Development plan for the aerodynamic linearization in OpenFAST

E. Branlard, J. Jonkman

December 16, 2019

Introduction

This document describes the implementation of a new dynamic stall model in *OpenFAST* and a variation of the dynamic wake model (DBEMT). Both implementations use a state-space formulation, which allows for the linearization of the model. Currently, steady state aerodynamics are used during the linearization of the *OpenFAST* model due to the difficulty in linearizing the current models. Unsteady aerodynamic effects such as the dynamic stall and dynamic wake effects are thus not accounted for in the linearization, leading to errors in the estimates of the frequencies, damping and stability of the system. The models presented in this document can be used both for linearization and time domain analyses, leading to consistent results between the two approaches.

The new dynamic stall model is directly based on the work of Hansen et al. [1] (further referred to as the HGM-model) which in turn is based on the work of Beddoes and Leishman [2] (BL-model). The HGM model may be seen as a simplified version of the BL model, but, with an additional account for the pitching motion of the airfoil. The HGM model requires a subsets of the inputs of the BL-model, which is currently implemented in *OpenFAST* . Oye's dynamic stall model may also be implemented since it can be seen as a straightforward simplification of the HGM model, with one state instead of four.

The new dynamic wake model is a variation of the current DBEMT model, which is attributed to the work of Øye. The current implementation uses finite differences between two discrete time steps. Simplifying assumptions are introduced in the new model to allow for a continuous state-space formulation.

The main corpus of the document presents the implementation steps for the new models, starting from a high level view and progressively focusing on the specific programming aspects. The appendix of the document provides some background on the various topics touched upon in this document and additional levels of details. Most notations have been kept consistent with the developments from Hansen et al. [1].

Contents

Notations/definitions **4**

1 Overview **6**
1.1 Problem statement and approach . . . 6
1.2 Implementation overview . . . 6

2 Implementation of the CUA module **8**
2.1 Relation to the original module by Hansen et al. . . . 8
2.2 Registry types . . . 9
2.3 Helper functions . . . 11
2.4 Initialization routine . . . 12
2.5 Update state routine . . . 12
2.6 Output calculation routine . . . 13
2.7 Jacobians . . . 13
2.8 Integration of the CUA module into AeroDyn/DBEMT . . . 16

3 Implementation of CDBEMT **17**
3.1 Introduction . . . 17
3.2 Changes to DBEMT . . . 17
3.3 Integration of CDBEMT into BEMT . . . 18

4 Implementation of BEMT CCSD **19**
4.1 Implementation changes . . . 19
4.2 Integration of BEMT into AeroDyn . . . 20

5 AeroDyn integration and linearization capability **22**
5.1 Conditions for time domain or linearization . . . 22
5.2 Changes to CalcOutput . . . 22
5.3 Changes to the Jacobain routines . . . 22

A Flowcharts of BEMT routines **23**

B Dynamic inflow model **25**
B.1 Model of Øye . . . 25
B.2 Implementation 1: Finite difference formulation . . . 26
B.3 Implementation 2: State-space formulation . . . 27

C Polar functions **27**
C.1 Basic lift parameters . . . 27
C.2 Separation function . . . 28
C.3 Fully-separated polar . . . 29
C.4 Inviscid polar . . . 30

D Øye's dynamic stall model **30**

E HGM Beddoes dynamic stall model **30**
E.1 Overview . . . 31
E.2 State equations . . . 31
E.3 Outputs . . . 32
E.4 Vectorial notations . . . 32
E.5 Jacobian . . . 33

F Sample source code for the HGM model **33**
F.1 Separation function and fully separated polar . . . 33
F.2 State equations . . . 33
F.3 Outputs . . . 34

G Investigations on the CUA model **34**
G.1 Neglecting the acceleration term $\dot{U}$ . . . 34

H General considerations for the linearization of the chord loads of an airfoil **34**

I Solution of a first order linear differential equation **36**

Conventions

Initialisms, acronyms, abbreviations

- 34: "three-quarter" chord point (see further description below)
- ac: aerodynamic center
- AD: AeroDyn
- BEM/BEMT: blade element momentum theory
- CCSD: Calculate continuous state derivative
- DB/DBEMT: dynamic BEM theory (dynamic inflow/wake)
- CDBEMT: continuous state formulation of the DBEMT
- dyn: dynamic (airfoil coefficients)
- inv: inviscid (airfoil coefficients)
- fs: fully-separated (airfoil coefficients)
- op: operating point
- qs: quasi-steady
- HGM: Hansen Gaunaa Madsen model, see [1]
- st: steady (airfoil coefficients)
- UA: unsteady aerodynamics (dynamic stall)
- CUA: unsteady aerodynamics module using a continuous states (new module)

Notations

A list of the main notations used in this document is given below. Figure 1 may be used to follow the notations.

- Aerodynamic Center (AC): point of the airfoil cross section where the aerodynamic forces and moment are assumed to act. Usually close to the 1/4 chord point for a regular airfoil and at the center for a circular cross section
- "3/4" chord point: in the original formulation this point refers to the point on the chord axis located 3/4 chord behind the leading edge. This concept is here generalized to the point located mid-way between the aerodynamic center and the trailing edge, to account for aerodynamic center positions that differ strongly from a 1/4 chord point[1]. The notation 3/4 is yet kept in this document.
- d_{34}: distance (positive) between the aerodynamic center and the 3/4 chord point (close to $c/2$ for a regular airfoil).
- $\boldsymbol{v}_{ac}$: velocity vector at the aerodynamic center $\boldsymbol{v}_{ac} \triangleq [v_{x,ac}, v_{y,ac}]$ (coordinates assumed to be expressed in the airfoil section coordinate system)
- $\boldsymbol{v}_{34}$: velocity vector at the 3/4 chord point $\boldsymbol{v}_{34} \triangleq [v_{x,34}, v_{y,34}]$(coordinates assumed to be expressed in the airfoil section coordinate system)
- ω: rotational speed of the airfoil section (pitching/torsional rate)
- U_{ac}: velocity norm at the aerodynamic center. $U_{ac} \triangleq \|\boldsymbol{v}_{ac}\| = \sqrt{v_{x,ac}^2 + v_{y,ac}^2}$

[1]These positions are currently not available in OpenFAST, and as a first approximation, the "3/4" point may be placed at a distance $c/2$ behind the aerodynamic center. A more permanent change would be to provide the relative position of the aerodynamic center with respect to the chord line in the AeroDyn blade input file. With this information, different relevant positions may be obtained: trailing edge, leading edge, 3/4 chord, etc.

- α_{ac}: angle of attack at the aerodynamic center $\alpha_{ac} \triangleq \text{atan2}(v_{x,ac}, v_{y,ac})$
- α_{34}: angle of attack at the 3/4 chord point $\alpha_{34} \triangleq \text{atan2}(v_{x,34}, v_{y,34})$
- $\boldsymbol{f}$: vector of airfoil section loads: $\boldsymbol{f} \triangleq [f_x, f_y, m_z]$ (coordinates assumed to be expressed in the airfoil section coordinate system), units: (N/m,N/m,N)
- $\boldsymbol{C}$: vector of airfoil aerodynamic lift, drag and moment coefficients $\boldsymbol{C} \triangleq [C_l, C_d, C_m]$

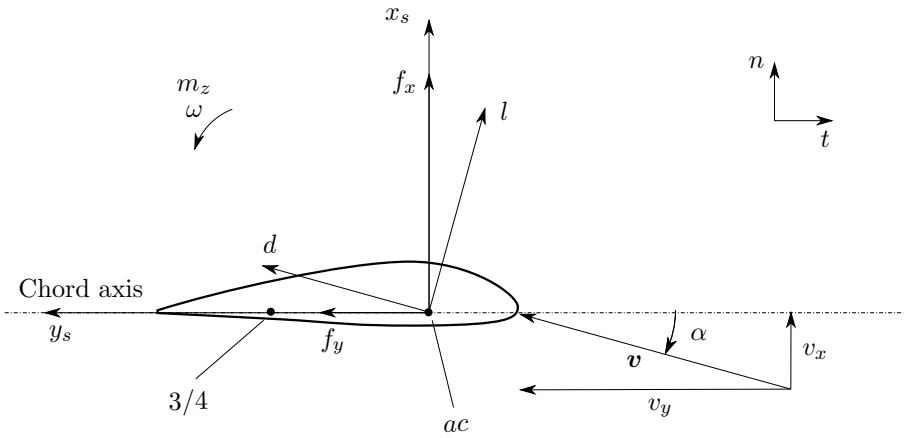

Figure 1: OpenFAST coordinate system for a blade cross section

Notations specific to the dynamic inflow module:

- $\boldsymbol{x}_{\text{DB}}$: the vector of states used by the dynamic BEM (dynamic inflow) module
- W_{qs}: the vector of quasi-steady induced velocities

Notations specific to the dynamic stall module:

- $\boldsymbol{x}_{\text{UA}}$: the vector of states (4 states per section) used by the unsteady aerodynamic (dynamic stall) module
- c, $C_{l,\alpha}$, α_0 are constant airfoil parameters: chord, lift slope and angle of attack of zero lift
- $f^{st}(\alpha)$ is the separation function, determined from the lift curve $C_L(\alpha)$ and which definition is given in subsection C.2
- A_1, A_2, b_1, b_2 are four constants, parameters of the HGM model, characteristic of the propagation of the wake vorticity for this airfoil
- T_f, T_p are two positive time constants, parameters of the HGM model, characteristic of the propagation time of the wake vorticity for this airfoil

1 Overview

1.1 Problem statement and approach

Linearization of the aerodynamics is not possible in AeroDyn since OpenFAST mostly supports linearization with continuous states, inputs and constraints, while currently:

- UA: uses discrete states.
- DBEMT: uses discrete inputs (and finite differences)

The following approach is followed to solve these issues:

- A new UA module is implemented (called CUA), which uses continuous states. This formulation is based on the model by Hansen et al. [1] (referred to as HGM model). The new module requires the structural rotational speed of each airfoil section ω.
- A variation of the DBEMT module is introduced (called CDBEMT), which only uses continuous values. This variations requires an estimation of the time derivative of the quasi-steady induced velocities. This time derivative is estimated based on the structural acceleration of each airfoil sections.

To support combinations of old and new modules the following features are required:

- CUA and CDBEMT need time integration routines (e.g. RK4, ABM4) to integrate the states independently of BEMT. Both modules involve a state equation of the form $\dot{\boldsymbol{x}} = \boldsymbol{X}(\boldsymbol{x}, \boldsymbol{u}, t)$, which is implemented as a routine called `CalcContinuousStateDerivative` (CCSD).
- Similarly, BEMT needs to have a CCSD routine and time integration routines. These routines are only used when both the new models are selected. For convenience, a routine called `BEMT_UpdateStates_CCSD` is added, and the responsibility to call this routine or the old routine (`BEMT_UpdateStates`) is given to AeroDyn.

The BEMT module is expressed using a constraint equation, and the constraint is tightly coupled to the UA and DBEMT modules. To ease the implementation of the BEMT CCSD routine, the constraint is solved for at the beginning of the routine, effectively removing the constraint from the state equation inputs (i.e. $\boldsymbol{X}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{z}, t) = \boldsymbol{X}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{z}|_{\boldsymbol{Z}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{z},t)=0}, t)$). A similar approach is used for the CalcOutput routine. The linearization is then performed at the AeroDyn level by perturbing the states and inputs given as arguments of the BEMT CCSD and CalcOutput routines. The amplitude of the perturbations need to be carefully selected as to be sufficiently large compared to the error associated with the resolution of the constraint.

1.2 Implementation overview

The following preliminary steps are needed:

- Implementation of the CUA module based on the HGM model. The implementation of this module is described in section 2, with the changes required to `AeroDyn/BEMT` presented in subsection 2.8. The model takes as inputs the relative velocity of the airfoil at the lifting line, and the pitching rate of the section, and outputs the unsteady airfoil coefficients. The main changes to AeroDyn/BEMT consist in: 1) adding the CUA module variables to the BEMT registry; 2) introducing conditional statements to switch between the UA and CUA module based on the input parameter `UAMod`; 3) providing the pitching rate of the airfoil sections to the CUA module.

- Update of DBEMT to include the continuous state option (CDBEMT). The implementation is detailed in section 3. The continuous states of the module are the induced velocities and accelerations. The module requires as input the quasi-steady induced accelerations. This information is approximated based on the structural acceleration of the blade sections, which need to be passed to AeroDyn and BEMT.

- Both modules may then be tested and compared to UA and DBEMT respectively.

The following steps are needed to allow for the linearization:

- Implementation of `BEMT_UpdateStates_CCSD` (see section 4). The integration is performed using "generic" time integration scheme routines (e.g. RK4, ABM4) and a CCSD routine which solve the BEMT constraint equation and call the CCSD of CUA and CDEMBT. The changes to AeroDyn are given in subsection 4.2, and simply consists of a switch between `BEMT_UpdateStates_CCSD` (if both continuous submodules are used) and `BEMT_UpdateStates` (otherwise).

- Implementation of the linearization routines in AeroDyn (see section 5). The linearization of the state and outputs is performed using finite differences. The method is similar to the one already implemented in AeroDyn (to linearize the constraints). Each state and input is perturbed and the corresponding partial derivative computed.

2 Implementation of the CUA module

A new submodule, named `CUA`, is added in `AeroDyn15`. This module is used when `UAMod=4`. This section provides an overview of the module and its differences with the model from Hansen et al.[1]. Then, the various components of the modules are described in successive sections: registry types, initialization, update of states, output calculation, and linearization.

2.1 Relation to the original module by Hansen et al.

The original model from Hansen et al. is described in Appendix E and in [1]. The current implementation differs from this model on different points. Figure 2 is used to illustrate the differences. Block 0 refer to the dynamics stall model of Hansen et al., block 1 is an intermediate block which is kept to ease future extension of the model, and block 2 is the interface of the dynamic stall module to be implemented. Using the notations introduced in section , the main differences from

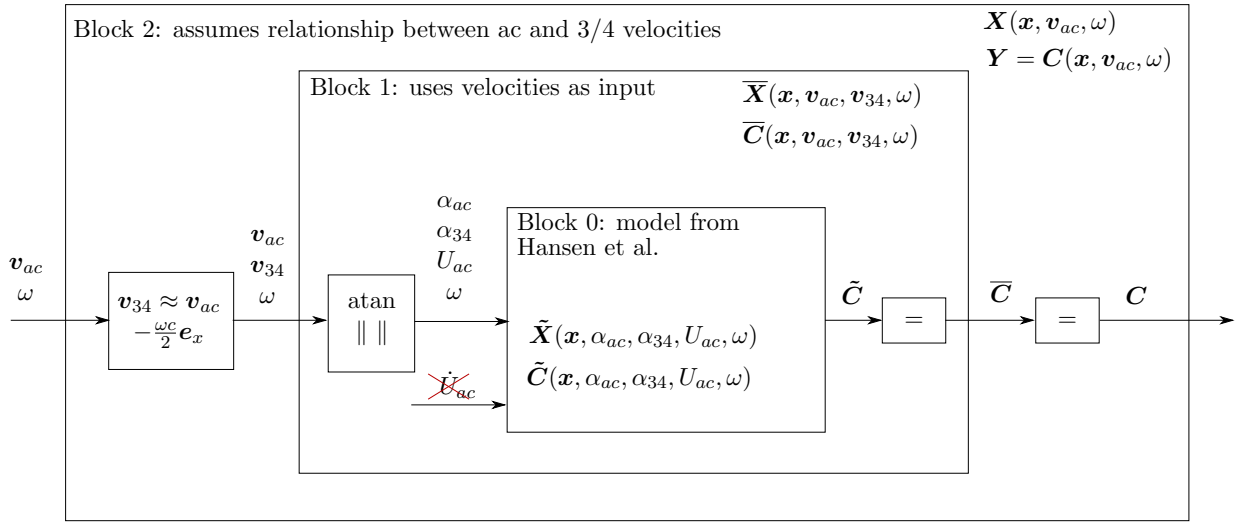

Figure 2: Inputs and outputs for different "blocks", where block 0 is the model from Hansen et al. [1], and block 2 represents the OpenFAST dynamics stall module.

the original model and the distinctions between the different blocks are as follows:

- The acceleration term $\dot{U}_{ac}$ is neglected in block 0, a choice justified in subsection G.1.
- Block 1 takes as input the velocity vectors at the aerodynamic center and the quarter chord ($\boldsymbol{v}_{ac}$, and $\boldsymbol{v}_{34}$). These are converted to angles of attacks and velocity norms before interacting with block 0.
- Block 2 takes as input the velocity at the aerodynamic center and derives a velocity at the "three-quarter" chord using the pitching rate ω[2]:

$$\boldsymbol{v}_{34} = \boldsymbol{v}_{ac} + \boldsymbol{\omega} \times \boldsymbol{r}_{ac \to 34} \approx \boldsymbol{v}_{ac} - \omega d_{34} \boldsymbol{e}_x \tag{1}$$

[2]This assumption amounts to neglecting the aerodynamic changes of the induced velocities (and turbulence inflow) along the chord. For a BEM calculation, this assumptions makes sense. A vortex code may only report the velocity at the lifting line or at the "3/4" point. Notations for block 1 are kept in this document if in the future this assumption is to be dropped.

where d_{34} is the distance between the aerodynamic point and the 3/4 point.

- Block 0 and block 1 return the airfoil coefficients $\boldsymbol{C} = [C_l, C_d, C_m]$.

To ease the linearization, different notations are introduced to distinguish between the different blocks. Different accents are added to the main functions of the blocks: the state function $\boldsymbol{X}$, the airfoil coefficient output $\boldsymbol{C}$ and the main output $\boldsymbol{Y}$. These functions return the same values but are function of different inputs:

$$\boldsymbol{X}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \omega) \triangleq \overline{\boldsymbol{X}}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \boldsymbol{v}_{34}, \omega) \equiv \tilde{\boldsymbol{X}}(\boldsymbol{x}, \alpha_{ac}, \alpha_{34}, U_{ac}, \omega) \tag{2}$$

$$\boldsymbol{Y}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \omega) \triangleq \boldsymbol{C}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \omega) \equiv \overline{\boldsymbol{C}}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \boldsymbol{v}_{34}, \omega) \equiv \tilde{\boldsymbol{C}}(\boldsymbol{x}, \alpha_{ac}, \alpha_{34}, U_{ac}, \omega) \tag{3}$$

$$\tag{4}$$

The distinction is only relevant for the linearization of the module.

2.2 Registry types

The variables and registry types of the module are given in Table 1. The main parameters are the airfoil parameters (polar data), and the same unsteady parameters as the ones defined for the Beddoes-Leishman model:

- A_1, A_2, b_1, b_2 are the four constants found in the profile input file of AeroDyn.
- T_f, T_p are the two constants also found in this profile file, and noted `T_f0` and `T_p`.

Different "misc" variables are used. These are temporary variables of the dynamic stall model, entirely determined by the states and the parameters. All the other variables should follow from the notations presented in section .

Table 1: Variables involved in the state space representation of the module

Symbol	Variable	Dim.	Description	Unit
			InitInputType	
c	`c`	-	Chord length at node	m
dt	`dt`	-	Time step used in numerical integration schemes	s
`AFI`	`AFI`	-	Airfoil parameters: C_l, C_d, C_m, $C_{l,\alpha}$, A_i, b_i, `BL_p:, T_f0, T_p0`	-
	`NumOuts`	-	The number of outputs for this module as requested in the input file	-
$\boldsymbol{x}$			**ContinuousStateType**	
x	`x`	4	States	rad
$\boldsymbol{u}$			**InputType**	
$\boldsymbol{v}_{ac}$	`v_ac`	2	Relative fluid velocity at the aerodynamics center	m/s
ω	`omega`	-	Pitching/twisting rate of the airfoil section	rad/s
$\boldsymbol{p}$			**ParameterType**	
c	`c`	-	Chord length	m
dt	`dt`	-	Time step used in numerical integration schemes	s
`AFI`	`AFI`	-	Airfoil parameters: C_l, C_d, C_m, $C_{l,\alpha}$, A_i, b_i, `BL_p:, T_f0, T_p0`	-
$\boldsymbol{y}$			**OutputType**	
$\boldsymbol{C}$	`Cl, Cd, Cm`	3	Airfoil coefficients $[C_{l,\mathrm{dyn}}, C_{d,\mathrm{dyn}}, C_{m,\mathrm{dyn}}]$	-
	`WriteOutput`	:	Outputs to be written to file	-

2.3 Helper functions

The dynamics stall module requires the evaluation of several functions which use interpolation based on discrete data (e.g. the polar data). These functions are described in this section and the best location to place them would appear to be the `AirfoilInfo` module. To speed up the execution of the repetitive evaluations of these functions, pre-computed interpolation weights may be used.

Static airfoil data The C_l, C_d, and C_m curves are already computed in OpenFAST using pre-computed interpolation weights. It appears that the routines `ComputeSteadyAirfoilCoefs` from `BEMTUncoupled` and `GetSteadyOutputs` from `UnsteadyAero` are very similar and it may be considered to merge them together (differences related to $C_{d,0}$ and Re). These functions may be merged and moved to the `AirfoilInfo` module.

Separation function The separation function $f_s^{st}(\alpha)$ described in subsection C.2 is similar from the one defined in the module `UnsteadyAero` via the functions `Get_f_from_Lookup` or `Get_f_from_Coeffs` depending on the variables `FLookup`[3]. In the current implementation, this function is defined using the lift coefficient and not the normal coefficient. The definition from Equation 60 is to be implemented. Care is required in the implementation of this function, as discussed in subsection C.2. A python source code to compute f_s^{st} is given in subsection F.1. Spline coefficients may be fitted to this function to speed up its evaluation.

Fully separated polar Once the separation function is known, the fully separated polar is defined using Equation 62. Spline coefficients may be fitted to this function to speed up its evaluation.

Derivatives of the polar functions For the linearization, the derivates of the polar functions with respect to α need to be evaluated. The derivative of the cubic splines can be used to create spline functions of the polar function derivatives. Alternatively, $\frac{dC_l^{st}}{d\alpha}$, $\frac{dC_d^{st}}{d\alpha}$, and $\frac{dC_m^{st}}{d\alpha}$ need to be obtained by finite differences, while the derivative of the separation function may be obtained as

$$\text{if } (C_{l,\alpha} \neq 0 \text{ and } \alpha \neq \alpha_0), \text{ or } C_l^{st} \neq 0, \quad \frac{df_s^{st}}{d\alpha} = \frac{2}{C_{l,\alpha}(\alpha-\alpha_0)}\left[\frac{dC_l^{st}}{d\alpha} - \frac{C_l^{st}(\alpha)}{\alpha-\alpha_0}\right]\left[2 - \sqrt{\frac{C_{l,\alpha}(\alpha-\alpha_0)}{C_l^{st}(\alpha)}}\right]$$

$$\text{else}, \quad \frac{df_s^{st}}{d\alpha} = 0$$

If $f_s^{st} \neq 1$, the derivative of the fully separated function is:

$$\frac{dC_l^{fs}}{d\alpha} = \frac{1}{(1-f_s^{st}(\alpha)^2)}\left[\left(\frac{dC_l^{st}}{d\alpha}(\alpha) - C_{l,\alpha}f_s^{st}(\alpha)\right)\left(1-f_s^{st}(\alpha)\right) + \left(C_l^{st}(\alpha) - C_{l,\alpha}(\alpha-\alpha_0)\right)\frac{df_s^{st}}{d\alpha}(\alpha)\right]$$

and $\frac{dC_l^{fs}}{d\alpha} = \frac{1}{2}\frac{dC_l^{st}}{d\alpha}$ otherwise. In the work of Hansen et al.[1, p.19] different recommendations are given if these functions are directly implemented.

[3] In this implementation, we suggest to use the lookup option, though it has been argued that a fit to a parametric function may be better suited given the uncertainty of the polar data, in particular in the stall region [3].

2.4 Initialization routine

The initialization routine performs the following steps:

- Initialize the chord, and time step based on inputs
- Stores a pointer to the airfoil information of the current section
- Initialize the four states $\boldsymbol{x} = [x_1, x_2, x_3, x_4]$ to 0.
- (Optional) Pre-compute the weights for the polar functions C_l, C_d, C_m, f_{st} and $C_{l,\text{fs}}$ (see subsection 2.3)

2.5 Update state routine

This routine integrates the states in time, calling the different time integration subroutines based on the integration scheme (`RK4`, `AB4`..). The core of all the time integration routines is the function `CalcContStateDeriv` which is described here. This function is only implemented for the case $U > 0$. The routine should abort if this is not the case, until an alternative formulation for this special case is found. This function computes $\dot{\boldsymbol{x}}$ using the right hand side of the equations 71-74. These equations are repeated below:

$$\begin{aligned}
\dot{x}_1 &= -T_u^{-1} b_1 \, x_1 + T_u^{-1} b_1 A_1 \alpha_{34} \\
\dot{x}_2 &= -T_u^{-1} b_2 \, x_2 + T_u^{-1} b_2 A_2 \alpha_{34} \\
\dot{x}_3 &= -T_p^{-1} x_3 + T_p^{-1} C_L^p \\
\dot{x}_4 &= -T_f^{-1} x_4 + T_f^{-1} f_s^{st}(\alpha_F), \qquad x_4 \in [0, 1]
\end{aligned}$$

with:

$$\begin{aligned}
T_u(t) &\triangleq \frac{1}{2} \min\left(\frac{c}{U_{ac}(t)}, 100\right) \\
\alpha_E(t) &\triangleq \alpha_{34}(t)(1 - A_1 - A_2) + x_1(t) + x_2(t) \\
C_L^p(t) &\triangleq C_{l,\alpha} \, (\alpha_E(t) - \alpha_0) + \pi T_u(t) \omega(t) \\
\alpha_F(t) &\triangleq \frac{x_3(t)}{C_{l,\alpha}} + \alpha_0
\end{aligned}$$

If $U_{ac} = 0$, then $T_u^{-1} = 0$, but other terms involving T_u are ill-defined. A limit was thus added to the definition of T_u so that it represents a large but finite time scale of 25s[4]. The validity of the dynamic stall model for low wind speed relative to the chord is yet questionable. Since the variable x_4 is effectively a separation parameter, the routine needs to ensure that this variables stays within the range $[0; 1]$. The airfoil parameters α_0 and $C_{l,\alpha}$ are found in the parameters, the functions C_l, C_d, C_m, f_{st} and $C_{l,fs}$ are functions based of the airfoil info parameter. A python code implementing this is given in subsection F.2).

[4]The value was set based on engineering judgment, the limit correspond for instance to a chord of 10m and a wind speed of 0.1m/s.

2.6 Output calculation routine

The function `CalcOutput` computes the unsteady airfoil coefficient, $\boldsymbol{C}$. The dynamic airfoil coefficients $C_{l,\text{dyn}}$, $C_{d,\text{dyn}}$, $C_{m,\text{dyn}}$ are obtained from the states as follows (see Appendix E or [1]):

$$C_{l,\text{dyn}}(t) = x_4(\alpha_E - \alpha_0)C_{l,\alpha} + (1 - x_4)C_{l,fs}(\alpha_E) + \pi T_u \omega \tag{5}$$

$$C_{d,\text{dyn}}(t) = C_d(\alpha_E) + (\alpha_{ac} - \alpha_E)C_{l,\text{dyn}} + [C_d(\alpha_E) - C_d(\alpha_0)]\,\Delta C''_{d,f} \tag{6}$$

$$C_{m,\text{dyn}}(t) = C_m(\alpha_E) - \frac{\pi}{2} T_u \omega \tag{7}$$

with

$$\Delta C''_{d,f} = \frac{\sqrt{f^{st}(\alpha_E)} - \sqrt{x_4}}{2} - \frac{f^{st}(\alpha_E) - x_4}{4}, \qquad x_4 \geq 0 \tag{8}$$

The variables T_u and α_E are recomputed within the routine based on equations 71-74. The airfoil parameters α_0 and $C_{l,\alpha}$ are found in the parameters, the functions C_l, C_d, C_m, f_{st} and $C_{l,fs}$ are functions based of the airfoil info parameter. An example of python source code for the output calculation is given in Appendix F.

2.7 Jacobians

NOTE: Jacobians do not need to be implemented. The linearization will be computed using finite differences at the aerodyn level.

This section provides the Jacobian of the state and output functions with respects to the inputs and states. Due to the change of variables involved between the different blocks (see Figure 2), some chain rules have to be applied between the functions and inputs of each block, requiring several intermediate variables to be defined. This approach is here chosen to help future extension of the code and ease the highlighting of potential errors in the derivations. To limit the number of steps though, the Jacobians of Block 1 are skipped and the link between block 2 and 0 is made directly.

The linearization is assumed to be performed about the operating point $(\boldsymbol{x}_{\text{op}}, \boldsymbol{v}_{ac,\text{op}}, \omega_{\text{op}})$, from which the following variables are directly derived by definitions $U_{ac,\text{op}}$, $\boldsymbol{v}_{34,\text{op}}$, α_{op}. The angle of attacks at the aerodynamic center and the 3/4 chord are linearized about the same angle of attack, noted, α_{op}. Based on these operating point values, the following variables are defined

$$T_{u,\text{op}} \triangleq \frac{c}{2U_{ac,\text{op}}}, \quad T_i \triangleq \frac{T_{u,\text{op}}}{b_i}, \quad f_{\text{op}} \triangleq f_s^{st}(\alpha_{\text{op}}) \tag{9}$$

$$C_{l,\text{op}} \triangleq C_l^{st}(\alpha_{\text{op}}), \quad C_{d,\text{op}} \triangleq C_d^{st}(\alpha_{\text{op}}), \quad C_{m,\text{op}} \triangleq C_m^{st}(\alpha_{\text{op}}) \tag{10}$$

Note: the code should check that the constants b_i are positive. The following variables are also

introduced (for the Jacobian of the outputs only):

$$c_{l,f} \triangleq C_{l,\alpha}(\alpha_{\text{op}} - \alpha_0) - C_{l,\text{op}} \tag{11}$$

$$c_{l,\alpha} \triangleq C_{l,\alpha} f_{\text{op}} + \left.\frac{dC_l^{st}}{d\alpha}\right|_{\alpha_{\text{op}}} (1 - f_{\text{op}}) \tag{12}$$

$$c_{d,f} \triangleq (C_{d,0} - C_{d,\text{op}}) \frac{1 - \sqrt{f_{\text{op}}}}{4\sqrt{f_{\text{op}}}} \quad \text{if } f_{\text{op}} \neq 0, \quad \text{else } c_{d,f} = 0 \tag{13}$$

$$c_{d,\alpha} \triangleq \left.\frac{dC_d^{st}}{d\alpha}\right|_{\alpha_{\text{op}}} - \left.\frac{df_s^{st}}{d\alpha}\right|_{\alpha_{\text{op}}} c_{d,f} \tag{14}$$

$$c_{m,\alpha} \triangleq \left.\frac{dC_m^{st}}{d\alpha}\right|_{\alpha_{\text{op}}} \tag{15}$$

"Block 0" jacobians For "block 0" (see Figure 2), the jacobians of the state function $\tilde{\boldsymbol{X}}$ and output function $\tilde{\boldsymbol{C}}$ are adapted from the report from Hansen et al.[1] as [5]:

$$\frac{\partial \tilde{\boldsymbol{X}}}{\partial \boldsymbol{x}} = \begin{pmatrix} -T_1^{-1} & 0 & 0 & 0 \\ 0 & -T_2^{-1} & 0 & 0 \\ T_p^{-1} C_{l,\alpha} & T_p^{-1} C_{l,\alpha} & -T_p^{-1} & 0 \\ 0 & 0 & T_f^{-1} \left.\frac{df_s^{st}}{d\alpha}\right|_{\alpha_{\text{op}}} C_{l,\alpha}^{-1} & -T_f^{-1} \end{pmatrix}, \quad \frac{\partial \tilde{\boldsymbol{X}}}{\partial U_{ac}} = 0 \tag{16}$$

$$\frac{\partial \tilde{\boldsymbol{X}}}{\partial \alpha_{34}} = \begin{pmatrix} T_1^{-1} A_1 \\ T_2^{-1} A_2 \\ T_p^{-1} C_{l,\alpha}(1 - A_1 - A_2) \\ 0 \end{pmatrix}, \qquad \frac{\partial \tilde{\boldsymbol{X}}}{\partial \omega} = \begin{pmatrix} 0 \\ 0 \\ \pi T_p^{-1} T_{u,\text{op}} \\ 0 \end{pmatrix}, \qquad \frac{\partial \tilde{\boldsymbol{X}}}{\partial \alpha_{ac}} = 0 \tag{17}$$

$$\frac{\partial \tilde{\boldsymbol{C}}}{\partial \boldsymbol{x}} = \begin{pmatrix} c_{l,\alpha} & c_{l,\alpha} & 0 & c_{l,f} \\ c_{d,\alpha} - C_{l,\text{op}} & c_{d,\alpha} - C_{l,\text{op}} & 0 & c_{d,f} \\ c_{m,\alpha} & c_{m,\alpha} & 0 & 0 \end{pmatrix}, \qquad \frac{\partial \tilde{\boldsymbol{C}}}{\partial \alpha_{34}} = (1 - A_1 - A_2) \begin{pmatrix} c_{l,\alpha} \\ c_{d,\alpha} - C_{l,\text{op}} \\ c_{m,\alpha} \end{pmatrix} \tag{18}$$

$$\frac{\partial \tilde{\boldsymbol{C}}}{\partial \alpha_{ac}} = \begin{pmatrix} 0 \\ C_{l,\text{op}} \\ 0 \end{pmatrix}, \qquad \frac{\partial \tilde{\boldsymbol{C}}}{\partial \omega} = \begin{pmatrix} \pi T_{u,\text{op}} \\ 0 \\ -\frac{\pi}{2} T_{u,\text{op}} \end{pmatrix}, \qquad \frac{\partial \tilde{\boldsymbol{C}}}{\partial U_{ac}} = 0 \tag{19}$$

Jacobians to relate Block 0 and Block 2 The jacobians involving the quantities at the aerodynamic center are obtained from the following definitions at the aerodynamic center:

$$\alpha_{ac} = \text{atan2}\, \frac{v_{x,ac}}{v_{y,ac}}, \quad U_{ac} = \sqrt{v_{x,ac}^2 + v_{y,ac}^2} \tag{20}$$

and similarly, using Equation 1, $v_{x,34} = v_{x,ac} - \omega d_{34}$, and $v_{y,34} = v_{y,ac}$, at the 3/4-chord point:

$$\alpha_{34} = \text{atan2}\, \frac{v_{x,34}}{v_{y,34}} = \text{atan2}\, \frac{v_{x,ac} - \omega d_{34}}{v_{y,ac}}, \quad U_{34} = \sqrt{v_{x,34}^2 + v_{y,34}^2} = \sqrt{(v_{x,ac} + \omega d_{34})^2 + v_{y,ac}^2} \tag{21}$$

[5] Some partial derivatives with respect to U_{ac} are taken as 0 in the equations below. This follows the HawcStab2 implementation, but sources of error are still being investigated. The following terms are zero: $\frac{\partial x_1}{\partial U_{ac}} = \frac{\partial x_2}{\partial U_{ac}} = 0$, but $\frac{\partial x_3}{\partial U_{ac}}$ may not be. A confusion on the definitions of the T_i variables is also possible, in which these variables would be constants and not functions of U_{ac}. More work is required to solve these issues.

The following partial derivatives follow:

$$\frac{\partial \alpha_{ac}}{\partial \boldsymbol{v}_{ac}} = \begin{bmatrix} \frac{v_{y,ac}}{U_{ac}^2} & \frac{-v_{x,ac}}{U_{ac}^2} \end{bmatrix}, \qquad \frac{\partial U_{ac}}{\partial \boldsymbol{v}_{ac}} = \begin{bmatrix} \frac{v_{x,ac}}{U_{ac}} & \frac{v_{y,ac}}{U_{ac}} \end{bmatrix} \tag{22}$$

$$\frac{\partial \alpha_{34}}{\partial \boldsymbol{v}_{ac}} = \begin{bmatrix} \frac{v_{y,ac}}{U_{34}^2} & \frac{-v_{x,ac} + \omega d_{34}}{U_{34}^2} \end{bmatrix} \tag{23}$$

$$\frac{\partial \alpha_{34}}{\partial \omega} = -\frac{c}{2}\frac{v_{y,ac}}{U_{34}^2}, \qquad \frac{\partial \alpha_{ac}}{\partial \omega} = \frac{\partial U_{ac}}{\partial \omega} = 0 \tag{24}$$

"Block 2" jacobians The jacobians of $\hat{\boldsymbol{X}}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \omega)$ and $\hat{\boldsymbol{C}}(\boldsymbol{x}, \boldsymbol{v}_{ac}, \omega)$, are expressed as functions of the ones of Block 0 (i.e. the ones of $\tilde{\boldsymbol{X}}(\boldsymbol{x}, \alpha_{ac}, \alpha_{34}, \omega)$ and $\tilde{\boldsymbol{C}}(\boldsymbol{x}, \alpha_{ac}, \alpha_{34}, \omega)$), using the chain rule and the relationships developed above between the velocities, angle of attacks and pitching rate.

$$\frac{\partial \hat{\boldsymbol{X}}}{\partial x} = \frac{\partial \tilde{\boldsymbol{X}}}{\partial x} \tag{25}$$

$$\frac{\partial \hat{\boldsymbol{C}}}{\partial x} = \frac{\partial \tilde{\boldsymbol{C}}}{\partial x} \tag{26}$$

$$\frac{\partial \hat{\boldsymbol{X}}}{\partial \boldsymbol{v}_{ac}} = \frac{\partial \tilde{\boldsymbol{X}}}{\partial \alpha_{ac}}\frac{\partial \alpha_{ac}}{\partial \boldsymbol{v}_{ac}} + \frac{\partial \tilde{\boldsymbol{X}}}{\partial \alpha_{34}}\frac{\partial \alpha_{34}}{\partial \boldsymbol{v}_{ac}} \tag{27}$$

$$\frac{\partial \hat{\boldsymbol{C}}}{\partial \boldsymbol{v}_{ac}} = \frac{\partial \tilde{\boldsymbol{C}}}{\partial \alpha_{ac}}\frac{\partial \alpha_{ac}}{\partial \boldsymbol{v}_{ac}} + \frac{\partial \tilde{\boldsymbol{C}}}{\partial \alpha_{34}}\frac{\partial \alpha_{34}}{\partial \boldsymbol{v}_{ac}} \tag{28}$$

$$\frac{\partial \hat{\boldsymbol{X}}}{\partial \omega} = \frac{\partial \tilde{\boldsymbol{X}}}{\partial \alpha_{34}}\frac{\partial \alpha_{34}}{\partial \omega} + \frac{\partial \tilde{\boldsymbol{X}}}{\partial \omega} \tag{29}$$

$$\frac{\partial \hat{\boldsymbol{C}}}{\partial \omega} = \frac{\partial \tilde{\boldsymbol{C}}}{\partial \alpha_{34}}\frac{\partial \alpha_{34}}{\partial \omega} + \frac{\partial \tilde{\boldsymbol{C}}}{\partial \omega} \tag{30}$$

Final outputs jacobians The final output of the module are the sectional loads $\boldsymbol{Y} = \boldsymbol{f} = \boldsymbol{MC}$, where $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{v}_{ac})$ is defined in Appendix H. The jacobians of the outputs are:

$$\frac{\partial \boldsymbol{Y}}{\partial x} = \boldsymbol{M}_{\text{op}} \left.\frac{\partial \hat{\boldsymbol{C}}}{\partial x}\right|_{\text{op}} \tag{31}$$

$$\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{v}_{ac}} = \left.\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{v}_{ac}}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} + \boldsymbol{M}_{\text{op}} \left.\frac{\partial \hat{\boldsymbol{C}}}{\partial \boldsymbol{v}_{ac}}\right|_{\text{op}} \tag{32}$$

$$\frac{\partial \boldsymbol{Y}}{\partial \omega} = \boldsymbol{M}_{\text{op}} \left.\frac{\partial \hat{\boldsymbol{C}}}{\partial \omega}\right|_{\text{op}} \tag{33}$$

where Equation 88 and Equation 93 provide:

$$\left.\frac{\partial \boldsymbol{M}}{\partial v_{x,ac}}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} = \frac{1}{2}\rho c \begin{bmatrix} \frac{v_{x,ac}}{U_{ac}}\left(v_{x,ac}C_l + U_{ac}v_{y,ac}C_d\right) + U_{ac}C_l \\ \frac{v_{x,ac}}{U_{ac}}\left(-v_{y,ac}C_l + U_{ac}v_{x,ac}C_d\right) + U_{ac}C_d \\ 2v_{x,ac}\, c\, C_m \end{bmatrix}_{\text{op}} \tag{34}$$

$$\left.\frac{\partial \boldsymbol{M}}{\partial v_{y,ac}}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} = \frac{1}{2}\rho c \begin{bmatrix} \frac{v_{y,ac}}{U_{ac}}\left(v_{x,ac}C_l + U_{ac}v_{y,ac}C_d\right) + U_{ac}C_d \\ \frac{v_{y,ac}}{U_{ac}}\left(-v_{y,ac}C_l + U_{ac}v_{x,ac}C_d\right) - U_{ac}C_l \\ 2v_{y,ac}\, c\, C_m \end{bmatrix}_{\text{op}} \tag{35}$$

$$\boldsymbol{M}_{\text{op}} = \frac{1}{2}\rho c U_{ac,\text{op}}^2 \begin{bmatrix} \cos\alpha_{\text{op}} & \sin\alpha_{\text{op}} & 0 \\ -\sin\alpha_{\text{op}} & \cos\alpha_{\text{op}} & 0 \\ 0 & 0 & c \end{bmatrix} \tag{36}$$

2.8 Integration of the CUA module into AeroDyn/DBEMT

The main changes to AeroDyn/BEMT to integrate to CUA module are:

- Adding the CUA module variables to the BEMT registry: misc, parameters and continuous states are added.
- Introduce conditional statements to switch between the UA and CUA module based on the input parameter `UAMod`. Such switches are introduced around all the calls to the `*UA_*` routines. The check for `UA_flag` can be kept common between UA and CUA. For ease of implementation, `BEMT_UpdateStates` may be split into several subroutines, each performing a dedicated task: computing the quasi-steady bem inductions, updating UA/UAC, and updating DBEMT. This will require the repetition of the loops over blades and sections, and will require the storage of some intermediate variables that are currently scalar variables computed at a given section and blade position (e.g. `phitmp`). Such split will ease the call to `UAC_UpdateStates`, if this module is implemented such that it computes all sections at once. The split will also help the implementation of `BEMT_CCSD` (see section 4).
- Provide the pitching rate of each airfoil sections to the CUA module. This information needs to be passed as input to the CUA module. The pitching rate variable is available in the AeroDyn input mesh and can be passed as BEMT inputs, or as CUA inputs directly, within the `SetInputsForBEMT` routine. Passing the variables from AeroDyn to CUA directly avoids adding an additional input to the BEMT registry.

3 Implementation of CDBEMT

3.1 Introduction

The main theory of the dynamic inflow model is provided in Appendix B together with a description of the current and new implementations. The state equation of CDBEMT is:

$$\begin{bmatrix} \dot{\boldsymbol{W}} \\ \ddot{\boldsymbol{W}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_2 \\ -\frac{1}{\tau_1\tau_2}\boldsymbol{I}_2 & -\frac{1}{\tau_1\tau_2}(\tau_1+\tau_2)\boldsymbol{I}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{W} \\ \dot{\boldsymbol{W}} \end{bmatrix} + \frac{1}{\tau_1\tau_2} \begin{bmatrix} 0 & 0 \\ 1 & k\tau_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_{\text{qs}} \\ \dot{\boldsymbol{W}}_{\text{qs}} \end{bmatrix} \tag{37}$$

where the state vector is $\boldsymbol{x} = [\boldsymbol{W}; \dot{\boldsymbol{W}}]$ (here, $\boldsymbol{W}$ corresponds to the variable `x%vind` in the code), and the input vector is $\boldsymbol{u} = [\boldsymbol{W}_{\text{qs}}; \dot{\boldsymbol{W}}_{\text{qs}}]$ (where $\boldsymbol{W}_{\text{qs}}$ corresponds to the variable `u%vind_s` in the code). The outputs are $\boldsymbol{y} = \boldsymbol{W}$. The module variant may be activated with the input option `DBEMT_Mod=3` [6]. A scheme of the main routines and variables of the submodule DBEMT is shown in Figure 3.

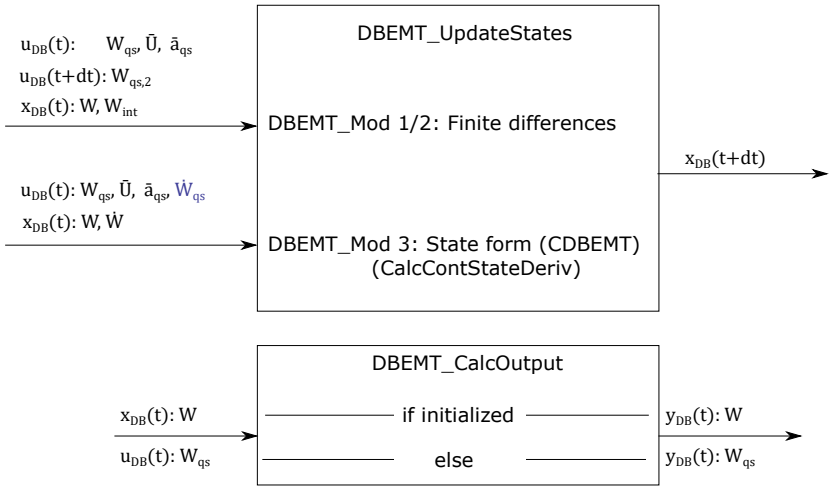

Figure 3: Scheme of DBEMT. The option `DBEMT_Mod`=3 is added

3.2 Changes to DBEMT

The following changes are required for the DBEMT module:

- Add the states and inputs to the registry (the time derivatives $\dot{\boldsymbol{W}}$ and $\dot{\boldsymbol{W}}_{\text{qs}}$)
- Implement a CCSD that returns the RHS of Equation 37
- Implement time-integration scheme routines (e.g. RK4, ABM4), to be used by the `UpdateStates` routine
- At init, the states are initialized to zero (unless the module callee can provide a guess). On the first call to update states, the inputs may be used to initialize the states (i.e. $\boldsymbol{W} = \boldsymbol{W}_{\text{qs}}$ and $\dot{\boldsymbol{W}} = \dot{\boldsymbol{W}}_{\text{qs}}$).

[6] In this variant it is assumed that τ_1 is constant. A variant `DBEMT_Mod=4` may be introduced with a varying τ_1. This would require adding τ_1 and $\dot{\tau}_1$ to the states. This approach is currently not described.

- Introduce switches in the code between the new and old methods based on `DBEMT_Mod`.

3.3 Integration of CDBEMT into BEMT

The inputs to DBEMT are set in the routine `calculate_Inductions_from_DBEMT`. The quasi steady induction is currently set as:

$$\boldsymbol{W}_{\text{qs}} = -a_{\text{qs}} V_x \boldsymbol{e}_x + a'_{\text{qs}} V_y \boldsymbol{e}_y \tag{38}$$

where V_x and V_y are the component of the relative wind without the wake inductions, $\boldsymbol{V} = \boldsymbol{V}_{\text{wind}} - \boldsymbol{V}_{\text{elast}}$, expressed in the section coordinates without pitch and sweep but with prebend, referred to as the no-sweep-pitch-twist coordinate system[7]. The time derivative of Equation 38 is:

$$\dot{\boldsymbol{W}}_{\text{qs}} = [-\dot{a}_{\text{qs}} V_x - a_{\text{qs}} \dot{V}_x - \omega_z a'_{\text{qs}} V_y] \boldsymbol{e}_x + [\dot{a'}_{\text{qs}} V_y + a'_{\text{qs}} \dot{V}_y - \omega_z a_{\text{qs}} V_x] \boldsymbol{e}_y \tag{39}$$

where ω_z is the rotation of the no-sweep-pitch-twist coordinate system around z. The relative acceleration, $\dot{\boldsymbol{V}}$, is approximated by neglecting the time change of the wind speed:

$$\dot{\boldsymbol{V}} = \dot{\boldsymbol{V}}_{\text{wind}} - \dot{\boldsymbol{V}}_{\text{elast}} \approx -\dot{\boldsymbol{V}}_{\text{elast}} \tag{40}$$

The time rate of the axial and tangential inductions are also neglected, i.e. $\dot{a}_{\text{qs}} \approx 0$ and $\dot{a}'_{\text{qs}} \approx 0$. With these assumptions, Equation 39 reduces to[8]:

$$\dot{\boldsymbol{W}}_{\text{qs}} \approx [a_{\text{qs}} \dot{V}_{x,\text{elast}} - \omega_z a'_{\text{qs}} V_y] \boldsymbol{e}_x + [-a'_{\text{qs}} \dot{V}_{y,\text{elast}} - \omega_z a_{\text{qs}} V_x] \boldsymbol{e}_y \tag{41}$$

The above equation is implemented in `calculate_Inductions_from_DBEMT`, to provide inputs to the DBEMT module.

[7] In the future this may be replaced by the relative wind normal and tangential to the rotor plane

[8] The impact of these assumptions should be evaluated in future work.

4 Implementation of BEMT CCSD

The implementation of the BEMT CCSD routine is described in this section. This routine is used both for time domain simulations and linearizations.

4.1 Implementation changes

The following steps are required:

- Implement a CCSD routine. The structure of this routine strongly follows the logic of the existing routine `DBEMT_UpdateStates`. The structure is illustrated in Figure 4.
- Implement time-integration scheme routines (e.g. RK4, ABM4)
- Add a routine called `BEMT_UpdateStates_CCSD` to use the above. This routine follows the framework and accepts several inputs at different times. The structure is illustrated in Figure 5.

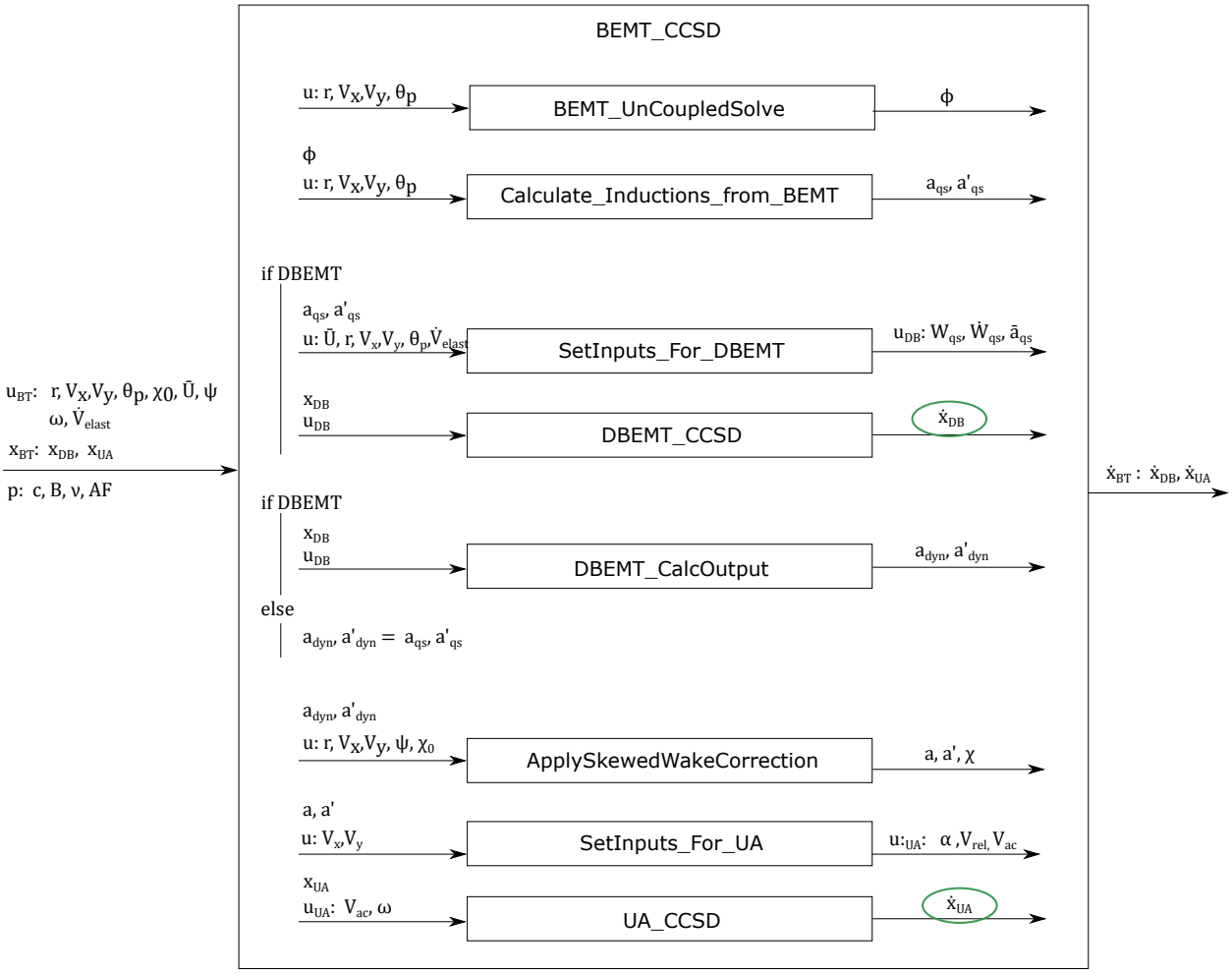

Figure 4: Structure of the `BEMT_CCSD` routine, called by `BEMT_UpdateStates_CCSD`.

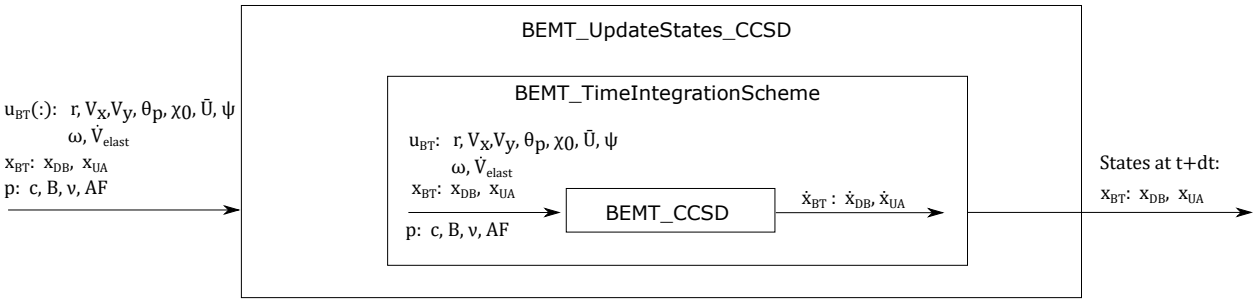

Figure 5: Structure of the `BEMT_UpdateStates_CCSD` routine

4.2 Integration of BEMT into AeroDyn

The changes needed to AeroDyn are the following:

- Provide the additional inputs to BEMT within the routine `SetInputs`: $\boldsymbol{\omega}$ the rotational velocity of each airfoil section, and $\dot{\boldsymbol{V}}_{\text{elast}}$ the structural acceleration of each section
- The routine `AD_UpdateStates` is rearranged such that, when the new modules are used (i.e. `UAMod=4` and `DBEMT_Mod`=3), then:
  - all inputs given to AeroDyn, i.e. $\boldsymbol{u}_{\text{AD}}(:)$, are converted to inputs for BEMT for all times, i.e. $\boldsymbol{u}_{\text{BT}}(:)$ (in the current implementation inputs are interpolated at t and $t + dt$ but not at all `utimes`)
  - the routine `BEMT_UpdateStates_CCSD` is used to integrate the states.

The `UpdateStates` routine of AeroDyn is illustrated in Figure 6.

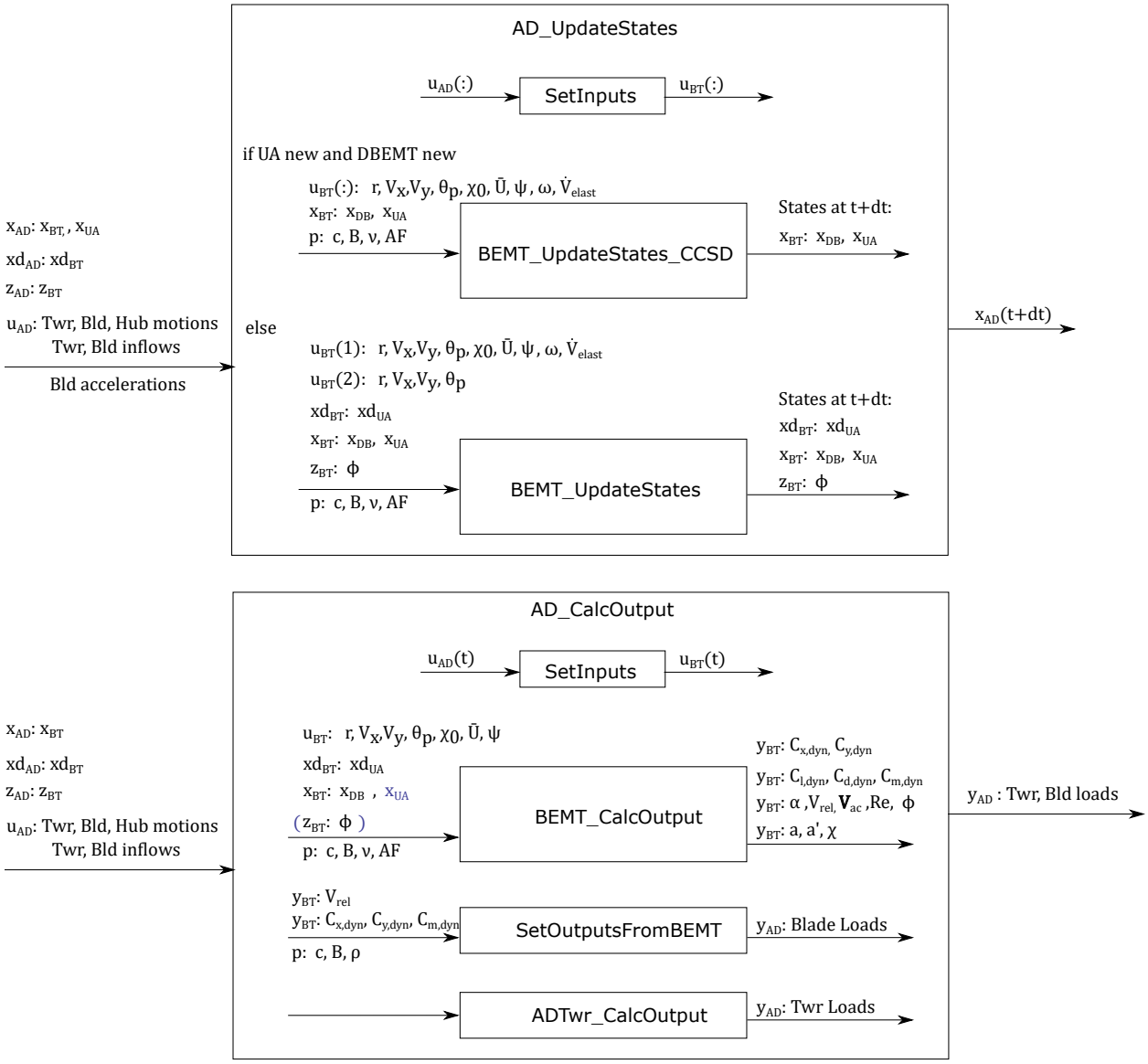

Figure 6: Scheme of AeroDyn states and output routines

5 AeroDyn integration and linearization capability

A "total derivative" approach is used for the linearization, and the constraint is assumed to be eliminated within the $\boldsymbol{X}$ and $\boldsymbol{Y}$ routines, such that the constraint state $\boldsymbol{z}$ is intrinsically handled in the linearization and doesn't result in associated Jacobians.

5.1 Conditions for time domain or linearization

No restrictions are present for time domain simulations, any combinations of inputs are allowed.(combinations of `DBEMT_Mod`, `UAMod` and `AFAeroMod`). For the linearization to be allowed, the following options are required: `DBEMT_Mod = 3` if `WakeMod = 2`, `UAMod = 4` if `AFAeroMod=2`

5.2 Changes to CalcOutput

The `BEMT_CalcOutput` routine needs to be adapted such that the constraint equation is solved for each time the routine is called when both new modules are used (i.e. when `UAMod=4` and `DBEMT_Mod`=3, either for linearization or time domain simulation). This condition can be gathered with the condition already present when the module is not initialized, which also triggers a computation of the constraint.

5.3 Changes to the Jacobain routines

The Jacobians routines of AeroDyn need to be modified to use the CCSD routine when both new modules are used. The partial derivatives are evaluated using finite differences, in harmony with the methods already in place. The changes to the different Jacobian routines are listed below[9].

`JacobianPInput`

- $\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{u}}$: this is now to be computed by perturbations of the inputs (similar to what is done for $\boldsymbol{Y}$). The CCSD function of BEMT is called to estimate the changes in the state function $\boldsymbol{X}$ for changes in inputs.
- $\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{u}}$: this should not be computed when the new modules are used
- $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{u}}$: no changes required, z should be ignored and recomputed by `AD_CalcOutput`.

`JacobianPContState`

- $\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{x}}$: computed by finite differences, perturbation of the states and evaluation of the CCSD routine.
- $\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{x}}$: not computed
- $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{x}}$: computed by finite differences, perturbation of the states and evaluation of the output routine.

`JacobianPConstrState`

- $\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{z}}$: not computed when both new modules are used
- $\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{z}}$: not computed when both new modules are used
- $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{z}}$: not computed when both new modules are used

[9]The changes mentioned refer to the OpenFAST branch and might not reflect the changes already incorporated in the Envision branch. In particular, the OpenFAST branch still contains partial derivatives associated with the constraint states.

A Flowcharts of BEMT routines

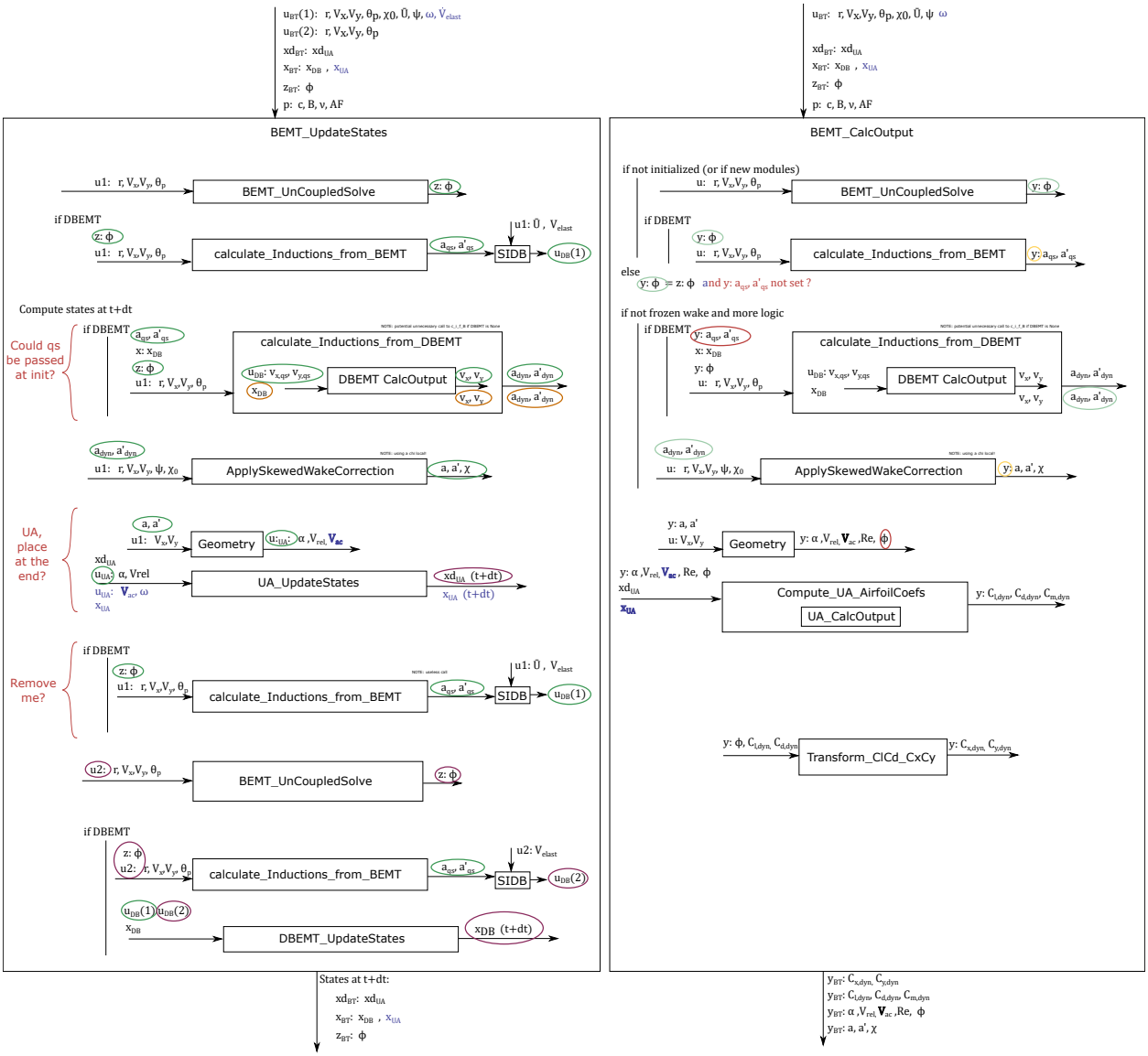

Figure 7: Scheme of BEMT update states and output routines

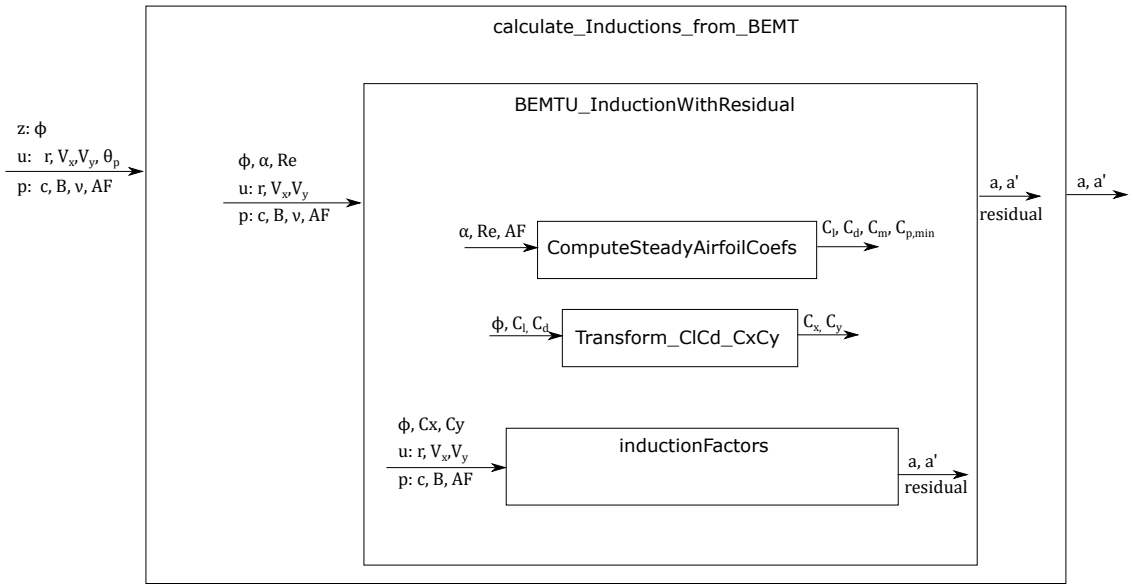

Figure 8: Scheme of main subroutines used in BEMT

B Dynamic inflow model

B.1 Model of Øye

Introduction A change in the rotor loading will result in a change in the wake configuration and the induced velocities. For a change between two loading configurations, it will take time for the wake to go from one equilibrium state to another. This phenomenon is referred to as the *dynamic inflow* or *dynamic wake*. Using a vorticity formulation, the change of loading will imply a change of vorticity emitted into the wake. The new value of the vorticity propagates progressively downstream replacing the old values and hence the induced velocity from this vorticity changes progressively. The time scales involved in the dynamic wake are thus related to the convection velocity of the vorticity in the wake. Due to the difference in convection velocity in the wake it is expected that the time delay towards the tip is shorter than towards the root [4].

Dynamic model of Øye The dynamic model of Øye is presented in the review of Snel and Schepers [4] and the book of Hansen [5]. The model is written using two first order differential equations:

$$\boldsymbol{W}_{\text{int}} + \tau_1 \frac{\mathrm{d}\boldsymbol{W}_{\text{int}}}{\mathrm{d}t} = \boldsymbol{W}_{\text{qs}} + k\tau_1 \frac{\mathrm{d}\boldsymbol{W}_{\text{qs}}}{\mathrm{d}t} \tag{42}$$

$$\boldsymbol{W} + \tau_2 \frac{\mathrm{d}\boldsymbol{W}}{\mathrm{d}t} = \boldsymbol{W}_{\text{int}} \tag{43}$$

where $\boldsymbol{W}$ is the actual induction at the rotor (at a given blade position and radial position), $\boldsymbol{W}_{\text{qs}}$ is the quasi-steady induction and $\boldsymbol{W}_{\text{int}}$ is an intermediate value coupling the quasi-steady and the actual inductions. The constant k is usually chosen as $k = 0.6$. The steady solution of the systems leads to $\boldsymbol{W} = \boldsymbol{W}_{\text{qs}}$. Within an unsteady BEM step, once the values of a_{qs} and a'_{qs} are computed, the quasi-steady induction vector is determined as

$$\boldsymbol{W}_{\text{qs}} = -a_{\text{qs}} V_x \boldsymbol{e}_x + a'_{\text{qs}} V_y \boldsymbol{e}_y \tag{44}$$

while the time constants are modelled as:

$$\tau_1 = \frac{1.1}{1 - 1.3\min(\overline{a}, 0.5)} \frac{R}{\overline{U}_0}, \qquad \tau_2 = \left[0.39 - 0.26\left(\frac{r}{R}\right)^2\right] \tau_1 \tag{45}$$

with $\overline{a}$ the mean axial induction over the rotor and $\overline{U}_0$ is the mean free stream velocity over the rotor[10]. The limit of 0.5 in Equation 46 is given without justifications in the original report [4]. Further work would be required to investigate this constraint. Equations 43-44 may be rewritten into a single differential equation as:

$$\boldsymbol{W}_{\text{qs}} + k\tau_1 \frac{\mathrm{d}\boldsymbol{W}_{\text{qs}}}{\mathrm{d}t} = \boldsymbol{W} + \left(\tau_1 + \tau_2 + \tau_1 \frac{\mathrm{d}\tau_2}{\mathrm{d}t}\right) \frac{\mathrm{d}\boldsymbol{W}}{\mathrm{d}t} + \tau_1 \tau_2 \frac{\mathrm{d}^2 \boldsymbol{W}}{\mathrm{d}t^2} \tag{46}$$

The above may be rearranged into a non-linear first order system:

$$\begin{bmatrix} \dot{\boldsymbol{W}} \\ \ddot{\boldsymbol{W}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_2 \\ -\frac{1}{\tau_1\tau_2}\boldsymbol{I}_2 & -\frac{1}{\tau_1\tau_2}(\tau_1 + \tau_2 + \tau_1\dot{\tau}_2)\boldsymbol{I}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{W} \\ \dot{\boldsymbol{W}} \end{bmatrix} + \frac{1}{\tau_1\tau_2} \begin{bmatrix} 0 & 0 \\ 1 & k\tau_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_{\text{qs}} \\ \dot{\boldsymbol{W}}_{\text{qs}} \end{bmatrix} \tag{47}$$

[10] In Bladed 4.7, $\overline{U}_0$ is estimated using the annulus at 70% radius. In AeroDyn15, the mean "over the rotor" is actually computed from the values at each blade node.

B.2 Implementation 1: Finite difference formulation

The numerical resolution of Equations 43-44 may be done in different ways. The method used in AeroDyn [6] and the method of Hansen [5] are given below. In both methods, the term $\dot{\boldsymbol{W}}_{\text{qs}}$ is approximated using finite differences of the values of the induced velocities at t and $t + dt$. For convenience, the RHS of 44 is written $\boldsymbol{H}(t)$:

$$\boldsymbol{H}(t) \triangleq \boldsymbol{W}_{\text{qs}} + k\tau_1 \frac{\mathrm{d}\boldsymbol{W}_{\text{qs}}}{\mathrm{d}t} \tag{48}$$

In the method of Hansen, this term is assumed to be constant between two time step, while a linear variation is assumed in the AeroDyn implementation.

AeroDyn implementation The following approach is followed in AeroDyn (see [6]). First, the term $\boldsymbol{W}_{\text{qs}}$ is assumed to vary linearly between t and $t+dt$, based on the estimated derivative, then, the derivative $\dot{\boldsymbol{W}}_{\text{qs}}$ is evaluated using backward differences as follows:

$$\boldsymbol{H}(t') \approx \boldsymbol{W}^i_{\text{qs}} + t' \frac{\mathrm{d}\boldsymbol{W}_{\text{qs}}}{\mathrm{d}t} + k\tau_1 \frac{\mathrm{d}\boldsymbol{W}_{\text{qs}}}{\mathrm{d}t} \approx \boldsymbol{A} + \boldsymbol{B}t', \qquad t' \in [0, \Delta t] \tag{49}$$

$$\text{with} \qquad \boldsymbol{A} = \boldsymbol{W}^i_{\text{qs}} + \boldsymbol{B}k\tau_1, \qquad \boldsymbol{B} = \frac{\boldsymbol{W}^{i+1}_{\text{qs}} - \boldsymbol{W}^i_{\text{qs}}}{\Delta t} \tag{50}$$

Equation 43 is then integrated using Equation 50 as RHS, assuming τ_1 constant in the interval, and using the general integration formula from Equation 96, to give:

$$\boldsymbol{W}_{\text{int}}(t') = \boldsymbol{A} + \boldsymbol{B}(t' - \tau_1) + \boldsymbol{C}_0 e^{t'/\tau_1}, \qquad \text{with} \quad \boldsymbol{C}_0 = \boldsymbol{W}^i_{\text{qs}} - \boldsymbol{A} + \boldsymbol{B}\tau_1 \tag{51}$$

The term $\boldsymbol{W}_{\text{int}}(t')$ is then used in the RHS of Equation 44, which is integrated using Equation 96, assuming τ_2 constant in the interval, giving:

$$\boldsymbol{W}(t') = \boldsymbol{C}_{0,2} e^{-t'/\tau_2} + \boldsymbol{A} + \boldsymbol{B}(t' - \tau_1 - \tau_2) + \frac{\boldsymbol{C}_0}{1 - k_\tau} e^{-t'/\tau_1} \tag{52}$$

$$\text{with} \quad \boldsymbol{C}_{0,2} = \boldsymbol{W}^i_{\text{int}} - \boldsymbol{A} + \boldsymbol{B}(\tau_1 + \tau_2) - \frac{C_0}{1 - k_\tau}$$

Equation 53 is expressed at $t' = \Delta t$ to obtain $\boldsymbol{W}^{i+1}$.

Method of Hansen The numerical resolution of Equations 43-44 is presented as follows by Hansen. The term involving $\dot{W}_{\text{qs}}$ is evaluated using backward differences, and the $\boldsymbol{W}_{\text{qs}}$ is assumed to be constant in the interval (taking its final value at $t + dt$):

$$\boldsymbol{H} \approx \boldsymbol{W}^{i+1}_{\text{qs}} + k\tau_1(\boldsymbol{W}^{i+1}_{\text{qs}} - \boldsymbol{W}^i_{\text{qs}})/\Delta t \tag{53}$$

where the upper script i and $i + 1$ represent two successive times separated by Δt. The term $\boldsymbol{H}$ is assumed constant, so that Equation 43 and Equation 44 can be successively integrated using Equation 96, leading:

$$\boldsymbol{W}^{i+1}_{\text{int}} = \boldsymbol{H} + (\boldsymbol{W}^i_{\text{int}} - \boldsymbol{H}) e^{-\frac{\Delta t}{\tau_1}}, \qquad \boldsymbol{W}^{i+1} = \boldsymbol{W}^{i+1}_{\text{int}} + (\boldsymbol{W}^i - \boldsymbol{W}^{i+1}_{\text{int}}) e^{-\frac{\Delta t}{\tau_2}} \tag{54}$$

B.3 Implementation 2: State-space formulation

To allow a full state space formulation and ease the linearization, the following approximations are made:

- $\dot{\tau}_2 = 0$: this approximation seem justified since the models were likely tuned at constant operating conditions
- $\dot{\boldsymbol{W}}_{\text{qs}}$ can be approximated using the structural acceleration of the blade (see Equation 41) and may be further improved by including the contributions from the BEM algorithm.

The state space formulation is then:

$$\begin{bmatrix} \dot{\boldsymbol{W}} \\ \ddot{\boldsymbol{W}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_2 \\ -\frac{1}{\tau_1\tau_2}\boldsymbol{I}_2 & -\frac{1}{\tau_1\tau_2}(\tau_1+\tau_2)\boldsymbol{I}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{W} \\ \dot{\boldsymbol{W}} \end{bmatrix} + \frac{1}{\tau_1\tau_2} \begin{bmatrix} 0 & 0 \\ 1 & k\tau_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_{\text{qs}} \\ \dot{\boldsymbol{W}}_{\text{qs}} \end{bmatrix} \tag{55}$$

C Polar functions

C.1 Basic lift parameters

Two important parameters characterize the lift coefficient: the angle of attach at zero lift noted α_0 and the lift slope about this angle noted $C_{l,\alpha}$:

$$C_{l,\alpha} \triangleq \frac{\mathrm{d}C_l}{\mathrm{d}\alpha}(\alpha_0) \tag{56}$$

Most airfoils have a linear lift coefficient region around $[\alpha_0 \, ; \alpha_0 + 5°]$. The lift slope is found as $C_{l,\alpha} = 2\pi$ for an inviscid flat plate. Real airfoils can have slopes value quite different than 2π. The point where the lift coefficient is maximum is also of relevance. After this point, the lift-coefficient drops, a phenomenon referred to as *stall*. Drastic changes of loads may occur depending on the stall behavior of the airfoil. It is usually desired to design an airfoil such that the maximum lift is high but the stall behavior is not abrupt.

C.2 Separation function

The separation function f_s usually represents the fraction of the lift coefficient C_l distributed between the inviscid lift coefficient $C_{l,\text{inv}}$ and the fully separated lift coefficient $C_{l,\text{fs}}$:

$$C_l(\alpha) = f_s \, C_{l,\text{inv}}(\alpha) + (1 - f_s) \, C_{l,\text{fs}}(\alpha) \tag{57}$$

The function f_s is either prescribed (Method 1) or determined from the fully separated polar (Method 2). The superscript st is used when a function refers to the steady lift coefficient. In the case of a constant lift coefficient, $C_l(\alpha) = c$, it is assumed that the flow is fully separated and hence: $C_{l,\alpha} = 0$, $C_{l,\text{inv}} = 0$, $C_{l,fs} = c$ and $f_s = 0$.

Method 1: separation function taken as the separation point for a Kichhoff flow Lift on a flat plate in a potential Kirchhoff flow is given by:

$$C_l^{st}(\alpha) = C_{l,\alpha} \left[\frac{1 + \sqrt{f_s^{st}(\alpha)}}{2} \right]^2 (\alpha - \alpha_0) \tag{58}$$

This expression is partially inverted as follows to define the separation function:

$$\begin{aligned} \text{if } C_{l,\alpha} = 0, \quad & f_s^{st} \equiv 0 \\ \text{if } C_{l,\alpha} \neq 0 \text{ and } \alpha \in [\alpha_1; \alpha_0[\cup]\alpha_0; \alpha_2] \ , \quad & f_s^{st} = \min\left\{ \left[2\sqrt{\frac{C_l^{st}(\alpha)}{C_{l,\alpha}(\alpha - \alpha_0)}} - 1 \right]^2 , 1 \right\} \\ \text{if } C_{l,\alpha} \neq 0 \text{ and } \alpha = \alpha_0 \ , \quad & f_s^{st} = 1 \\ \text{else}, \quad & f_s^{st} = 0 \end{aligned} \tag{59}$$

Where, as mentioned by Hansen et al. [1], there are two issues with this definition, and essentially, the implementation should ensure that the function remains between 0 and 1:

- To avoid the function exceeding 1 in the linear region, the term $C_{l,\alpha}(\alpha - \alpha_0)$ needs to be higher than $C_{l,\alpha}$. To achieve this, the slope is defined as the maximum of the slope within the linear region. Then, any values of f_s^{st} above 1 in this region is capped to 1 via the "min" function.
- Outside of the linear region, the separation function progressively decays. As soon as it reaches 0 or negative values from both side of α_0, this function needs to be bounded to 0. The values where the separation function reaches 0 on both sides of α_0 are noted α_1 and α_2.

A python source code to compute f_s^{st} is given in subsection F.1.

Method 2: Definition in terms of full separated polar Assuming the steady lift coefficient, the inviscid and fully separated coefficients are known, the separation function is directly obtained as:

$$C_l^{st}(\alpha) \triangleq f_s^{st}\, C_{l,\text{inv}}(\alpha) + (1 - f_s^{st})\, C_{l,\text{fs}}(\alpha) \quad \Rightarrow \quad f_s^{st}(\alpha) \triangleq \frac{C_l^{st}(\alpha) - C_{l,\text{fs}}(\alpha)}{C_{l,\text{inv}}(\alpha) - C_{l,\text{fs}}(\alpha)} \tag{60}$$

C.3 Fully-separated polar

Method using f_s^{st} If the separation function is known, the fully separated polar is directly obtained from Equation 58:

$$C_{l,\text{fs}}(\alpha) = \frac{C_l^{st}(\alpha) - C_{l,\alpha}(\alpha - \alpha_0) f_s^{st}(\alpha)}{1 - f_s^{st}(\alpha)} \text{ when } f_s^{st} \neq 1, \qquad C_{l,\text{fs}}(\alpha) = \frac{C_l^{st}(\alpha)}{2} \text{ when } f_s^{st} = 1 \tag{61}$$

A python source code to compute $C_{l,\text{fs}}$ is given in subsection F.1.

Method from Øye The following model is used e.g. by Øye dynamic stall model [7]. More details on the implementation is provided in [8].

$$s_{\text{fs}} = \left.\frac{\partial C_l}{\partial \alpha}\right|_{\text{fs}} \qquad \left(= \frac{1}{2} C_{l,\alpha} \right) \tag{62}$$

$$\text{for } \alpha > \alpha_0, \quad C_{l,\text{fs}} = \begin{cases} s_{\text{fs}}(\alpha - \alpha_0) & \alpha < \alpha_1 - \Delta\alpha \\ s_{\text{fs}}(\alpha - \alpha_0) - s_{\text{fs}} \frac{(\alpha - \alpha_1 + \Delta\alpha)^2}{4\Delta\alpha} & \alpha < \alpha_1 + \Delta\alpha \\ C_{l_1} & \alpha < \alpha_{\text{merge}} \\ C_l & \text{otherwise} \end{cases} \tag{63}$$

An example of application of this engineering model is shown in Figure 9.

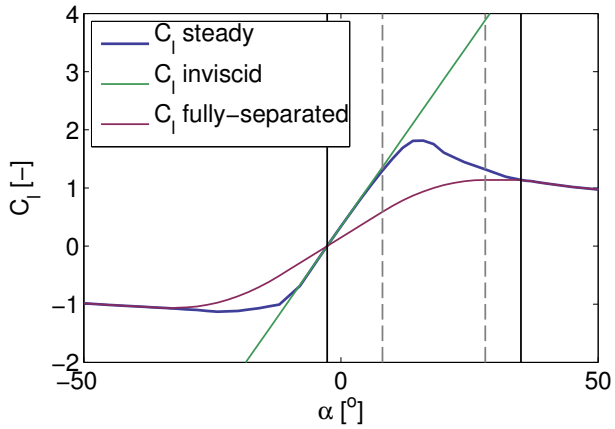

Figure 9: Illustration of the engineering model to derive a fully-separated polar from steady data. The original polar is the one of FFA-W3-241 airfoil at $Re = 12 \times 10^6$ as given by the DTU 10-MW-RWT. Parameters for the fully-separated model were $\alpha_{\text{merge}} = 35°$, $\Delta\alpha = 10°$, and $s_{\text{fs}} = \pi$. The solid vertical lines mark α_0 and α_{merge}, while the dashed lines mark $\alpha_1 \pm \Delta\alpha$. The linear region of the steady polar has a slope higher than 2π.

C.4 Inviscid polar

From the 2D viscous tabulated data, α_0 and $C_{l,\alpha}$ can be determined and the inviscid lift coefficient is obtained as:

$$C_{l,\text{inv}} = C_{l,\alpha} \sin(\alpha - \alpha_0) \approx C_{l,\alpha}(\alpha - \alpha_0) \tag{64}$$

D Øye's dynamic stall model

$$\dot{f}_s(t) = -T_f^{-1} f_s(t) + T_f^{-1} f_s^{st}(\alpha(t)) \tag{65}$$

The model of Øye [7] considers trailing edge stall, that is, when the flow separation starts at the trailing edge and gradually increases upstream for higher angles of attack. The suggested way to model this is to use an interpolation of aerodynamic data between two extreme cases for which the flow is either fully separated or entirely inviscid. The differences between these two cases and the steady data are shown in Figure 9. The lift coefficient is computed as a linear combination of

the two extremes cases where the linear parameter is function of time to account for the adaptation of the flow to a configuration or another. This is written as:

$$C_l(\alpha, t) = f_s(t)\, C_{l,\text{inv}}(\alpha) + (1 - f_s(t))\, C_{l,\text{fs}}(\alpha) \tag{66}$$

where α is the instantaneous angle of attack, f_s is called the separation function which acts as a relaxation factor, and $C_{l,\text{inv}}$ and $C_{l,\text{fs}}$ are the lift coefficients for inviscid flow without separation and for fully separated flow respectively. The inviscid lift coefficient is given by Equation 65 and the fully separated coefficient by Equation 64. The separation function is modelled as a first order model which tends to the equilibrium (static) value f_s^{fs}:

$$f_s(t + \Delta t) = f_s^{\text{st}}(\alpha(t)) + \left(f_s(t) - f_s^{\text{st}}(\alpha(t))\right) e^{-\frac{\Delta t}{\tau}} \tag{67}$$

The equilibrium or static value is obtained from the inviscid and fully separated lift curve as:

$$f_s^{\text{st}} = \frac{C_l - C_{l,\text{fs}}}{C_{l,\text{inv}} - C_{l,\text{fs}}} \tag{68}$$

The application of Equation 67 for $f_s = f_s^{\text{st}}$ gives then the steady curve.The time constant of the flow adaptation, τ, is assumed to depend on the chord and the relative wind speed:

$$\tau = \frac{k_\tau c}{V_{\text{rel}}} \tag{69}$$

where usual values of k_τ are 3, 4 or 10 [5]. For $k = 1$, the time constant provides a representative time for the flow to go from the leading edge to the trailing edge, hence "replacing" the boundary layer.

E HGM Beddoes dynamic stall model

The model developed by Hansen et al. is described in details in [1]. The notations from this document are adopted and the key results are mentioned in this section. The model will be further referred to as the HGM-Beddoes dynamics stall model, or HGM model, similar to the appellation used in HAWC2 [9].

The HGM model has been used and extended by Bergami et al. to model trailing-edge flaps [10].

E.1 Overview

An overview of the HGM model is provided in this paragraph. Some notations are introduced to familiarize the reader with the nomenclature but their definitions will be provided in more details in the subsequent paragraphs. The HGM model uses 4 states variables $x_1..x_4$ to model the unsteady aerodynamics of an airfoil. The unsteadiness can be attributed to the motion of the airfoil or changes in inflow conditions. The two first states x_1 and x_2 have the dimension of an angle of attack and they capture the time history of the wake vorticity. Any change of the airfoil circulation indeed results in vorticity being shed in the wake and propagated downstream. In turn, the vorticity in the wake affects the flow around the airfoil. The time scale characteristic of the circulation propagation is noted T_u. The values of x_1 and x_2 contribute to an "effective" angle of attack α_E, which would be obtained if the flow around the airfoil was fully attached and reacted

instantaneously to the changes in the wake vorticity. The lift coefficient that would be obtained under these assumptions is noted C_L^p. The state variable x_3 has the dimension of a lift coefficient and it captures the fact that the pressure around the airfoil needs time to adapt to a change in effective angle of attack and it is thus lagging compared to the fully-attached lift value. The time constant for this time-lag is T_p, where the subscript p stands for pressure time-lag. The angle of attack that would correspond to the quasi-steady lift coefficient x_3 is noted α_F. The variable x_4 captures the dynamics of the boundary layer and represents a dimensionless separation point, a quantity noted with the variable f. Two "steady-state-stall" configurations of the airfoil will have two separation point locations. The time scale for the boundary layer to change its separation point between two stall configurations is noted T_f. The state variable x_4 represents the evolution of the separation from the current state of the boundary layer to the quasi-steady value defined by $f^{st}(\alpha_F)$.

The model uses six constants to be provided by the user for each airfoil: four constants A_1, A_2, b_1, b_2 characteristic of the propagation of the wake vorticity, the pressure time constant T_p and the boundary layer time constant T_f.

E.2 State equations

The state equations of the HGM model are given in equations 71-74 with the definitions provided in equations 75-78. They are linear first order ordinary differential equations. Equations 71-72 are uncoupled and the coefficients of x_i are time dependent via the velocity $U(t)$ (and $T_u \triangleq c/2U$). Equation 73 is coupled to eqs. 71-72 via C_L^p, and equation 74 is coupled to eq. 73 via α_F. It is readily seen that the steady state values are $x_1 \to A_1\alpha_{3/4}$, $x_2 \to A_2\alpha_{3/4}$, $x_3 \to C_L^p$ and $x_4 \to f^{st}(\alpha_F)$ and they are reached respectively with the time constants T_u/b_i, T_p and T_f.

$$\dot{x}_1 = -T_u^{-1}\left[b_1 + \frac{c\dot{U}}{2U^2}\right]x_1 + T_u^{-1}b_1A_1\alpha_{3/4} \tag{70}$$

$$\dot{x}_2 = -T_u^{-1}\left[b_2 + \frac{c\dot{U}}{2U^2}\right]x_2 + T_u^{-1}b_2A_2\alpha_{3/4} \tag{71}$$

$$\dot{x}_3 = -T_p^{-1}x_3 + T_p^{-1}C_L^p \tag{72}$$

$$\dot{x}_4 = -T_f^{-1}x_4 + T_f^{-1}f^{st}(\alpha_F) \tag{73}$$

with:

$$T_u(t) \triangleq \frac{c}{2U(t)} \tag{74}$$

$$\alpha_E(t) \triangleq \alpha_{3/4}(t)(1 - A_1 - A_2) + x_1(t) + x_2(t) \tag{75}$$

$$C_L^p(t) \triangleq C_{l,\alpha}\ (\alpha_E(t) - \alpha_0) + \pi T_u(t)\omega(t) \tag{76}$$

$$\alpha_F(t) \triangleq \frac{x_3(t)}{C_{l,\alpha}} + \alpha_0 \tag{77}$$

and where:

- c, $C_{l,\alpha}$, α_0 are constant airfoil parameters: chord, lift slope and angle of attack of zero lift

- $f^{st}(\alpha)$ is the separation function, determined from the lift curve $C_L(\alpha)$ and which definition is given in subsection C.2
- A_1, A_2, b_1, b_2 are four constants, parameters of the HGM model, characteristic of the propagation of the wake vorticity for this airfoil
- T_f, T_p are two constants, parameters of the HGM model, characteristic of the propagation of the wake vorticity for this airfoil
- $\alpha(t)$, $\alpha_{3/4}(t)$ and $U(t)$ (with ω and $\dot{U}$) are time-varying inputs of the HGM model, they correspond respectively to: the angle of attack, the angle of attack at the 3/4-chord location and the relative airfoil speed. These values can be derived from the outputs of the structural model, the inflow model and aerodyn.

The source code for the state derivatives is given in Appendix F.

E.3 Outputs

The unsteady airfoil loads are derived from the states as follows (see [1]):

$$C_{l,\text{dyn}}(t) = x_4(\alpha_E - \alpha_0)C_{l,\alpha} + (1 - x_4)C_{l,fs}(\alpha_E) + \pi T_u \omega \tag{78}$$

$$C_{d,\text{dyn}}(t) = C_d(\alpha_E) + (\alpha - \alpha_E)C_{l,\text{dyn}} + [C_d(\alpha_E) - C_d(\alpha_0)]\,\Delta C''_{d,f} \tag{79}$$

$$C_{m,\text{dyn}}(t) = C_m(\alpha_E) + C_{l,\text{dyn}}\Delta C''_{m,f} - \frac{\pi}{2}T_u\omega \tag{80}$$

with

$$\Delta C''_{d,f} = \frac{\sqrt{f^{st}(\alpha_E)} - \sqrt{x_4}}{2} - \frac{f^{st}(\alpha_E) - x_4}{4}, \qquad \Delta C''_{m,f} = 0 \tag{81}$$

The source code for the output calculation is given in Appendix F.

E.4 Vectorial notations

The state and output equations are written in a vectorial form as:

$$\dot{\boldsymbol{x}} = \boldsymbol{X}(\boldsymbol{x}, \boldsymbol{u}, t), \qquad \boldsymbol{y} = \boldsymbol{Y}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{82}$$

with:

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4], \qquad \boldsymbol{u} = [\alpha, \omega, \alpha_{3/4}, U, \dot{U}] \tag{83}$$

E.5 Jacobian

The linearization of the states and output equations are provided in [1] and repeated in the paragraph "Block 0" of subsection 2.7.

F Sample source code for the HGM model

F.1 Separation function and fully separated polar

Source code to compute f_s^{st} and $C_{l,\text{fs}}$. Taken from an airfoil library by the author `https://github.com/ebranlard/welib/blob/master/welib/airfoils/Polar.py`. The linear slope $C_{l,\alpha}$ is be obtained as the maximum slope within the linear region.

```
def cl_fully_separated(self):
    alpha0 = self.alpha0()
    cla,_, = self.cl_linear_slope(method='max')
    if cla==0:
        cl_fs    = self.cl # when f_st ==1
        f_st     = self.cl*0
    else:
        cl_ratio = self.cl / ( cla*(self.alpha-alpha0))
        cl_ratio[ np.where(cl_ratio<0)]=0
        f_st = ( 2 *np.sqrt(cl_ratio)-1)**2
        f_st[np.where(f_st<1e-15)] = 0
        # Initialize to linear region (in fact only at singularity, where f_st=1)
        cl_fs    = self.cl/2.0 # when f_st ==1
        # Region where f_st<1, merge
        I=np.where(f_st<1)
        cl_fs[I] =(self.cl[I] - cla* (self.alpha[I]-alpha0)*f_st[I])/(1.-f_st[I]) # when f_st<1, otherwise
        # Outside region
        iHig=np.ma.argmin( np.ma.MaskedArray(f_st,self.alpha<alpha0) );
        iLow=np.ma.argmin( np.ma.MaskedArray(f_st,self.alpha>alpha0) );
        cl_fs[0:iLow+1]  = self.cl[0:iLow+1]
        cl_fs[iHig+1:-1] = self.cl[iHig+1:-1]
        #print(iLow,iHig)

    # Ensuring everything is in harmony
    cl_inv = cla*(self.alpha - alpha0)
    f_st=(self.cl-cl_fs)/(cl_inv-cl_fs);
    f_st[np.where(f_st<1e-15)] = 0
    # Storing
    self.f_st  = f_st
    self.cl_fs = cl_fs
    return cl_fs,f_st
```

F.2 State equations

Calculation of state derivatives:

```
    Tu     = c/(2*U)                                          # Eq. 23
    # Variables derived from states
    alphaE  = alpha_34*(1-A1-A2)+ x1 + x2                     # Eq. 12
    Clp     = Cla * (alphaE-alpha0) + np.pi * Tu * alpha_dot  # Eq. 13
    alphaF  = x3/Cla+alpha0                                   # p. 13
    fs_aF   = F_st(alphaF)                                     # p. 13

    # State equation
    xdot = [0]*4
    xdot[0] = -1/Tu * (b1 + c * U_dot/(2*U**2)) * x1 + b1 * A1 / Tu * alpha_34
    xdot[1] = -1/Tu * (b2 + c * U_dot/(2*U**2)) * x2 + b2 * A2 / Tu * alpha_34
    xdot[2] = -1/Tp                             * x3 + 1/Tp * Clp
    xdot[3] = -1/Tf                             * x4 + 1/Tf * fs_aF
```

F.3 Outputs

Calculation of outputs from states:

```
    alphaE = alpha_34*(1-A1-A2)+ x1 + x2                  # Eq. 12
    faE = F_st(alphaE)
    DeltaCdfpp = (np.sqrt(faE)-np.sqrt(x4))/2 - (faE-x4)/4
    #ast_x4  = (fCm(x4)  - fCm(alpha0))/Cl(x4)
    #ast_faE = (fCm(faE) - fCm(alpha0))/Cl(faE)
    #DeltaCmfpp = (fa_st(x4) - fa_st(faE))
```

```
    DeltaCmfpp = 0
    # Outputs
    Cl_dyn =  Cla * (alphaE-alpha0)*x4 + Cl_fs(alphaE)*(1-x4) + np.pi*Tu*alpha_dot
    Cd_dyn =  Cd(alphaE) + (alpha-alphaE)*Cl_dyn + (Cd(alphaE)-Cd(alpha0))*DeltaCdfpp
    Cm_dyn =  Cm(alphaE) + Cl_dyn*DeltaCmfpp - np.pi/2*Tu*alpha_dot
```

G Investigations on the CUA model

G.1 Neglecting the acceleration term $\dot{U}$

The acceleration term $\dot{U}$ appears in the original HGM-model in Equation 71 and Equation 72 as: $\left[b_i + \frac{c\dot{U}}{2U^2}\right]$, with $i = 1, 2$. One way to evaluate the important of the acceleration term is thus to compare the term $b_{\dot{U}} = \frac{c\dot{U}}{2U^2}$ with the value of b_i, that are typically $b_1 \approx 0.05$ and $b_2 \approx 0.3$.

Using several 10min simulations of the NREL5MW turbine at 7, 13, and 23m/s under normal operating conditions (with turbulence) and using the radial positions at 17, 43, 70, 98% it was observed that the term $b_{\dot{U}}$ did not exceed 0.0013, that is less than 3% of b_1. This example case tend to show that the term $\dot{U}$ can be omitted in the model. The maximum value is in general obtained at the root of the blade, where the velocity U is lower and the chord bigger. Neglecting the term $\dot{U}$ is thus likely to have more impact at the inboard part.

The inflow time-series of the above-mentioned simulation were used as inputs to the HGM dynamic stall model to determine the dynamic aerodynamic coefficients as a post-processing step, with or without the term $\dot{U}$. The mean relative error of the lift, drag and moment coefficients along the span of the blade were below 3.5%, 6% and 2.5% respectively.

H General considerations for the linearization of the chord loads of an airfoil

Notations are given in Figure 1. The relative wind speed and angle of attack are:

$$\alpha = \operatorname{atan2} \frac{v_x}{v_y}, \qquad U = \sqrt{v_x^2 + v_y^2} \tag{84}$$

leading to the following Jacobians with respect to $\boldsymbol{v} = [v_x, v_y]^t$

$$\frac{\partial U}{\partial \boldsymbol{v}} = \begin{bmatrix} \frac{v_x}{U} & \frac{v_y}{U} \end{bmatrix}, \qquad \frac{\partial \alpha}{\partial \boldsymbol{v}} = \begin{bmatrix} \frac{v_y}{U^2} & \frac{-v_x}{U^2} \end{bmatrix} \tag{85}$$

The aerodynamic lift, drag and moment and given by:

$$L = \frac{1}{2}\rho U^2 c C_{l,\mathrm{dyn}}, \qquad D = \frac{1}{2}\rho U^2 c C_{d,\mathrm{dyn}}, \qquad M = \frac{1}{2}\rho U^2 c^2 C_{m,\mathrm{dyn}} \tag{86}$$

They are projected onto the airfoil coordinate system as:

$$\begin{bmatrix} f_x \\ f_y \\ m_z \end{bmatrix} = \frac{1}{2}\rho c U^2 \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} C_{l,\mathrm{dyn}} \\ C_{d,\mathrm{dyn}} \\ C_{m,\mathrm{dyn}} \end{bmatrix} \quad \Leftrightarrow \quad \boldsymbol{f} = \boldsymbol{M}\boldsymbol{C} \tag{87}$$

The aerodynamic coefficients $\boldsymbol{C}$, the transformation matrix $\boldsymbol{M}$ and the outputs loads are linearized as follows:

$$\boldsymbol{M} = \boldsymbol{M}_{\text{op}} + \left.\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{q}}\right|_{\text{op}} \delta\boldsymbol{q}, \qquad \boldsymbol{C} = \boldsymbol{C}_{\text{op}} + \left.\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}}\right|_{\text{op}} \delta\boldsymbol{q}, \qquad \boldsymbol{f} = \boldsymbol{f}_{\text{op}} + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\text{op}} \delta\boldsymbol{q} \tag{88}$$

where $\boldsymbol{q}$ represent the different degrees of freedom (states, inputs). Using $\boldsymbol{f} = \boldsymbol{M}\boldsymbol{C}$, $\boldsymbol{f}_{\text{op}} = \boldsymbol{M}_{\text{op}}\boldsymbol{C}_{\text{op}}$ and keeping the first order terms, the Jacobian of the output loads is identified as:

$$\left.\frac{\partial \boldsymbol{f}}{\partial q_i}\right|_{\text{op}} = \left.\frac{\partial \boldsymbol{M}}{\partial q_i}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} + \boldsymbol{M}_{\text{op}} \left.\frac{\partial \boldsymbol{C}}{\partial q_i}\right|_{\text{op}} \tag{89}$$

The first term can be determined independently of the dynamic stall model, and its expression is made explicit in the next paragraph. The expression for the second term is a function of the dynamic stall model.

First term ($\partial(MC_{\text{op}})$) Using $v_x = U\cos\alpha$ and $v_y = U\sin\alpha$, the product $\boldsymbol{M}C_{\text{op}}$ is

$$\boldsymbol{M}\boldsymbol{C}_{\text{op}} = \frac{1}{2}\rho c \begin{bmatrix} Uv_xC_{l,\text{op}} + Uv_yC_{d,\text{op}} \\ -Uv_yC_{l,\text{op}} + Uv_xC_{d,\text{op}} \\ cU^2C_{m,\text{op}} \end{bmatrix} \tag{90}$$

Using Equation 86, the Jacobians are obtained as: Since $\boldsymbol{M}$ only depends on v_x and v_y:

$$\left.\frac{\partial \boldsymbol{M}}{\partial v_x}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} = \frac{1}{2}\rho c \begin{bmatrix} \frac{v_x}{U}\left(v_xC_{l,\text{op}} + Uv_yC_{d,\text{op}}\right) + UC_{l,\text{op}} \\ \frac{v_x}{U}\left(-v_yC_{l,\text{op}} + Uv_xC_{d,\text{op}}\right) + UC_{d,\text{op}} \\ 2v_xcC_{m,\text{op}} \end{bmatrix} \tag{91}$$

$$\left.\frac{\partial \boldsymbol{M}}{\partial v_y}\right|_{\text{op}} \boldsymbol{C}_{\text{op}} = \frac{1}{2}\rho c \begin{bmatrix} \frac{v_y}{U}\left(v_xC_{l,\text{op}} + Uv_yC_{d,\text{op}}\right) + UC_{d,\text{op}} \\ \frac{v_y}{U}\left(-v_yC_{l,\text{op}} + Uv_xC_{d,\text{op}}\right) - UC_{l,\text{op}} \\ 2v_ycC_{m,\text{op}} \end{bmatrix} \tag{92}$$

I Solution of a first order linear differential equation

General equation and its solution A first order linear differential equation is written in its general form as:

$$\dot{x} + a(t)x(t) = b(t) \tag{93}$$

Solving first for the homogeneous solution and then using the method of variation of the constant leads to the following general solution, where t_0 is the initial time:

$$x(t) = e^{-\int_{t_0}^{t} a(t')dt'} \left[x(t_0) + \int_{t_0}^{t} b(t')\, e^{\int_{t_0}^{t'} a(t'')dt''} dt'\right] \tag{94}$$

Particular cases

- If a is constant:

$$x(t) = e^{-a(t-t_0)} \left[x(t_0) + \int_{t_0}^{t} b(t')\, e^{a(t'-t_0)} dt'\right] \tag{95}$$

- If b is also constant:

$$x(t) = e^{-a(t-t_0)} \left[x(t_0) + \frac{b}{a} \left[e^{a(t-t_0)} - 1 \right] \right] = \frac{b}{a} + \left[x(t_0) - \frac{b}{a} \right] e^{-a(t-t_0)} \tag{96}$$

- If a and b are constants, integrating between t and $t + \Delta t$:

$$x(t + \Delta t) = \frac{b}{a} + \left[x(t) - \frac{b}{a} \right] e^{-a\Delta t} \tag{97}$$

Numerical integration

- If a and b are constants, integrating between t and $t + \Delta t$:

$$x(t + \Delta t) = \frac{b}{a} + \left[x(t) - \frac{b}{a} \right] e^{-a\Delta t} \tag{98}$$

  This form is for instance recommended by Hansen [5] in the implementation of Oye's dynamic stall model described in Appendix D.

References

[1] M.H. Hansen, Mac Gaunaa, and Helge Aagaard Madsen. A beddoes-leishman type dynamic stall model in state-space and indicial formulations. Technical report, Risø National Laboratory, Roskilde, Denmark, 2004.

[2] T. S. Beddoes. A near wake dynamic model. *proc. of the AHS national specialist meeting on aerodynamics and aeroacoustics*, 1987.

[3] J. Gordon Leishman. Assessment of aerodyn theory basis including unsteady aerodynamics modules. Technical report, Rotor Systems Research LLC, 2011.

[4] H. Snel and J. G. Schepers. Joint investigation of dynamic inflow effects and implementation of an engineering method. Technical report, ECN-C–94-107, Energy Research Centre of the Netherlands, Petten, 1995.

[5] M. O. L. Hansen. *Aerodynamics of Wind Turbines - Second Edition.* Earthscan, London, Sterling, VA, 2008.

[6] R. Damiani and J. Jonkman. DBEMT theory manual, rev.3. Technical report, National Renewable Energy Laboratory, 2017. Unpublished NREL report.

[7] S. Øye. Dynamic stall, simulated as a time lag of separation. *Proceedings of the 4th IEA Symposium on the Aerodynamics of Wind Turbines*, 1991.

[8] E. Branlard. *Wind Turbine Aerodynamics and Vorticity-Based Methods.* Springer, 2017.

[9] T. J. Larsen and A. M. Hansen. *HAWC2 - User manual.* DTU-Risø-R-1597, 2007.

[10] Leonardo Bergami and Mac Gaunaa. Ateflap aerodynamic model, a dynamic stall model including the effects of trailing edge flap deflection. Technical report, DTU - Risø National Laboratory, 2012.