



# CCBlade Documentation

## *Release 0.2.0*

S. Andrew Ning

**NREL is a national laboratory of the U.S. Department of Energy  
Office of Energy Efficiency & Renewable Energy  
Operated by the Alliance for Sustainable Energy, LLC**

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at [www.nrel.gov/publications](http://www.nrel.gov/publications).

**Technical Report**  
NREL/TP-5000-58819  
June 2013

Contract No. DE-AC36-08GO28308



NATIONAL RENEWABLE ENERGY LABORATORY

# CCBlade Documentation

## *Release 0.2.0*

S. Andrew Ning

Prepared under Task No(s). WE11.0341

**NREL is a national laboratory of the U.S. Department of Energy  
Office of Energy Efficiency & Renewable Energy  
Operated by the Alliance for Sustainable Energy, LLC**

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at [www.nrel.gov/publications](http://www.nrel.gov/publications).

National Renewable Energy Laboratory  
15013 Denver West Parkway  
Golden, CO 80401  
303-275-3000 • [www.nrel.gov](http://www.nrel.gov)

**Technical Report**  
NREL/TP-5000-58819  
June 2013

Contract No. DE-AC36-08GO28308

## NOTICE

This report was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or any agency thereof.

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at [www.nrel.gov/publications](http://www.nrel.gov/publications).

Available electronically at <http://www.osti.gov/bridge>

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831-0062  
phone: 865.576.8401  
fax: 865.576.5728  
email: <mailto:reports@adonis.osti.gov>

Available for sale to the public, in paper, from:

U.S. Department of Commerce  
National Technical Information Service  
5285 Port Royal Road  
Springfield, VA 22161  
phone: 800.553.6847  
fax: 703.605.6900  
email: [orders@ntis.fedworld.gov](mailto:orders@ntis.fedworld.gov)  
online ordering: <http://www.ntis.gov/help/ordermethods.aspx>

*Cover Photos: (left to right) photo by Pat Corkery, NREL 16416, photo from SunEdison, NREL 17423, photo by Pat Corkery, NREL 16560, photo by Dennis Schroeder, NREL 17613, photo by Dean Armstrong, NREL 17436, photo by Pat Corkery, NREL 17721.*



Printed on paper containing at least 50% wastepaper, including 10% post consumer waste.

## Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>2</b>
<b>3</b>	<b>Tutorial</b>	<b>3</b>
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
4.1	Airfoil Interface	7
4.2	CCAirfoil Class	7
4.3	CCBlade Class	9
<b>5</b>	<b>Theory</b>	<b>13</b>
	<b>Coordinate System</b>	<b>15</b>
	<b>Bibliography</b>	<b>22</b>

## List of Figures

Figure 1.	Flapwise and lead-lag aerodynamic loads along blade. . . . .	5
Figure 2.	Power coefficient as a function of tip-speed ratio. . . . .	6
Figure 3.	Parameters specifying inflow conditions of a rotating blade section. . . . .	13
Figure 4.	Residual function of BEM equations using new methodology. Solution point is where $f(\phi) = 0$ . . . . .	14
Figure 5.	Inertial and Wind-aligned axes. . . . .	15
Figure 6.	Wind-aligned and yaw-aligned axes. $\Psi$ is the rotor yaw angle. . . . .	16
Figure 7.	Yaw-aligned and hub-aligned axes. $\Theta$ is the rotor tilt angle. . . . .	17
Figure 8.	Hub-aligned and azimuth-aligned axes. $\Lambda$ is the (local) blade azimuth angle. . . . .	18
Figure 9.	Azimuth-aligned and blade-aligned axes. $\Phi$ is the (local) blade precone angle. . . . .	19
Figure 10.	Blade-aligned and airfoil-aligned coordinate systems. $\theta$ is the airfoil twist + pitch angle. For convenience the local wind vector and angle of attack is shown. . . . .	20
Figure 11.	Airfoil-aligned and profile coordinate systems. . . . .	20

## List of Tables

Table 1.	Degree of spline across Reynolds number. . . . .	8
Table 2.	Inertial-Wind conversion methods . . . . .	16
Table 3.	Wind-Yaw conversion methods . . . . .	17
Table 4.	Yaw-Hub conversion methods . . . . .	17
Table 5.	Hub-Azimuth conversion methods . . . . .	18
Table 6.	Azimuth-Blade conversion methods . . . . .	19
Table 7.	Blade-Airfoil conversion methods . . . . .	20
Table 8.	Airfoil-Profile conversion methods . . . . .	21

# 1 Introduction

CCBlade predicts aerodynamic loading of wind turbine blades using blade element momentum (BEM) theory. CC stands for continuity and convergence. CCBleade was developed primarily for use in gradient-based optimization applications where  $C^1$  continuity and robust convergence are essential.

Typical BEM implementations use iterative solution methods to converge the induction factors (e.g., fixed-point iteration or Newton's method). Some more complex implementations use numerical optimization to minimize the error in the induction factors. These methods can be fairly robust, but all have at least some regions where the algorithm fails to converge. A new methodology was developed that is provably convergent in every instance (see *Theory*). This robustness is particularly important for gradient-based optimization. To ensure  $C^1$  continuity, lift and drag coefficients are computed using a bivariate cubic spline across angle of attack and Reynolds number.

CCBlade is primarily written in Python, but iteration-heavy sections are written in Fortran in order to improve performance. The Fortran code is called from Python as an extension module using f2py. The module AirfoilPrep.py is also included with the source. Although not directly used by CCBleade, the airfoil preprocessing capabilities are often useful for this application. This is the stand-alone version of CCBleade. A version exists that is packaged with NREL\_WISDEM and allows for the computation of power-regulated performance (e.g., power curves, annual energy production) for any arbitrary aerodynamics code.

## 2 Installation

---

### Prerequisites

C compiler, Fortran compiler, NumPy, SciPy

---

Download either CCBlade.py-0.2.0.tar.gz or CCBlade.py-0.2.0.zip and uncompress/unpack it.

Install CCBlade with the following command.

```
$ python setup.py install
```

To check if installation was successful run the unit tests for the NREL 5-MW model

```
$ python test/test_ccblade.py
```

An “OK” signifies that all the tests passed.

To access an HTML version of this documentation that contains further details and links to the source code, open docs/index.html.

---

**Note:** The CCBlade installation also installs the module *AirfoilPrep.py*. Although it is not necessary to use Airfoil-Prep.py with CCBlade, its inclusion is convenient when working with AeroDyn input files or doing any aerodynamic preprocessing of airfoil data.

---

## 3 Tutorial

One example of a CCBlade application is the simulation of the NREL 5-MW reference model's aerodynamic performance. First, define the geometry and atmospheric properties.

```
import numpy as np
from math import pi
import matplotlib.pyplot as plt

from ccblade_sa import CCAirfoil, CCBlade


# geometry
Rhub = 1.5
Rtip = 63.0

r = np.array([2.8667, 5.6000, 8.3333, 11.7500, 15.8500, 19.9500, 24.0500,
              28.1500, 32.2500, 36.3500, 40.4500, 44.5500, 48.6500, 52.7500,
              56.1667, 58.9000, 61.6333])
chord = np.array([3.542, 3.854, 4.167, 4.557, 4.652, 4.458, 4.249, 4.007, 3.748,
                  3.502, 3.256, 3.010, 2.764, 2.518, 2.313, 2.086, 1.419])
theta = np.array([13.308, 13.308, 13.308, 13.308, 11.480, 10.162, 9.011, 7.795,
                  6.544, 5.361, 4.188, 3.125, 2.319, 1.526, 0.863, 0.370, 0.106])
B = 3 # number of blades

tilt = 5.0
precone = 2.5
yaw = 0.0

nSector = 8 # azimuthal discretization

# atmosphere
rho = 1.225
mu = 1.81206e-5

# power-law wind shear profile
shearExp = 0.2
hubHt = 90.0
```

Airfoil aerodynamic data is specified using the `CCAirfoil` class. Rather than use the default constructor, this example uses the special constructor designed to read AeroDyn files directly `CCAirfoil.initFromAerodynFile()`.

```
afinit = CCAirfoil.initFromAerodynFile # just for shorthand

# load all airfoils
airfoil_types = [0]*8
airfoil_types[0] = afinit('Cylinder1.dat')
airfoil_types[1] = afinit('Cylinder2.dat')
airfoil_types[2] = afinit('DU40_A17.dat')
airfoil_types[3] = afinit('DU35_A17.dat')
airfoil_types[4] = afinit('DU30_A17.dat')
airfoil_types[5] = afinit('DU25_A17.dat')
airfoil_types[6] = afinit('DU21_A17.dat')
airfoil_types[7] = afinit('NACA64_A17.dat')
```

```

# place at appropriate radial stations
af_idx = [0, 0, 1, 2, 3, 3, 4, 5, 5, 6, 6, 7, 7, 7, 7, 7, 7]

af = [0]*len(r)
for i in range(len(r)):
    af[i] = airfoil_types[af_idx[i]]

```

Next, construct the CCBlade object.

```

# create CCBlade object
rotor = CCBlade(r, chord, theta, af, Rhub, Rtip, B, rho, mu,
                 precone, tilt, yaw, shearExp, hubHt, nSector)

```

Evaluate the distributed loads at a chosen set of operating conditions.

```

# set conditions
Uinf = 10.0
tsr = 7.55
pitch = 0.0
Omega = Uinf*tsr/Rtip * 30.0/pi # convert to RPM
azimuth = 0.0

# evaluate distributed loads
r, theta, Tp, Np = rotor.distributedAeroLoads(Uinf, Omega, pitch, azimuth)

```

Plot the flapwise and lead-lag aerodynamic loading

```

# plot
rstar = (r - Rhub) / (Rtip - Rhub)
plt.plot(rstar, Tp/1e3, label='lead-lag')
plt.plot(rstar, Np/1e3, label='flapwise')
plt.xlabel('blade fraction')
plt.ylabel('distributed aerodynamic loads (kN)')
plt.legend(loc='upper left')
plt.grid()
plt.show()

```

as shown in Figure 1.

To get the power, thrust, and torque at the same conditions (in both absolute and coefficient form), use the `evaluate` method. This is generally used for generating power curves so it expects `array_like` input. For this example a list of size one is used.

```

P, T, Q = rotor.evaluate([Uinf], [Omega], [pitch])

CP, CT, CQ = rotor.evaluate([Uinf], [Omega], [pitch], coefficient=True)

print CP, CT, CQ

```

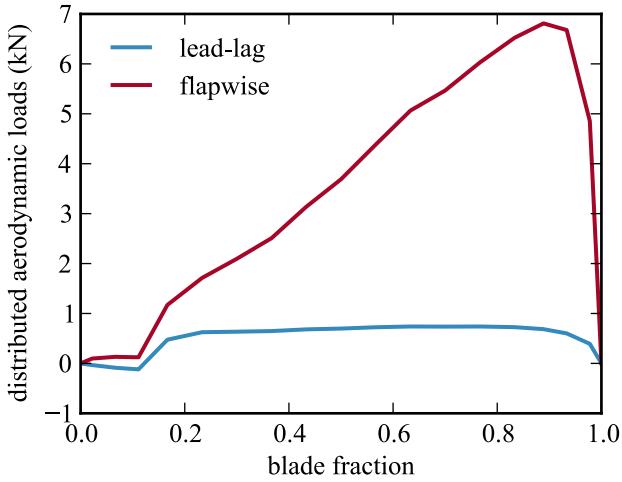
The result is

```

>>> CP = [ 0.48329808]
>>> CT = [ 0.7772276]
>>> CQ = [ 0.06401299]

```

Note that the outputs are numpy arrays (of length 1 for this example). To generate a nondimensional power curve ( $\lambda$  vs  $c_p$ ):



**Figure 1. Flapwise and lead-lag aerodynamic loads along blade.**

```
# velocity has a small amount of Reynolds number dependence
tsr = np.linspace(2, 14, 50)
Omega = 10.0 * np.ones_like(tsr)
Uinf = Omega*pi/30.0 * Rtip/tsr
pitch = np.zeros_like(tsr)

CP, CT, CQ = rotor.evaluate(Uinf, Omega, pitch, coefficient=True)

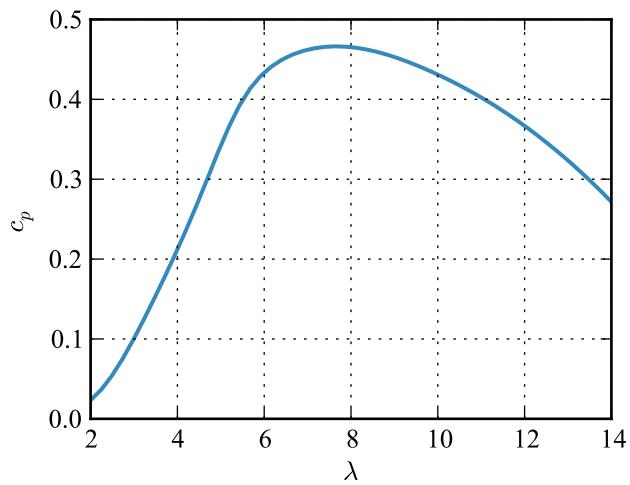
plt.figure()
plt.plot(tsr, CP)
plt.xlabel('$\lambda$')
plt.ylabel('$c_p$')
plt.show()
```

Figure 2 shows the resulting plot.

CCBlade provides a few additional options in its constructor. The other options are shown in the following example with their default values.

```
# create CCBlae object
rotor = CCBlae(r, chord, theta, af, Rhub, Rtip, B, rho, mu,
                precone, tilt, yaw, shearExp, hubHt, nSector
                tiploss=True, hubloss=True, wakerotation=True, usecd=True, iterRe=1)
```

The parameters `tiploss` and `hubloss` toggle Prandtl tip and hub losses respectively. The parameter `wakerotation` toggles wake swirl (i.e.,  $a' = 0$ ). The parameter `usecd` can be used to disable the inclusion of drag in the calculation of the induction factors (it is always used in calculations of the distributed loads). However, doing so may cause potential failure in the solution methodology (see (Ning, 2013)). In practice, it should work fine, but special care for that particular case has not yet been examined, and the default implementation allows for the possibility of convergence failure. All four of these parameters are `True` by default. The parameter `iterRe` is for advanced usage. Referring to (Ning, 2013), this parameter controls the number of internal iterations on the Reynolds number. One iteration is almost always sufficient, but for high accuracy in the Reynolds number `iterRe` could be set at 2. Anything larger than that is unnecessary.



**Figure 2. Power coefficient as a function of tip-speed ratio.**

## 4 Module Documentation

The main methodology is contained in *CCBlade*. Airfoil data is provided by any object that implements *AirfoilInterface*. The helper class *CCAirfoil* is provided as a useful default implementation for *AirfoilInterface*. If *CCAirfoil* is not used, the user must provide an implementation that produces  $C^1$  continuous output (or else accept non-smooth aerodynamic calculations from *CCBlade*). Some of the underlying implementation for *CCBlade* is written in Fortran for computational efficiency.

An HTML version of this documentaion is available that is better formatted for reading the code documentation and contains hyperlinks to the source code.

### 4.1 Airfoil Interface

The airfoil objects used in *CCBlade* need only implement the following `evaluate()` method. Although using *CCAirfoil* for the implementation is recommended, any custom class can be used.

#### Class Summary:

**interface** `ccblade_sa.AirfoilInterface`

Interface for airfoil aerodynamic analysis.

**evaluate** (`alpha, Re`)

Get lift/drag coefficient at the specified angle of attack and Reynolds number

#### Parameters

`alpha` : float (rad)

angle of attack

`Re` : float

Reynolds number

#### Returns

`cl` : float

lift coefficient

`cd` : float

drag coefficient

#### Notes

Any implementation can be used, but to keep the smooth properties of *CCBlade*, the implementation should be  $C^1$  continuous.

### 4.2 CCAirfoil Class

*CCAirfoil* is a helper class used to evaluate airfoil data with a continuously differentiable bivariate spline across the angle of attack and Reynolds number. The degree of the spline polynomials across the Reynolds number is summarized in the following table (the same applies to the angle of attack although generally, the number of points for the angle of attack is much larger).

**Table 1. Degree of spline across Reynolds number.**

len(Re)	degree of spline
1	constant
2	linear
3	quadratic
4+	cubic

**Class Summary:**

**class ccblade\_sa.CCAirfoil (alpha, Re, cl, cd)**

Setup CCAirfoil from raw airfoil data on a grid.

**Parameters**

**alpha** : array\_like (deg)

angles of attack where airfoil data are defined (should be defined from -180 to +180 degrees)

**Re** : array\_like

Reynolds numbers where airfoil data are defined (can be empty or of length one if not Reynolds number dependent)

**cl** : array\_like

lift coefficient 2-D array with shape (alpha.size, Re.size) cl[i, j] is the lift coefficient at alpha[i] and Re[j]

**cd** : array\_like

drag coefficient 2-D array with shape (alpha.size, Re.size) cd[i, j] is the drag coefficient at alpha[i] and Re[j]

**evaluate (alpha, Re)**

Get lift/drag coefficient at the specified angle of attack and Reynolds number.

**Parameters**

**alpha** : float (rad)

angle of attack

**Re** : float

Reynolds number

**Returns**

**cl** : float

lift coefficient

**cd** : float

drag coefficient

## Notes

This method uses a spline so that the output is continuously differentiable, and also uses a small amount of smoothing to help remove spurious multiple solutions.

**classmethod `initFromAerodynFile`(*aerodynFile*)**  
convenience method for initializing with AeroDyn formatted files

### Parameters

**aerodynFile** : str

location of AeroDyn style airfoil file

### Returns

**af** : CCAirfoil

a constructed CCAirfoil object

## 4.3 CCBBlade Class

This class provides aerodynamic analysis of wind turbine rotor blades using BEM theory. It can compute distributed aerodynamic loads and integrated quantities such as power, thrust, and torque. An emphasis is placed on convergence robustness and differentiable output so that it can be used with gradient-based optimization.

### Class Summary:

**class ccblade\_sa.CCBBlade(*r, chord, theta, af, Rhub, Rtip, B=3, rho=1.225, mu=1.81206e-05, precone=0.0, tilt=0.0, yaw=0.0, shearExp=0.2, hubHt=80.0, nSector=8, tiploss=True, hubloss=True, wakerotation=True, usecd=True, iterRe=1*)**  
Constructor for aerodynamic rotor analysis

### Parameters

**r** : array\_like (m)

radial locations where blade is defined (should be increasing)

**chord** : array\_like (m)

corresponding chord length at each section

**theta** : array\_like (deg)

corresponding *twist angle* at each section— positive twist decreases angle of attack.

**af** : list(AirfoilInterface)

list of *AirfoilInterface* objects at each section

**Rhub** : float (m)

radial location of hub

**Rtip** : float (m)

radial location of tip

**B** : int, optional

number of blades

**rho** : float, optional (kg/m^3)

freestream fluid density  
**mu** : float, optional (kg/m/s)  
 dynamic viscosity of fluid  
**precone** : float, optional (deg)  
 blade *precone angle*  
**tilt** : float, optional (deg)  
 nacelle *tilt angle*  
**yaw** : float, optional (deg)  
 nacelle *yaw angle*  
**shearExp** : float, optional  
 shear exponent for a power-law wind profile across hub  
**hubHt** : float, optional  
 hub height used for power-law wind profile.  $U = U_{ref} * (z/z_0)^{\text{shearExp}}$ , where  $z = \text{hubHt} + r * \cos(\text{azimuth})$ ,  $z_0 = \text{hubHt}$   
**nSector** : int, optional  
 number of azimuthal sectors to discretize aerodynamic calculation. automatically set to 1 if tilt, yaw, and shearExp are all 0.0. Otherwise set to a minimum of 4.  
**tiploss** : boolean, optional  
 if True, include Prandtl tip loss model  
**hubloss** : boolean, optional  
 if True, include Prandtl hub loss model  
**wakerotation** : boolean, optional  
 if True, include effect of wake rotation (i.e., tangential induction factor is nonzero)  
**usecd** : boolean, optional  
 If True, use drag coefficient in computing induction factors (always used in evaluating distributed loads from the induction factors). Note that the default implementation may fail at certain points if drag is not included (see Section 4.2 in (Ning, 2013)). This can be worked around, but has not been implemented.  
**iterRe** : int, optional  
 The number of iterations to use to converge Reynolds number. Generally iterRe=1 is sufficient, but for high accuracy in Reynolds number, iterRe=2 iterations can be used. More than that should not be necessary.  
**distributedAeroLoads** (*Uinf*, *Omega*, *pitch*, *azimuth*)  
 Compute distributed aerodynamic loads along blade.

**Parameters**

**Uinf** : float (m/s)  
 freestream velocity

**Omega** : float (RPM)

    rotor rotation speed

**pitch** : float (deg)

    blade pitch in same direction as *twist* (positive decreases angle of attack)

**azimuth** : float (deg)

*azimuth angle* where aerodynamic loads should be computed at

#### Returns

**r** : ndarray (m)

    radial stations where force is specified (should go all the way from hub to tip)

**theta** : ndarray (deg)

    corresponding geometric twist angle (not including pitch)— positive twists nose into the wind

**Tp** : ndarray (N/m)

    force per unit length tangential to the section in the direction of rotation

**Np** : ndarray (N/m)

    force per unit length normal to the section on downwind side

**evaluate** (*Uinf*, *Omega*, *pitch*, *coefficient=False*)

Run the aerodynamic analysis at the specified conditions.

#### Parameters

**Uinf** : array\_like (m/s)

    freestream velocity

**Omega** : array\_like (RPM)

    rotor rotation speed

**pitch** : array\_like (deg)

    blade pitch setting

**coefficient** : bool, optional

    if True, results are returned in nondimensional form

#### Returns

**P or CP** : ndarray (W)

    power or power coefficient

**T or CT** : ndarray (N)

    thrust or thrust coefficient (magnitude)

**Q or CQ** : ndarray (N\*m)

    torque or torque coefficient (magnitude)

### **Notes**

Normalization uses Rtip and rho provided in the constructor:

$$CP = P / (q * Uinf * A)$$

$$CT = T / (q * A)$$

$$CQ = Q / (q * A * Rtip)$$

where  $A = \pi * Rtip^{**2}$  and  $q = 0.5 * \rho * Uinf^{**2}$

## 5 Theory

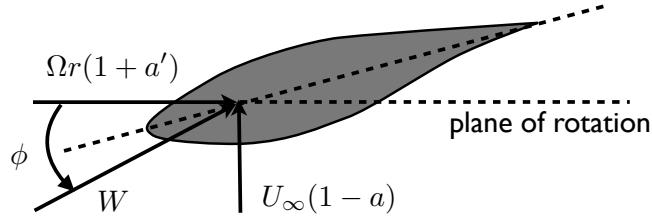
**Note:** Only an overview of the theory is included here; details can be found in Ning (2013).

The rotor aerodynamic analysis is based on blade element momentum (BEM) theory. Using BEM theory in a gradient-based rotor optimization problem can be challenging because of occasional convergence difficulties of the BEM equations. The standard approach to solving the BEM equations is to arrange the equations as functions of the axial and tangential induction factors and solve the fixed-point problem:

$$(a, a') = f_{fp}(a, a')$$

using either fixed-point iteration, Newton's method, or a related fixed-point algorithm. An alternative approach is to use nonlinear optimization to minimize the sum of the squares of the residuals of the induction factors (or normal and tangential loads). Although these approaches are generally successful, they suffer from instabilities and failure to converge in some regions of the design space. Thus, they require increased complexity and/or heuristics (but may still not converge).

The new BEM methodology transforms the two-variable, fixed-point problem into an equivalent one-dimensional root-finding problem. This is enormously beneficial as methods exist for one-dimensional root-finding problems that are guaranteed to converge as long as an appropriate bracket can be found. The key insight to this reduction is to use the local inflow angle  $\phi$  and the magnitude of the inflow velocity  $W$  as the two unknowns in specifying the inflow conditions, rather than the traditional axial and tangential induction factors (see Figure 3).



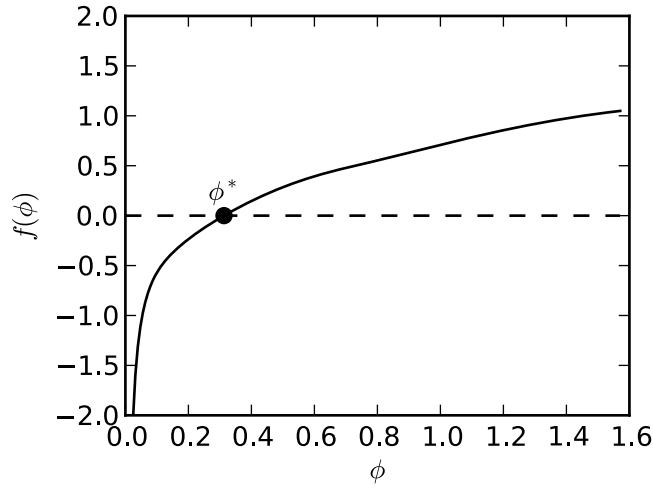
**Figure 3. Parameters specifying inflow conditions of a rotating blade section.**

This approach allows the BEM equations to be reduced to a one-dimensional residual function as a function of  $\phi$ :

$$f(\phi) = \frac{\sin \phi}{1 - a(\phi)} - \frac{\cos \phi}{\lambda_r(1 + a'(\phi))} = 0$$

Figure 4 shows the typical behavior of  $f(\phi)$  over the range  $\phi \in (0, \pi/2]$ . Almost all solutions for wind turbines fall within this range (for the provable convergence properties to be true, solutions outside of this range must also be considered). The referenced paper (Ning, 2013) demonstrates through mathematical proof that the methodology will always find a bracket to a zero of  $f(\phi)$  without any singularities in the interior. This proof, along with existing proofs for root-finding methods like Brent's method (Brent, 1971), implies that a solution is guaranteed. Furthermore, not only is the solution guaranteed, but it can be found efficiently and in a continuous manner. This behavior allows the use of gradient-based algorithms to solve rotor optimization problems much more effectively than with traditional BEM solution approaches.

Any corrections to the BEM method can be used with this methodology (e.g., finite number of blades and skewed wake) as long as the axial induction factor can be expressed as a function of  $\phi$  (either explicitly or through a numerical solution). CCBlade chooses to include both hub and tip losses using Prandtl's method (Glauert, 1935) and a



**Figure 4. Residual function of BEM equations using new methodology. Solution point is where  $f(\phi) = 0$ .**

high-induction factor correction by Buhl (2005). Drag is included in the computation of the induction factors. However, all of these options can be toggled on or off. For a given wind speed, a spline is fit to the normal and tangential forces along the radial discretization of the blade before integrating for thrust and torque. This allows for smoother variation in thrust and torque for improved gradient estimation.

## Coordinate System

This module defines coordinate systems for horizontal axis wind turbines and provides convenience methods for transforming vectors between the various coordinate systems. The supplied transformation methods are for *rotation only* and do not account for any offsets that may be necessary depending on the vector quantity (e.g., transfer of forces between coordinate system does not depend on the location where the force is defined, but position, velocity, moments, etc. do). In other words the vectors are treated as directions only and are independent of the defined position. How the vector should transform based on position is not generalizable and depends on the quantity of interest. All coordinate systems obey the right-hand rule,  $x \times y = z$ , and all angles must be input in **degrees**. The turbine can be either an upwind or downwind configuration, but in either case it is assumed that the blades rotate in the **clockwise** direction when looking downwind (more specifically the rotor is assumed to rotate about the  $+x_h$  axis in Figure 7). The vectors allow for elementary operations ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ) between other vectors of the same type, or with scalars (e.g., `force_total = force1 + force2`).

```
class csystem.DirectionVector (x, y, z)
```

3-Dimensional vector that depends on direction only (not position).

### Parameters

**x** : float or ndarray

  x-direction of vector(s)

**y** : float or ndarray

  y-direction of vector(s)

**z** : float or ndarray

  z-direction of vector(s)

## Inertial and Wind-aligned

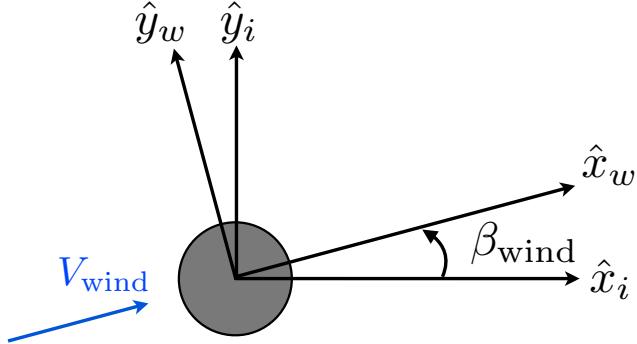


Figure 5. Inertial and Wind-aligned axes.

Figure 5 defines the transformation between the inertial and wind-aligned coordinate systems. The two coordinate systems share a common origin, and a common z-direction. The wind angle  $\beta$  is positive for rotation about the  $+z$  axis. The direction of wave loads are defined similarly to the wind loads, but there is no wave-aligned coordinate system.

*Inertial coordinate system*

**origin:** center of the tower base (ground-level or sea-bed level)

**x-axis:** any direction as long as used consistently, but convenient to be in primary wind direction

**y-axis:** follows from the right-hand rule

**z-axis:** up the tower (opposite to gravity vector)

#### *Wind-aligned coordinate system*

**origin:** center of the tower base (ground-level or sea-bed level)

**x-axis:** in direction of the wind

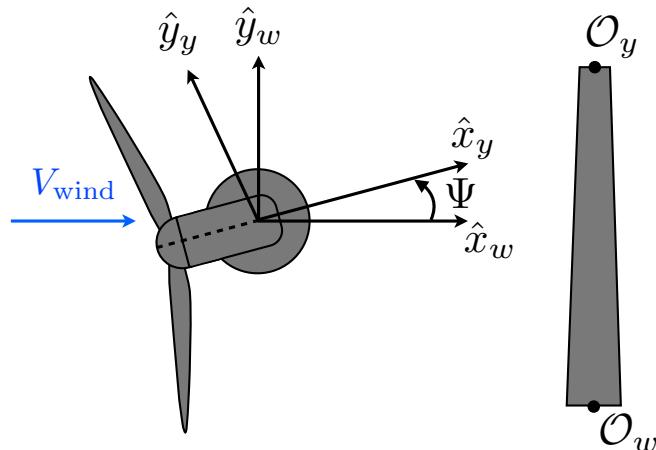
**y-axis:** follows from the right-hand rule

**z-axis:** up the tower (opposite to gravity vector), coincident with inertial z-axis

**Table 2. Inertial-Wind conversion methods**

inertialToWind(beta)	Rotates from inertial to wind-aligned
windToInertial(beta)	Rotates from wind-aligned to inertial

#### **Wind-aligned and Yaw-aligned**



**Figure 6. Wind-aligned and yaw-aligned axes.  $\Psi$  is the rotor yaw angle.**

Figure 6 defines the transformation between the wind-aligned and yaw-aligned coordinate systems. The two coordinate systems are offset by the height  $h_t$  along the common z-axis. The yaw angle  $\Psi$  is positive when rotating about the +z axis, and should be between -180 and +180 degrees.

#### *Yaw-aligned coordinate system*

**origin:** Tower top (center of the yaw bearing system)

**x-axis:** along projection of rotor shaft in horizontal plane (aligned with rotor shaft for zero tilt angle).

The positive direction is defined such that the x-axis points downwind at its design operating orientation (i.e., at zero yaw  $x_y$  is the same direction as  $x_w$ ). Thus, for a downwind machine the  $x_y$  axis would still

be downwind at zero yaw, but in terms of nacelle orientation it would point from the back of the nacelle toward the hub.

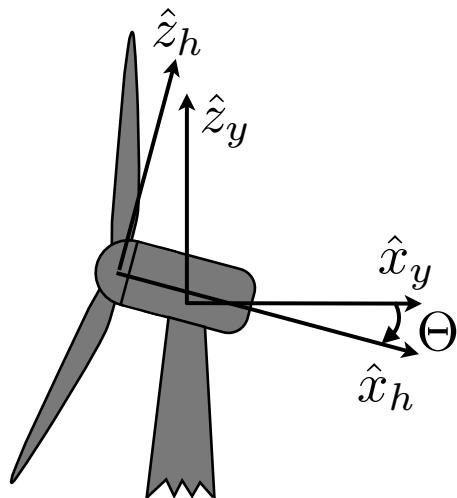
**y-axis:** follows from the right-hand rule

**z-axis:** points up the tower (opposite to gravity vector), coincident with wind-aligned z-axis

**Table 3. Wind-Yaw conversion methods**

windToYaw(Psi)	Rotates from wind-aligned to yaw-aligned
yawToWind(Psi)	Rotates from yaw-aligned to wind-aligned

## Yaw-aligned and Hub-aligned



**Figure 7. Yaw-aligned and hub-aligned axes.  $\Theta$  is the rotor tilt angle.**

Figure 7 defines the transformation between the yaw-aligned and hub-aligned coordinate systems. The two coordinate systems share a common y axis. The tilt angle  $\Theta$  is positive when rotating about the +y axis, which tilts the rotor up for an upwind machine (tilts the rotor down for a downwind machine).

### Hub-aligned coordinate system

**origin:** center of the rotor.

**x-axis:** along the rotor shaft toward the nominal downwind direction (aligned with  $x_y$  for zero tilt)

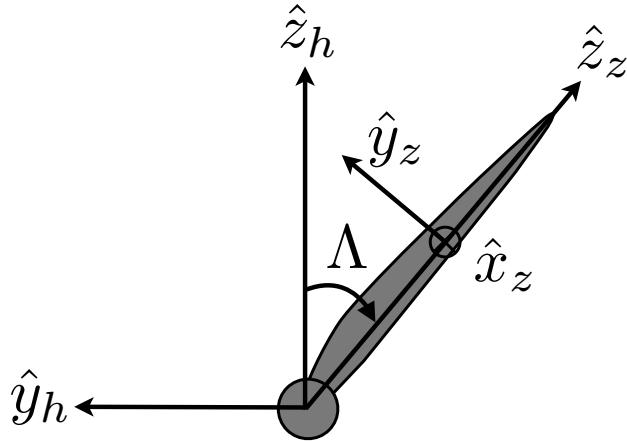
**y-axis:** coincident with yaw-aligned y-axis

**z-axis:** right-hand rule (vertical if zero tilt)

**Table 4. Yaw-Hub conversion methods**

yawToHub(Theta)	Rotates from yaw-aligned to hub-aligned
hubToYaw(Theta)	Rotates from hub-aligned to yaw-aligned

## Hub-aligned and Azimuth-aligned



**Figure 8. Hub-aligned and azimuth-aligned axes.**  $\Lambda$  is the (local) blade azimuth angle.

Figure 8 defines the transformation between the hub-aligned and azimuth-aligned coordinate systems. The two coordinate systems share a common x-axis. The azimuth angle  $\Lambda$  is positive when rotating about the  $+x$  axis. The blade can employ a variable azimuth angle along the blade axis, to allow for swept blades.

### Azimuth-aligned coordinate system

A rotating coordinate system—about the  $x_h$  axis. The coordinate-system is locally-defined for the case of a variable-swept blade.

**origin:** blade pitch axis, local to the blade section

**x-axis:** aligned with the hub-aligned x-axis

**y-axis:** right-hand rule

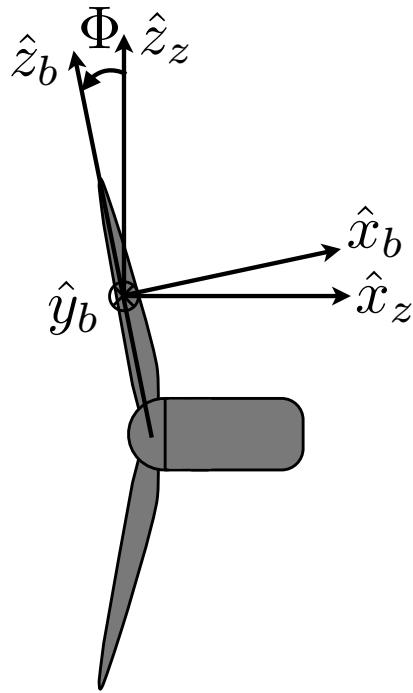
**z-axis:** along projection of blade from root to tip in the  $y_h - z_h$  plane (aligned with blade only for zero precone)

**Table 5. Hub-Azimuth conversion methods**

hubToAzimuth(Lambda)	Rotates from hub-aligned to azimuth-aligned
azimuthToHub(Lambda)	Rotates from azimuth-aligned to hub-aligned

## Azimuth-aligned and Blade-aligned

Figure 9 defines the transformation between the azimuth-aligned and blade-aligned coordinate systems. The  $y_b$  and  $z_b$  axes are in the same direction. The two coordinate systems rotate together such that the  $x_b - z_b$  plane is always coplanar with the  $x_z - z_z$  plane. The precone angle  $\Phi$  is positive when rotating about the  $-y_z$  axis, and causes the blades to tilt away from the nacelle/tower for a downwind machine (tilts toward tower for upwind machine). The blade can employ a variable precone angle along the blade axis. The blade-aligned coordinate system is considered local to a section of the blade. *Blade-aligned coordinate system*



**Figure 9. Azimuth-aligned and blade-aligned axes.  $\Phi$  is the (local) blade precone angle.**

A rotating coordinate system that rotates with the azimuth-aligned coordinate system. The coordinate-system is locally-defined along the blade radius. The direction of blade rotation is in the negative y-axis. A force in the x-axis would be a flapwise shear, and a force in the y-axis would be a lead-lag shear.

**origin:** blade pitch axis, local to the blade section

**x-axis:** follows from the right-hand rule (in nominal downwind direction)

**y-axis:** opposite to rotation direction, positive from section leading edge to trailing edge (for no twist)

**z-axis:** along the blade pitch axis in increasing radius

**Table 6. Azimuth-Blade conversion methods**

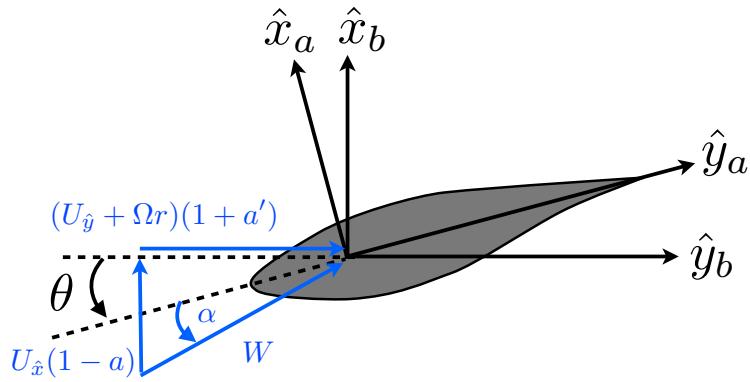
azimuthToBlade(Phi)	Rotates from azimuth-aligned to blade-aligned
bladeToAzimuth(Phi)	Rotates from blade-aligned to azimuth-aligned

## Blade-aligned and Airfoil-aligned

Figure 10 defines the transformation between the blade-aligned and airfoil-aligned coordinate systems. The  $z_b$  and  $z_a$  axes are in the same direction. The twist angle  $\theta$  is positive when rotating about the  $-z_a$  axis, and causes the angle of attack to decrease.

### Airfoil-aligned coordinate system

A force in the x-axis would be a flapwise shear, and a force in the y-axis would be an edgewise shear.



**Figure 10. Blade-aligned and airfoil-aligned coordinate systems.**  $\theta$  is the airfoil twist + pitch angle. For convenience the local wind vector and angle of attack is shown.

**origin:** blade pitch axis, local to the blade section

**x-axis:** follows from the right-hand rule

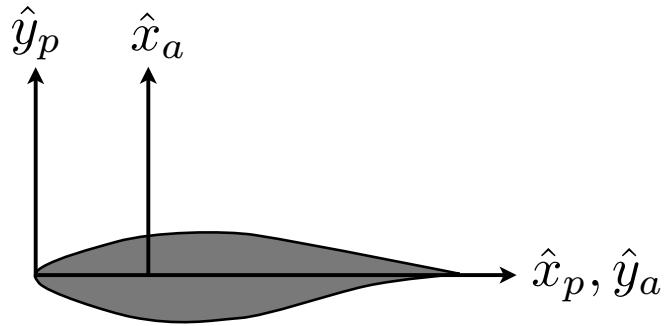
**y-axis:** along chord line in direction of trailing edge

**z-axis:** along the blade pitch axis in increasing radius, same as  $z_b$  (into the page in above figure)

**Table 7. Blade-Airfoil conversion methods**

bladeToAirfoil(theta)	Rotates from blade-aligned to airfoil-aligned
airfoilToBlade(theta)	Rotates from airfoil-aligned to blade-aligned

## Airfoil-aligned and Profile



**Figure 11. Airfoil-aligned and profile coordinate systems.**

Figure 11 defines the transformation between the airfoil-aligned and profile coordinate systems. The profile coordinate system is generally used only to define airfoil profile data.

*Profile coordinate system*

**origin:** airfoil nose

**x-axis:** positive from nose to trailing edge along chord line

**y-axis:** orthogonal to x-axis, positive from lower to upper surface

**z-axis:** n/a (profile is a 2-dimensional coordinate system)

**Table 8. Airfoil-Profile conversion methods**

airfoilToProfile()	Rotates from airfoil-aligned to profile
profileToAirfoil()	Rotates from profile to airfoil-aligned

## Bibliography

- Brent, R.P. (1971). "An Algorithm with Guaranteed Convergence for Finding a Zero of a Function." *The Computer Journal* 14(4); pp. 422–425.
- Buhl, M.L. (August 2005). *A New Empirical Relationship between Thrust Coefficient and Induction Factor for the Turbulent Windmill State*. NREL/TP-500-36834, National Renewable Energy Laboratory, Golden, CO.
- Glauert, H. (1935). *Airplane Propellers*, Vol. 4. Springer Verlag.
- Ning, S.A. (2013). "A Simple Solution Method for the Blade Element Momentum Equations with Guaranteed Convergence." *Wind Energy* (in press).