***ClCd_XRotor.m* function user guide**

1 Introduction and description

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.

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

2 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.

- $\boldsymbol{C_{d_{min}}}$
  minimum blade element's drag coefficient

- $\boldsymbol{\frac{dC_d}{dC_l^2}}$
  quadratic coefficient of the parable that approximates the $C_d(C_l)$ law

- $\boldsymbol{C_l\left(C_{d_{min}}\right)}$
  blade element's lift coefficient for which the drag coefficient assumes its minimum value

- $\boldsymbol{Re_{ref}}$
  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 Ωr where Ω is the propeller's angular velocity and r is the radial position of the blade element taken into account)

- $\boldsymbol{Re_\infty}$
  asymptotic Reynolds number of the phenomena

- $\boldsymbol{f}$
  Reynolds number scaling exponent, according to XRotor documentation

- $\boldsymbol{C_{l_{max}}}$
  maximum blade element's lift coefficient

- $\boldsymbol{C_{l_{min}}}$
  minimum blade element's lift coefficient

- ***C_l 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. *Cl_Cd_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_l vector***
  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

3 *Code description*

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

```
Cl_vec = Cl_min:.01:Cl_max;
```

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

```
Cd = (Cd_min + dCd_dCl2*(Cl_Cd_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.

4 *Usage examples*

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.

5 *References*

[1] *XRotor* software user guide
http://web.mit.edu/drela/Public/web/xrotor/xrotor_doc.txt

6 *Complete script*

The above explained script is thus reported in the following lines.

```
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
```