maximum possible power output at any wind speed. Therefore it must operate at its design tip speed ratio X_{design} which links wind speed and speed of rotation by $\Omega = v_\infty \cdot X_{design} / R$. Then the power coefficient would always be the maximum of $P_{C\,design}$ and the resulting power increases with the 3rd power of the wind speed:

$$P = P_{C\,design} \cdot \rho / 2 \cdot v_\infty^3 \cdot \pi \cdot R^2 .$$

At a certain wind speed the maximum power of the generator is reached and the machine must be relieved by reducing the power coefficient. This must be done by moving the tip speed ratio off the optimum point, either by actively braking to reduce Ω or by reducing the generator load and letting the machine spin up. The latter is probably only possible for small machines where high speeds of rotation are not dangerous.

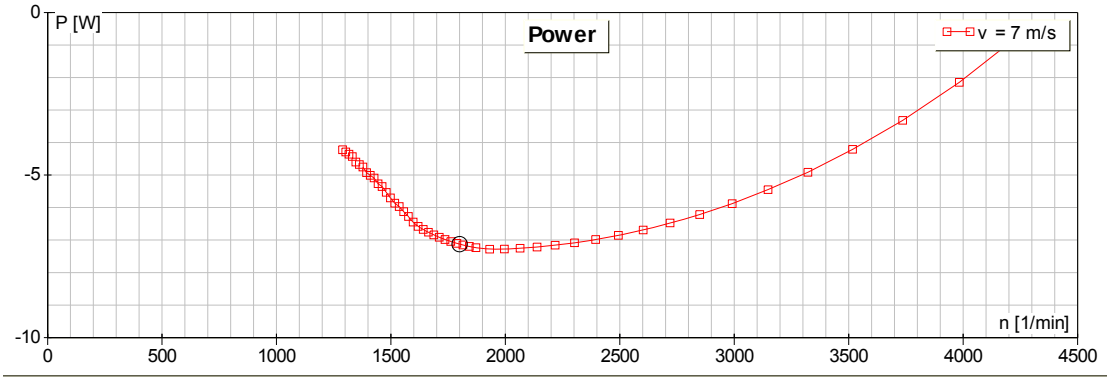


Figure 19: Variation of the power generation of the small wind turbine example with speed of rotation for a constant wind speed $v_\infty = 7\ m/s$. Circle indicates design point close to maximum power coefficient. Power can be reduced by increasing or by decreasing n.

Validation of JAVAPROP

To compare the results of JAVAPROP with experimental data, a set of test results for a propeller according to design 5868-9 of the Navy Bureau of Aeronautics was selected. The propeller geometry as well as the test data can be found in NACA Report 594. The propeller used airfoils of the Clark Y type and had 3 blades. Data shown in Figure 21 are for the configuration "Nose 6, Propeller C". The blade geometry according to the NACA report has been imported into JAVAPROP via the geometry card. The blade angle was adjusted to match

the given angles at 75% of the radius and the results produced by the Multi-Analysis card were collected in an Excel spreadsheet. No further tweaking was performed.
The comparison shows that JAVAPROP predicts the general performance characteristics in the typical "linear" operating range quite well. For this example, thrust and power are somewhat under-predicted, indicating that possibly the zero lift angle of the Clark Y airfoil in JAVAPROP might be too low. A more likely explanation however, is that the blade angle of the NACA tests refers to the flat underside of the blade while JAVAPROP uses the x-axis of the airfoil section for reference. Unfortunately the NACA reports do not give a clear indication, how exactly the blade angle was measured. In case of a Clark Y airfoil having 12% thickness, the difference amounts to about 2°.

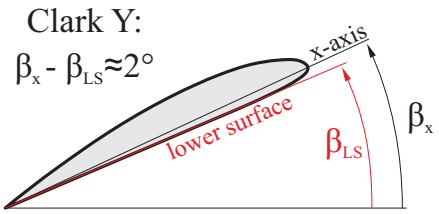
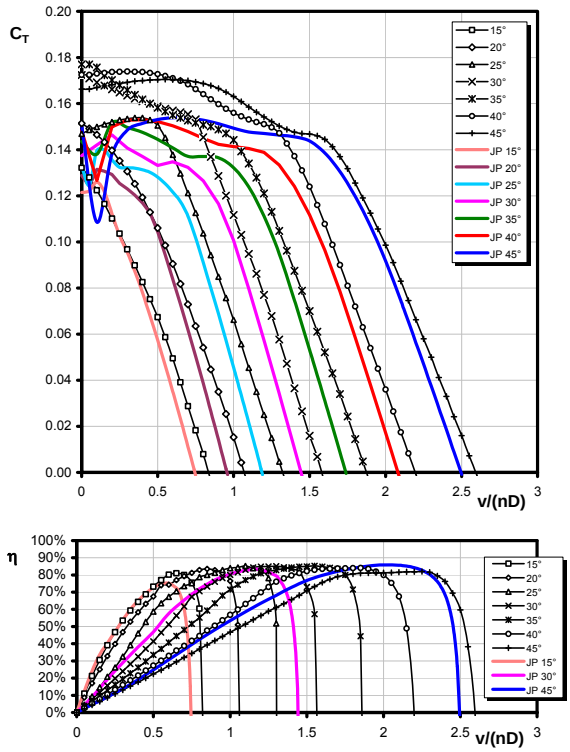


Figure 20: Possible reference lines for blade angle measurement.

Similar levels of the efficiency indicate that the lift to drag ratio of the Clark Y airfoil model in JAVAFOIL corresponds well to the tests.
Large deviations occur in the regions towards the left, where the propeller stalls. Here the flow is largely separated, three dimensional, unsteady and also depending on the external flow field (e.g. crosswind, wind tunnel interference). Such flow regimes are beyond the assumptions of the underling theory so that no good match can be expected here. It should be noted, that the experimental data show considerable scatter and irregular behavior in this regime too.



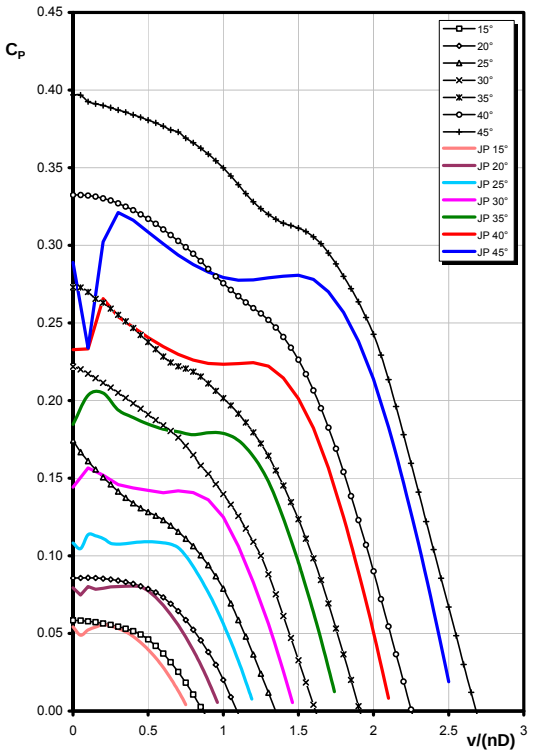


Figure 21: Prediction by JAVAPROP and experimental data from NACA R-594 (symbols).

Another comparison was performed for the propeller described in NACA Report 350. The results compare slightly worse, but still acceptable. The power coefficient predicted by JAVAPROP drops steeper with increasing advance ratio than the experiments indicate. This may be due to differences in the airfoil polars towards lower and negative lift coefficients.

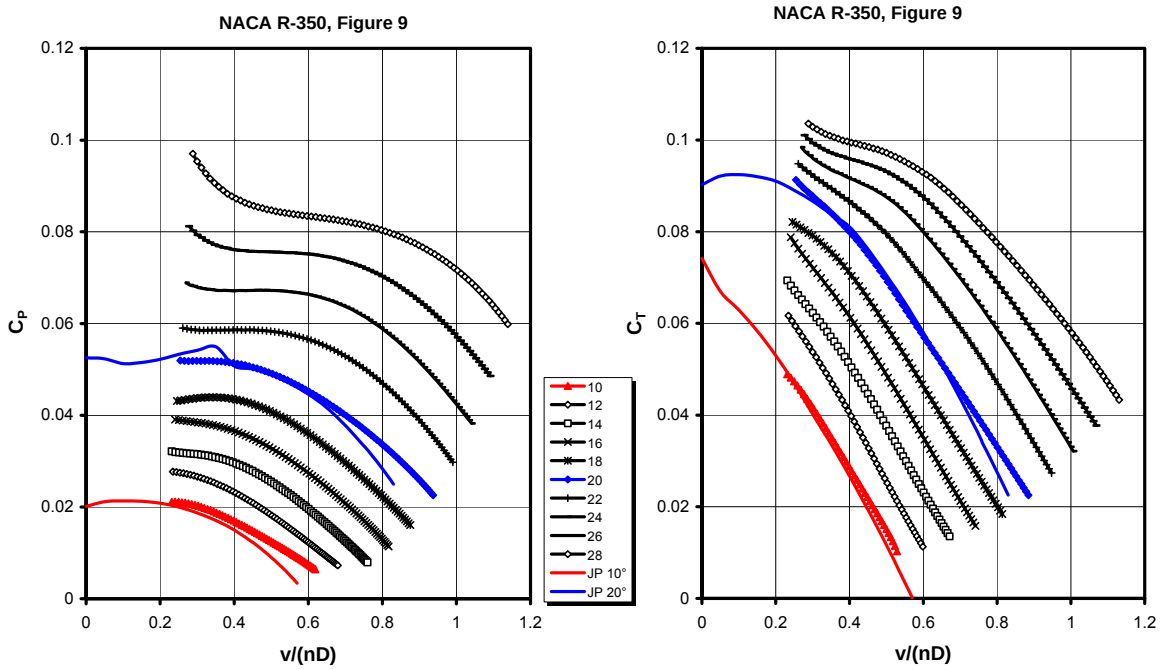


Figure 22: Data predicted by JAVAPROP and experimental data from NACA R-350 (symbols).

Controlling JAVAPROP from other applications

Using JavaProp with GNU OCTAVE

GNU OCTAVE [10] is very similar to Matlab and also allows you to use JAVAPROP directly from an OCTAVE script. This enables you to perform more complex tasks without using the JAVAPROP GUI, for example parameter sweeps or numerical optimization. The application is almost identical to MATLAB.

```
%-------------------------------------------------------------------------
%
% A function to encapsulate the JavaProp propeller design code
%
%-------------------------------------------------------------------------
% Can be used with MatLab (tested with version 7.9.0)
% as well as with GNU Octave (tested with version 3.2.4).
%
% Requirements:
% For Octave the package java-1.2.7 or higher is required.
%
% Written by Martin Hepperle, December 2010.
% The initial version of this example has been developed by
% Ed Waugh, University of Southampton, 2007.
%-------------------------------------------------------------------------
```

```
%
% JavaProp is Copyright 2001-2010 Martin Hepperle
% http://www.mh-aerotools.de/airfoils/javaprop.htm
%
%--------------------------------------------------------------------------

%% DesignProp

function DesignProp ()

    % add the archives to the Java classpath (must be done only once
    % per session, but can be called multiple times)
    basepath = '..\java';
    javaaddpath([basepath, '\', 'JavaProp.jar']);
    javaaddpath([basepath, '\', 'MHClasses.jar']);

    % Number of airfoil sections to use, the online JavaProp GUI uses 4
    % If you change this number you also need to change the setAirfoil section
    % to define all the desired airfoils
    % Usually 2-4 sections are sufficient to define the airfoil distribution
    % along the radius
    airfoil_sections = 4;

    % Number of blade sections, increasing this means a longer run time
    % but greater accuracy.
    % Sufficiently accurate results are obtained with 25 to 50 blade elements.
    blade_sections = 21;

    % Create a Propeller object to work on and name it
    PropDesign = javaObject('MH.JavaProp.Propeller', ...
                            blade_sections, airfoil_sections);
    PropDesign.Name = 'JavaProp-Test';

    % environmental parameters (atmosphere at sea level)
    PropDesign.Density = 1.225;                     % [kg/m^3]
    PropDesign.KinematicViscosity = 0.000014607;    % [m^2/s]
    PropDesign.SpeedOfSound = 340.29;               % [m/s]

    % Define the airfoil distribution
    theAirfoil = createAirfoil ( 3 );
    PropDesign.setAirfoil(0, 0.000, theAirfoil);

    theAirfoil = createAirfoil ( 3 );
    PropDesign.setAirfoil(1, (1/3), theAirfoil);

    theAirfoil = createAirfoil ( 4 );
    PropDesign.setAirfoil(2, (2/3), theAirfoil);

    theAirfoil = createAirfoil ( 4 );
    PropDesign.setAirfoil(3, 1.000, theAirfoil);

    % define the design angles of attack (degrees)
    PropDesign.setAlfa(0, 0.000, 3.0);
    PropDesign.setAlfa(1, (1/3), 3.0);
    PropDesign.setAlfa(2, (2/3), 3.0);
    PropDesign.setAlfa(3, 1.000, 3.0);

    % set prop dimensions [m]
    Diameter = 0.457;
    Radius = Diameter / 2;

    % number of blades
```

```
    PropDesign.BladeCount = 2;

    % set the spinner size [m]
    SpinDiameter = 0.126;
    PropDesign.rRSpinner = SpinDiameter / Diameter;

    % Set the design conditions for the propeller
    % The geometry of the resulting prop will be optimized to give the maximum
    % efficiency at the point specified. Either Thrust or Power can be defined
    % but not both. One must be set to zero. To use a Torque value, set both
    % Power and Thrust to zero.

    Airspeed = 30;                  % [m/s]
    RPM = 5000;                     % [revs/min]

    Frequency = (RPM / 60);         % frequency [Hz]
    Omega = 2 * pi * Frequency;     % angular velocity [rad/s]

    % specify one of these three, set the others to zero
    Power = 0;                      % [Watt}
    Thrust = 0;                     % [Newton}
    Torque = 0.915;                 % [Nm}

    if Power == 0                % if power and thrust are zero use torque
        if Thrust == 0
            Power = Torque * Omega;
        end
    end

    % create a propeller
    PropDesign.performPropellerDesign(Airspeed, Omega, ...
         Radius, Power, Thrust);

    % perform an analysis to get some of the parameters we need
    PropDesign.performAnalysis(Airspeed / (Frequency * Diameter));

    % assign the PropDesign object to the base workspace
    assignin('base','PropDesign',PropDesign);

end

%--------------------------------------------------------------------------

function [theAirfoil] = createAirfoil ( AirfoilNo )
   %
   % Generate and initialize an airfoil.
   %
   % It is also possible to set a base directory which is later used,
   % when Init(n) is called with n >= 14  to read airfoil polars
   % from file af_1.afl (or, if not found from af_1.xml) in JP directory
   %
   % theAirfoil.setBaseDir("c:/...");
   % theAirfoil.Init ( 14 );
   %

   theAirfoil = javaObject('MH.AeroTools.Airfoils.Airfoil');
   theAirfoil.Init ( AirfoilNo );

end
```

Using JavaProp with MATLAB

Because JAVAPROP is written in Java it is possible to access its classes directly from a MATLAB [11] script. The application is almost identical to GNU OCTAVE.

Using JavaProp with MATHEMATICA

WOLFRAM MATHEMATICA [12] can also interface to Java classes. As I am not an expert in MATHEMATICA, the following notebook is probably not the most elegant way to use MATHEMATICA, but it should suffice to demonstrate how to interface JAVAPROP and MATHEMATICA.

```
(* This simple example demonstrates how JavaProp can be used from
   Wolfram Mathematica 7.0 *)

(* prepare environment *)
path = "C:\\Java\\JavaProp\\java";
Needs["JLink`"];
InstallJava[];
AddToClassPath[path <> "\\JavaProp.jar"];
AddToClassPath[path <> "\\JavaClasses.jar"];

(* define number of airfoils along blade, see below for
   defining these 4 sections *)
airfoilSections=4;
(* define number of blade sections for elemental discretization,
   usually 20-40 is sufficient *)
bladeSections = 20;

(* Create a Propeller object to work on and name it *)
PropDesign = JavaNew["MH.JavaProp.Propeller",bladeSections,airfoilSections];
PropDesign@Name= "JavaProp from Mathematica";

(* define environment - note: metric units are used throughout JavaProp *)
(* density of air in kg/m^3 *)
PropDesign@Density = 1.2210;
(* kinematic voscosity of air in m^2/s *)
PropDesign@KinematicViscosity = 0.000014607;
(* speed of sound in air in m/s *)
PropDesign@SpeedOfSound = 340.29;

(* Define the airfoil distribution by 4 airfoil sections *)
AirfoilNo=3;
theAirfoil = JavaNew["MH.AeroTools.Airfoils.Airfoil"];
theAirfoil@Init [AirfoilNo];
PropDesign@setAirfoil[0,0.0,theAirfoil];

theAirfoil = JavaNew["MH.AeroTools.Airfoils.Airfoil"];
theAirfoil@Init [AirfoilNo];
PropDesign@setAirfoil[1,(1.0/3.0),theAirfoil];

theAirfoil = JavaNew["MH.AeroTools.Airfoils.Airfoil"];
theAirfoil@Init [AirfoilNo];
PropDesign@setAirfoil[2,(2.0/3.0),theAirfoil];

theAirfoil = JavaNew["MH.AeroTools.Airfoils.Airfoil"];
theAirfoil@Init [AirfoilNo];
PropDesign@setAirfoil[3,1.0,theAirfoil];

(* define the design angles of attack (in degrees) *)
```

```
PropDesign@setAlfa[0,0.000,4.0];
PropDesign@setAlfa[1,(1.0/3.0),3.0];
PropDesign@setAlfa[2,(2.0/3.0),2.0];
PropDesign@setAlfa[3,1.000,1.0];

 (* prop dimensions (meters) *)
Diameter=0.457;
Radius=Diameter/2;

(* number of blades *)
PropDesign@BladeCount=2;

(*spinner size (in meters) *)
SpinDiameter=0.126;
PropDesign@rRSpinner=SpinDiameter/Diameter;

(* Set the design conditions for the propeller
   The geometry of the resulting prop will be optimised to give the maximum
   efficiency at the point specified. Either Thrust or Power can be defined
   but not both. One must be set to zero. To use a torque value, set both
   power and thrust to zero. *)

(* flight speed meters/second *)
Airspeed=50.0;
(* propeller shaft revolutions per minute *)
RPM=5000;

(* angular velocity rad/s *)
Omega=2.0*Pi*(RPM/60);

(* either: supplied power in Watt *)
ShaftPower=0;
(* or: thrust in Newton *)
Thrust=0;
(* or: torque in Nm *)
Torque=0.915;

(* if power and thrust are zero: use torque *)
if [ShaftPower==0  , if [  Thrust==0 , ShaftPower=Torque*Omega  ] ];

(* create a propeller geometry by design *)
PropDesign@performPropellerDesign[ Airspeed, Omega, Radius,
    ShaftPower, Thrust];

(* create a nice table of results *)
resultTable1=Text@Grid[
{
{PropDesign@Name,SpanFromLeft},
{"Blades",PropDesign@BladeCount},
{"RPM",RPM,"1/min"},
{"v",PropDesign@V,"m/s"},
{"Diameter",Diameter,"m"},
{"Thrust",PropDesign@Thrust,"N"},
{"Power",PropDesign@Power,"W"},
{"Torque",PropDesign@getTorque[],"Nm"},
{"v/(nD)",Airspeed/(RPM/60.0*Diameter)},
{"Pitch",PropDesign@getBladePitch[],"m"},
{"Blade Angle",PropDesign@getBladeAngle[], "\[Degree]"},
{"CP",PropDesign@CP},
{"CT",PropDesign@CT},
{"Eta",PropDesign@Eta}
},
```

```
 Frame ->All];

cr = Transpose[{PropDesign@rR,PropDesign@Chord}];
br = Transpose[{PropDesign@rR,180.0*PropDesign@Beta/Pi}];

(* Perform an off-design analysis at prescribed v/(nD) *)
PropDesign@performAnalysis[0.5*Airspeed/((RPM/60)*Diameter)];

(* create a nice table with off-design results *)
resultTable2=Text@Grid[
{
{"Off-Design",SpanFromLeft},
{"Blades",PropDesign@BladeCount},
{"RPM",RPM,"1/min"},
{"v",PropDesign@V,"m/s"},
{"Diameter",Diameter,"m"},
{"Thrust",PropDesign@Thrust,"N"},
{"Power",PropDesign@Power,"W"},
{"Torque",PropDesign@getTorque[],"Nm"},
{"v/(nD)",Airspeed/(RPM/60.0*Diameter)},
{"Pitch",PropDesign@getBladePitch[],"m"},
{"Blade Angle",PropDesign@getBladeAngle[], "\[Degree]"},
{"CP",PropDesign@CP},
{"CT",PropDesign@CT},
{"Eta",PropDesign@Eta}
},
 Frame ->All];

(* terminate any Java virtual machines and processes *)
UninstallJava[];

(* show results *)
Join[resultTable1,resultTable2]
ListLinePlot[cr,PlotLabel-> "chord length distribution",
    AxesLabel-> {"r/R","c/R"}]
ListLinePlot[br,PlotLabel-> "blade angle distribution",
    AxesLabel-> {"r/R","beta [\[Degree]]"}]
```

The Propeller Object Model

This section describes the public fields, properties and methods available in the Propeller object.

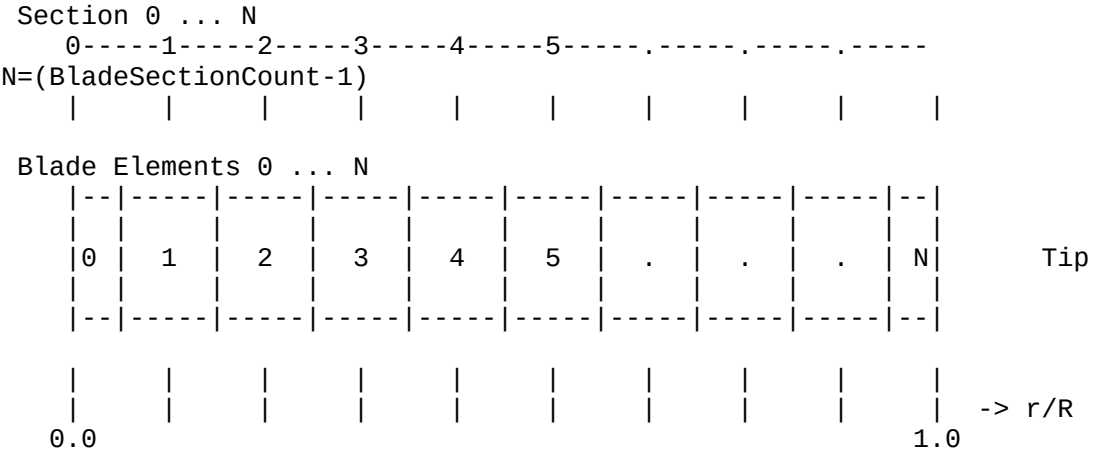
---

public class MH.JavaProp.Propeller

This class describes a propeller object.

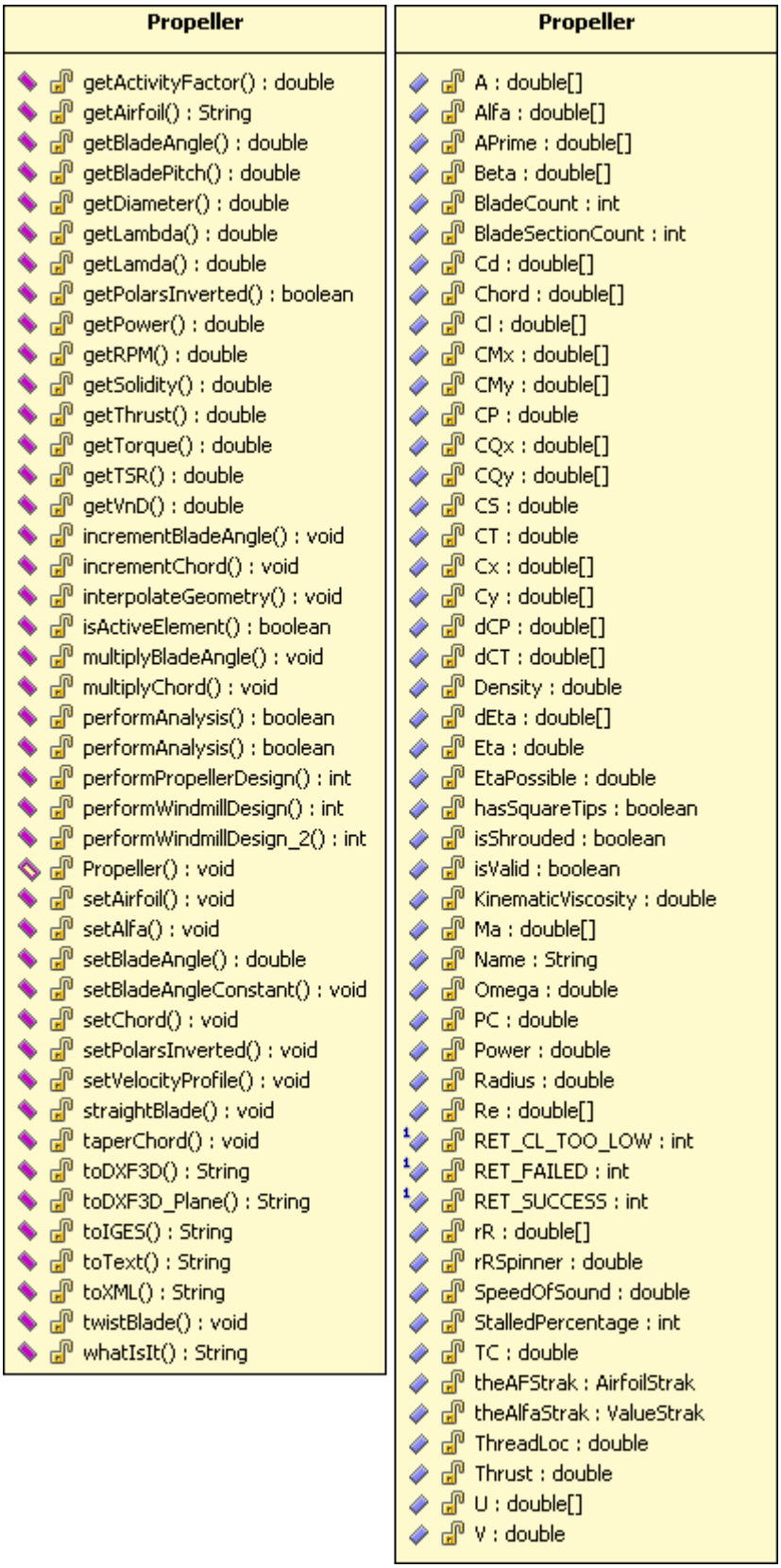
It contains methods to create a propeller or a windmill by design or by interpolating a given geometry. A designed or given propeller can also be analyzed at off-design conditions. Propellers are usually used as a means of propulsion for vehicles in air or water. A special form of a propeller is a windmill. Here the design converts power from the wind into shaft power. This Propeller object can also design and analyze windmills, but one must be aware of the fact that the airfoils on a windmill are mounted "upside down" compared to a propeller. This means, that airfoil polars have to be mirrored before they can be used. The propeller is represented by blade elements which are radially stacked. The method is based on the so

called blade element theory, which connects two dimensional airfoil characteristics with momentum equations. The blade element theory is well suited for fast conceptual and preliminary design and analysis. It has proven its applicability for typical general aviation, man powered aircraft and model aircraft applications. For a highly sophisticated detailed design, when strong transonic flow effects are expected or strong 3D effects due to highly curved or thick blades are present, application of a full blown CFD code should be considered.
The optimum design methodology implemented here is based on work by Betz, Prandtl and Glauert.



Notes:

- Each blade element is centered around the corresponding section.
- The width of element 0 and element N is half the usual element width.
- Element 0 is at r/R=0.0 and Element N at r/R=1.0.



Public methods and properties of the propeller class.

Author:

Martin Hepperle, Germany

Field Summary

double[]	**A** The distribution of the axial induction factor a along the radius [-].
double[]	**Alfa** The distribution of the angle of attack along the radius [rad].
double[]	**APrime** The distribution of the circumferential induction factor a' along the radius [-].
double[]	**Beta** The distribution of the blade angle ß along the radius [rad].
int	**BladeCount** The number of blades of this propeller.
int	**BladeSectionCount** The number of blade sections used for design and analysis.
double[]	**Cd** The distribution of the drag coefficient Cd along the radius [-].
double[]	**Chord** The distribution of the blade chord length c/R along the radius [-].
double[]	**Cl** The distribution of the lift coefficient Cl along the radius [-].
double[]	**CMx** The distribution of the in plane moment coefficient Mx along the radius [-].
double[]	**CMy** The distribution of the out of plane moment coefficient My along the radius [-].
double	**CP** The power coefficient CP [-].
double[]	**CQx** The distribution of the shear force coefficient Qx along the radius [-].
double[]	**CQy** The distribution of the shear force coefficient Qy along the radius [-].
double	**CS** The speed power coefficient CS [-].
double	**CT** The thrust coefficient CT [-].
double[]	**Cx** The distribution of the tangential force coefficient Cx along the radius [-].

double[]	**Cy** The distribution of the axial force coefficient Cy along the radius [-].
double[]	**dCP** The distribution of the local power coefficient dCP/d(r/R) along the radius [-].
double[]	**dCT** The distribution of the local thrust coefficient dCT/d(r/R) along the radius [-].
double	**Density** The density of the medium in [kg/m^3].
double[]	**dEta** The distribution of the local efficiency (v/nD) * dCT/dCP along the radius [-].

double	**Eta**
	The propulsive efficiency [-]
	This efficiency is defined by:
	`Eta = v*T/P`
	In case of windmilling operation (`Power < 0`) the efficiency is calculated as the ratio of delivered power to the power delivered by the ideal wind mill according to Albert Betz (1926):
	`Eta = P / ((16/27)*Density/2 * V^3 *D^2 * PI / 4)`

double	**EtaPossible** The maximum possible efficiency for the same thrust according to momentum theory [-].
boolean	**hasSquareTips** Whether the design method should produce a blade with pointed or square tips.
boolean	**isShrouded** Whether the blade operates inside a shroud.
boolean	**isValid**
double	**KinematicViscosity** The kinematic viscosity of the medium in [m^2/s].
double[]	**Ma** The distribution of the Mach number M along the radius [-].
java.lang.String	**Name** The name of this propeller.
double	**Omega** The angular velocity.
double	**PC** The power coefficient PC [-].
double	**Power** The power consumed by the propeller under the

	"current "operating conditions in [W].
double	**Radius** The radius of the propeller in [m].
double[]	**Re** The distribution of the Reynolds number Re along the radius [-].
static int	**RET_CL_TOO_LOW** Error code: design lift coefficient must be positive.
static int	**RET_FAILED** performPropellerDesign or performWindmillDesign failed
static int	**RET_SUCCESS** performPropellerDesign or performWindmillDesign was successful
double[]	**rR** The distribution of the relative radius r/R along radius [-].
double	**rRSpinner** The relative spinner radius r/R [-].
double	**SpeedOfSound** The speed of sound of the medium in [m/s].
int	**StalledPercentage** The percentage of blade elements for which the previous analysis predicted separated flow.
double	**TC** The thrust coefficient TC [-].
MH.AeroTools.Airfoils.AirfoilStrak	**theAFStrak** Interpolates the airfoils over the radius r/R.
MH.AeroTools.Util.ValueStrak	**theAlfaStrak** Interpolates the design angle of attack over the radius r/R.
double	**ThreadLoc** The threading line x/c of the sections along the radius [-].
double	**Thrust** The thrust produced by the propeller under the "current" operating conditions in [N].
double[]	**U** The distribution of the axial velocity ratio u/v along the radius [-].
double	**V** The axial (flight) speed in [m/s].

Constructor Summary

Propeller(int BladeSections, int AirfoilSections) Creates a Propeller having the given number of blade and airfoil definition sections.

Method Summary

double	**getActivityFactor**() Determine the Activity factor AF of the propeller.
java.lang.String	**getAirfoil**(double Position) Return the name of the airfoil at a given radial position.
double	**getBladeAngle**() Return the nominal blade angle at 75% of the radius.
double	**getBladePitch**() Return the nominal blade pitch H at 75% of the radius.
double	**getDiameter**()
double	**getLambda**()
double	**getLamda**()
boolean	**getPolarsInverted**()
double	**getPower**()
double	**getRPM**()
double	**getSolidity**() The solidity is the sum of area of all blades divided by disc area.
double	**getThrust**()
double	**getTorque**()
double	**getTSR**()
double	**getVnD**()
void	**incrementBladeAngle**(double DeltaDeg) Increments the local blade angle ß by a constant Delta-ß.
void	**incrementChord**(double DeltaChord) Increments the local chord c/R by a constant Delta-c/R.
void	**interpolateGeometry**(double[] R, double[] Chord, double[] BetaDeg, int nSections) Populates this Propeller object by interpolation of the geometry supplied in the given control sections.
boolean	**isActiveElement**(double rR)
void	**multiplyBladeAngle**(double AngleFactor)

	Multiplies the local blade angle ß with a constant ß-Factor.
void	**multiplyChord**(double ChordFactor) Multiply the current local chord c/R by a constant factor.
boolean	**performAnalysis**(double VnD) Performs the analysis of this propeller for a given advance ratio J.
boolean	**performAnalysis**(double V, double Omega, double Radius, double Density, double KinematicViscosity, double SpeedOfSound) Performs the analysis of this propeller for a given advance ratio J.
int	**performPropellerDesign**(double vDesign, double omegaDesign, double radiusDesign, double powerDesign, double thrustDesign) Perform the design of a minimum induced loss propeller.
int	**performWindmillDesign_2**(double V, double Omega, double Radius) Design an optimum windmill with swirl losses according to Glauert.
int	**performWindmillDesign**(double V, double Omega, double Radius) Design an optimum windmill with swirl losses according to Wiederhöft.
void	**setAirfoil**(int nSection, double Position, MH.AeroTools.Airfoils.Airfoil theAirfoil) Places the given airfoil at the given position of the airfoil strak.
void	**setAlfa**(int nSection, double Position, double AlfaDeg) Sets the angle of attack for the given section.
double	**setBladeAngle**(double AngleDeg) Sets the nominal blade angle at 75% of the radius.
void	**setBladeAngleConstant**(double AngleDeg) Define a blade having constant blade angle.
void	**setChord**(double ChordLength) Define a blade having a constant chord length.
void	**setPolarsInverted**(boolean isInverted) Set whether polars are to be used in "upside down" form.
void	**setVelocityProfile**(double[] vV) Define an axially symmetric velocity profile v/V of the onset flow.
void	**straightBlade**(double TaperRatio, double TwistAngleDeg) Create a geometry from straight generator lines connecting root and tip section.
void	**straightBlade**(double RootChord, double RootAngleDeg, double TaperRatio, double TwistAngleDeg) Create a geometry from straight generator lines connecting root and tip section.
void	**taperChord**(double TaperRatio) Multiply the local chord c/R by a linear varying c/R-Factor.

java.lang.String	**toDXF3D_Plane**() Create a 2D DXF representation of the propeller twist distribution.
java.lang.String	**toDXF3D**() Create a 3D representation of the propeller geometry.
java.lang.String	**toIGES**() Converts the 3D airfoil sections of this propeller into an IGES representation.
java.lang.String	**toText**(boolean IncludeHeader, boolean IncludeAeroData, java.lang.String LineStart, java.lang.String LineEnd, java.lang.String Separator) Creates a tabular representation of the propeller object.
java.lang.String	**toXML**()
void	**twistBlade**(double TwistAngleDeg) Modifies the local blade angle beta by a linear twist.
java.lang.String	**whatIsIt**() Find out what the current propeller is.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

RET_SUCCESS

public static final int **RET_SUCCESS**

performPropellerDesign or performWindmillDesign was successful

See Also:

Constant Field Values

RET_CL_TOO_LOW

public static final int **RET_CL_TOO_LOW**

Error code: design lift coefficient must be positive.

See Also:

Constant Field Values

RET_FAILED

public static final int **RET_FAILED**

performPropellerDesign or performWindmillDesign failed

See Also:

Constant Field Values

Name

public java.lang.String **Name**

The name of this propeller.

BladeSectionCount

`public int` **`BladeSectionCount`**

The number of blade sections used for design and analysis. The blade radius is cut into `BladeSectionCount` blade elements.

BladeCount

`public int` **`BladeCount`**

The number of blades of this propeller.

StalledPercentage

`public int` **`StalledPercentage`**

The percentage of blade elements for which the previous analysis predicted separated flow.

isShrouded

`public boolean` **`isShrouded`**

Whether the blade operates inside a shroud. If set to true, the Prandtl tip loss model will be switched off. This suppresses the tip loss towards blade tip, which approximately models a shrouded rotor.

hasSquareTips

`public boolean` **`hasSquareTips`**

Whether the design method should produce a blade with pointed or square tips. If set to true, the tip chord length will set to 90% of the value linearly extrapolated from the two next to last stations. If this extrapolation would lead to a negative tip chord length, the chord length of 0 is retained.

theAFStrak

`public MH.AeroTools.Airfoils.AirfoilStrak` **`theAFStrak`**

Interpolates the airfoils over the radius r/R.

---

theAlfaStrak

`public MH.AeroTools.Util.ValueStrak` **`theAlfaStrak`**

Interpolates the design angle of attack over the radius r/R.

Omega

`public double` **`Omega`**

The angular velocity. Omega = 2*PI*(n/60) in [rad/s].

V

`public double` **`V`**

The axial (flight) speed in [m/s].

Power

`public double` **`Power`**

The power consumed by the propeller under the "current "operating conditions in [W]. The "current" operating conditions are defined by the values set for `V`, `Diameter`, `Omega` and `Density`.

Thrust

`public double` **`Thrust`**

The thrust produced by the propeller under the "current" operating conditions in [N]. The "current" operating conditions are defined by the values set for `V`, `Diameter`, `Omega` and `Density`.

Radius

`public double` **`Radius`**

The radius of the propeller in [m]. `R=D/2`.

---

rRSpinner

`public double` **`rRSpinner`**

The relative spinner radius r/R [-]. `rRSpinner = RSpinner / R = DSpinner / D`.

PC

`public double` **`PC`**

The power coefficient PC [-].

```
PC = 2 * P / (Density * V^3 * PI * R^2)
```

CP

`public double` **`CP`**

The power coefficient CP [-].

```
CP = P / (Density * n^3 * D^5)
```

CS

`public double` **`CS`**

The speed power coefficient CS [-].

```
CS = v * (Density / (P * n^2))^5
```

TC

`public double` **`TC`**

The thrust coefficient TC [-].

```
TC = 2 * T / (Density * V^2 * PI * R^2)
```

CT

`public double` **`CT`**

The thrust coefficient CT [-].

```
CT = T / (Density * n^2 * D^4)
```

Eta

`public double` **`Eta`**

The propulsive efficiency [-]
This efficiency is defined by:
`Eta = v*T/P`
In case of windmilling operation (`Power < 0`) the efficiency is calculated as the ratio of delivered power to the power delivered by the ideal wind mill according to Albert Betz (1926):
`Eta = P / ((16/27)*Density/2 * V^3 *D^2 * PI / 4)`

EtaPossible

`public double` **`EtaPossible`**

The maximum possible efficiency for the same thrust according to momentum theory [-].

Density

`public double` **`Density`**

The density of the medium in [kg/m^3]. Will be used for the calculation of thrust and power.

KinematicViscosity

`public double` **`KinematicViscosity`**

The kinematic viscosity of the medium in [m^2/s]. Will be used for the calculation of the Reynolds number distribution.

SpeedOfSound

`public double` **`SpeedOfSound`**

The speed of sound of the medium in [m/s]. Is used for the determination of the Mach number distribution.

ThreadLoc

`public double` **`ThreadLoc`**

The threading line x/c of the sections along the radius [-]. All airfoil sections are threaded onto a straight radial line. This value defines the point where this line intersects the x-axis of each airfoil section. A value of 0.0 produces a straight leading edge, whereas a value of 1.0 corresponds to a straight trailing edge. Note that the blade element method as implemented in the Propeller object does not take planform details into account. They are only used for graphical output.

---

rR

`public double[]` **`rR`**

The distribution of the relative radius r/R along radius [-].

Beta

`public double[]` **`Beta`**

The distribution of the blade angle ß along the radius [rad].

Chord

`public double[] Chord`

The distribution of the blade chord length c/R along the radius [-].

Alfa

`public double[] Alfa`

The distribution of the angle of attack along the radius [rad].

Cl

`public double[] Cl`

The distribution of the lift coefficient Cl along the radius [-].

Cd

`public double[] Cd`

The distribution of the drag coefficient Cd along the radius [-].

Cx

`public double[] Cx`

The distribution of the tangential force coefficient Cx along the radius [-].

Cy

`public double[] Cy`

The distribution of the axial force coefficient Cy along the radius [-].

dCT

`public double[] dCT`

The distribution of the local thrust coefficient dCT/d(r/R) along the radius [-].

dCP

`public double[] dCP`

The distribution of the local power coefficient dCP/d(r/R) along the radius [-].

dEta

`public double[] dEta`

The distribution of the local efficiency (v/nD) * dCT/dCP along the radius [-].

CQx

`public double[] CQx`

The distribution of the shear force coefficient Qx along the radius [-].

CQy

`public double[] CQy`

The distribution of the shear force coefficient Qy along the radius [-].

CMx

`public double[] CMx`

The distribution of the in plane moment coefficient Mx along the radius [-].

CMy

```
public double[] CMy
```

The distribution of the out of plane moment coefficient My along the radius [-].

A

```
public double[] A
```

The distribution of the axial induction factor a along the radius [-].

APrime

```
public double[] APrime
```

The distribution of the circumferential induction factor a' along the radius [-].

U

```
public double[] U
```

The distribution of the axial velocity ratio u/v along the radius [-].

Re

```
public double[] Re
```

The distribution of the Reynolds number Re along the radius [-].

Ma

```
public double[] Ma
```

The distribution of the Mach number M along the radius [-].

isValid

```
public boolean isValid
```

Constructor Detail

Propeller

```
public Propeller(int BladeSections,
                 int AirfoilSections)
```

Creates a Propeller having the given number of blade and airfoil definition sections. The blade is defined by a number of BladeSections. Each of these sections defines the middle section of a blade element. Only for the first (root) and last (tip) element these sections are located at the ends of the element. These two elements having indices [0] and [BladeSections-1] have half the width of the regular elements.
The number of airfoil sections and blade sections are usually not identical. After creating a Propeller object, the airfoil sections should be defined by calling the setAirfoil() method for each airfoil section. This creator method inserts the flat plate airfoil #1 at an angle of attack of 3° as the default airfoil distribution.
The actual blade geometry can then be created in two ways:

- by calling the optimum propeller design method, or
- by specifying the geometry of a given propeller.

Geometry by Optimum Design

For the optimum design a distribution of the design angle of attack must be prescribed along the radius. This angle defines a design lift coefficient for each station. Use the setAlfa() method for this purpose.

```
// create a propeller object
// the blade will be discretized by BLADE_SECTIONS elements
// the airfoil distribution will be defined
// by AIRFOIL_SECTIONS sections
Propeller thePropeller = new Propeller(BLADE_SECTIONS,
                                       AIRFOIL_SECTIONS);

// air speed in [m/s]
double RPM = 20000.0;
// propeller diameter in [m]
double Diameter = 0.28;
// speed of rotation in [1/min]
double Velocity = 40.0;
// design power in [W]
double Power = 2000.0;
// propeller thrust in [N] (set either Power or Thrust to 0)
double Thrust = 0.0;

// derived parameters
// speed of rotation in [rad/s]
double Omega = 2.0 * Math.PI * RPM / 60.0;
// propeller radius in [m]
double Radius = Diameter / 2.0;

// define the airfoil distribution by AIRFOIL_SECTIONS,
// airfoil #1 is the first airfoil in JavaProp
//            (flat plate, Re=100'000)
// section #0 at r/R=0.0
thePropeller.setAirfoil(0, 0.0, 1);
// airfoil #2 is the second airfoil in JavaProp
//            (flat plate, Re=500'000)
// section #1 at r/R=1.0
thePropeller.setAirfoil(1, 1.0, 2);

// DEBUG: output the names of the airfoil sections
System.out.println(thePropeller.getAirfoil(0.0));
System.out.println(thePropeller.getAirfoil(1.0));

// define the design angles of attack
// section #0 at r/R=0.0
thePropeller.setAlfa(0, 0.0, 3.0);
// section #1 at r/R=1.0
thePropeller.setAlfa(1, 1.0, 3.0);
// create the propeller geometry
thePropeller.Blades = 2;
thePropeller.performPropellerDesign(Velocity, Omega, Radius,
                                    Power, Thrust);
// performance parameters can be obtained via the following
// class members:
// Lambda, PC, CP, TC, CT, Eta, EtaPossible
```

Given Geometry

The geometry can be prescribed by specifying chord length and blade angle along the radius. Use the Interpolate() method to interpolate any arbitrary distribution to the blade sections of the Propeller object.

Parameters:
`BladeSections` - int - the desired number of blade sections. Sufficiently accurate results are obtained with 25 to 50 blade elements.
`AirfoilSections` - int - the desired number of airfoil sections. Usually 2-4 sections are sufficient to define the airfoil distribution along the radius.

Method Detail

interpolateGeometry

```
public void interpolateGeometry(double[] R,
                                double[] Chord,
                                double[] BetaDeg,
                                int nSections)
```

Populates this Propeller object by interpolation of the geometry supplied in the given control sections. These sections are defined by radius, chord and blade angle. The method is used to prepare the analysis of a given propeller geometry instead of using the performPropellerDesign() or performWindmillDesign() methods to create a propeller or windmill.
The method normalizes the input values to unit radius so that the units of R and Chord are arbitrary.
Parameters:
`R` - double[] - an array containing the radial stations (ordered from root to tip).
`Chord` - double[] - an array containing the chord length at each radial station (ordered from root to tip).
`BetaDeg` - double[] - an array containing the blade angle in degrees at each station (ordered from root to tip).
`nSections` - int - the number of stations in the 3 arrays above

getActivityFactor

```
public double getActivityFactor()
```

Determine the Activity factor AF.
Returns:
double - the Activity factor of the complete propeller (NOT per blade).

getSolidity

```
public double getSolidity()
```

The solidity is the sum of area of all blades divided by disc area.
Returns:
double - the solidity of the complete propeller (NOT per blade).

setAirfoil

```
public void setAirfoil(int nSection,
                       double Position,
                       MH.AeroTools.Airfoils.Airfoil theAirfoil)
```

Places the given airfoil at the given position of the airfoil strak.
Parameters:
`nSection` - int - zero-based section number, must be lower than the value used for nSections in the constructor of this propeller object.

`Position` - double - the radial station r/R of this section. Must be in the range 0.0 ... 1.0.

`theAirfoil` - int - the airfoil object to be used at this section

See Also:

`getAirfoil(double Position)`

---

getAirfoil

```
public java.lang.String getAirfoil(double Position)
```

Returns the name of the airfoil at a given radial position.

Parameters:

`Position` - double - the relative station r/R.

Returns:

String - the airfoil name. Returns "interpolated", when there is no definition section at the given station.

See Also:

`setAirfoil(int nSection, double Position, int nAirfoil)`

---

setAlfa

```
public void setAlfa(int nSection,
                    double Position,
                    double AlfaDeg)
```

Sets the angle of attack for the given section. Used to prepare a subsequent call of the performPropellerDesign or the performWindmillDesign methods.

Parameters:

`nSection` - int - zero-based section number, must be lower than the value used for nSections in the constructor of this propeller object.

`Position` - double - the radial station r/R of this section. Must be in the range 0.0 ... 1.0.

`AlfaDeg` - double - the angle of attack at this station in [degrees].

---

setVelocityProfile

```
public void setVelocityProfile(double[] vV)
```

Define an axially symmetric velocity profile v/V of the onset flow.

Parameters:

`vV` - double[] - velocity ratio (1.0 = undisturbed flow, vV < 1 = slowed down inflow. This array must have the length `BladeSectionCount` so that a value is defined for each station.

---

performPropellerDesign

```
public int performPropellerDesign(double vDesign,
                                  double omegaDesign,
                                  double radiusDesign,
                                  double powerDesign,
                                  double thrustDesign)
```

Perform the design of a minimum induced loss propeller. Can perform a design either for given shaft power or given thrust. If a design for prescribed power consumption is desired, thrust must be set to zero and vice versa.

Parameters:

`vDesign` - double - the design speed in [m/s].

`omegaDesign` - double - the design speed in [rad/s].

radiusDesign - double - the radius of the propeller in [m].
powerDesign - double - the design shaft power in [W] or zero.
thrustDesign - double - the design thrust in [N] or zero.
Returns:
int - return code: RET_SUCCESS or RET_FAILED.

---

performWindmillDesign

```
public int performWindmillDesign(double V,
                                 double Omega,
                                 double Radius)
```

Design an optimum windmill with swirl losses according to Wiederhöft.
Parameters:
V - double - wind speed
Omega - double - the design speed [rad/s]
Radius - double - the radius of the propeller [m]
Returns:
int - return code: RET_SUCCESS or RET_CL_TOO_LOW

---

performWindmillDesign_2

```
public int performWindmillDesign_2(double V,
                                   double Omega,
                                   double Radius)
```

Design an optimum windmill with swirl losses according to Glauert. Note that friction drag is neglected, and therefore the design may not work as a windmill when analyzed afterwards. Then the analysis returns an efficiency of zero (negative thrust, but still positive power). This is usually the case, when the asvance ratio is too small (tip speed ratio too large) for the given aerfoil L/D ratio.
Parameters:
V - double - wind speed
Omega - double - the design speed [rad/s]
Radius - double - the radius of the propeller [m]
Returns:
int - return code: RET_SUCCESS or RET_CL_TOO_LOW

---

setPolarsInverted

```
public void setPolarsInverted(boolean isInverted)
```

Set whether polars are to be used in "upside down" form.
Parameters:
isInverted - boolean - true if upside down mode shall be used (wind turbine).

---

getPolarsInverted

```
public boolean getPolarsInverted()
```

Returns:
boolean - true if upside down mode is used (wind turbine).

---

performAnalysis

```
public boolean performAnalysis(double V,
                               double Omega,
                               double Radius,
```

```
                              double Density,
                              double KinematicViscosity,
                              double SpeedOfSound)
```

Performs the analysis of this propeller for a given advance ratio J. The results are stored in the propeller object and can be retrieved after returning from this method. The values for `V`, `Omega`, `Radius`, `Density`, `KinematicViscosity` and `SpeedOfSound` are stored in this Propeller object. Any subsequent analysis with the method performAnalysis(double VnD) will use these values.

Parameters:

`V` - double - The axial (flight) speed in [m/s].

`Omega` - double - The angular velocity. Omega = 2*PI*(n/60) in [rad(s].

`Radius` - double - The radius of the propeller in [m]. R=D/2.

`Density` - double - The density of the medium in [kg/m^3].

`KinematicViscosity` - double - The kinematic viscosity of the medium in [m^2/s].

`SpeedOfSound` - double - The speed of sound of the medium in [m/s].

Returns:

boolean - see performAnalysis(V/nD)

---

performAnalysis

`public boolean` **`performAnalysis`**`(double VnD)`

Performs the analysis of this propeller for a given advance ratio J. The results are stored in the propeller object and can be retrieved after returning from this method. besides the coefficients, `Thrust` and `Power` are calculated. For this purpose the values set for `V`, `Omega`, `Radius` and `Density` are used. The current setting for `KinematicViscosity` is used to fill the `Re[]` array and `SpeedOfSound` provides the base for the results in `Ma[]`.

A typical application would be to call this method in a loop, incrementing VnD until the efficiency Eta becomes negative.

```
// assume a Propeller object has been created and populated by
performPropellerDesign() or by specifying a geometry.

// assume constant RPM for this analysis
double RPM = 20000.0;

// loop over advance ratio J = v/(n*D)
for ( double VnD=0.0 ; VnD < 100.0 ; VnD += 0.05 )
{
    // do the analysis for this advance ratio
    // will produce CT and CP as well as thrust and
    // power for whatever values are currently set for
    // Omega, Vms and Diameter the Propeller object.
    thePropeller.performAnalysis(VnD);
    double V = VnD * (RPM / 60.0 * Diameter);
    double T = CT * Density * Math.pow(RPM / 60.0, 2) *
                              Math.pow(Diameter, 4);
    double P = CP * Density * Math.pow(RPM / 60.0, 3) *
                              Math.pow(Diameter, 5);
    if ( thePropeller.Eta < 0.0 )
       break;
}
```

Parameters:

`VnD` - double - the advance ratio J=v/(nD)

Returns:
boolean - always true)

---

getBladeAngle

`public double getBladeAngle()`

Returns the nominal blade angle at 75% of the radius. Note that the blade angle is referred to the x-axis of the local airfoil section, which is not the possibly flat lower surface.

Returns:
double - the blade angle in degrees.

---

setBladeAngle

`public double setBladeAngle(double AngleDeg)`

Sets the nominal blade angle at 75% of the radius.

Parameters:
`AngleDeg` - double - the blade angle in degrees.

Returns:
double

---

getBladePitch

`public double getBladePitch()`

Returns the nominal blade pitch H at 75% of the radius.

Returns:
double - the pitch height.

---

incrementBladeAngle

`public void incrementBladeAngle(double DeltaDeg)`

Increments the local blade angle ß by a constant Delta-ß. Can be used to simulate a variable pitch propeller.

Parameters:
`DeltaDeg` - double - the angle increment in degrees (can also be negative).

---

multiplyBladeAngle

`public void multiplyBladeAngle(double AngleFactor)`

Multiplies the local blade angle ß with a constant ß-Factor.

Parameters:
`AngleFactor` - double - the blade angle multiplier.

---

setBladeAngleConstant

`public void setBladeAngleConstant(double AngleDeg)`

Define a blade having constant blade angle.

Parameters:
`AngleDeg` - double - blade angle in degrees.

---

twistBlade

`public void twistBlade(double TwistAngleDeg)`

Modifies the local blade angle beta by a linear twist. The twist angle varies from 0 at r/R = 0 to TwistAngleDeg at r/R = 1.

Parameters:

`TwistAngleDeg` - double - the twist angle at the tip section.

---

straightBlade

```
public void straightBlade(double RootChord,
                          double RootAngleDeg,
                          double TaperRatio,
                          double TwistAngleDeg)
```

Create a geometry from straight generator lines connecting root and tip section.

Parameters:

`RootChord` - double - the relative chord length c/R at r/R = 0.

`RootAngle` - double - the blade angle beta at r/R = 0 in degrees.

`TaperRatio` - double - the taper ratio ratio c/R(r/R=1) / c/R(r/R=0).

`TwistAngleDeg` - double - the twist angle beta(r/R=1) - beta(r/R=0) in degrees.

---

straightBlade

```
public void straightBlade(double TaperRatio,
                          double TwistAngleDeg)
```

Create a geometry from straight generator lines connecting root and tip section. Twist and taper distribution are based on the current chord length Chord[0] and the blade angle Beta[0].

Parameters:

`TaperRatio` - double - the taper ratio ratio c/R(r/R=1) / c/R(r/R=0).

`TwistAngleDeg` - double - the twist angle beta(r/R=1) - beta(r/R=0) in degrees.

---

incrementChord

```
public void incrementChord(double DeltaChord)
```

Increments the local chord c/R by a constant Delta-c/R.

Parameters:

`DeltaChord` - double - the additional c/R.

---

multiplyChord

```
public void multiplyChord(double ChordFactor)
```

Multiply the current local chord c/R by a constant factor.

Parameters:

`ChordFactor` - double - the multiplier for the chord distribution.

---

taperChord

```
public void taperChord(double TaperRatio)
```

Multiply the local chord c/R by a linear varying c/R-Factor. The tapering factor varies from 1.0 at the root to TaperRatio of the local chord at the tip.

Parameters:

`TaperRatio` - double - the taper ratio at the tip section.

---

setChord

```
public void setChord(double ChordLength)
```

Define a blade having a constant chord length.

Parameters:

`ChordLength` - double - the c/R ratio.

---

toDXF3D_Plane

```
public java.lang.String toDXF3D_Plane()
```

Create a 2D DXF representation of the propeller twist distribution.

Returns:

String - a geometry in AutoCad DXF format.

---

toDXF3D

```
public java.lang.String toDXF3D()
```

Create a 3D representation of the propeller geometry.

Returns:

String - a simple model of the blade surface in AutoCad DXF format.

---

toIGES

```
public java.lang.String toIGES()
```

Converts the 3D airfoil sections of this propeller into an IGES representation.

Returns:

String - contains the propeller geometry in IGES format.

---

toText

```
public java.lang.String toText(boolean IncludeHeader,
                               boolean IncludeAeroData,
                               java.lang.String LineStart,
                               java.lang.String LineEnd,
                               java.lang.String Separator)
```

Creates a tabular representation of the propeller object. The returned string contains a table organized in columns which are separated by a given separator characters. The columns contain

- r/R - the local relative radius
- c/R - the local relative chord length
- ß - the local blade angle in degrees

Parameters:

`IncludeHeader` - boolean - if set to true, a header block with column titles is included. The tokens are separated by the same strings as they are used for the data lines (see below).

`IncludeAeroData` - boolean - if set to true, additional columns are appended to each line. When called immediately after a design, these columns contain aerodynamic and loads data valid for the condition used for `performPropellerDesign()`. Otherwise the table contains the results of the last call to one of the `performAnalysis()` methods.

- alfa - the local angle of attack in degrees
- Cl - the local lift coefficient
- Cd - the local drag coefficient
- a - the local axial induction factor

- a' - the local circumferential induction factor
- Re - the local Reynolds number
- Ma - the local Mach number
- Cx - the elemental force coefficient in tangential direction (contributes to torque)
- Cy - the elemental force coefficient in normal direction (contributes to thrust)
- CQx - the shear force coefficient in tangential direction (due to torque)
- CQy - the shear force coefficient in normal direction (due to thrust)
- CMx - the bending moment coefficient in tangential direction (due to torque)
- CMy - the bending moment coefficient in normal direction (due to thrust)

`LineStart` - String - a string to be inserted at the start of each line.
`LineEnd` - String - a string to be appended to the end of each line.
`Separator` - String - the string to be used for separating columns.

Returns:

String - a multi-line string. A typical result of a call of `toText(false, ";")` may look like:

```
N;r/R;c/R;ß
1;0.0;0.0;0.0
2;0.05;0.0;0.0
... lines omitted ...
20;0.95;0.038477771508231735;17.758865921679522
21;1.0;0.0;17.007335190131688
```

A call of the form `toText(true, false, "<station><data>", "</data></station>\n", "</data><data>"))` may produce XML style output like:

```
<station><data>N</data><data>r/R</data><data>c/R</data><data>ß</data>
</station>

<station><data>1</data><data>0.0</data><data>0.0</data><data>0.0</dat
a></station>

<station><data>2</data><data>0.05</data><data>0.0</data><data>0.0</da
ta></station>
   ... lines omitted ...

<station><data>20</data><data>0.95</data><data>0.038477771508231735</
data><data>17.758865921679522</data></station>

<station><data>21</data><data>1.0</data><data>0.0</data><data>17.0073
35190131688</data></station>
```

Method added 24 November 2007 by MH.

---

isActiveElement

```
public boolean isActiveElement(double rR)
```

Parameters:

`rR` - double

Returns:

boolean

---

toXML

```
public java.lang.String toXML()
```

Returns:
String

---

whatIsIt

```
public java.lang.String whatIsIt()
```

Find out what the current propeller is. It can be either

- a propeller, delivering thrust,
- a wind turbine, delivering power, or
- something in between, consuming power, producing drag.

Returns:
String - a string explaining what kind of machine this propeller is.

---

getRPM

```
public double getRPM()
```

Returns:
double - the speed in units of revolutions per minute [1/min]

---

getVnD

```
public double getVnD()
```

Returns:
double - the advance ratio J = v/(n*D)

---

getLamda

```
public double getLamda()
```

Returns:
double - the advance ratio v/(Omega*R)

---

getLambda

```
public double getLambda()
```

Returns:
double - the advance ratio v/(Omega*R)

---

getTSR

```
public double getTSR()
```

Returns:
double - the tip speed ratio TSR = Omega*R/v

---

getTorque

```
public double getTorque()
```

Returns:
double - the torque [Nm]

---

getPower

```
public double getPower()
```

Returns:
double - the power [W]

---

getThrust

```
public double getThrust()
```

Returns:
double - the thrust [N]

---

getDiameter

```
public double getDiameter()
```

Returns:
double - the diameter [m]

References

[1] A. Betz: “Schraubenpropeller mit geringstem Energieverlust” (with an Addendum by L. Prandtl) in “Vier Abhandlungen zur Hydrodynamik und Aerodynamik”, reprint of the 1927 edition, Göttingen, 1944.

[2] H.Glauert: “Die Grundlagen der Tragflügel und Luftschraubentheorie”, Translation by H. Holl, Berlin 1929.

[3] H. B. Helmbold: “Über die Goldstein’sche Lösung des problems der Luftschraube mit endlicher Flügelzahl”, Zeitschrift für Flugtechnik und Motorluftschiffahrt, 14. Heft, 1931.

[4] Larrabee, E., “Practical Design of Minimum Induced Loss Propellers”, Society of Automotive Engineers Business Aircraft Meeting and Exposition, Wichita, KS, SAE paper 790585, April 3-6, 1979.

[5] Adkins, Liebeck, N., R. H., “Design of Optimum Propellers”, Journal of Propulsion and Power, Vol. 10, No. 5, 1994, pp. 676-682.

[6] Adkins, N., Liebeck, R. H., “Design of Optimum Propellers”, 21st Aerospace Sciences Meeting, Reno, AIAA Paper 83-0190, Jan. 1983.

[7] Wald, Q. R., “The aerodynamics of propellers”, Progress in Aerospace Sciences, 42, 2006, pp. 85-128, doi:10/1016/j.paerosci.2006.04.001.

[8] R. Eppler and M. Hepperle: “A procedure for Propeller Design by Inverse Methods”, in G.S. Dulikravich: Proceedings of the “International Conference on Inverse Design Concepts in Engineering Sciences” (ICIDES), pp. 445-460, Austin, October 17-18, 1984.

[9] K. Wiederhöft, “60 Jahre nach Hütter. Aerodynamische Radialschnitttheorie für Windenergieanlagen”, DEWEK 2002.

[10] GNU Octave: see http://www.octave.org [retrieved 20 December 2010].

[11] Mathworks Matlab: see http://www.mathworks.com/ [retrieved 20 December 2010].

[12] Wolfram Mathematica: see http://www.wolfram.com/ [retrieved 20 December 2010].

Localization

The language resources of JavaProp have been translated by the persons listed in the table below.

Language	Code	Author	Year

English	en	Martin Hepperle, Braunschweig, Germany	2001
German	de	Martin Hepperle, Braunschweig, Germany	2001
French	fr	Giorgio Toso, Montreal, Canada	2002
Italian	it	Giorgio Toso, Montreal, Canada	2002
Portuguese	pt	João Alveirinho Correia, Portugal	
Simplified Chinese	zh_CN	J. X. Ding, Taipei, Republic of China	2011
Traditional Chinese	zh_TW	J. X. Ding, Taipei, Republic of China	2011

Incomplete Version History

Version	**Date**	**Comments.**
1.57	25. Aug 11	Added Chinese language features.
		Added some additional resource strings.
		Cleaned up expiration reminder code.
		Added saving and restoring of most GUI settings on exit and startup to application preferences.
		Added styled labels to graphs for sub- and superscripts to mhclasses.jar.
1.56	07. Jan 11	Introduced sweep angle as a function of local Mach number
1.55	30. Jun 10	Fix: airfoil list was not updated when working directory was changed by WorkDir=.... command line option.
		Fix: Geometry table was not updated when airfoil was selected.
		PROPPY compatible XML export of propeller geometry added.
1.54	31. Mar 2010	Added local thrust, power, efficiency added to single analysis card, windmill design method Wiederhöft added.
1.53	01. Jan 10	Refined Multi-Analysis card, added V=const. analysis mode.
1.52	10. Oct 2009	IGES export of 3D blade section geometry added
1.51	20. Sep 09	Added shear force and bending moment graphs to single analysis card.
1.5	02. Sep 09	Added windmill design and analysis.
1.41		Additional output of shear force and bending moment coefficients on single analysis card.
1.4		Refinement of Multi-Analysis card, Q=const. added.
1.39		Refinement of Multi-Analysis card, RPM graph added.
1.38		Fix: error in save/restore of Torque setting on design card
1.37		Fix: error in DXF 3D export (spinner was neglected in
		DXF j-index).
1.36		Fix: angle of attack transfer error on Airfoil card.
		Added iteration on Design card so that analysis matches design parameters accurately.
1.35		Modified clipping of induction factor a during analysis.
1.33		Extended Single-Analysis CopyText and CopyHTML commands to include more data.
1.3		Changed XYCanvas sizing.
1.29		Fix: Error in Modify card: angle always went to 90°
1.27		Fix: Flow Field card calculation of "a" (dA).
1.26		Added Flow Field card.

1	2001	Initial version