The `dmst.m` function:
theory and code documentation

Gabriele Lucci, M53000957

February 17, 2022

Contents

1 Double-Multiple Streamtube methods description

Performance assessment of a Darrieus turbine can be improved through Double-Multiple Streamtube (DMST) methods class. Although these methods are based on a more accurate and realistic modeling of the turbine, they still keep a reasonable computational cost (Paraschivoiu, 2002). DMST methods rely on the following considerations:

- each blade element, throughout a whole rotation about turbine axis, passes twice through the same streamtube, the first time moving upwind, the second downwind;
- the conditions it "sees" in its second passage are clearly different from the ones of the first, since part of the energy pertaining to the undisturbed current has already been extracted.

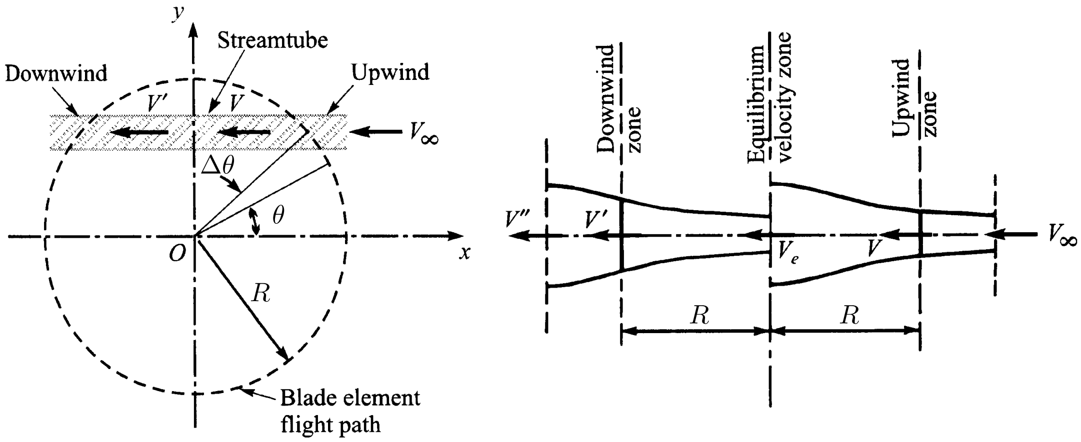


Figure 1: DMST model (adapted from Paraschivoiu, 2002).

DMST methods consist, thus, in modeling the Darrieus turbine through two *series* of actuator disks in tandem (a pair for each elemental streamtube), to which correspond two axially constant — but different — axial induction factors. It is furthermore assumed that each streamtube is not influenced by the others, that their sections don't vary in the axial direction, and that flow expansion due to the interaction with the upwind actuator disks completes before the current begins to interact with the downwind ones. In other words, the conditions of current entering the downwind streamtubes coincide withe the ones in the *far wake* of the upwind ones.

1.1 Theoretical notes

The Paraschivoiu (2002) approach will be followed in these notes (restricting it to the case of straight-bladed Darrieus rotors), and the same notation will be kept (see figure 1). The azimuthal position $\theta = 0$ is such that the blade is straight upwind, with its arm parallel to the freestream direction.

Streamlines intersection points with the blade path are equally spaced of the angle

$$\Delta\theta = \frac{\pi}{n_{\mathrm{st}}}. \tag{1}$$

There will be, thus, n_{st} values of θ for each half-cycle identifying the intersection of each streamtube axis with the blade element trajectory.

For each streamtube, five *axial* current velocities are defined, standing in the relation

$$V_\infty > V > V_{\mathrm{e}} > V' > V''$$

Interference factors $u = V/V_\infty$ and $u' = V'/V_{\mathrm{e}}$ are also defined, and it can be stated that

- V_∞ is the freestream undisturbed velocity;
- $V = uV_\infty$ is the flow velocity at the upwind actuator disk (thus considering a first slow-down due to axial induction);
- $V_{\mathrm{e}} = (2u - 1)V_\infty$ is the *equilibrium* velocity, equal to the the velocity in the far wake of the first actuator disk, and thus to the one with wich it begins interacting with the second;

- $V' = u'V_\mathrm{e} = u'(2u-1)V_\infty$ is the flow velocity at the second actuator disk;
- $V'' = (2u'-1)V_\mathrm{e} = (2u'-1)(2u-1)V_\infty$ is the velocity in the far wake of the second atuator disk, i.e. of the turbine itself.

Note that $a = 1-u$, $a' = 1-u'$.

Two problems must be solved separatedly and in sequence. The upwind problem results will be the "input" of the downwind one.

Upwind half of the rotor: $-\pi/2 \le \theta \le \pi/2$

Through geometrical considerations and referring to figure 2 on the following page, one gets the expression below for the velocity ratio

$$W^2 = V^2\left[(\lambda_\theta - \sin\theta)^2 + \cos^2\theta\right] \tag{2}$$

where $\lambda_\theta = R\Omega/V$ is the *local* tip speed ratio (i.e. the blade peripheral velocity is compared to the previously defined V). As for the angle of attack, one has

$$\alpha = \arcsin\left[\frac{\cos\theta}{\sqrt{(\lambda_\theta - \sin\theta)^2 + \cos^2\theta}}\right] \tag{3}$$

For each streamtube, composing the aerodynamic force in its normal and tangential directions and accounting for local velocitites composition, the following integral equation can be written

$$f_\mathrm{up}u = \pi(1-u) \tag{4}$$

which has to be solved iteratively in u with numerical techniques, and where

$$f_\mathrm{up} = \frac{\sigma}{8\Delta\theta}\int_{\theta-\Delta\theta/2}^{\theta+\Delta\theta/2}\left(C_n\frac{\cos\theta}{|\cos\theta|} - C_t\frac{\sin\theta}{|\cos\theta|}\right)\left(\frac{W}{V}\right)^2 \mathrm{d}\theta \tag{5}$$

Equation (5) can be derived by imposing, as usual in BEMT methods, that the thrust calculated by momentum conservation through the streamtube equals the one coming from the aerodynamic forces acting on the blade. The force coefficients present in equation (5) depend, clearly, on the local angle of attack and on the local Reynolds number. Once the value of u is found for each streamtube, V_e is known and the downwind problem can be solved.

Downwind half of the rotor: $\pi/2 \le \theta \le 3\pi/2$

With the same logic of the upwind cycle, the foregoing relations are obtained

$$W'^2 = V'^2\left[(\lambda'_\theta - \sin\theta)^2 + \cos^2\theta\right] \tag{6}$$

where $\lambda'_\theta = R\Omega/V'$,

$$\alpha' = \arcsin\left[\frac{\cos\theta}{\sqrt{(\lambda'_\theta - \sin\theta)^2 + \cos^2\theta}}\right]. \tag{7}$$

Furthermore,

$$f_\mathrm{dw}u' = \pi(1-u') \tag{8}$$

to be solved numerically in u', with

$$f_\mathrm{dw} = \frac{\sigma}{8\Delta\theta}\int_{\theta-\Delta\theta/2}^{\theta+\Delta\theta/2}\left(C'_n\frac{\cos\theta}{|\cos\theta|} - C'_t\frac{\sin\theta}{|\cos\theta|}\right)\left(\frac{W'}{V'}\right)^2 \mathrm{d}\theta \tag{9}$$

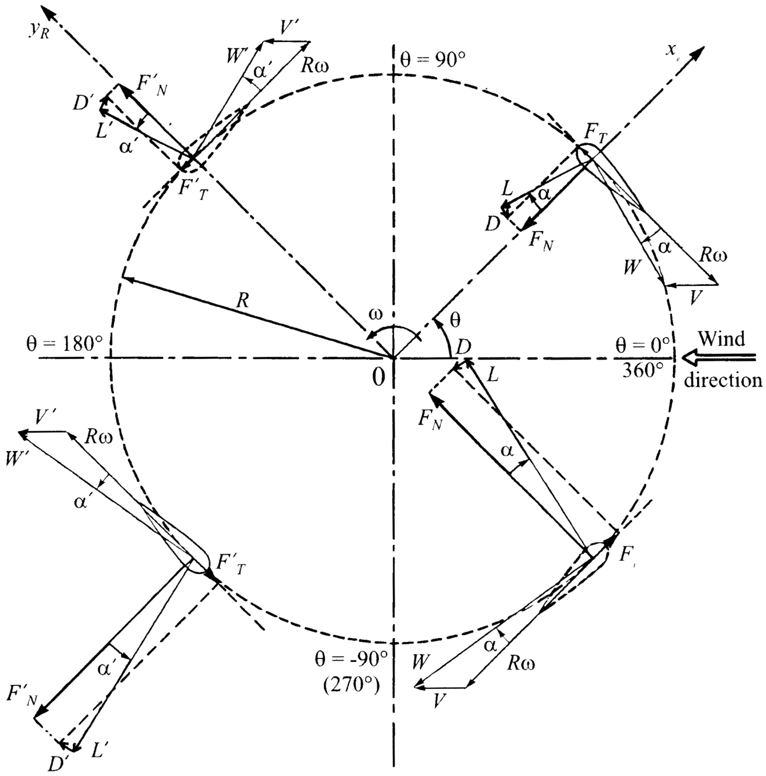


Figure 2: Upwind and Downwind blade rotation phases and corresponding azimuth angle ranges. Local velocities compositions and aerodynamic forces projections for each quadrant (Paraschivoiu, 2002).

Torque and power coefficients calculation

Once the factors u and u' are known for each streamtube, torque coeffients can be found (averaging on a whole rotor cycle) for each one of the two halves, using the expressions

$$\overline{C}_{Q,\mathrm{up}} = \frac{\sigma}{8\pi} \int_{-\pi/2}^{\pi/2} C_t \left(\frac{W}{V_\infty} \right)^2 \mathrm{d}\theta \tag{10a}$$

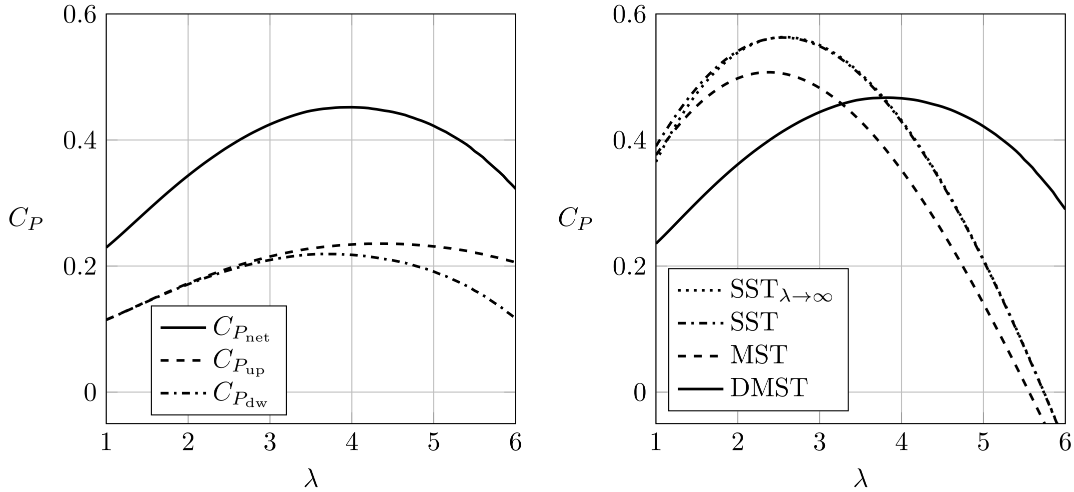
$$\overline{C}_{Q,\mathrm{dw}} = \frac{\sigma}{8\pi} \int_{\pi/2}^{3\pi/2} C'_t \left(\frac{W'}{V_\infty} \right)^2 \mathrm{d}\theta \tag{10b}$$

Finally, power coefficients are

$$C_{P,\mathrm{up}} = \frac{R\Omega}{V_\infty} \overline{C}_{Q,\mathrm{up}} = \lambda \overline{C}_{Q,\mathrm{up}} \tag{11a}$$

$$C_{P,\mathrm{dw}} = \frac{R\Omega}{V_\infty} \overline{C}_{Q,\mathrm{dw}} = \lambda \overline{C}_{Q,\mathrm{dw}} \tag{11b}$$

$$C_P = C_{P,\mathrm{up}} + C_{P,\mathrm{dw}} \tag{11c}$$



(a) DMST model results. Linear law for C_l, XROTOR model for C_d. $V_\infty = 5$ m/s.

(b) Comparison with MST, SST and $\text{SST}_{\lambda\to\infty}$ models. Linear law for C_l, $C_d = 100$ DC.

Figure 3: $C_P(\lambda)$ for a straight-bladed Darrieus turbine with $\sigma = 0.3$.

1.2 Streamtube theories limitations

All streamtube methods have the same theoretical limitation: Rankine-Froude theory, upon which they rely, is valid only until $a < 0.5$. Beyond that values, the turbulent wake state is entered and the theory provides nonphysical results (Tognaccini, 2021). Such results are valid, thus, only for lightly-loaded rotors (i.e. for low solidity and TSR values). Furthermore, this kind oh theories are not capable to distiguish among different B, c and R combinations to which correspond a single σ value. This makes impossible to take into account the "flow curvature" effects and the fact that, for increasing B, each blade is affected by the wake of the one it follows.

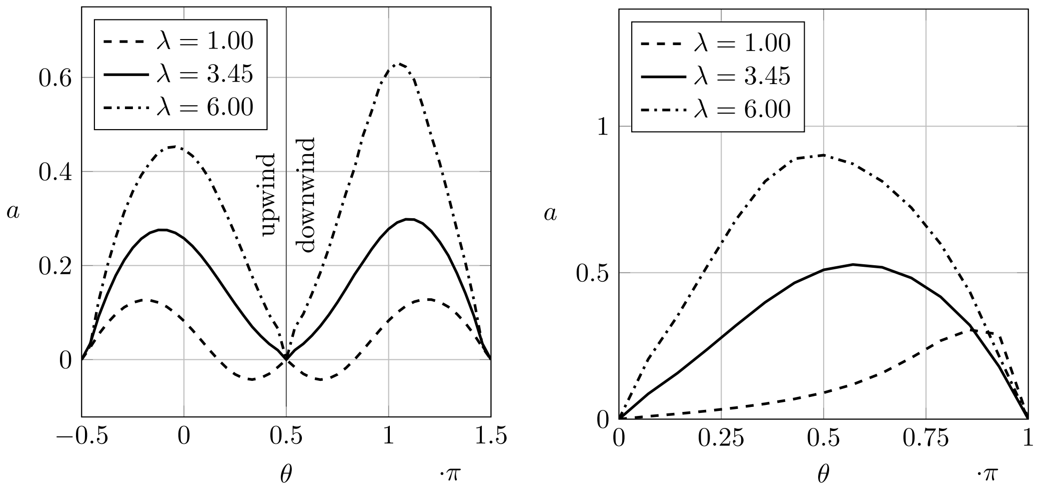
Another important limitation lies into the assumption that the flow field is steady. This hypothesis is quite far from reality in many functioning conditions. Therefore, an interesting way to improve DMST methods accuracy would be to implement a dynamic stall prevision technique.

2 Results

Figure 3a shows power coefficient curve versus TSR for a straight-bladed Darrieus turbine with $\sigma = 0.3$. The aerodynamic model employed is such that $C_l = 2\pi\,\alpha$ and C_d is obtained through XROTOR formula. Upwind and downwind contributions to the power coefficient are also shown. Such curve is of merely theoretical interest, since it is not limited to blade aerodynamic stall (for low λ values), neither to momentum theory validity condition ($a < 0.5$).

Of greater — though still theoretical — interest is figure 3b, where the results of different models are compared at fixed solidity and aerodynamic model. It can be noted that simpler theories overestimate maximum C_P value, while underestimating λ design value (at which the machine is capable of extracting maximum power from wind).

Furthermore, comparing calculated values for axial inductions (see figure 4 on the next page) at different TSRs, it can be seen that MST theory, by assuming kinetic energy extraction is accomplished through a single actuator disk for each streamtube, provides greater values for a



(a) Axial induction calculated through DMST theory.

(b) Axial induction calculated through MST theory.

Figure 4: $a(\theta)$ for a straight-bladed Darrieus turbine with $\sigma = 0.3$. Linear law for C_l, XROTOR drag polar formula for C_d. $V_\infty = 5$ m/s.

(all other conditions being equal), and an anticipated violation of the MT validity limit.

In figure 5 on the following page, also, ratios between axial velocities (accounting for inductions) and freestream velocity is reported versus TSR, for each rotor half. Blades angular positions are, namely, $\theta = 0$ and $\theta = \pi$ (i.e. the streamtube is the same).

It can be noted that a significant difference between the two ratios exists, and such difference grows with λ. It can be stated, therefore, that MST and DMST theory provide quite similar results for low TSRs; for high tip speed ratios, though, MST theory appears rather inadequate, since it falls short of modeling the great difference in operative conditions between the upwind and the downwind rotor zones (Paraschivoiu, 2002).

3 `dmst.m` code validation: a case study

A realistic aerodynamic model to determine the forces acting on the blade can be implemented in a DMST code. Experimental data gathered by Sheldahl and Klimas (1981) of a NACA 0012 airfoil, for angles of attack ranging between 0° and 180° and for Reynolds number from 10^4 to 10^7, has been adopted. The performance of a rotor consisting in $B = 3$ blades of $c = 0.2$ m chord length and whose radius is $R = 2$ m (thus $\sigma = 0.3$) have been calculated for TSRs ranging in the interval $[1.5, 5.8]$. Freestream velocity has been taken to be $V_\infty = 5$ m/s. In said conditions and taking axial inductions into account, local Reynolds number values vary between $3.33 \cdot 10^4$ for $\lambda = 1.5$ and a maximum value of $1.29 \cdot 10^6$ for $\lambda = 5.8$.

Considering the same turbine and wind speed, Saber et al. (2018) performed the same calculations through a modified DMST method, which employs a different expression for equilibrium velocity V_e. The results obtained with such method (previously validated in turn through experimental results comparison) have been taken as reference. The comparison is shown in figure 6a on page 8.

A substantial agreement (except for the low λ interval) between the results provided by the

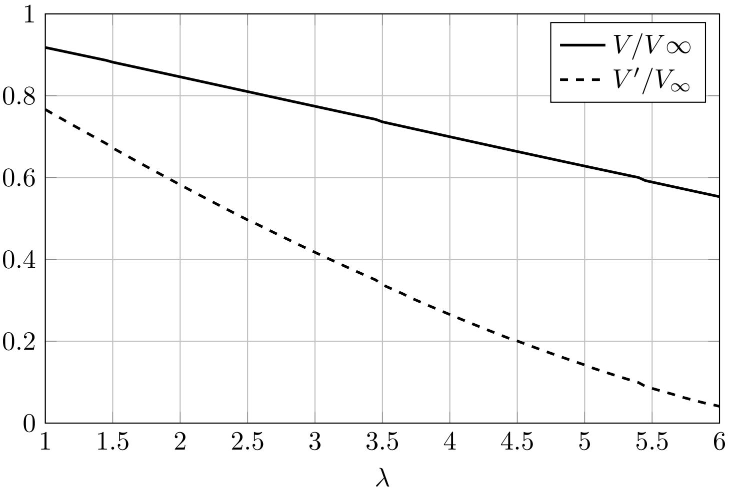


Figure 5: Ratio between axial velocities (accounting for inductions) and freestream velocity, with respect to tip speed ratio, for the upstream and downstream halves of the rotor (respectively, for $\theta = 0$ e per $\theta = \pi$, calulated for the same turbine considered in figures 3 and 4). Linear law for C_l, XROTOR model for C_d. $V_\infty = 5$ m/s.

two methods can be noted. In particular, it is noteworthy that, for a fixed V_∞ value, at low rotational speeds (i.e. TSRs) the power coefficient is negative: this confirms that such turbines are not capable of self-starting. It can be furthermore observed that, left of maximum C_P value, the curve exhibits quite a steep slope, which indicates a rather sudden blades stall; after the maximum value, power coefficient decreases gradually due to parasite drag.

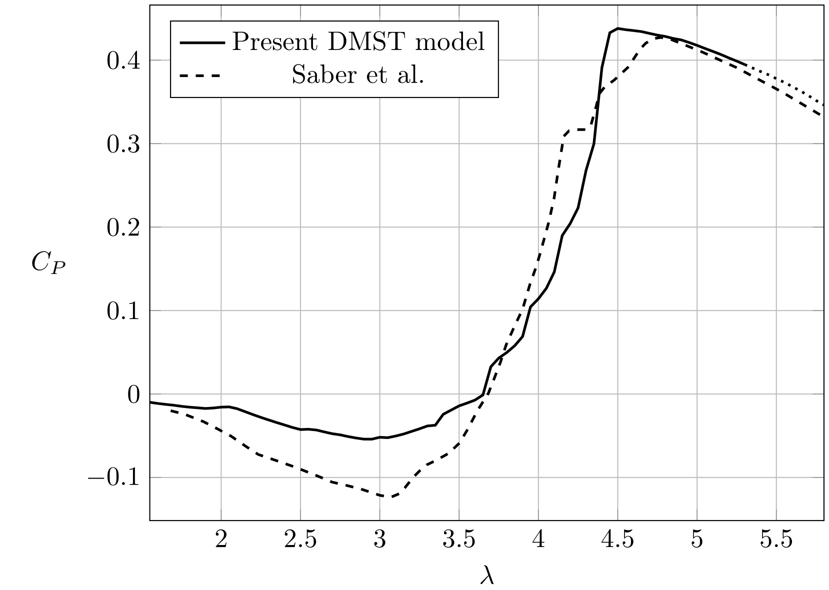
3.1 Instantaneous torque

Figure 6b on the following page shows the instantaneous torque coefficient $C_Q(\theta)$, for $\lambda = 1.5$ and for $\lambda = \lambda_{C_{P_{max}}} = 4.50$, calculated as

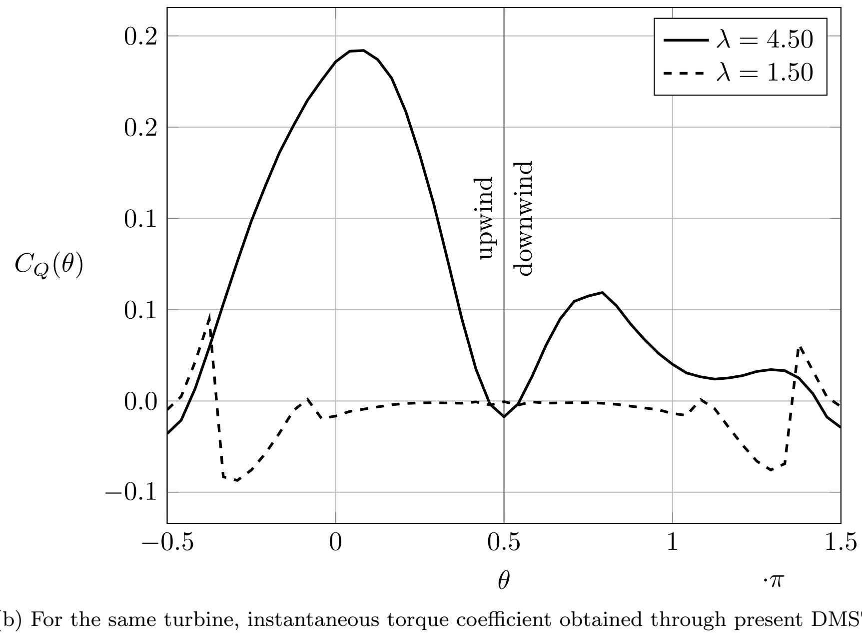
$$C_Q(\theta) = \frac{\sigma}{4} C_t(\theta) \left[\frac{W}{V_\infty}(\theta) \right]^2, \quad -\pi/2 \leq \theta \leq \pi/2 \tag{12a}$$

$$C_Q(\theta) = \frac{\sigma}{4} C_t'(\theta) \left[\frac{W'}{V_\infty}(\theta) \right]^2, \quad \pi/2 \leq \theta \leq 3\pi/2 \tag{12b}$$

Referring to figure 6a it is observable that, for $\lambda = 0.15$, the torque acting on the blade is negative, both in the upwind and in the downwind cycles. In this functioning condition, the turbine acts like a propeller: power must be fed to its shaft in order to keep (at least) a constant rotational speed. On the other hand, for $\lambda = 4.50$ the blade acts with a positive torque on the shaft for almost the whole rotation, the major contribution being the one coming from the upwind cycle. Indeed, it can be observed that the turbine drains power only for very small azimuth angle ranges, close to the cycle phases in which the blade section chord is almost parallel to the direction of the asymptotic wind.



(a) Results comparison between present DMST model, based on Paraschivoiu (2002) theory, and the "modified DMST" method, developed by Saber et al. (2018). The dashed part of the curve relative to the present method indicates the λ values for which $a > 0.5$ and Momentum Theory breaks down.



(b) For the same turbine, instantaneous torque coefficient obtained through present DMST method for $\lambda = 1.50$ and $\lambda = \lambda_{C_{P_{max}}} = 4.50$.

Figure 6: Calculation results for a Darrieus turbine with three straight blades of $c = 0.20$ m chord, whose section is a NACA 0012 airfoil, with radius $R = 2$ m (thus $\sigma = 0.3$). Freestream velocity is $V_\infty = 5$ m/s. Aerodynamic forces calculation is based on Sheldahl and Klimas (1981) experimental data.

4 `dmst.m` function description and usage

This section contains a brief description of the `dmst.m` function, written in MATLAB language. The input arguments are the following:

- `n_st`, integer, number of streamtubes pairs;
- `B`, integer, number of blades;
- `c`, double, blade chord length m;
- `R`, double, rotor radius m;
- `lambda`, double, Tip Speed Ratio;
- `Vinf`, double, wind speed m/s;
- `aeroflag`, string. To be chosen among `'xrotor'`, `'skdata'` and `'simple'`. Selects between Cl and Cd calculation through a linear law and through `ClCd_Xrotor.m` function included in the Eli-TAARG library (`'xrotor'`), through two-variable interpolation on Sheldahl and Klimas (1981) experimental data (`'skdata'`), or through a linear law for C_l and a constant 100 DC value for C_d (`'simple'`).
- `varargin`, 7-by-1 double array. Input parameters for `ClCd_XRotor.m` function (see documentation).

while the output is

- `lambda_flag_us`, boolean, `1` if $a > 0.5$ somewhere upwind, `0` otherwise;
- `lambda_flag_ds`, boolean, `1` if $a > 0.5$ somewhere downwind, `0` otherwise;
- `lambda_eff_us`, `n_st`-by-1 double array, upwind local TSR;
- `lambda_eff_ds`, `n_st`-by-1 double array, downwind local TSR;
- `Vratiosq_us`, `n_st`-by-1 double array, upwind local velocity ratio squared;
- `Vratiosq_ds`, `n_st`-by-1 double array, downwind local velocity ratio squared;
- `counter_us`, `n_st`-by-1 integer array, upwind loop iteration counter;
- `counter_ds`, `n_st`-by-1 integer array, downwind loop iteration counter;
- `alpha_us`, `n_st`-by-1 double array, upwind local angle of attack;
- `alpha_ds`, `n_st`-by-1 double array, downwind local angle of attack;
- `Re_us`, `n_st`-by-1 double array, upwind local Reynolds number;
- `Re_ds`, `n_st`-by-1 double array, downwind local Reynolds number;
- `Cn_us`, `n_st`-by-1 double array, upwind local normal force coefficient;
- `Cn_ds`, `n_st`-by-1 double array, downwind local normal force coefficient;
- `Ct_us`, `n_st`-by-1 double array, upwind local tangential force coefficient;
- `Ct_ds`, `n_st`-by-1 double array, downwind local tangential force coefficient;
- `a_us`, `n_st`-by-1 double array, upwind local axial induction factor;
- `a_ds`, `n_st`-by-1 double array, downwind local axial induction factor;
- `instCq_us`, `n_st`-by-1 double array, instantaneous torque coefficient for the upwind cycle;
- `instCq_ds`, `n_st`-by-1 double array, instantaneous torque coefficient for the downwind cycle;
- `CP_us`, double, power coefficient generated by `B` blades in the upwind passage, averaged on the whole rotor revolution;
- `CP_ds`, double, power coefficient generated by `B` blades in the downwind passage, averaged on the whole rotor revolution;
- `CP`, double, average net power coefficient.

Once called, `dmst.m` performs some input checks, defines fundamental variables such as the angular spacing between streamtubes axes `Delta_theta` and the two arrays with the `n_st` values of the angular coordinates. If user set `aeroflag` to `'skdata'` and it was not previously called, `ReadAeroData.m` function is run to load experimental aerodynamic data (see section 4.1).

Then the upwind calculation loop begins. The loop is initialized by guessing induction factor u is equal to 1. This value is stored in the `u_us_old` variable. Following that, a `while` loop begins. `lambda_eff_us`, `Vratiosq_us`, `alpha_us` and `Re_us` are calculated. If the experimental aerodynamic model is selected, a check on Reynolds number is performed to ensure it does not exceed the minimum and maximum values of the available data, in order to avoid infinite looping due to `NaN` values that would come from data interpolation.

`Cn_us` and `Ct_us` values are then calculated. If the streamtube corresponds to the $-\pi/2$ or to the $\pi/2$ positions, the loop is exited. This is because it would not be possible to iterate over the induction factors, since equation (5) is singular for such values of θ; otherwise, equation (4) will be solved in u through the MATLAB `integral` routine and the result will be stored in the `u_us_new` variable.

A check on the absolute value of the difference between the new and the old induction values is performed. Should it be less then a certain tolerance (set to 10^{-2}), the characteristic variables will be updated with the new induction values and the `while` loop will be exited. Otherwise, `u_us_old` value will be updated with `u_us_new` and the iterations will continue.

If convergence is reached, instantaneous torque is caculated. This goes on for every θ value of the upstream cycle. Once done, average torque is calculated with equation (10a) through MATLAB `trapz` routine; then power and axial induction values `a_us` are computed, and if any of the `n_st` values of `a_us` is greater than 0.5, `lambda_flag_us` is set equal to `1`.

The downwind problem is solved with the same logic. The only differences are that equations (6), (7), (9) and (10b) require previously found values of upstream inductions (which therefore become inputs of the downwind problem and initialize downwind induction values), being $\lambda'_\theta = \frac{\lambda}{(2u-1)u'}$.

4.1 `ReadAeroData.m` function

This is a small function with no output arguments, which just loads aerodynamic data contained in a spreadsheet whose path is `filepath` (the only input argument), through MATLAB `readtable` function. Once called, it sets the flag `RADrunflag` to true, to ensure it will not be called again if aerodynamic data has been already loaded.

Data is stored in `Cl_data`, `Cd_data`, `ALPHA` and `RE` global variables. The last two are obtained through MATLAB `meshgrid` function, making everything ready for the two-variables interpolation operated by `Cl_dsmt.m` and `Cd_dmst.m` functions.

4.2 `Cl_dsmt.m` and `Cd_dmst.m` functions

If `aeroflag` is set to `'xrotor'`, these functions provide lift and drag 2-D coefficients employing a linear law for the former and the XROTOR model for the latter (Drela & Youngren, 2003). If it is set to `'skdata'`, instead, they interpolate over gridded data through MATLAB `interp2` function. Laslty, if `aeroflag` is set to `'simple'`, the same linear law is used for C_l and it is assumed that $C_d = 100$ DC. Thus, the input arguments are the local Reynolds number `Re`, the angle of attack `alpha`, the `aeroflag` string and the 7-by-1 input parameters array for the `ClCd_XRotor.m` function.

References

Drela, M., & Youngren, H. (2003). XROTOR 7.55 (Unix) *user guide*. Massachussetts Institute of Technology. https://web.mit.edu/drela/Public/web/xrotor/xrotor_doc.txt

Paraschivoiu, I. (2002). *Wind turbine design with emphasis on darrieus concept*. Polytechnic International Press.

Saber, E., Afify, R., & Elgamal, H. (2018). Performance of sb-vawt using a modified double multiple streamtube model. *Alexandria Engineering Journal.*

Sheldahl, R. E., & Klimas, P. C. (1981). Aerodynamic characteristics of seven symmetrical airfoil sections through 180-degree angle of attack for use in aerodynamic analysis of vertical axis wind turbines.

Tognaccini, R. (2021). *Lezioni di aerodinamica dell'ala rotante*. Università degli Studi di Napoli "Federico II".

A Code listings

dmst.m

```
function [ ...
    lambda_flag_us,lambda_flag_ds, ...
    lambda_eff_us,lambda_eff_ds, ...
    Vratiosq_us,Vratiosq_ds, ...
    counter_us,counter_ds, ...
    alpha_us,alpha_ds, ...
    Re_us,Re_ds, ...
    Cn_us,Cn_ds, ...
    Ct_us,Ct_ds, ...
    a_us,a_ds, ...
    instCQ_us,instCQ_ds, ...
    CP_us,CP_ds,CP ...
    ] = dmst(n_st,B,c,R,lambda,Vinf,aeroflag,varargin)

global RADrunflag RE

% Input checks
narginchk(7,8);

if strcmpi(aeroflag,'xrotor')

    % Check wether 'xrotor' model has been chosen, but 'input_v' was not
    % provided.
    if nargin == 7

        error(['<strong>ClCd_XRotor</strong> needs an input vector. ', ...
            'See function documentation.']);

    else

        input_v = varargin{1};

    end

elseif ~strcmpi(aeroflag,'xrotor')

    if nargin <= 8

        input_v = [];

    end

    if strcmpi(aeroflag,'skdata') && isempty(RADrunflag)

        % Check wether realistic model has been chosen, but aerodynamic
        % data was not previoulsy loaded.
        input_v  = [];
        filename = 'sandia0012data.xlsx';
        filepath = [cd,'/ExperimentalData/',filename];
        ReadAeroData(filepath);

    end

end
% End of input checks

Delta_theta = pi/n_st;
```

```

theta_us_seq = linspace(pi/2,-pi/2,n_st);
theta_ds_seq = linspace(pi/2,3*pi/2,n_st);

sigma = B*c/R;
nu = 1.5e-5;

%% Vars init
lambda_eff_us = zeros(n_st,1);
lambda_eff_ds = zeros(n_st,1);
Vratiosq_us   = zeros(n_st,1);
Vratiosq_ds   = zeros(n_st,1);
counter_us    = zeros(n_st,1);
counter_ds    = zeros(n_st,1);
instCQ_us     = zeros(n_st,1);
instCQ_ds     = zeros(n_st,1);
alpha_us      = zeros(n_st,1);
alpha_ds      = zeros(n_st,1);
Re_us         = zeros(n_st,1);
Re_ds         = zeros(n_st,1);
Cn_us         = zeros(n_st,1);
Cn_ds         = zeros(n_st,1);
Ct_us         = zeros(n_st,1);
Ct_ds         = zeros(n_st,1);
u_us          = zeros(n_st,1);
u_ds          = zeros(n_st,1);

%% Calc loops
disp('Entering upwind loop...');
for ind_theta = 1:n_st

    theta = theta_us_seq(ind_theta);

    u_us_old = 1;

    exitflag = -1;

    while exitflag == -1

        counter_us(ind_theta) = counter_us(ind_theta) + 1;

        lambda_eff_us(ind_theta) = lambda/u_us_old;

        Vratiosq_us(ind_theta) = ...
            (lambda_eff_us(ind_theta) - ...
            sin(theta))^2 + cos(theta)^2;

        alpha_us(ind_theta) = ...
            asin(cos(theta)/ ...
            sqrt(Vratiosq_us(ind_theta)));

        Re_us(ind_theta) = ...
            Vinf*sqrt(Vratiosq_us(ind_theta))*c/nu;

        % Check if local Reynolds number outranges tabulated values
        if strcmpi(aeroflag,'skdata')

            if Re_us(ind_theta) < RE(1)

                error(['Local Reynolds number is lower than minimum ', ...
                    'value (',num2str(RE(1)),') in the available data.' ...
                    ' Cannot lookup aerodynamics table and continue.', ...
```

```
                    newline, 'theta = ',num2str(theta), ...
                    ', lambda = ',num2str(lambda), ...
                    ', Re = ',num2str(Re_us(ind_theta)),'.']);

            elseif Re_us(ind_theta) > RE(end)

                error(['Local Reynolds number is grater than maximum', ...
                    'value (',num2str(RE(end)), ...
                    ') in the available data. Cannot lookup  ',...
                    'aerodynamics table and continue.', ...
                    newline, 'theta = ',num2str(theta), ...
                    ', lambda = ',num2str(lambda), ...
                    ', Re = ',num2str(Re_us(ind_theta)),'.']);

            end

        end

        Cn_us(ind_theta) = ...
            Cl_dmst(Re_us(ind_theta), ...
            alpha_us(ind_theta), ...
            aeroflag)* ...
            cos(alpha_us(ind_theta)) + ...
            Cd_dmst(Re_us(ind_theta), ...
            alpha_us(ind_theta), ...
            aeroflag, ...
            input_v)* ...
            sin(alpha_us(ind_theta));

        Ct_us(ind_theta) = ...
            Cl_dmst(Re_us(ind_theta), ...
            alpha_us(ind_theta), ...
            aeroflag)* ...
            sin(alpha_us(ind_theta)) - ...
            Cd_dmst(Re_us(ind_theta), ...
            alpha_us(ind_theta), ...
            aeroflag, ...
            input_v)* ...
            cos(alpha_us(ind_theta));


        if ind_theta == 1 || ind_theta == n_st

            % Exit loop where F_us would be singular...
            exitflag = 1;
            u_us(ind_theta) = 1;

        else

            % ...or find new induction value
            intfun = @(theta) ...
                Vratiosq_us(ind_theta).* ...
                (Cn_us(ind_theta).*cos(theta)./ ...
                abs(cos(theta)) - ...
                Ct_us(ind_theta).*sin(theta)./ ...
                abs(cos(theta)));

            F_us = sigma/ ...
                (8*Delta_theta)* ...
                integral(intfun, ...
                theta-Delta_theta/2,theta+Delta_theta/2);

```

```
            u_us_new = pi/(F_us + pi);

            % Convergence check
            if abs(u_us_old - u_us_new) < 1e-2

                exitflag = 1;

                % Update variables
                u_us(ind_theta) = u_us_new;

                lambda_eff_us(ind_theta) = lambda/u_us_new;

                Vratiosq_us(ind_theta) = ...
                    (lambda_eff_us(ind_theta) - ...
                    sin(theta))^2 + cos(theta)^2;

                alpha_us(ind_theta) = ...
                    asin(cos(theta)/ ...
                    sqrt(Vratiosq_us(ind_theta)));

                Re_us(ind_theta) =  ...
                    Vinf*sqrt(Vratiosq_us(ind_theta))*c/nu;

                Cn_us(ind_theta) = ...
                    Cl_dmst(Re_us(ind_theta), ...
                    alpha_us(ind_theta), ...
                    aeroflag)* ...
                    cos(alpha_us(ind_theta)) + ...
                    Cd_dmst(Re_us(ind_theta), ...
                    alpha_us(ind_theta), ...
                    aeroflag, ...
                    input_v)* ...
                    sin(alpha_us(ind_theta));

                Ct_us(ind_theta) =  ...
                    Cl_dmst(Re_us(ind_theta), ...
                    alpha_us(ind_theta), ...
                    aeroflag)* ...
                    sin(alpha_us(ind_theta)) - ...
                    Cd_dmst(Re_us(ind_theta), ...
                    alpha_us(ind_theta), ...
                    aeroflag, ...
                    input_v)* ...
                    cos(alpha_us(ind_theta));

            end

            u_us_old = u_us_new;

        end

    end

    instCQ_us(ind_theta) = ...
        sigma/4*u_us(ind_theta)^2*Vratiosq_us(ind_theta)*Ct_us(ind_theta);

end

CQ_us = sigma/(8*pi)* ...
    trapz(flip(theta_us_seq), ...
    Ct_us(:).*u_us(:).^2.* ...
    Vratiosq_us(:));
```

```
CP_us = CQ_us*lambda;

a_us = 1 - u_us;

disp(['...upwind problem solved in ', ...
    num2str(sum(counter_us)),' total iterations.']);

if any(a_us > 0.5)

    lambda_flag_us = 1;
    warning(['Rankine-Froude theory limit (a = 0.5) exceeded ', ...
        '<strong>upwind</strong>.',newline, ...
        'Entering downwind loop...']);

else

    lambda_flag_us = 0;
    disp('Entering downwind loop...');

end

for ind_theta = 1:n_st

    theta = theta_ds_seq(ind_theta);

    u_ds_old = u_us(ind_theta);

    exitflag = -1;

    while exitflag == -1

        counter_ds(ind_theta) = counter_us(ind_theta) + 1;

        u_us_local = u_us(ind_theta);

        lambda_eff_ds(ind_theta) = lambda/(2*u_us_local - 1)*u_ds_old;

        Vratiosq_ds(ind_theta) = ...
            (lambda_eff_ds(ind_theta) - ...
            sin(theta))^2 + cos(theta)^2;

        alpha_ds(ind_theta) = ...
            asin(cos(theta)/ ...
            sqrt(Vratiosq_ds(ind_theta)));

        Re_ds(ind_theta) = ...
            Vinf*sqrt(Vratiosq_ds(ind_theta))/nu;

        % Check if local Reynolds number outranges tabulated values
        if strcmpi(aeroflag,'skdata')

            if Re_ds(ind_theta) < RE(1)

                error(['Local Reynolds number is lower than minimum ', ...
                    'value (',num2str(RE(1)),') in the available data.' ...
                    ' Cannot lookup aerodynamics table and continue.', ...
                    newline, 'theta = ',num2str(theta), ...
                    ', lambda = ',num2str(lambda), ...
                    ', Re = ',num2str(Re_ds(ind_theta)),'.']);

            elseif Re_ds(ind_theta) > RE(end)
```

```

                error(['Local Reynolds number is grater than maximum', ...
                    'value (',num2str(RE(end)), ...
                    ') in the available data. Cannot lookup  ',...
                    'aerodynamics table and continue.', ...
                    newline, 'theta = ',num2str(theta), ...
                    ', lambda = ',num2str(lambda), ...
                    ', Re = ',num2str(Re_ds(ind_theta)),'.']);

            end

        end

        Cn_ds(ind_theta) = ...
            Cl_dmst(Re_ds(ind_theta), ...
            alpha_ds(ind_theta), ...
            aeroflag)* ...
            cos(alpha_ds(ind_theta)) + ...
            Cd_dmst(Re_ds(ind_theta), ...
            alpha_ds(ind_theta), ...
            aeroflag, ...
            input_v)* ...
            sin(alpha_ds(ind_theta));

        Ct_ds(ind_theta) = ...
            Cl_dmst(Re_ds(ind_theta), ...
            alpha_ds(ind_theta), ...
            aeroflag)* ...
            sin(alpha_ds(ind_theta)) - ...
            Cd_dmst(Re_ds(ind_theta), ...
            alpha_ds(ind_theta), ...
            aeroflag, ...
            input_v)* ...
            cos(alpha_ds(ind_theta));

        if ind_theta == 1 || ind_theta == n_st

            % Exit loop where F_ds would be singular
            exitflag = 1;
            u_ds(ind_theta) = 1;

        else

            % ...or find new induction value
            intfun = @(theta) ...
                Vratiosq_ds(ind_theta).* ...
                (Cn_ds(ind_theta).*cos(theta)./ ...
                abs(cos(theta)) - ...
                Ct_ds(ind_theta).*sin(theta)./ ...
                abs(cos(theta)));

            F_ds = sigma/(8*Delta_theta)* ...
                integral(intfun, ...
                theta-Delta_theta/2,theta+Delta_theta/2);

            u_ds_new = pi/(F_ds + pi);

            % Convergence check
            if abs(u_ds_old - u_ds_new) < 1e-2

                exitflag = 1;

```

```
                % Update variables
                u_ds(ind_theta) = u_ds_new;

                lambda_eff_ds(ind_theta) = ...
                    lambda/(2*u_us_local - 1)*u_ds_new;

                Vratiosq_ds(ind_theta) = ...
                    (lambda_eff_ds(ind_theta) - ...
                    sin(theta))^2 + cos(theta)^2;

                alpha_ds(ind_theta) = ...
                    asin(cos(theta)/ ...
                    sqrt(Vratiosq_ds(ind_theta)));

                Re_ds(ind_theta) = ...
                    Vinf*sqrt(Vratiosq_ds(ind_theta))*c/nu;

                Cn_ds(ind_theta) = ...
                    Cl_dmst(Re_ds(ind_theta), ...
                    alpha_ds(ind_theta), ...
                    aeroflag)* ...
                    cos(alpha_ds(ind_theta)) + ...
                    Cd_dmst(Re_ds(ind_theta), ...
                    alpha_ds(ind_theta), ...
                    aeroflag, ...
                    input_v)* ...
                    sin(alpha_ds(ind_theta));

                Ct_ds(ind_theta) = ...
                    Cl_dmst(Re_ds(ind_theta), ...
                    alpha_ds(ind_theta), ...
                    aeroflag)* ...
                    sin(alpha_ds(ind_theta)) - ...
                    Cd_dmst(Re_ds(ind_theta), ...
                    alpha_ds(ind_theta), ...
                    aeroflag, ...
                    input_v)* ...
                    cos(alpha_ds(ind_theta));

            end

            u_ds_old = u_ds_new;

        end

    end

    instCQ_ds(ind_theta) = ...
        sigma/4*...
        (u_ds(ind_theta)*(2*u_us(ind_theta) - 1))^2* ...
        Vratiosq_ds(ind_theta)*Ct_ds(ind_theta);

end

CQ_ds = sigma/(8*pi)*trapz(theta_ds_seq, ...
    Ct_ds(:).* ...
    ((2*u_us(ind_theta) - 1)*u_ds(:)).^2.* ...
    Vratiosq_ds(:));

CP_ds = CQ_ds*lambda;

CP = CP_us + CP_ds;
```

```
a_ds = 1 - u_ds;

disp(['...downwind problem solved in ', ...
    num2str(sum(counter_ds)),' total iterartions.']);

if any(a_ds > 0.5)

    lambda_flag_ds = 1;
    warning(['Rankine-Froude theory limit (a = 0.5) exceeded ', ...
        '<strong>downwind</strong>.']);

else

    lambda_flag_ds = 0;

end

end
```

ReadAeroData.m

```
function ReadAeroData(filepath)

global Cl_data Cd_data ALPHA RE RADrunflag

RADrunflag = true;

aerodata = readtable(filepath);

Re_data = aerodata{~isnan(aerodata{:,end}),end};

for i = 1:11

    j = 2*i;
    Cl_data(:,i) = aerodata{:,j};

end

for i = 1:11

    j = (2*i+1);
    Cd_data(:,i) = aerodata{:,j};

end

alpha = [flipud(-aerodata.alpha(2:end)); aerodata.alpha];

Cl_data = [flipud(-Cl_data(2:end,:));Cl_data];
Cd_data = [flipud(Cd_data(2:end,:));Cd_data];

[RE,ALPHA] = meshgrid(Re_data,alpha);

end
```

Cl_dmst.m

```
function Cl_val = Cl_dmst(Re,alpha,aeroflag)

global RE ALPHA Cl_data

if strcmpi(aeroflag,'skdata')

    Cl_val = interp2(RE,ALPHA,Cl_data,Re,alpha);

elseif strcmpi(aeroflag,'xrotor') || strcmpi(aeroflag,'simple')

    Cl_val = 2*pi*alpha;

else

    error("Spellcheck 'aeroflag'");

end

end
```

Cd_dmst.m

```
function Cd_val = Cd_dmst(Re,alpha,aeroflag,varargin)

narginchk(3,4);

if strcmpi(aeroflag,'xrotor') && nargin == 3

    error(['<strong>ClCd_XRotor</strong> also needs an input vector. ', ...
        'See function documentation.'])

end

global RE ALPHA Cd_data

if strcmpi(aeroflag,'skdata')

    Cd_val = interp2(RE,ALPHA,Cd_data,Re,alpha);

elseif strcmpi(aeroflag,'simple')

    Cd_val = 0.01;

elseif strcmpi(aeroflag,'xrotor')

    Cl           = Cl_dmst(Re,alpha,aeroflag);
    input_v      = varargin{1};
    [~,~,Cd_val] = ClCd_XRotor(input_v,Re,Cl);

else

    error("Spellcheck 'aeroflag'");

end

end
```

B XROTOR model for drag coefficient

The XROTOR software drag polar model is based upon the fact that, in the unstalled region, the blade element C_d has a quadratic dependence on C_l and a power-law dependence on Reynolds number as follows:

$$C_d = \left[C_{d_0} + b\,(C_{l_0} - C_l)^2\right]\left(\frac{Re}{Re_{\mathrm{ref}}}\right)^f \tag{13}$$

where C_{d_0} is the minimum drag coefficient, C_{l_0} is the lift coefficient value at which $C_d = C_{d_0}$, b is a coefficient for the quadratic term, Re is the local Reynolds number, Re_{ref} is the Reynolds number at which equation (13) is applied and f is a scaling exponent.

All said parameters were set according to XROTOR documentation (Drela & Youngren, 2003) and trimmed in such a way that the C_d resulting values were similar to the ones obtainable through the XFOIL software, in the linear region of the lift curve, and with $Re = Re_{\mathrm{ref}}$.