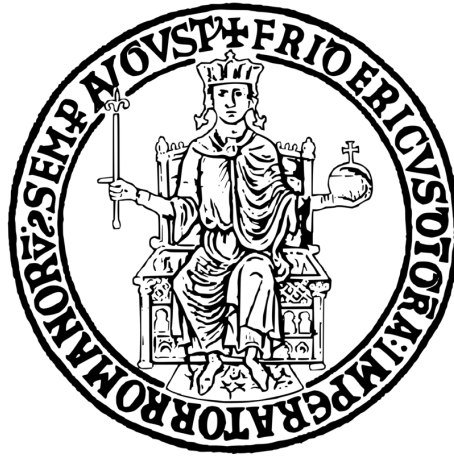


University of Naples Federico II



# User's Guide of Eli-TAARG

Department of Aerospace Engineering

# List of Functions

Page N.

Introduction	1
Cl-Cd Xrotor function	3
Flapping angles	6
RotorFF function	9
Geometry reader function	12
Optimum propeller	21
Geometry input	29
Output function	31
Multiple streamtube Darrieus turnines function	34

## Introduction

---

The main scope of this library is the collection of MatLab's functions for educational purpose. The manual could be considered as a guide to obtain the principal results of the aerodynamic rotary wing theory. In the library, the functions are meant to evaluate different aspects of rotary wing. It is possible to assign geometrical information about different propellers and more specific information as flapping angles. Also there will be the possibility to get Cl-Cd polars thanks to XFoil implementation, characteristics curves and to realize optimal propeller design function. The documentation has been developed by the students of the Department of Aerospace Engineering, University of Naples Federico II, over the course of the Aerodynamic Rotary Wing lectures in the 2020/2021 academic year.

Eli-TAARG is a free software. You can distribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. Eli-TAARG is developed by the TAARG Educational organization for learning purposes only. Theoretical and Applied Aerodynamic Research Group - University of Naples Federico II.

Eli-TAARG is distributed in the hope that it will be useful, but without any warranty. It lacks the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details: <http://www.gnu.org/licenses>

## Functions Description

---

The following functions are represented in all their characteristics in the other pages of the paper. The main scope for each function is:

### **- Cl-Cd XRotor function:**

This function evaluates the aerodynamic model of a propeller's or rotor's element blade, according to the software XRotor by Prof. M. Drela at MIT. The theory uses the following law to build the polar parabola representing the drag coefficient as function of the lift coefficient. The meaning of each one of these quantities is presented in the I/O section.

### **-Flapping angles function:**

This script evaluates a rotor blade's flapping angles as a function of the azimuth angle and its derivative. In this scope, the first harmonic flapping motion has been determined with the assumptions based on small flapping angles and linear aerodynamics.

**-RotorFF function:**

This function returns the characteristic curves for rotor in forward flight for both constant thrust and power and gives in output also the relative x and y values. It requires as input the angle of attack expressed in degrees.

**-Geometry reader:**

This function has the scope to read a standard text file (which the user has to compile with the geometry informations about the propeller) and to allocate the geometry parameters in local variables. Moreover, the class has been provided with two more functions that allow to analyze the propeller thanks to xrotor and also to plot the main results.

**-Optimum propeller design function:**

This function allows to design an optimum propeller by considering the approximated optimum propeller theory developed by Prandtl. It is possible to suppose that the propeller is lightly loaded, the number of blades has been considered of number  $N$  much higher than 1, in order to neglect the wake contraction. By considering the over mentioned hypothesis we can define the thrust  $T$  and the power  $P$  distributions along the non-dimensional radius as functions of the rotational speed, the number of blades and two induction factors  $a$  and  $a'$  that will be analyzed in the relative sections.

**-Input data function:**

This function provide the geomtry input. Starting from a database, a file *.txt* with all the propeller/rotor/turbine geometry information, the function provides different variables that can be needed for different analysis.

**-Output data function:**

Using the vectors of the other functions of the library as input parameters, this function, suitably called by the others, has two fundamental purposes:

1. generate a plot (or more plots) relating to the specific operating curves of propellers, rotors and turbines;
2. generate a text file with all output vectors in column.

**- Multiple streamtube Darrieus turbines function:**

This function implements the multiple streamtubes theory which gives a more accurate prediction of the wind velocity variations across the Darrieus rotor with respect to the single streamtube model.

## Cl-Cd XRotor function

---

This function evaluates the aerodynamic model of a propeller's or rotor's element blade according to the software XRotor [1], by Prof. M. Drela at MIT. The theory uses the following law to build the polar parabola representing the drag coefficient as function of the lift coefficient. The meaning of each one of these quantities is presented in the I/O section.

$$C_d = \{Cd_{min} + \frac{dC_d}{dC_l^2} \cdot [C_l(Cd_{min}) - C_l]^2\} \cdot \left(\frac{Re_{\infty}}{Re_{ref}}\right)^f \quad (1)$$

### I/O

---

The function is intended to take in input the following values. Note that, if the contrary is not stated, the values are represented by scalar quantities.

- $Cd_{min}$  minimum blade element's drag coefficient
- $(dC_d)/(dC_l^2)$  quadratic coefficient of the parable that approximates the  $C_d(C_l)_{law}$
- $C_l(Cd_{min})$  blade element's lift coefficient for which the drag coefficient assumes its minimum value
- $Re_{ref}$  blade element's Reynolds number computed taking into account the radius at which the blade element is intended to be (i.e. the reference velocity is  $\Omega r$  where  $\Omega$  is the propeller's angular velocity and  $r$  is the radial position of the blade element taken into account)
- $Re_{\infty}$  asymptotic Reynolds number of the phenomena
- $f$  Reynolds number scaling exponent, according to XRotor documentation
- $Cl_{max}$  maximum blade element's lift coefficient
- $Cl_{min}$  minimum blade element's lift coefficient
- $Cl_{breakpoints}$  blade element's lift coefficient breakpoints; this input can be both a vector or a single value; a breakpoint is the value at which the drag coefficient evaluation is requested, according to the  $C_d(C_l)$  law built through the function script

Input data inserting mode is very strict: the function must be carefully called because the order of the input vectors can only be the following one. Input data given in a different order could result in wrong outputs. The function takes 2 inputs; the first one must be ordered in the following way:

1.  $Cd_{min}$
2.  $dCd_dCl2$
3.  $ClCd_{min}$

4.  $Re_{ref}$

5.  $Re_{inf}$

6.  $f$

7.  $Cl_{max}$

8.  $Cl_{min}$

the second input can be either a vector or a single value, representing the blade element's lift coefficient breakpoints (i.e. it is the variable previously called  $C_l$  breakpoints)

The outputs of the function are:

- $C_{lvector}$  this is the lift coefficients vector used to build the  $C_d(C_l)$  law; it is through this vector that the parable is plotted
- $C_d$  this is the drag coefficients vector evaluated at  $C_l$  vector
- $C_d$  breakpoints this is a vector or a single value, depending on the nature of the  $C_l$  breakpoints input, whose components are the drag coefficients requested at lift coefficients breakpoints

#### Algorithm Description

---

Once the inputs are stored, the script firstly generates a lift coefficient vector through the line:

$$Cl_{vec} = Cl_{min} : 0.01 : Cl_{max};$$

and then builds the aerodynamic model implementing the above-mentioned formula:

$$Cd = (Cd_{min} + dCd_dCl2 * (ClCd_{min} - Cl_{vec}).^2) * (Re_{inf}/Re_{ref})^f;$$

After these first passages, the script evaluates the requested drag coefficients breakpoints at the given lift coefficients input values. To do that, the code firstly define an anonymous function through the handle command in MATLAB using the built-in function `interp1` applying a piecewise cubic Hermite interpolating polynomial, and then uses this interpolation to evaluates the desired drag coefficients values. The lines referred in this explanation are:

$$Cd_{interp} = @(Cl_{interp})interp1(Cl_{vec}, Cd, Cl_{interp}, 'pchip'); Cd_{bp} = Cd_{interp}(Cl_{bp});$$

The function finally plots the  $C_d(C_l)$  law.

#### Example

---

An useful test case input is represented by the following values:

$$input_{vec} = [.0068, .0023, .69, 750000, 750000, -1.5, 1.57, -.86]; Cl_{bp} = [0.8, 1]; [Cl_{vec}, Cd, Cd_{bp}] = ClCd_XRotor(input_{vec}, Cl_{bp});$$

Note that, to work, this calling must be in the same directory as the `ClCd_XRotor.m` script.

Results obtained in this case have been validated through the software XRotor itself.

Outputs from XRotor and from the `ClCd_XRotor.m` script are reported below.

It is also reported the display of the aerodynamics parameters of the blade element used in XRotor: values not modified as written above are set as default by the software XRotor.

### Code listing

```
function [varargout] = flappingangles(V,alpha,W,gamma,...
sigma,theta_tw,cla,omegaR,eR,R,options)

function [Cl_vec, Cd, Cd_bp] = ClCd_XRotor(input_v, Cl_bp)
% taking input data
Cd_min = input_v(1);
dCd_dCl2 = input_v(2);
Cl_Cd_min = input_v(3);
Re_ref = input_v(4);
Re_inf = input_v(5);
f = input_v(6);
Cl_max = input_v(7);
Cl_min = input_v(8);
% lift coefficient vector creation
Cl_vec = Cl_min:.01:Cl_max;
% aerodynamic model building
Cd = (Cd_min + dCd_dCl2*(Cl_Cd_min - Cl_vec).^2)*(Re_inf/Re_ref)^f;
%% evaluation of Cd values @ requested Cl breakpoints
% note that Cl_bp can be both a single value or a vector
% definition of an anonymous function through the handle (@) symbol
Cd_interp = @(Cl_interp) interp1(Cl_vec, Cd, Cl_interp, 'pchip');
% anonymous fcn used to compute requested Cd
Cd_bp = Cd_interp(Cl_bp);
%% plots section
% blade element's polar diagram
% along x: drag coefficient
% along y: lift coefficient
plot(Cd, Cl_vec);
end
```

## Flapping Angles

This script evaluates a rotor blade's flapping angles as a function of the azimuth angle, and its derivative. In this scope, the first harmonic flapping motion has been determined with the assumptions of small flapping angles and linear aerodynamics. **[prouty]**

$$\beta = \beta_0 + \beta_{1c} \cos(\psi) + \beta_{1s} \sin(\psi) \quad (1)$$

### Algorithm Description

The default syntax of the function is shown below; all inputs are scalar values.

```
[psi,beta,beta_dot] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,
    omegaR,eR,R)
```

Where:

INPUT	
V	Airspeed [m/s]
alpha	Angle of attack [deg]
W	Helicopter ( <i>Rotor</i> ) weight [N]
gamma	Rotor Lock number
sigma	Rotor solidity
theta_tw	Rotor blade twist ( <i>Assumed linear</i> ) [deg]
cla	Rotor blade airfoil lift curve slope [1/rad]
omegaR	Rotor blade tip speed $\Omega R$ [m/s]
eR	Adimensional flapping hinge eccentricity $e/R$
R	Rotor blade radius [m]
OUTPUT	
psi	Azimuth angles [deg]
beta	Rotor blade flapping angles [deg]
beta_dot	Rotor blade flapping angle derivative $\frac{d\beta}{d\psi}$

By default  $\psi$  is an array of 200 equally spaced points from  $0^\circ$  to  $360^\circ$ . The user may also choose to specify the query points through the '*sample*' keyword.

```
[~,beta,beta_dot] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,omegaR,
    eR,R,'sample',psi)
```

By adding '*output*','*coefficients*' to the function call, the output will change to the flapping angle coefficients used in 1. That is, respectively:

beta_0	Rotor blade coning [deg]
beta_1c	Rotor blade longitudinal flapping [deg]
beta_1s	Rotor blade lateral flapping [deg]

```
[beta_0,beta_1c,beta_1s] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,
    omegaR,eR,R,'output','coefficients')
```



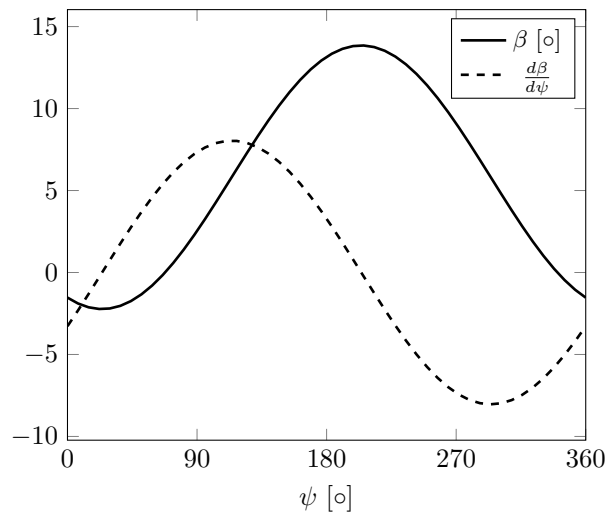
### Example

A test case for the function is shown below, where it is called to obtain the flapping angles at the requested azimuths. Also, it is possible to call the function in a cycle in order to obtain the flapping coefficients  $\beta_0, \beta_{1c}$  and  $\beta_{1s}$  as functions of the advance ratio  $\mu$ .

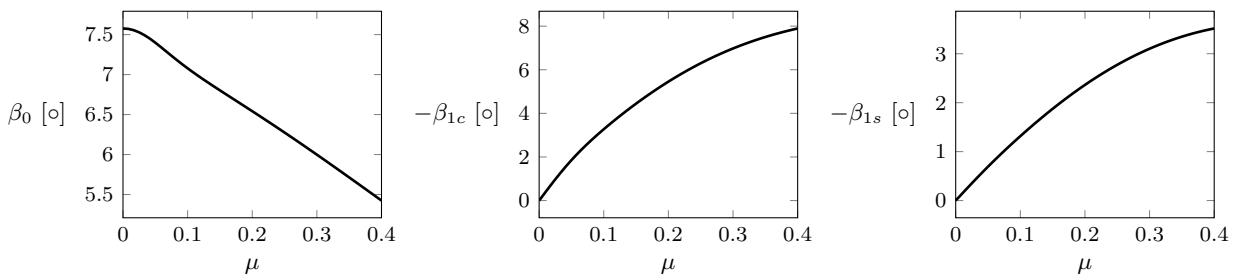
```
psi = linspace(0,360,50);
[~,beta,beta_dot] = flappingangles(100,0,33523.27,8,0.1,-8,5.7,213,0,4,...
'sample',psi);

[b0,b1c,b1s,V] = deal(linspace(0,120,200));
mi = V/213;

for i = 1:length(V)
    [b0(i),b1c(i),b1s(i)] = flappingangles(V(i)
    ,0,33523.27,8,0.1,-8,5.7,213,0,4,...
    'output','coefficients');
end
```



**Figure 1:** Flapping angles of a rotor blade



**Figure 2:** Flapping coefficients as functions of advance ratio [johnson]

## Code listing

```
function [varargout] = flappingangles(V,alpha,W,gamma,...
sigma,theta_tw,cla,omegaR,eR,R,options)

% Argument validation
arguments
V      (1,1) {mustBeNumeric,mustBeReal}
alpha  (1,1) {mustBeNumeric,mustBeReal}
W      (1,1) {mustBeNumeric,mustBeNonnegative}
gamma  (1,1) {mustBeNumeric,mustBeNonnegative}
sigma  (1,1) {mustBeNumeric,mustBeNonnegative}
theta_tw (1,1) {mustBeNumeric,mustBeReal}
cla    (1,1) {mustBeNumeric,mustBeReal}
omegaR (1,1) {mustBeNumeric,mustBeNonnegative}
eR     (1,1) {mustBeNumeric,mustBeNonnegative}
R      (1,1) {mustBeNumeric,mustBeNonnegative}
options.output (1,1) string {mustBeMember(options.output,{'angles','
coefficients'})} = 'angles'
options.sample (:,1) {mustBeNumeric,mustBeReal} = linspace(0,360,200)'
end

%% Preliminary calculations
altitude = 0;
[~,~,~,rho] = atmosisa(altitude);

% Converting to radians
theta_tw = convang(theta_tw,'deg','rad');

% Thrust coefficient
Tc = W/(rho*omegaR^2*pi*R^2);

% Advance ratio
mi = V*cosd(alpha)/omegaR;

% Adimensional inflow
lambda_i = sqrt(-.5*V.^2+.5*sqrt(V.^4+4*(W/(2*rho*pi*R^2)).^2))/omegaR;
lambda = mi.*tand(alpha)+ lambda_i;

% Collective pitch
theta_0 = 6*Tc./(sigma*cla*(1+3/2*mi.^2)*(1-eR))...
-3/4*theta_tw*(1+mi.^2)./(1+3/2*mi.^2)+1.5*lambda./(1+3/2*mi.^2);

%% Flapping coefficients

% Coning
beta_0 = 1/6*gamma*(3/4*theta_0.*(1+mi.^2)+theta_tw*(3/5+mi.^2/2)-lambda)...
*(1-eR)^2;

% Longitudinal flapping
beta_1c = -2*mi.*(4/3*theta_0+theta_tw-lambda)./(1-mi.^2/2)...
-12*eR./(gamma*(1-eR)^3*(1+mi.^4/4)).*(4/3*Tc/sigma*...
(2/3*mi*gamma/(cla*(1+3/2*eR))));

% Lateral flapping
beta_1s = -4/3*mi.*beta_0.*(1+eR/2)./((1+mi.^2/2)*(1-eR)^2)...
-12*eR*(1+eR/2)./(gamma*(1-eR)^3*(1+mi.^4/4))*...
2.*mi.*(4/3*theta_0+theta_tw-lambda);

%% Output
if strcmpi(options.output,'coefficients')
varargout = {convang(beta_0,'rad','deg'),...
convang(beta_1c,'rad','deg'),...
convang(beta_1s,'rad','deg')};
else
psi = options.sample;
beta = (beta_0 + beta_1c.*cosd(psi) + beta_1s.*sind(psi));
beta_dot = (-beta_1c.*sind(psi) + beta_1s.*cosd(psi));
varargout = {psi,convang(beta,'rad','deg'),...
convang(beta_dot,'rad','deg')};
end
end
```

## RotorFF function

---

### I/O

---

Alfa – angle of attack

Constant Thrust

Vt – asymptotic velocity

wt – induction

Pt– power

Constant Power

Vp – asymptotic velocity

wp– induction

Tt– power

All the values are non-dimensional in respect to their value in hovering (for V is used induction in hovering).

### Algorithm Description

---

[Vt,wt,Vp,wp,Pt,Tp] = RotorFF(alfa) returns the characteristic curves for rotor in forward flight for both constant thrust and power and gives in output also the relative x and y values. It requires in input the angle of attack in degrees.

The plot available are:

- For constant Thrust:

-w versus V

-P versus V

- For constant Power:

-w versus V

-T versus V

where w = induction, V = asymptotic velocity, T = Thrust, P = Power

The algorithm implements the following equations:

Constant Thrust

$$(v_{\infty}^2 w \sin \alpha + w^2) + v_{\infty}^2 w^2 \cos^2 \alpha = 1 \quad (1)$$

$$P_l = v_{\infty}^2 + \sin \alpha + w \quad (2)$$

Constant Power

$$[(v_{\infty}^2 w \sin \alpha + w^2)^2 + v_{\infty}^2 w^2 \cos^2 \alpha] + (v_{\infty}^2 \sin \alpha + w) = 1 \quad (3)$$

$$T = (v_{\infty}^2 \sin \alpha + w)^{-1} \quad (4)$$

The code begins with the function call.

```
function [Vt,wt,Vp,wp,Pt,Tp] = RotorFF(alfa)
```

The angle of attack given as input in degrees is converted in radiant.

```
alfa=deg2rad(alfa);
```

The two implicit functions are managed through a function handle.

```
f1 = @(x,y) (((x.*cos(alfa)).^2 + (x.*sin(alfa)+y).^2).*(y.^2))-1;
f2 = @(x,y) (((x.*cos(alfa)).^2 + (x.*sin(alfa)+y).^2).*(y.^2).*((x.*sin(
    alfa)+y))-1 ;
```

They are plotted using the fimplicit function.

```
h1=fimplicit(f1,[0 10 0 1],'k');
h2=fimplicit(f2,[0 5 0 1],'k');
```

After that, the values of the axis are extracted through the get function.

```
Vt = (get(h1, 'XData'));
wt = (get(h1, 'YData'));

Vp= (get(h2, 'XData'));
wp= (get(h2, 'YData'));
```

These values are used to define the remaining function and they are also available in output.

```
Pt=Vt.*sin(alfa)+wt;
Tp=(Vp*sin(alfa)+wp).^(-1);
```

These two functions are finally plotted.

```
plot(Vt,Pt,'k')
plot(Vt,Pt,'k')
```

### Code listing

```
function [Vt,wt,Vp,wp,Pt,Tp] = RotorFF(alfa)

%%Angle conversion to radiant
alfa=deg2rad(alfa);

%% Constant thrust
%%Function definition
f1 = @(x,y) (((x.*cos(alfa)).^2 + (x.*sin(alfa)+y).^2).*(y.^2))-1;

%%Plot setting
subplot(2,2,1);
h1=fimplicit(f1,[0 10 0 1],'k');
axis square;
xlabel('$\tilde{V}_{\infty}$','Interpreter','latex');ylabel('$\tilde{w}$','Interpreter','latex');
title('Characteristic curve $\tilde{w} = \tilde{w}(\tilde{V}_{\infty})$, constant Thrust', 'Interpreter','latex') ;

%%Axis values
Vt = (get(h1, 'XData'));
wt = (get(h1, 'YData'));

%%Function of Power versus speed for constant Thrust
Pt=Vt.*sin(alfa)+wt;
%%Plot setting
subplot(2,2,3);
plot(Vt,Pt,'k')
axis([0 max(Vt) 0 max(Pt)]);
axis square;
xlabel('$\tilde{V}_{\infty}$','Interpreter','latex');ylabel('$\tilde{P}$','Interpreter','latex');
```

```

title('Characteristic curve  $\tilde{P} = \tilde{P}(\tilde{V}_{\infty})$ ,
      constant Thrust', 'Interpreter','latex') ;

%% Constant Power

%Function Definition
f2 = @(x,y) (((x.*cos(alfa)).^2 + (x.*sin(alfa)+y).^2)).*(y.^2).*((x.*sin(
    alfa)+y))-1 ;

%Plot setting
subplot(2,2,2);
h2=fimplicit(f2,[0 5 0 1],'k');
axis square;
xlabel('$\tilde{V}_{\infty}$','Interpreter','latex');ylabel('$\tilde{w}$','
    Interpreter','latex');
title('Characteristic curve  $\tilde{w} = \tilde{w}(\tilde{V}_{\infty})$ ,
      constant Power', 'Interpreter','latex') ;

%Axis values
Vp= (get(h2, 'XData'));
wp= (get(h2, 'YData'));

%Function of Thrust versus speed for costant Power
Tp=(Vp*sin(alfa)+wp).^(-1);

%Plot setting
subplot(2,2,4);
plot(Vt,Pt,'k')
axis([0 max(Vp) 0 max(Tp)]);
axis square;
xlabel('$\tilde{V}_{\infty}$','Interpreter','latex');ylabel('$\tilde{T}$','
    Interpreter','latex');
title('Characteristic curve  $\tilde{T} = \tilde{T}(\tilde{V}_{\infty})$ ,
      constant Power', 'Interpreter','latex') ;

end

```

## Geometry reader function

---

The aim of this function is to read a standard text file (which the user has to compile with the geometry informations about the propeller) and to allocate the geometry parameters in local variables. Moreover, the class has been provided with two more functions that allow to analyse the propeller via *xrotor* and also to plot the main results.

### Algorithm Description

---

For the correct usage of the tool, user has to put in the same folder:

- the "*geometryreader.m*" class (whose Matlab script is fully reported in appendix );
- a standard text file;
- the *xrotor* executable;
- a "*main.m*" Matlab file (whose script is showed in the listing below) in which user can choose the desired operations.

```
PropDataFileName='propgeometry.txt';    %.txt standard geometry file
myProp=geometryreader(PropDataFileName);    %class call
myProp.xRotorAnalysis;
myProp.plotResults;
```

The user has three usage options:

- by typing the first two lines of the listing, the class will just read the input file and allocate the parameters in local variables;
- by typing the first three lines of the listing, the class will interface with *xrotor.exe*, analyze the geometry and allocate the results in local variables;
- by typing the first two and the last lines of the listing, the class will also plot the results of the analysis.

In figure fig:standard text file is showed the standard text file that the user has to compile with the features of the propeller.

#### PROPELLER MAIN GEOMETRIC DATA

2 Number of blades  
0.81 Tip radius, (m)  
0.14 Hub radius, (m)  
0 Hub wake displacement body radius, (m)

#### PROPELLER PERFORMANCE

0 Thrust, (N)  
55000 Power, (W)

#### AERODYNAMICS

1 Medium lift coefficient, (adimensional)  
0 Lock number  
0 Equivalent parasite area

#### FLIGHT CONDITIONS

45 Airspeed (m/s)  
0.21 Advance Ratio  
4 Height asl, (km)

#### BLADE SECTIONS

1	2	3	4	5	6	7	8	9	10	SEZ
0.25	0.5	0.75	0.97							r/R
0.13	0.17	0.11	0.02							c/R
55	35	25	0							theta(deg)

#### XROTOR ANALYSIS PARAMETERS

500 Number of iterations  
0 Power choice (set 0 for power and 1 for thrust)  
0.143 First advance ratio value  
0.363 Last advance ratio value  
0.006 Advance ratio step

**Figure 1:** Standart text file example

Finally, in the listing below, taken from the *geometryreader.m* script, the user can find any variable associated with their name.

```
%
% Geometry
%
N      %Number of blades
t_r    %Tip radius
h_r    %Hub radius
hwb    %Hub wake displacement body radius, (m)
%
% Propeller performance
%
T      %Thrust, (N)
P      %Power, (W)
%
% Aerodynamics
%
Cl_ave  %Average lift coefficient
lock_number %Lock number
f
%
% Flight conditions
%
v_inf  %Airspeed, (m/s)
adv    %Advance ratio
h      %Height asl, (km)
%
%Propeller sections
%
r      %Adimensional radius
c      %Adimensional chord
theta  %Pitch angle
%
%ANALYSIS PARAMETERS
%
iter    %Number of iterations
Powerchoice %Choice between Power and Thrust
firstadv %first advance ratio value
lastadv  %last advance ratio value
advstep  %advance ratio step
%
%ANALYSIS RESULTS
%
%Results of the analysis performed at the project advance ratio
chords  %chords distribution along the radius
pitch   %pitch distribution along the radius (deg)
radius  %adimensional radius vector
Reynolds %Reynolds number along the radius
Mach    %Mach number along the radius
Cd      %drag coefficient along the radius
%
%Results of the analysis performed for different advance ratios
advanceratio %advance ratio vector
Thrust       %Thrust vector (N)
Power        %Power vector(kW)
Torque       %Torque vector
Efficiency   %Efficiency vector
rpm          %rpm vector
```



## Code listing

---

The *geometryreader.m* script is shown in full in the following listing.

```
classdef geometryreader < handle
properties

Name = 'geometryreader';
err = 0;

%
% Geometry
%
N      %Number of blades
t_r    %Tip radius
h_r    %Hub radius
hwb    %Hub wake displacement body radius, (m)

%
% Propeller performance
%
T      %Thrust, (N)
P      %Power, (W)

%
% Aerodynamics
%
Cl_ave %Average lift coefficient
lock_number %Lock number
f

%
% Flight conditions
%
v_inf %Airspeed, (m/s)
adv   %Advance ratio
h     %Height asl, (km)

%
%Propeller sections
%
r      %Adimensional radius
c      %Adimensional chord
theta  %Pitch

%
%ANALYSIS PARAMETERS
%
iter      %Number of iterations
Powerchoice %Choice between Power and Thrust
firstadv  %first advance ratio value
lastadv   %last advance ratio value
advstep   %advance ratio step

%
%ANALYSIS RESULTS
%
%Results of the analysis performed at the project advance ratio
chords    %chords distribution along the radius
pitch     %pitch distribution along the radius (deg)
radius    %adimensional radius vector
Reynolds  %Reynolds number along the radius
Mach      %Mach number along the radius
Cd        %drag coefficient along the radius

%Results of the analysis performed for different advance ratios
advanceratio %advance ratio vector
Thrust       %Thrust vector (N)
Power        %Power vector(kW)
Torque       %Torque vector
Efficiency   %Efficiency vector
rpm          %rpm vector

end
```

```

methods
%% Constructor, populates class properties reading from propgeometry file
function obj = geometryreader (dataFileName)

f_id = fopen(dataFileName,'r');
    % verifies the .txt file opening
if (f_id==1)
    obj.err = -1;
    disp(['geometryreader :: initFromFile __ Could NOT open file ', ...
        dataFileName, ' ...'])
else
    disp(['geometryreader :: initFromFile __ Opening file ', ...
        dataFileName, ' ... OK.'])
end
%% File open, OK
for i=1:3
    fgetl(f_id);
end
%% Geometric data
obj.N = fscanf(f_id,'%f'); fgetl(f_id);
obj.t_r = fscanf(f_id,'%f') ;fgetl(f_id);
obj.h_r = fscanf(f_id,'%f');fgetl(f_id);
obj.hwb = fscanf(f_id,'%f');fgetl(f_id);
for i=1:4
    fgetl(f_id);
end
%% Performance data
obj.T = fscanf(f_id,'%f '); fgetl(f_id);
obj.P = fscanf(f_id,'%f '); fgetl(f_id);
for i=1:4
    fgetl(f_id);
end
%% Aerodynamics
obj.Cl_ave = fscanf(f_id,'%f'); fgetl(f_id);
obj.lock.number= fscanf(f_id,'%f'); fgetl(f_id);
obj.f=fscanf(f_id,'%f'); fgetl(f_id);
for i=1:4
    fgetl(f_id);
end
%% Flight conditions
obj.v_inf = fscanf(f_id,'%f'); fgetl(f_id);
obj.adv = fscanf(f_id,'%f');fgetl(f_id);
obj.h = fscanf(f_id,'%f');
for i=1:6
    fgetl(f_id);
end
%% Propeller sections
%% radius
dataBuffer = textscan(f_id,'%s\n','CollectOutput',1,...
    'Delimiter','');
r=dataBuffer{:};
newStr = split(r);
newStr=newStr(1:end-1);
obj.r=str2double(newStr)';
fgetl(f_id);

%% chord
dataBuffer = textscan(f_id,'%s\n','CollectOutput',1,...
    'Delimiter','');
c=dataBuffer{:};
newStr = split(c);
newStr=newStr(1:end-1);
obj.c=str2double(newStr)';
fgetl(f_id);

%% theta
dataBuffer = textscan(f_id,'%s\n','CollectOutput',1,...
    'Delimiter','');
theta=dataBuffer{:};
newStr = split(theta);

```

```

newStr=newStr(1:end-1);
obj.theta=str2double(newStr)';
fgetl(f_id);

%% Xrotor analysis parameter
for i=1:4
    fgetl(f_id);
end
obj.iter = fscanf(f_id,'%f');fgetl(f_id);
obj.Powerchoice = fscanf(f_id,'%f');fgetl(f_id);
obj.firstadv = fscanf(f_id,'%f');fgetl(f_id);
obj.lastadv = fscanf(f_id,'%f');fgetl(f_id);
obj.advstep = fscanf(f_id,'%f');fgetl(f_id);
%% finally, sets the error tag
obj.err = 0;
end

%% performs analysis via xrotor, if requested from user

function xRotorAnalysis(obj)

%% creates .txt input file for xrotor
fid = fopen('xrotor_input.txt','w');
fprintf(fid,'ATMO\n');
fprintf(fid,'%f\n',obj.h);
fprintf(fid,'DESI\n');
fprintf(fid,'INPU\n');
fprintf(fid,'%d\n',obj.N);
fprintf(fid,'%f\n',obj.t_r);
fprintf(fid,'%f\n',obj.h_r);
fprintf(fid,'%f\n',obj.hwb);
fprintf(fid,'%f\n',obj.v_inf);
fprintf(fid,'%f\n',obj.adv);
fprintf(fid,'%d\n',obj.Powerchoice);
fprintf(fid,'%f\n',obj.P);
fprintf(fid,'%f\n',obj.Cl_ave);
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'OPER\n');
fprintf(fid,'ITER\n');
fprintf(fid,'%d\n',obj.iter);
fprintf(fid,'ADVA\n');
fprintf(fid,'%f\n',obj.adv);
fprintf(fid,'WRIT\n');
fprintf(fid,'newprop.txt\n');
fprintf(fid,'ASEQ\n');
fprintf(fid,'%f\n',obj.firstadv);
fprintf(fid,'%f\n',obj.lastadv);
fprintf(fid,'%f\n',obj.advstep);
fprintf(fid,'CPUT\n');
fprintf(fid,'prest\n');
% Close file
fclose(fid);

%% Runs Xrotor using input file, if requested from user in the file 'main'
cmd = 'xrotor.exe < xrotor_input.txt';
[status,result] = system(cmd);

%% Reads results saved from xrotor and allocates them in local variables
newprop='newprop.txt';
fidprop = fopen(newprop); % Open file for reading

dataBuffer = textscan(fidprop,'%f%f%f%f%f%f%f%f','CollectOutput',1,...
    % Read data from file
    'Delimiter',' ','HeaderLines',17);

fclose(fidprop);
% Close file
delete('newprop.txt');
A=dataBuffer{:};
obj.radius=A(:,2);
obj.chords=A(:,3);

```

```

obj.pitch=A(:,4);
obj.Reynolds = A(:,7);
obj.Mach=A(:,8);
obj.Cd=A(:,6);

prest='prest.txt';
fidprop1 = fopen('prest');
dataBuffer1 = textscan(fidprop1,'%f%f%f%f%f%f%f%f%f', 'CollectOutput'
    ,1,...
    'Delimiter',' ','HeaderLines',3);

fclose(fidprop1);
delete('prest');
B=dataBuffer1{:};
obj.advanceratio=B(:,2);
obj.rpm=B(:,5);
obj.Power=B(:,10);
obj.Thrust=B(:,11);
obj.Torque=B(:,12);
obj.Efficiency=B(:,13);

end

%% Generates plots, if requested from user in the file 'main'
function plotResults(obj)

%% creates .txt input file for xrotor
fid = fopen('xrotor_input.txt','w');
fprintf(fid,'ATMO\n');
fprintf(fid,'%f\n',obj.h);
fprintf(fid,'DESI\n');
fprintf(fid,'INPU\n');
fprintf(fid,'%d\n',obj.N);
fprintf(fid,'%f\n',obj.t_r);
fprintf(fid,'%f\n',obj.h_r);
fprintf(fid,'%f\n',obj.hwb);
fprintf(fid,'%f\n',obj.v_inf);
fprintf(fid,'%f\n',obj.adv);
fprintf(fid,'%d\n',obj.Powerchoice);
fprintf(fid,'%f\n',obj.P);
fprintf(fid,'%f\n',obj.CL_ave);
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'OPER\n');
fprintf(fid,'ITER\n');
fprintf(fid,'%d\n',obj.iter);
fprintf(fid,'ADVA\n');
fprintf(fid,'%f\n',obj.adv);
fprintf(fid,'WRIT\n');
fprintf(fid,'newprop.txt\n');
fprintf(fid,'ASEQ\n');
fprintf(fid,'%f\n',obj.firstadv);
fprintf(fid,'%f\n',obj.lastadv);
fprintf(fid,'%f\n',obj.advstep);
fprintf(fid,'CPUT\n');
fprintf(fid,'prest\n');
% Close file
fclose(fid);

%% Runs Xrotor using input file, if requested from user in the file 'main'
cmd = 'xrotor.exe < xrotor_input.txt';
[status,result] = system(cmd);

%% Reads results saved from xrotor and allocates them in local variables
newprop='newprop.txt';
fidprop = fopen(newprop); % Open file for reading
dataBuffer = textscan(fidprop,'%f%f%f%f%f%f%f%f', 'CollectOutput',1,...
    % Read data from file
    'Delimiter',' ','HeaderLines',17);

fclose(fidprop);
% Close file
delete('newprop.txt');

```

```

A=dataBuffer{:};
obj.radius=A(:,2);
obj.chords=A(:,3);
obj.pitch=A(:,4);
obj.Reynolds=A(:,7);
obj.Mach=A(:,8);
obj.Cd=A(:,6);

prest='prest.txt';
fidprop1 = fopen('prest');
dataBuffer1 = textscan(fidprop1, '%f%f%f%f%f%f%f%f', 'CollectOutput'
    ,1,...
    'Delimiter', ',', 'HeaderLines', 3);

fclose(fidprop1);
delete('prest');
B=dataBuffer1{:};
obj.advanceratio=B(:,2);
obj.rpm=B(:,5);
obj.Power=B(:,10);
obj.Thrust=B(:,11);
obj.Torque=B(:,12);
obj.Efficiency=B(:,13);

figure(1)
title('\theta (deg)');
plot(obj.radius,obj.pitch);
grid on;
xlabel('r/R');
ylabel('\theta (deg)');

figure(2)
title('chord');
plot(obj.radius,obj.chords);
grid on;
xlabel('r/R');
ylabel('c/R');

figure(3)
title('Reynolds Number');
plot(obj.radius,obj.Reynolds);
grid on;
xlabel('r/R');
ylabel('Re*10^3');

figure(4)
title('Mach Number');
plot(obj.radius,obj.Mach);
grid on;
xlabel('r/R');
ylabel('M');

figure(5)
title('Cd');
plot(obj.radius,obj.Cd);
grid on;
xlabel('r/R');
ylabel('C_d');

figure(6)
title('efficienza');
plot(obj.advanceratio,obj.Efficiency);
grid on;
xlabel('J');
ylabel('\eta');

figure(7)
title('Power');
plot(obj.advanceratio,obj.Power);
grid on;
xlabel('J');
ylabel('P (kW)');

```

```
figure(8)
title('Thrust');
plot(obj.advanceratio,obj.Thrust);
grid on;
xlabel('J');
ylabel('T (N)');

end %plotResults
end %methods
end %constructor of geometryreader
```

## Optimum propeller

This function allows to design an optimum propeller by considering the approximated optimum propeller theory developed by Prandtl. It is possible to suppose that the propeller is lightly loaded, the number of blades has been consider  $N \gg 1$  in order to neglect the wake contraction; the viscous losses are neglected as well. By considering the overmentioned hypothesis we can define the thrust  $T$  and the power  $P$  distributions along the non-dimensional radius as functions of the rotational speed, the number of blades and two induction factors  $a$  and  $a'$  that will be analyzed in the following sections.

$$\begin{cases} dT = N\rho\Omega\bar{r}(1 - a')\Gamma R_{tip}^2 d(\bar{r}) \\ dP = N\rho\Omega\bar{r}V_\infty(1 + a)\Gamma R_{tip}^2 d(\bar{r}) \end{cases} \quad (1)$$

It is important to notice that  $a$ ,  $a'$  represent the axial and rotational induction distributions along the non dimensional radius  $\chi$  which can be defined as:

$$a = \frac{w_0}{V_\infty} \frac{\chi^2}{(1 + \frac{w_0}{V_\infty})^2 + \chi^2} \quad (2)$$

$$a' = \frac{w_0}{V_\infty} \frac{1 + \frac{w_0}{V_\infty}}{(1 + \frac{w_0}{V_\infty})^2 + \chi^2} \quad (3)$$

$$\chi = \frac{\Omega r}{V_\infty} \quad (4)$$

We can now define the circulation along the blade :

$$\frac{N\Gamma}{\Omega R_{tip}^2} = F(\bar{r})4\pi\bar{r}^2 a(\bar{r}), \quad (5)$$

In this expression it is possible to emphasize the  $F$  function which represents the Prandtl correction function that allows to take into account the effect related to a finite number of blades.

$$F(\bar{r}) = \frac{2}{\pi} \arccos[e^{\frac{N}{2\lambda}(\frac{r-R_{tip}}{R_{tip}})}]; \quad \lambda = \frac{V_\infty}{\Omega R_{tip}} \quad (6)$$

## I/O

In this section a list of both input and output parameters will be provided. The whole function consists of two different subfunctions:

- **Opti\_Prop\_T.m** which solves the Euler constrained minimum problem. In this case we are supposing to fix the thrust coefficient  $C_T$  and obtaining the maximum  $C_P$  coefficient.
- **Opti\_Prop\_P.m** which solves the Euler constrained minimum problem. In this case we are supposing to fix the power coefficient  $C_P$  and obtaining the maximum  $C_T$  coefficient.

**Opti\_Prop\_T Input** This function requires the following input values

- **N\_blade** = blade's number
- **R\_hub** = Hub percentage value with respect to the radius
- **R\_tip** = Tip Radius
- **N\_rpm** = Revolutions per minute
- **V\_inf** = Asymptotic speed
- **C\_T** = Thrust coefficient

- **h** = Altitude

**Opti\_Prop\_T Output** This function provides the following output values

- **r\_adim\_T** = non dimensional radius
- **chi\_T** = non dimensional radius
- **a\_corr\_chi\_T** = corrected axial induction vs  $\chi$
- **a\_first\_corr\_chi\_T** = corrected rotational induction vs  $\chi$
- **dCt\_dradim\_T** = Thrust coefficient distribution along the non dimensional radius
- **dCp\_dradim\_T** = Power coefficient distribution along the non dimensional radius
- **Cp** = Power coefficient

**Opti\_Prop\_P Input** This function requires the following input values

- **N\_blade** = blade's number
- **R\_hub** = Hub percentage value with respect to the radius
- **R\_tip** = Tip Radius
- **N\_rpm** = Revolutions per minute
- **V\_inf** = Asymptotic speed
- **C\_p** = Power coefficient
- **h** = Altitude

**Opti\_Prop\_P Output** This function provides the following output values

- **r\_adim\_P** = non dimensional radius
- **chi\_P** = non dimensional radius
- **a\_corr\_chi\_P** = corrected axial induction vs  $\chi$
- **a\_first\_corr\_chi\_P** = corrected rotational induction vs  $\chi$
- **dCp\_dradim\_P** = Power coefficient distribution along the non dimensional radius
- **dCt\_dradim\_P** = Thrust coefficient distribution along the non dimensional radius
- **Ct** = Thrust coefficient

### Datasheet

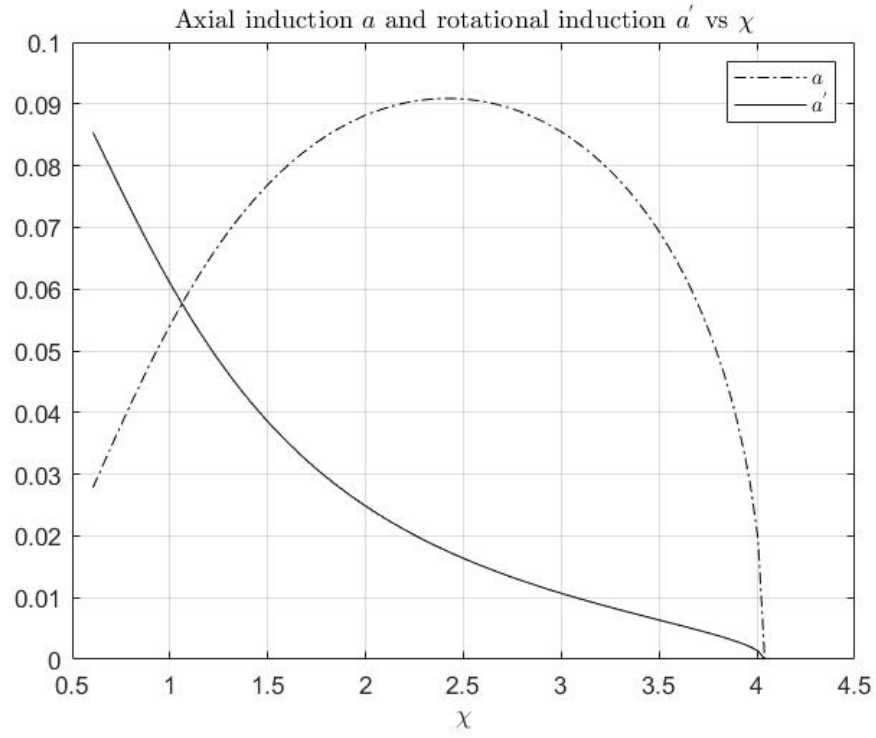
Both functions provide an output datasheet file in which all these parameters are collected together with the efficiency calculated at the project advance ratio and the convergency induction velocity  $w$ . By default, the datasheets are named as

- **Data\_Opti\_Prop\_T.txt**
- **Data\_Opti\_Prop\_P.txt**

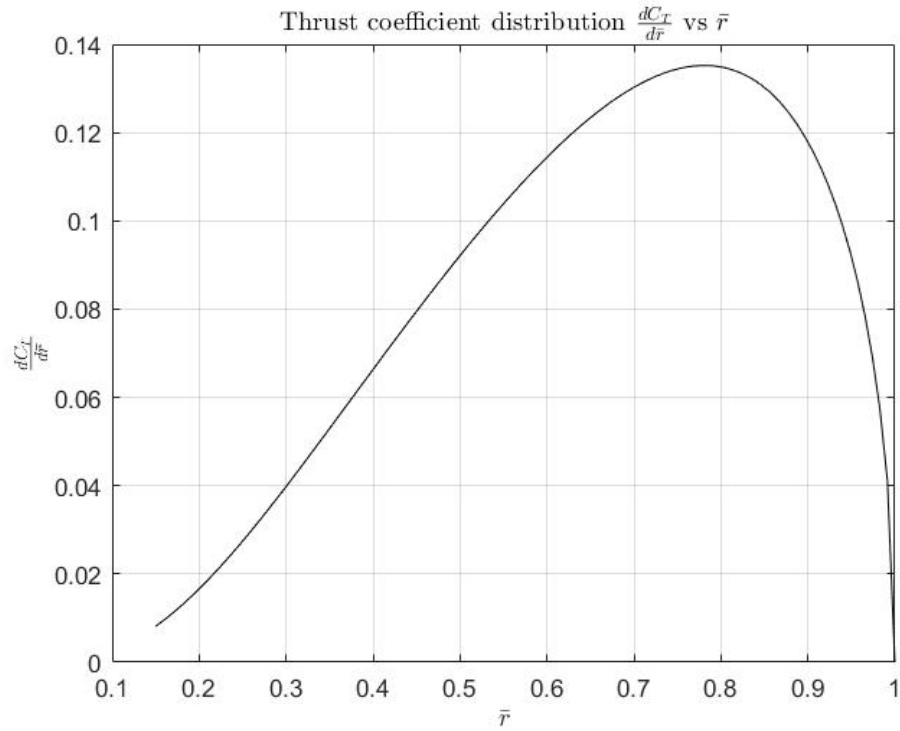
### Plot - Opti\_Prop\_T(P)

Both functions provides the following charts that are reported just for an illustrative purpose; further information will be provided in the Test Cases section.

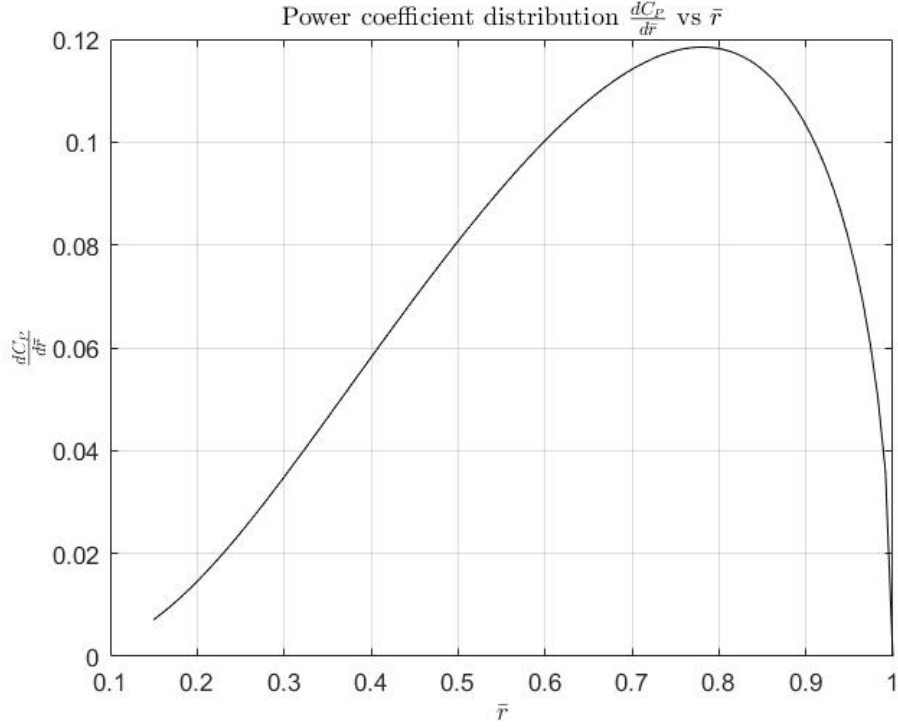




**Figure 1:** Axial  $a$  and rotational  $a'$  distribution scaled with the  $F$  Prandtl function vs  $\chi$



**Figure 2:**  $C_T$  thrust coefficient distribution vs non dimensional radius  $\bar{r}$



**Figure 3:**  $C_P$  power coefficient distribution vs non dimensional radius  $\bar{r}$

## Algorithm Description

In this chapter a detailed description of the algorithm will be provided. Since both subfunctions are similar, only the Opti\_Prop\_T subfunction will be analyzed while the Opti\_Prop\_P subfunction will be highlighted only in its peculiar aspects.

### Opti\_Prop\_T

The two first attempt values of the axial induction  $w_0$  and  $w_1$  have been set.  $w_0 = aV_\infty$  where  $a$  is the axial induction and  $w_1 = 2w_0$ . Two different first attempt values are necessary in order to initialize the false position method.

```
T = Ct*rho*(n_rps^2)*(D^4);
syms w0;
eqn1 = w0 > 0;
eqn2 = T == 2*rho*pi*(R_tip^2)*(V_inf+w0)*w0;
eqn = [eqn1 eqn2];
S = solve(eqn,w0,'ReturnConditions',true);
w_0 = double(S.w0);
w_1 = 1.5*w_0;
T_1 = 2*rho*pi*(R_tip^2)*(V_inf+w_1)*w_1;
Ct_1 = T_1/(rho*(n_rps^2)*(D^4));
Ct_0 = 0;
Ct_new = 0;
```

The false position method has been chosen in order to implement the iterative cycle. The first value of the axial induction speed, has been calculated according to the classical method formulation by using the first attempt axial induction speed values. A while loop cycle has been implemented in order to exploit the false position method.

```
w = (w_1*error_0 - w_0*error_1)/(error_0 - error_1);
k = 0; %cycle counter
while abs((Ct - Ct_new)/Ct) > tao
```

The Prandtl correction function for finite blade number has been implemented.  $\lambda = \chi^{-1}$ . According to the momentum theory:  $w_j = 2w$ ;

```
lambda = (n_rads*R_tip/V_inf)^-1;
F       = (2/pi)*acos(exp((N_blade/(2*lambda))*(r_adim-1)));
a       = (w/V_inf).*((chi.^2)./((1+w/V_inf)^2+(chi.^2)));
a_first = (w/V_inf).*((1+w/V_inf)./((1+w/V_inf)^2+chi.^2));
```

The non-dimensional aerodynamic optimal load Gamma, already scaled with the Prandtl correction function is implemented.

```
GAMMA = (4*pi.*F.*(r_adim*R_tip).^2.*a_first*n_rads)/N_blade';
```

At each step,  $\frac{dT}{dr}$  or  $\frac{dP}{dr}$  are calculated. These values are lately integrated along the non dimensional radius in order to calculate a step T or P value that must be confronted with the design T or P value.

```
dT_drDIM = N_blade*rho*n_rads*(R_tip^2)*r_adim.*(1-a_first).*GAMMA;
dCt_drDIM = dT_drDIM/(rho*(n_rps^2)*(D^4));
Ct_new    = trapz(r_adim,dCt_drDIM);
error_new = Ct_new - Ct;
```

In this step, the induction speed values and errors are updated at each cycle and then the new induction speed value is calculated in order to reiterate the calculation until the while loop exit condition is verified.

```
w_0    = w_1;
w_1    = w;
error_0 = error_1;
error_1 = error_new;
w       = (w_1*error_0-w_0*error_1)/(error_0-error_1);
k       = k+1;
end
```

The corrected inductions are defined and the power distribution and coefficients are calculated as well. The efficiency  $\eta$  and the design advance ratio are then summared in the output file.

```
a_corr = a.*F;
a_first_corr = a_first.*F;
dCp_drDIM = dT_drDIM*(V_inf+w)/((rho*(n_rps^3)*(D^5)));
Cp        = trapz(r_adim,dCp_drDIM);
J         = (V_inf/(n_rps*D));
eta       = J*(Ct_new/Cp);
error_perc_Ct = abs((error_new))/Ct*100
```

## Opti\_Prop\_P

It is possible to highlight the 3 different sections with respect to the previously analyzed script. The following code block is substitutive of the last 3 steps of the previous section.

```
%% Step 2
%{At each step, dT/dr_adim or dP/dr_adim are calculated. These values are
  lately integrated along the non dimensional radius in order to calculate
  a step T or P value that must be confronted with the design T or P
  value.
%}
dP_drDIM = N_blade*rho*V_inf*n_rads*(R_tip^2)*r_adim.*(1+a).*GAMMA;
dCp_drDIM = dP_drDIM/(rho*(n_rps^3)*(D^5));
Cp_new    = trapz(r_adim,dCp_drDIM);
error_new = Cp_new - Cp;

%% Step 3 – FALSE POSITION METHOD
%{In this step, the induction speed values and errors are updated at each
  cycle and then the new induction speed value is calculated in order to
  reiterate the calculation until the while loop exit condition is
  verified.
%}
w_0 = w_1;
w_1 = w;
```

```

error_0 = error_1;
error_1 = error_new;
w = (w_1*error_0-w_0*error_1)/(error_0-error_1);
k = k+1;

end

a_corr = a.*F;
a_first_corr = a_first.*F;
dCt_dradim = dP_dradim/(((V_inf+w)*(rho*(n_rps^2)*(D^4)))));
Ct = trapz(r_adim,dCt_dradim);
J = (V_inf/(n_rps*D));
eta = J*(Ct/Cp_new);
error_perc_Cp = abs((error_new))/Cp*100

```

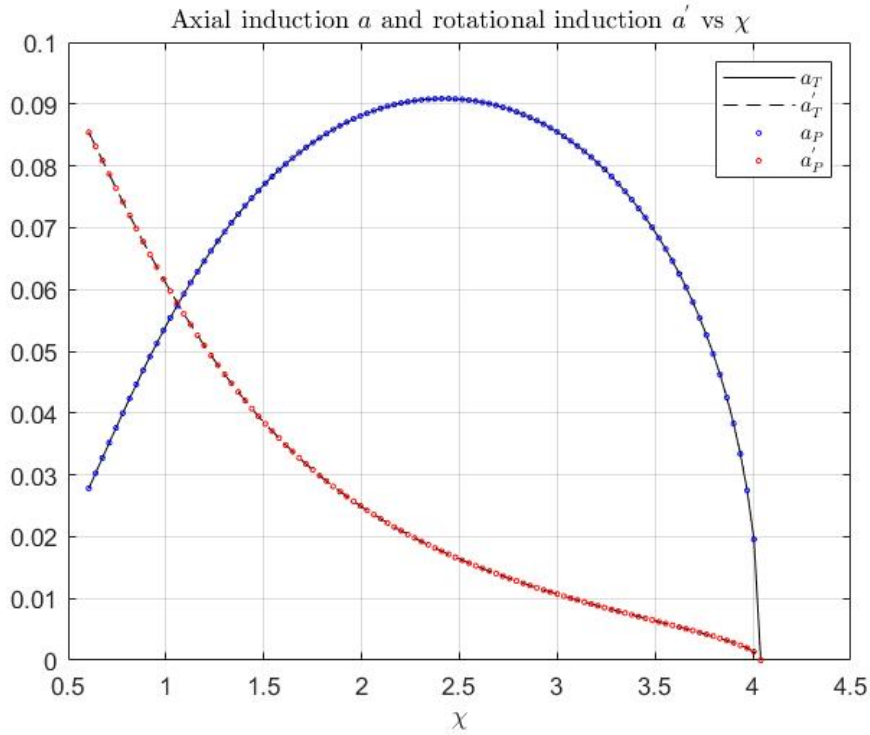
## Example

In this chapter a test case is considered in order to validate both the Opti\_Prop\_T and Opti\_Prop\_P functions.

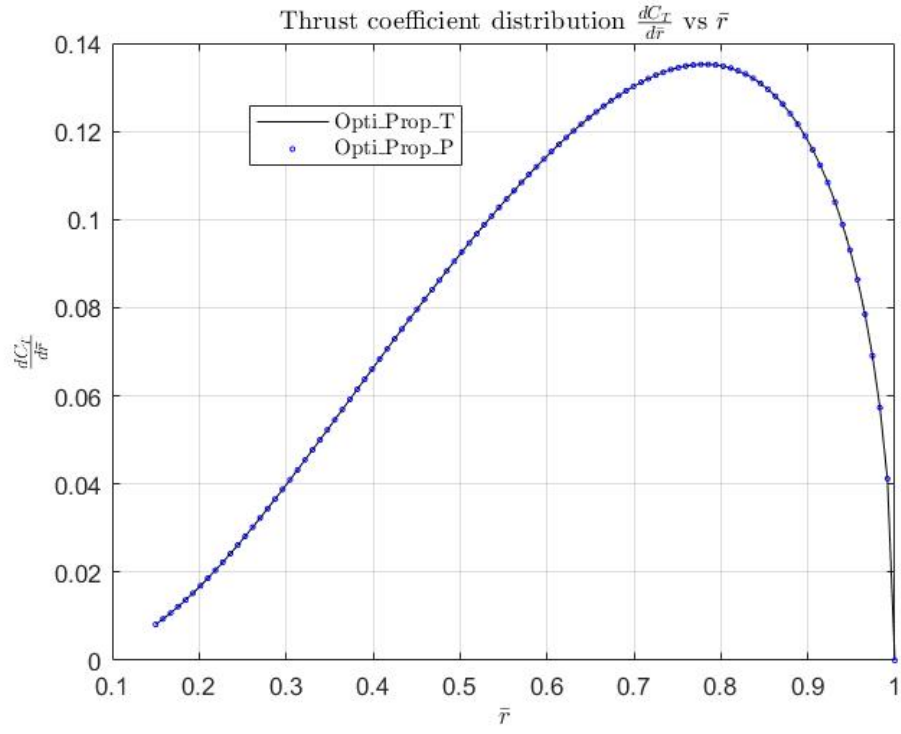
```
%% INPUT
N_blade = 2;           %[ ]
R_hub   = 0.15;        %[%]
R_tip   = 0.9;         %[m]
n_rpm   = 2500;        %[rpm]
V_inf   = 58.33 ;      %[m/s]
Ct       = 0.0740;      %[]
h        = 4510;       %[m]

r = linspace(R_hub*R_tip,R_tip,1000)/R_tip;
[r_adim_T,chi_T,a_corr_T,a_first_corr_T,...
dCt_dradim_T,dCp_dradim_T,Cp]= Opti_prop_T(N_blade,R_hub,R_tip,n_rpm,V_inf,Ct
,h);
[r_adim_P,chi_P,a_corr_P,a_first_corr_P,...
,dCp_dradim_P,dCt_dradim_P,Ct]= Opti_prop_P(N_blade,R_hub,R_tip,n_rpm,V_inf,
Cp,h);
```

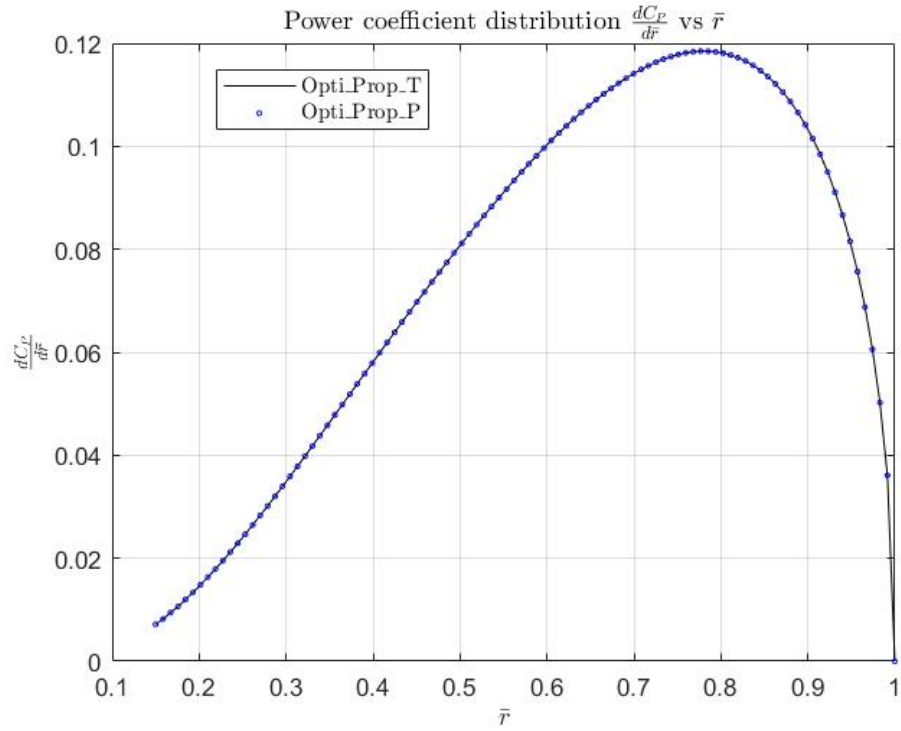
In order to verify that both functions provide almost the same output values, a generic input data set has been defined. The  $C_p$  coefficient has been calculated by using the Opti\_Prop\_T; this value has been lately given as an input parameter to the Opti\_Prop\_P function.



**Figure 4:** Axial  $a$  and rotational  $a'$  distribution scaled with the  $F$  Prandtl function vs  $\chi$



**Figure 5:**  $C_T$  thrust coefficient distribution vs non dimensional radius  $\bar{r}$



**Figure 6:**  $C_P$  power coefficient distribution vs non dimensional radius  $\bar{r}$

## Geometry input data function

Starting from a database, a file *.txt* with all the propeller/rotor/turbine geometry information, the function provides different variables that can be needed for different analysis.

### Algorithm Description

All inputs are the function identifier as a text, while outputs are structure arrays that contains all the variables needed by each function.

```
[X]= input_per_la_geometria(txt)
```

Where:

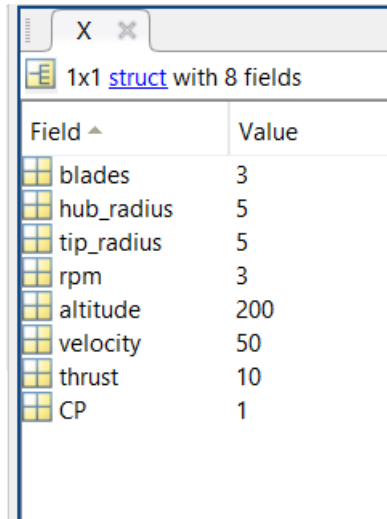
INPUT  
txt function identifier

OUTPUT  
X structure array

### Example

A test case is shown below, where it is called in input the identifier '*Opti – Prop*' to which it corresponds the function that provides the axial and the rotational inductions and the thrust and power coefficient distributions of the optimal propeller. In output the function for geometry data input provides the variables needed in input by the function *OptiProp.m* as shown in figure 1.

```
clc; close all; clear all;  
X= input_per_la_geometria('Opti_Prop');
```



The image shows a MATLAB variable viewer window titled 'X'. It displays a 1x1 struct with 8 fields. The fields and their values are listed in a table below:

Field	Value
blades	3
hub_radius	5
tip_radius	5
rpm	3
altitude	200
velocity	50
thrust	10
CP	1

**Figure 1:** X, structure arrays, output of the function for geometry input data in case of *OptiProp.m*

## Code listing

```
function [X]= Geometry_input_data(txt)

PropDataFileName='propgeometry.txt';    %.txt standard geometry file
myProp=geometryreader(PropDataFileName);    %class call

numdipale = myProp.N ;
raggio= myProp.r ; %adimensional
corda=myProp.c; %adimensional
chords=myProp.chords; %distribution along the radius
radius=myProp.radius; %distribution along the radius
tip_radius= myProp.t_r ;
hub_radius=myProp.h_r ;
thrust=myProp.T;
power=myProp.P;
locknum=myProp.lock_number;
v=myProp.v_inf;
h=myProp.h;
theta=myProp.theta;
pitch=myProp.pitch;
Reynolds=myProp.Reynolds;
Mach=myProp.Mach;
Cd=myProp.Cd;
rpm=myProp.rpm;
f=myProp.f;
advanceratio=myProp.advanceratio;

function_name=txt;
switch function_name
case 'BEMT'
    X=struct('blades',numdipale, 'r',raggio, 'hub_radius', hub_radius, 'velocity', v, 'rpm',rpm);
case 'Darrieus_flusso_moltiplo'
    X=struct('corda', corda, 'R', r, 'blades',numdipale,'Cd', Cd);
case 'Axial_rotor'
    X=struct('r',raggio);
case 'Ang_attacco_effettivo'
    X=struct('theta',theta);
case 'elica_intubata'
    X=struct('v',v,'thrust',thrust);
case 'adim_coeff'
    X=struct('velocity',v,'altitude',h,'blades', numdipale);
case 'Opti_Prop'
    X=struct('blades',numdipale,'hub_radius', hub_radius,'tip_radius', tip_radius,'rpm',rpm,'altitude',h,'velocity', v,'thrust',thrust, 'power', power);
case 'RvortexInt'
    X=struct('corda',corda);
case 'flappingangles'
    X=struct('velocity',v, 'Lock number', locknum,'R',r);
case 'Cdcl_xfoil'
    X=struct('Reynolds number', Reynolds);
case 'RotorFF'
    X=[]; %they only need in input the angle of attack
case 'Axial_Descent_Ascent'
    X=struct('R',r);
case 'Cdcl_xrotor'
    X=struct('Reynolds_number', Reynolds, 'f',f);

end

end
```



## Output Function

Using the vectors of the other functions of the library as input parameters, this function, suitably called by the others, has two fundamental purposes:

1. generate a plot (or more plots) relating to the specific operating curves of propellers, rotors and turbines;
2. generate a text file with all output vectors in column.

### Algorithm Description

The inputs of the function are the function identifier as a string and vectors that need to be plotted or transcribed into a text file.

```
[y1,y2] = FunzionidiOutput(txt,v1,v2)
```

Where:

INPUT	
txt	function identifier
v1	x-axis vector
v2	y-axis vector

OUTPUT	
yi	figure(i)

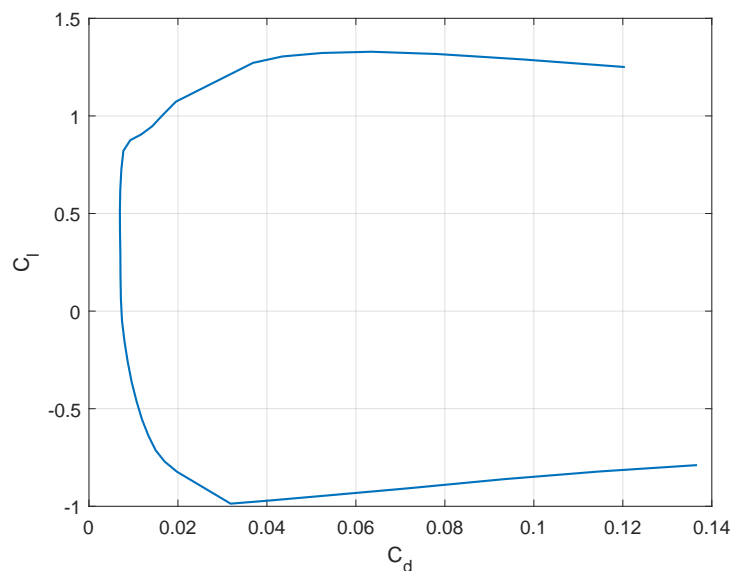
### Example

A test case for the function is shown below.

It is called the function identifier 'ClCd Xrotor' whose purpose is to output the polar of the airfoil according to the software Xrotor.

```
clc; close all; clear all;  
function [y1,y2] = FunzionidiOutput('CdCl_xfoil',v1,v2)
```

An example of a function with an output graph is below: it is underlined that the graph is right as an example but there is no reference to the real polar.



**Figure 1:** Example of output graph

## Code listing

```
function [y1,y2,y3] = Outputfunction(txt,v1,v2,v3,v4,v5,v6)

%% Default case
% function y = FunzionidiOutput(input1,input2,axisname)
% [rows,columns]=size(input1);
% for i=1:rows
%     xaxisname=string(axisname(1,i));
%     yaxisname=string(axisname(2,i));
%     y(i)=figure(i);
%     plot(x(i,:),y(i,:),'-k');
%     grid on;
%     xlabel(xaxisname);
%     ylabel(yaxisname);
% end
% end

%% Specifics cases
function_name=txt;
switch function_name
case 'ClCd_XRotor'
    y1=figure(1)
    plot(v1,v2,'linewidth',1.1);
    grid on;
    xlabel('C_d');
    ylabel('C_l');
    v3==v4==v5==v6==[];
    y2=[];
    y3=[];
case 'Characteristics_Curve_H0_Windmill'
    y1=figure(1)
    plot(v1,v2,'linewidth',1.1);
    grid on;
    xlabel('\lambda')
    ylabel('C_T')
    y2=figure(2)
    plot(v3,v4,'linewidth',1.1);
    grid on;
    xlabel('\lambda')
    ylabel('C_Q')
    y3=figure(3)
    plot(v5,v6,'linewidth',1.1);
    grid on;
    xlabel('\lambda')
    ylabel('C_P')
case 'Axial_Descent_Ascent'
    y1=figure(1)
    plot(v1,v2,'linewidth',1.1);
    grid on;
    xlabel('$\widetilde{V}$','Interpreter','latex','FontSize',15);
    ylabel('$\widetilde{w}$','Interpreter','latex','FontSize',15);
    y2=figure(2)
    plot(v3,v4,'linewidth',1.1);
    grid on;
    xlabel('$\widetilde{V}$','Interpreter','latex','FontSize',15);
    ylabel('$\widetilde{P}$','Interpreter','latex','FontSize',15);
    v5==v6==[];
    y3=[];
case 'RVortexInt'
    y1=figure(1)
    plot(v1,v2);
    grid on;
    axis ([0 1 0 1]);
    text(0.25,1,'Velocity induced by vortex ring:');
    text(0.25,0.90,['fx= ',num2str(fx)]);
    v3==v4==v5==v6==[];
    y2=[];
    y3=[];
case 'CdCL_xfoil'
    y1=figure(1)
    plot(v1,v2);
```

```

xlabel('Drag coefficient C_d');
ylabel('Lift coefficient C_l');
grid on;
v3==v4==v5==v6==[];
y2=[];
y3=[];
case 'RotorFF' %subplot
case 'Opti_prop_P'
y1 = ['Data_Opti_Prop_P.txt'];
fid = fopen(y1, 'wt');
fprintf(fid, '%s\t%s', ' efficiency =', eta, ' at J =', J); % header
fprintf(fid, '\n');
fprintf(fid, '%s\t%s', ' w_conv =', w); % header
fprintf(fid, '\n');
fprintf(fid, '%s\t%s\t%s\t%s\t%s\t%s\t%s\n', ' r_adim', ' chi', ' a(chi)
', ' a''(chi)', ' dCt/dr_adim', ' dCp/dr_adim'); % header
fclose(fid);
dlmwrite(y1, DATA, 'delimiter', '\t', 'precision', ['%10.', num2str(6), 'f'], '-
append');
v3==v4==v5==v6==[];
y2=[];
y3=[];
case 'Opti_prop_T'
y1 = ['Data_Opti_Prop_T.txt'];
fid = fopen(y1, 'wt');
fprintf(fid, '%s\t%s', ' efficiency =', eta, ' at J =', J); % header
fprintf(fid, '\n');
fprintf(fid, '%s\t%s', ' w_conv =', w); % header
fprintf(fid, '\n');
fprintf(fid, '%s\t%s\t%s\t%s\t%s\t%s\t%s\n', ' r_adim', ' chi', ' a(chi)
', ' a''(chi)', ' dCt/dr_adim', ' dCp/dr_adim'); % header
fclose(fid);
dlmwrite(y1, DATA, 'delimiter', '\t', 'precision', ['%10.', num2str(6), 'f'], '-
append');
v3==v4==v5==v6==[];
y2=[];
y3=[];
end
end

```

## Multiple streamtube Darrieus turbines function

This function implements the multiple streamtubes theory which gives a more accurate prediction of the wind velocity variations across the Darrieus rotor with respect to the single streamtube model.

### I/O

The default syntax of the function is shown below;

```
[cp,cq,lambdav] = Multiple_Streamtube_Darrieus_Turbines_Theory(alpha_max,c,R,cla,N,cd)
```

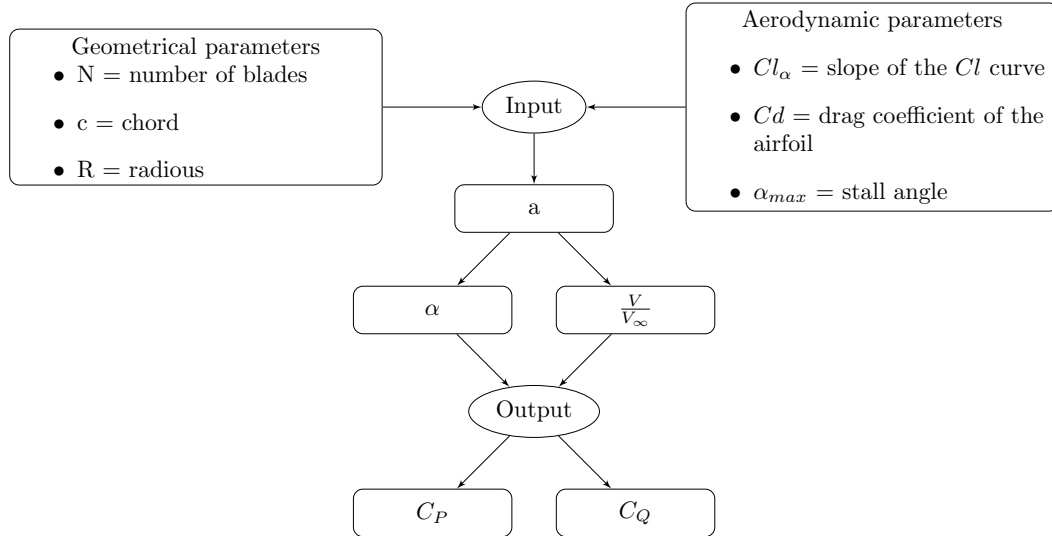
Where:

#### INPUT

alpha\_max angle of attack at the stall of the considered airfoil [deg]  
c chord of the blade [m]  
R blade radius [m]  
Cla slope coefficient of the  $C_l$  curve for the considered airfoil [1/rad]  
N number of blades  
Cd drag coefficient of the considered airfoil

#### OUTPUT

cp vector of power coefficient values for the turbine  
cq vector of torque coefficient values for the turbine  
lambdav vector of tip speed values corresponding to the cp and cq coefficients



**Figure 1:** Flow chart of the function code.

## Theory

This function implements the multiple streamtubes theory which gives a more accurate prediction of the wind velocity variations across the Darrieus rotor with respect to the single streamtube model.

The multiple streamtubes theory is derived as a generalization of the single streamtube one. The induction  $a$  is considered variable with the radius,

$$a = a(r) \quad (1)$$

where

$$r = R \sin(\phi) \quad (2)$$

The induction comes from the equations of drag force from the differential momentum theory,

$$dD_R = 2 \rho V_\infty^2 (1 - a) a R \cos(\phi) d\phi \quad (3)$$

and the drag force from the blade element theory.

$$dD_R = \frac{1}{2} \rho V^2 c Cl \cos(\phi + \alpha) d\phi \quad (4)$$

Matching equations (3) and (4), as in eq. (5), and substituting the working velocity of the airfoil, eq. (6), and the angle of attack expression, eq. (7), the induction is obtained.

$$2 \rho V_\infty^2 (1 - a) a R \cos(\phi) d\phi = \frac{1}{2} \rho V^2 c Cl \cos(\phi + \alpha) d\phi \quad (5)$$

$$\frac{V}{V_\infty} = \sqrt{[\lambda + (1 - a) \sin(\phi)]^2 + (1 - a)^2 \cos(\phi)^2} \quad (6)$$

$$\alpha = \arctan \frac{(1 - a) \cos(\phi)}{\lambda + (1 - a) \sin(\phi)} \quad (7)$$

Moreover, the  $C_P$  and the  $C_Q$  coefficients are derived by integrating the forces acting on the blade element during the rotation, as in single streamtube theory.

$$C_P = \frac{N c \lambda}{4 \pi R} \int_0^{2\pi} \left( \frac{V}{V_\infty} \right)^2 Cl \sin(\alpha) \left( 1 - \frac{Cd}{Cl} \cot(\alpha) \right) d\phi \quad (8)$$

$$C_Q = \frac{C_P}{\lambda} \quad (9)$$

The characteristic  $\lambda - C_P$  curves obtained through the proposed method is not valid for any  $\lambda$ . It is possible to define a  $\lambda_{min}$  when  $\alpha = \alpha_{max}$ , at the stall of the airfoil and a  $\lambda_{max}$  by imposing that the turbine must provide power.

## Algorithm Description

At the beginning of the function, the vector of the  $\phi$  and  $\lambda$  domains are created and then all the needed variables  $v/v_{inf}$ ,  $\alpha$ ,  $a$ ,  $C_P$  and  $C_Q$  are initialized.

After that, we enter in a while cycle where it is defined a value of  $\lambda$  and only the positive values of  $C_P$  are considered because for negative  $C_P$  the rotor will not provide energy but it'll need energy.

```
1 while cp >= 0
2     j = j + 1;
3     lambda = lambdav(j);
```

Inside this cycle, we enter a for cycle in which for every value of  $\phi$  the velocity induction is evaluated thanks to the matlab function *Fzero* since the equation that has to be solved is not linear.

```
1     for i = 1 : numel(phiv)
2         phi = phiv(i);
3         %anonymous function
4         eq = @(a) ((1-a).*a) - (c/(4*R))* ...
5             ( (lambda + (1-a).*sin(phi)).^2+...
6             (1-a).^2.*cos(phi).^2).*cla.*...
7             atan2(((1-a).*cos(phi)), (lambda + ...
8             (1-a).*sin(phi))).*(cos(phi+(atan2...
9             (((1-a).*cos(phi)), (lambda + ...
10            (1-a).*sin(phi))))./cos(phi)));
11        %find zero of the previous function
12        a(1,i) = (fzero(eq,0.2));
```

Then  $v/v_{inf}$  and  $\alpha$  can be evaluated, and in particular, following the rows, these parameters change with  $\phi$ , instead, following the column, they change with  $\lambda$ .

```
1     v_vinf(j,:) = sqrt((lambda + (1-a).*...
2         sin(phiv)).^2 + (1-a).^2 .*cos(phiv).^2);
3     alpha(j,:) = atan2(((1-a).*cos(phiv)),...
4         (lambda + (1-a).*sin(phiv)));
```

Once evaluated these parameters,  $C_P$  and  $C_Q$  can be evaluated with the matlab function *trapz* because to obtain both of them, integrals have to be made.

```
1     cost_p = (N*c*lambda)/(4*pi*R);
2     % computing Cp - numerical integration
3     cp = cost_p.* trapz(phiv, (...
4         v_vinf(j,:).^2*cla.*alpha(j,:).*...
5         sin(alpha(j,:)).*(1-(cd./...
6         (cla.*alpha(j,:)).*cot(alpha(j,:)))));
7     cq = cp/lambda; % Cq value
8
9     % Allocating values in corresponding vectors
10    cpv(j,1) = cp;
11    cqv(j,1) = cq;
```

At the end of the code, there is a control on the minimum  $\lambda$  because the  $\alpha$  of the blade elements have always to be smaller than the  $\alpha_{max}$ , otherwise the wing will not work properly and will not generate any power.

```
1 max_v = zeros(numel(lambdav),1);
2 alphadeg = rad2deg(alpha); % [deg]
3
4 %Find max alpha for each row of the matrix alpha
5 for h = 1 : numel(phiv)
6     max_v(h) = max(alphadeg(h,:));
7 end
```

```

8
9 %initial value for lambda min index
10 contatorelambdamin = 0;
11
12 %check on stall angle
13 for f = 1 : numel(phiv)
14     if alphamax < max_v(f)
15         contatorelambdamin = contatorelambdamin + 1;
16     end
17 end

```

For values of  $\sigma$  that are greater than 0.25, there can be possibilities of error when  $\lambda$  becomes much larger than one.

For this reason, in the code there is a control on the induction vector that allows the function to exit the cycle when one NaN appears.

So, from the first value of  $\lambda$  characterized by this behavior, the  $C_P$  and  $C_Q$  are not evaluated.

In the end, this behaviour appears when lambda is really great and so for values of  $C_P$  that are close to zero.

But when a rotor is designed, the idea is to size it in order to work where  $C_P$  is close to the  $C_{P,max}$ , so the error of the function occurs far away from the design point.

```

1     if sum(isnan(ind(j,:))) >= 1
2         disp('Warning: from this lambda on,');
3         disp('the results are not reliable,');
4         disp('so the results are not reported');
5         cp = -1;
6         j = j-1;
7     else

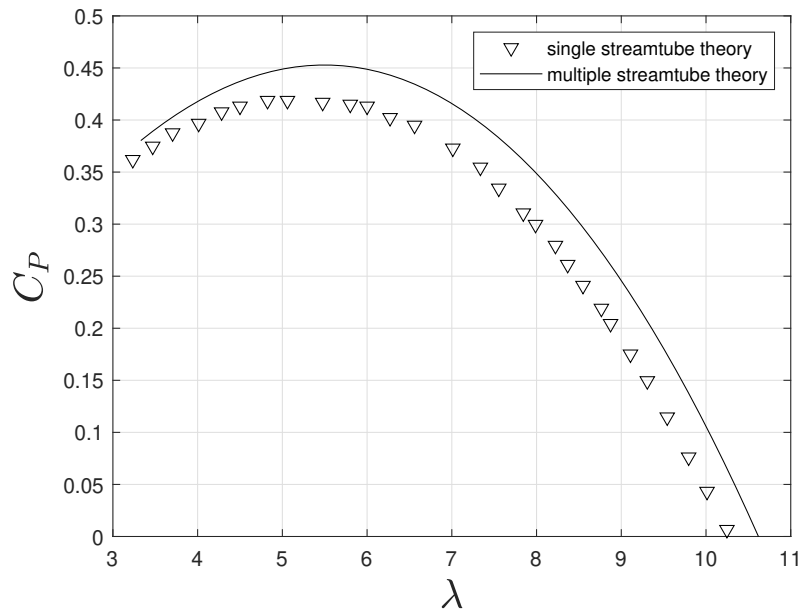
```

### Example

In order to validate the function, some test cases have been run. The obtained results have been compared with the values of De Vries,[2], which refers to the single streamtube theory. The table 1 shows the whole parameters taken into account to carry out the validation procedure.

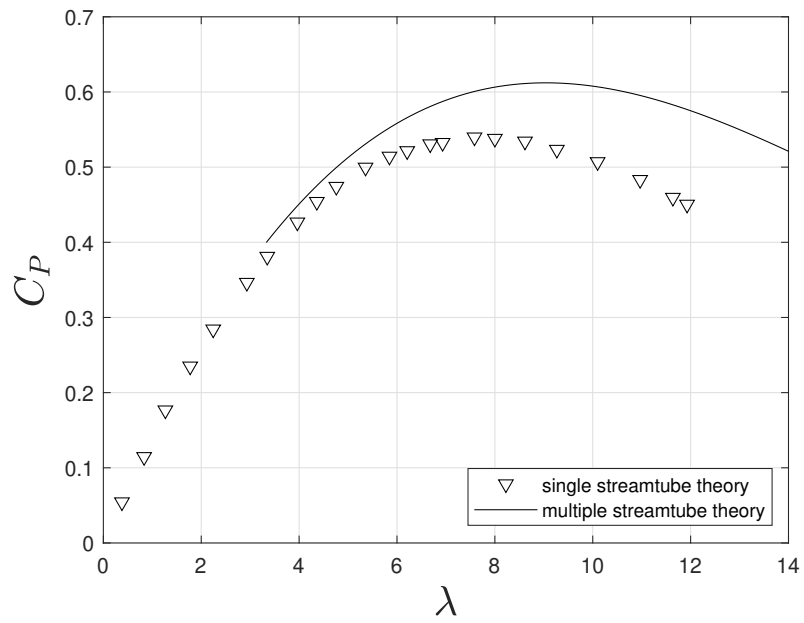
test	$\sigma$	$\alpha_{max}$	c	R	$Cl_\alpha$	N	Cd
1	0.1	14°	1	30	6.28	3	0.01
2	0.1	14°	1	30	6.28	3	0
3	0.2	14°	1	15	6.28	3	0.01
4	0.2	14°	1	15	6.28	3	0

**Table 1:** Test case values.

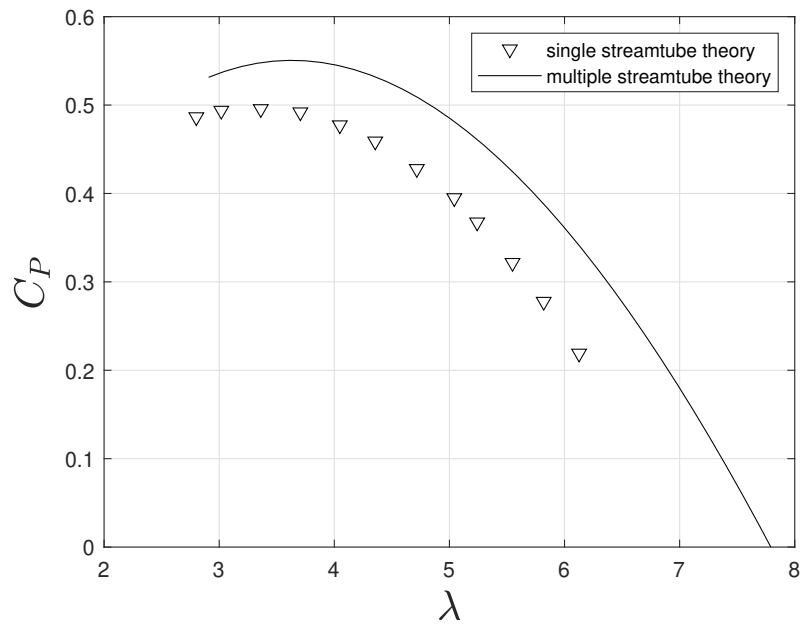


**Figure 2:** Test 1. Values compared with single streamtube theory from De Vries, [2].

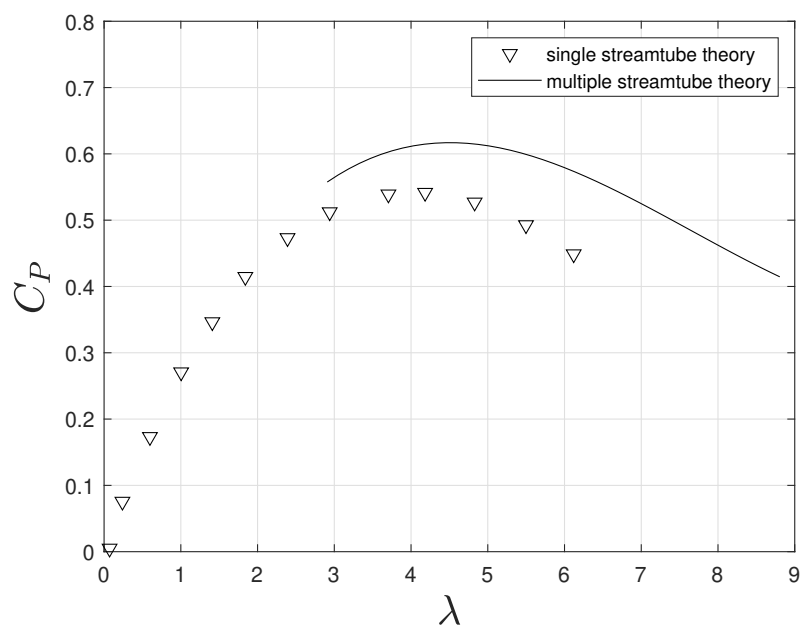




**Figure 3:** Test 2. Values compared with single streamtube theory from De Vries, [2].



**Figure 4:** Test 3. Values compared with single streamtube theory from De Vries, [2].



**Figure 5:** Test 4. Values compared with single streamtube theory from De Vries, [2].

## Code listing

```
function [cp,cq,lambdav] =
    Multiple_Streamtube_Darrieus_Turbines_Theory(alphamax,c,R,cla,N,cd)

%% Initialization of vectors used in the code
phiv = linspace(0,360,100); %phi domain [deg]
phiv = deg2rad(phiv); % [rad]
lambdav = linspace(0.1,12,100); %tip speed
cpv = zeros(numel(lambdav),1); %cp vector
cq = zeros(numel(lambdav),1); %cq vector
a = zeros(1,numel(phiv)); %induction
v_vinf = zeros(numel(phiv),numel(phiv)); %v/v_inf
alpha = zeros(numel(phiv),numel(phiv)); %alpha
ind = zeros(numel(phiv),numel(phiv)); %indices vector
%% Values for entering the loop
cp = 0;
j = 0;
%% a, CP, CQ computing
while cp >= 0
    j = j + 1;
    lambda = lambdav(j);
    for i = 1 : numel(phiv)
        phi = phiv(i);
        eq = @(a) ((1-a).*a) - (c/(4*R))* ( (lambda + (1-a).*sin(phi)).^2 + (1-a)
            .^2.*cos(phi).^2).*cla.*atan2(((1-a).*cos(phi)), (lambda +(1-a).*sin(phi)
            )).*cos(phi)+(atan2(((1-a).*cos(phi)), (lambda +(1-a).*sin(phi)))./cos(
            phi))); %anonymous function
        %find zero of the previous function
        a(1,i) = (fzero(eq,0.5));
    end
    ind(j,:) = a;
    v_vinf(j,:) = sqrt((lambda + (1-a).*sin(phiv)).^2+ (1-a).^2 .*cos(phiv)
        .^2);
    alpha(j,:) = atan2(((1-a).*cos(phiv)), (lambda + (1-a).*sin(phiv)));
    cost_p = (N*c*lambda)./(4*pi*R);
    cp = cost_p.* trapz(phiv, (v_vinf(j,:).^2*cla.*alpha(j,:).*sin(alpha(j,:))
        ).*(1-(cd./(cla.*alpha(j,:)).*cot(alpha(j,:)))));
    % computing Cp - numerical integration
    cq = cp/lambda; % Cq value

    % Allocating values in corresponding vectors
    cpv(j,1) = cp;
    cqv(j,1) = cq;

    %Condition to exit the cycle when lambda
    % is the last value of the
    %vector
    if lambda == lambdav(end)
        cp = -1;
    end
end
%%Allocating vector of max values
max = zeros(numel(lambdav),1);
alphadeg = rad2deg(alpha); % [deg]

%Find max alpha for each row of the matrix alpha
for h = 1 : numel(phiv)
    max(h) = max(alphadeg(h,:));
end

lambdamin_ind = 0; %intial value for lambda min index

%check on stall angle
for f = 1 : numel(phiv)
    if alphamax < max(f)
        lambdamin_ind = lambdamin_ind + 1;
    end
end
%cp, cq and lambdav are downsized according to conditions
```

```
% of stall and power positive value
cq = cqv(lambdamin_ind:j,1);
cp = cpv(lambdamin_ind:j,1);
lambdav = lambdav(1,lambdamin_ind:j);
end
```