# AirfoilPrep.py Documentation

## *Release 0.1.0*

S.A. 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.

**Technical Report**
NREL/TP-5000-58817
September 2013

Contract No. DE-AC36-08GO28308

NATIONAL RENEWABLE ENERGY LABORATORY

# AirfoilPrep.py Documentation

## *Release 0.1.0*

S.A. Ning

Prepared under Task No. 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.

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

**Technical Report**
NREL/TP-5000-58817
September 2013

Contract No. DE-AC36-08GO28308

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at 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
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

## List of Figures



## List of Tables

# 1 Introduction

AirfoilPrep.py (pronounced Airfoil Preppy) provides functionality to preprocess aerodynamic airfoil data. Essentially, the module is an object oriented version of the AirfoilPrep spreadsheet with additional functionality and is written in the Python language. The intent is to provide the functionality of the AirfoilPrep spreadsheet, but in an easy-to-use format both for stand-alone preprocessing through scripting and for direct implementation within other codes such as blade element momentum methods.

AirfoilPrep.py allows the user to read in two-dimensional (2-D) aerodynamic airfoil data (i.e., from wind tunnel data or numerical simulation), apply three-dimensional (3-D) rotation corrections for wind turbine applications, and extend the data to very large angles of attack. Airfoil data can also be blended together to define intermediate sections between linearly lofted sections. Capabilities unique to the Python version include the ability to read and write to AeroDyn format files directly. The only feature that is contained in the spreadsheet version but is currently missing in AirfoilPrep.py, is handling of pitching moment coefficients.

This document discusses installation, usage, and documentation of the module. Because the theory is simplistic, only a brief overview is provided in the documentation section with corresponding references that contain further detail.

# 2 Installation

---

**Prerequisites**

NumPy

---

Download either AirfoilPrep.py-0.1.0.tar.gz or AirfoilPrep.py-0.1.0.zip and uncompress/unpack it.

If you are only going to use AirfoilPrep.py from the *command-line* for simple preprocessing, no installation is necessary. The `airfoilprep.py` file in the `src` directory can be copied to any location on your computer and used directly. For convenience you may want to add the directory it is contained in to the system path. If you will use AirfoilPrep.py from within Python for more advanced preprocessing or for integration with other codes, AirfoilPrep.py should be installed using:

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

To verify that the installation was successful and to run all the unit tests:

```
$ python test/test_airfoilprep.py
```

An “OK” signifies that all the tests passed.

See *module documentation* for more details on usage within Python. To access an HTML version of this documentation with improved formatting and links to the source code, open `docs/index.html`.

# 3 Tutorial

AirfoilPrep.py can be accessed either through the *command line* or through *Python*. The command-line interface is the simplest but provides only a limited number of options. The Python interface is useful for more advanced preprocessing and for integration with other codes.

## 3.1 Command-Line Usage

From the terminal, to see the options, invoke help:

```
$ python airfoilprep.py -h
```

When using the command-line options, all files must be AeroDyn formatted files. The command line provides three main methods for working with files directly: 3-D stall corrections, high angle of attack extrapolation, and a blending operation. In all cases, you first specify the name (and path if necessary) of the file you want to work with:

```
$ python airfoilprep.py airfoil.dat
```

The following optional arguments are available

**Table 1. Available flags for using AirfoilPrep.py in command-line mode.**

| flag | arguments | description |
|---|---|---|
| `-h` | | display help |
| `--stall3D` | r/R c/r tsr | 3-D rotational corrections near stall |
| `--extrap` | cdmax | high angle of attack extrapolation |
| `--blend` | other weight | blend with other file using specified weight |
| `--out` | outfile | specify a different name for output file |
| `--plot` | | plot data (for diagnostic purposes) using matplotlib |
| `--common` | | output airfoil data using a common set of angles of attack |

### *3.1.1 Stall Corrections*

The first method available from the command line is `--stall3D`, which reads the file, applies rotational corrections, and then writes the data to a separate file. This argument must specify the parameters used for the correction in the format `--stall3D r/R c/r tsr`, where `r/R` is the local radius normalized by the rotor radius, `c/r` is the local chord normalized by the local radius, and `tsr` is the local tip-speed ratio. For example, if `airfoil.dat` contained 2-D data with `r/R=0.5`, `c/r=0.15`, `tsr=5.0`, then we would apply rotational corrections to the airfoil using:

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0
```

By default the output file will append _3D to the name. In the above example, the output file would be `airfoil_3D.dat`. However, this can be overriden with the `--out` option. To output to a file at `/Users/Me/Airfoils/my_new_airfoil.dat`:

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0 \
> --out /Users/Me/Airfoils/my_new_airfoil.dat
```

Optionally, you can also plot the results (matplotlib must be installed) with the `--plot` flag. For example,

```
$ python airfoilprep.py DU21_A17.dat --stall3D 0.2 0.3 5.0 --plot
```

displays Figure 1 (only one Reynolds number shown) along with producing the output file. AirfoilPrep.py can

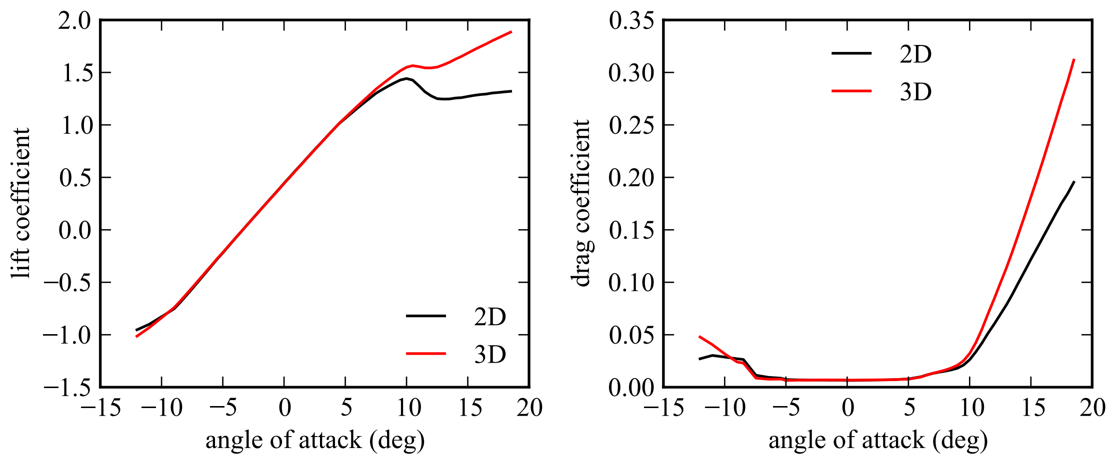


**Figure 1. Lift and drag coefficient with 3-D stall corrections applied.**

utilize data for which every Reynolds number uses a different set of angles of attack. However, some codes need data on a uniform grid of Reynolds number and angle of attack. To output the data on a common set of angles of attack, use the `--common` flag.

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0 --common
```

### 3.1.2 Angle of Attack Extrapolation

The second method available from the command line is `--extrap`, which reads the file, applies high angle of attack extrapolations, and then writes the data to a separate file. This argument must specify the maximum drag coefficient to use in the extrapolation across the full +/- 180-degree range `--extrap cdmax`. For example, if `airfoil_3D.dat` contained 3D stall corrected data and `cdmax=1.3`, then we could extrapolate the airfoil using:

```
$ python airfoilprep.py airfoil_3D.dat --extrap 1.3
```

By default the output file will append _extrap to the name. In the above example, the output file would be `airfoil_-3D_extrap.dat`. However, this can also be overriden with the `--out` flag. The `--common` flag is also useful here if a common set of angles of attack is needed.

The output can be plotted with the –plot flag. The command

```
$ python airfoilprep.py DU21_A17_3D.dat --extrap 1.3 --plot
```

displays Figure 2 (only one Reynolds number shown) along with producing the output file.

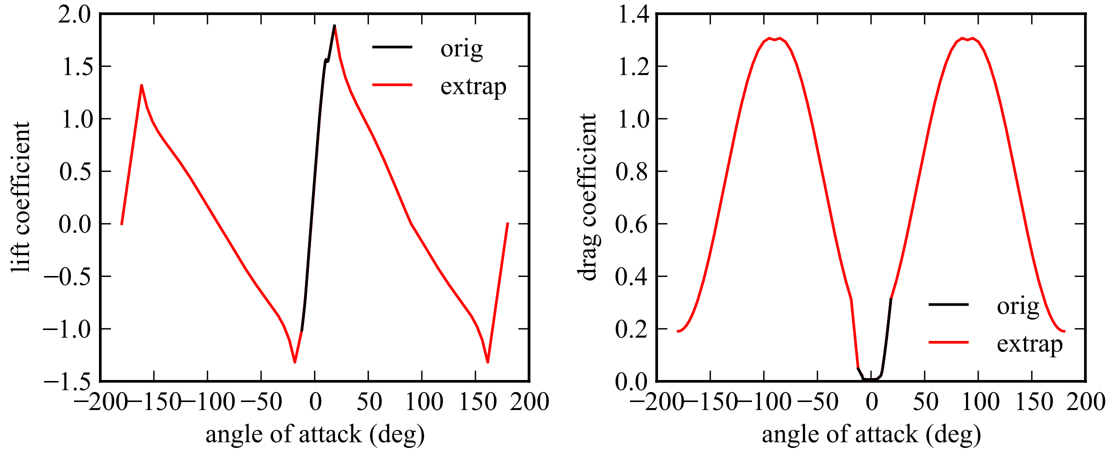


**Figure 2. Airfoil data extrapolated to high angles of attack.**

#### *3.1.3 Blending*

The final capability accessible from the command line is blending of airfoils. This is invoked through `--blend filename weight`, where `filename` is the name (and path if necessary) of a second file to blend with, and `weight` is the weighting used in the blending. The weight ranges on a scale of 0 to 1 where 0 returns the first airfoil and 1 the second airfoil. For example, the following command blends airfoil1.dat with airfoil2.dat with a weighting of 0.3 (conceptually the new airfoil would equal 0.7*airfoil1.dat + 0.3*airfoil2.dat).

```
$ python airfoilprep.py airfoil1.dat --blend airfoil2.dat 0.3
```

By default, the output file appends the names of the two files with a '+' sign, then appends the weighting using '_blend' and the value for the weight. In this example, the output file would be `airfoil1+airfoil2_-blend0.3.dat`. Just like the previous case, the name of the output file can be overridden by using the `--out` flag. The `--common` flag is also useful here if a common set of angles of attack is needed. This data can also be plotted, but only the blended airfoil data will be shown. Direct comparison to the original data is not always possible, because the blend method allows for the specified airfoils to be defined at different Reynolds numbers. Blending first occurs across Reynolds numbers and then across angle of attack.

## 3.2 Python Usage

The Python interface allows for more flexible usage or integration with other programs. Descriptions of the interfaces for the classes contained in the module are contained in *Module Documentation*.

Airfoils can be created from AeroDyn formatted files,

```
from airfoilprep import Polar, Airfoil
import numpy as np

airfoil = Airfoil.initFromAerodynFile('DU21_A17.dat')
```

or they can be created directly from airfoil data.

```
# first polar
Re = 7e6
```

```
alpha = [-14.50, -12.01, -11.00, -9.98, -8.12, -7.62, -7.11, -6.60, -6.50,
         -6.00, -5.50, -5.00, -4.50, -4.00, -3.50, -3.00, -2.50, -2.00, -1.50,
         -1.00, -0.50, 0.00, 0.50, 1.00, 1.50, 2.00, 2.50, 3.00, 3.50, 4.00,
         4.50, 5.00, 5.50, 6.00, 6.50, 7.00, 7.50, 8.00, 8.50, 9.00, 9.50,
         10.00, 10.50, 11.00, 11.50, 12.00, 12.50, 13.00, 13.50, 14.00, 14.50,
         15.00, 15.50, 16.00, 16.50, 17.00, 17.50, 18.00, 18.50, 19.00, 19.50,
         20.00, 20.50]
cl = [-1.050, -0.953, -0.900, -0.827, -0.536, -0.467, -0.393, -0.323, -0.311,
      -0.245, -0.178, -0.113, -0.048, 0.016, 0.080, 0.145, 0.208, 0.270, 0.333,
      0.396, 0.458, 0.521, 0.583, 0.645, 0.706, 0.768, 0.828, 0.888, 0.948,
      0.996, 1.046, 1.095, 1.145, 1.192, 1.239, 1.283, 1.324, 1.358, 1.385,
      1.403, 1.401, 1.358, 1.313, 1.287, 1.274, 1.272, 1.273, 1.273, 1.273,
      1.272, 1.273, 1.275, 1.281, 1.284, 1.296, 1.306, 1.308, 1.308, 1.308,
      1.308, 1.307, 1.311, 1.325]
cd = [0.0567, 0.0271, 0.0303, 0.0287, 0.0124, 0.0109, 0.0092, 0.0083, 0.0089,
      0.0082, 0.0074, 0.0069, 0.0065, 0.0063, 0.0061, 0.0058, 0.0057, 0.0057,
      0.0057, 0.0057, 0.0057, 0.0057, 0.0057, 0.0058, 0.0058, 0.0059, 0.0061,
      0.0063, 0.0066, 0.0071, 0.0079, 0.0090, 0.0103, 0.0113, 0.0122, 0.0131,
      0.0139, 0.0147, 0.0158, 0.0181, 0.0211, 0.0255, 0.0301, 0.0347, 0.0401,
      0.0468, 0.0545, 0.0633, 0.0722, 0.0806, 0.0900, 0.0987, 0.1075, 0.1170,
      0.1270, 0.1368, 0.1464, 0.1562, 0.1664, 0.1770, 0.1878, 0.1987, 0.2100]

p1 = Polar(Re, alpha, cl, cd)

# second polar
Re = 9e6
alpha = [-14.24, -13.24, -12.22, -11.22, -10.19, -9.70, -9.18, -8.18, -7.19,
         -6.65, -6.13, -6.00, -5.50, -5.00, -4.50, -4.00, -3.50, -3.00, -2.50,
         -2.00, -1.50, -1.00, -0.50, 0.00, 0.50, 1.00, 1.50, 2.00, 2.50, 3.00,
         3.50, 4.00, 4.50, 5.00, 5.50, 6.00, 6.50, 7.00, 7.50, 8.00, 9.00,
         9.50, 10.00, 10.50, 11.00, 11.50, 12.00, 12.50, 13.00, 13.50, 14.00,
         14.50, 15.00, 15.50, 16.00, 16.50, 17.00, 17.50, 18.00, 18.50, 19.00]
cl = [-1.229, -1.148, -1.052, -0.965, -0.867, -0.822, -0.769, -0.756, -0.690,
      -0.616, -0.542, -0.525, -0.451, -0.382, -0.314, -0.251, -0.189, -0.120,
      -0.051, 0.017, 0.085, 0.152, 0.219, 0.288, 0.354, 0.421, 0.487, 0.554,
      0.619, 0.685, 0.749, 0.815, 0.879, 0.944, 1.008, 1.072, 1.135, 1.197,
      1.256, 1.305, 1.390, 1.424, 1.458, 1.488, 1.512, 1.533, 1.549, 1.558,
      1.470, 1.398, 1.354, 1.336, 1.333, 1.326, 1.329, 1.326, 1.321, 1.331,
      1.333, 1.340, 1.362]
cd = [0.1461, 0.1263, 0.1051, 0.0886, 0.0740, 0.0684, 0.0605, 0.0270, 0.0180,
      0.0166, 0.0152, 0.0117, 0.0105, 0.0097, 0.0092, 0.0091, 0.0089, 0.0089,
      0.0088, 0.0088, 0.0088, 0.0088, 0.0088, 0.0087, 0.0087, 0.0088, 0.0089,
      0.0090, 0.0091, 0.0092, 0.0093, 0.0095, 0.0096, 0.0097, 0.0099, 0.0101,
      0.0103, 0.0107, 0.0112, 0.0125, 0.0155, 0.0171, 0.0192, 0.0219, 0.0255,
      0.0307, 0.0370, 0.0452, 0.0630, 0.0784, 0.0931, 0.1081, 0.1239, 0.1415,
      0.1592, 0.1743, 0.1903, 0.2044, 0.2186, 0.2324, 0.2455]
p2 = Polar(Re, alpha, cl, cd)

# create airfoil object (can contain as many polars as desired)
af = Airfoil([p1, p2])
```

Blending is easily accomplished just like in the *command-line interface*. There is no requirement that the two airfoils share a common set of angles of attack.

```python
airfoil1 = Airfoil.initFromAerodynFile('DU21_A17.dat')
airfoil2 = Airfoil.initFromAerodynFile('DU25_A17.dat')

# blend the two airfoils
airfoil_blend = airfoil1.blend(airfoil2, 0.3)
```

Applying 3-D corrections and high alpha extensions directly in Python, allows for a few additional options as compared to the command-line version. The following example performs the same 3-D correction as in the *command-line version*, followed by an alternative 3-D correction that utilizes some of the optional inputs. See `correction3D` for more details on the optional parameters.

```python
r_over_R = 0.5
chord_over_r = 0.15
tsr = 5.0

# 3D stall correction
af3D_ex1 = af.correction3D(r_over_R, chord_over_r, tsr)

# a second example using the optional inputs
alpha_max_corr = 25  # apply full rotational correction only up to this angle of attack
alpha_linear_min = -3  # angle of attack to start evaluating slope of linear region
alpha_linear_max = 7  # angle of attack to stop evaluating slope of linear region

af3D_ex2 = af.correction3D(r_over_R, chord_over_r, tsr,
                           alpha_max_corr=alpha_max_corr,
                           alpha_linear_min=alpha_linear_min,
                           alpha_linear_max=alpha_linear_max)
```

The airfoil data can be extended to high angles of attack using the `extrapolate` method. Just like the previous method, a few optional parameters are available through the Python interface. The following example performs the same extrapolation as in the *command-line version*, followed by an alternative extrapolation that utilizes some of the optional inputs.

```python
cdmax = 1.3

# compute a 3D corrected and extended airfoil
af_extrap1 = af.extrapolate(cdmax)

# a second example using the optional inputs
AR = 17  # blade aspect ratio. If provided, cdmax is estimated using the aspect ratio.
cdmin = 0.001  # minimum drag coefficient.  Viterna's method can occasionally produce
               # negative drag coefficients.  A minimum is used to prevent unphysical data.
               # The passed in value is used to override the default.

af_extrap2 = af.extrapolate(cdmax, AR=AR, cdmin=cdmin)
```

Some codes need to use the same set of angles of attack data for every Reynolds number defined in the airfoil. The following example performs the same method as in the *command-line version* followed by an alternate approach where the user can specify the set of angles of attack to use.

```python
# create new airfoil that uses the same angles of attack at each Reynolds number
af_common1 = af.interpToCommonAlpha()

# default approach uses a union of all defined angles of attack
# alternatively, specify the exact angles to use
```

```
alpha = np.arange(-180, 180)
af_common2 = af.interpToCommonAlpha(alpha)
```

For direct access to the underlying data in a grid format (if not already a grid, it is interpolated to a grid first), use the `createDataGrid` method as follows:

```
# extract a data grid from airfoil
alpha, Re, cl, cd = af.createDataGrid()

# cl[i, j] is the lift coefficient for alpha[i] and Re[j]
```

Finally, writing AeroDyn formatted files is straightforward.

```
af.writeToAerodynFile('output.dat')
```

# 4 Module Documentation

Two classes are provided in the module: *Polar* and *Airfoil*. Generally, the Polar class is not needed for direct usage except for its constructor. All objects in this module are **immutable**. In other words, calling Airfoil.correct3D() creates a new modified airfoil object rather than editing the existing object.

This PDF version of the documentation only provides an summary of the classes and methods. Further details are found in the HTML version of this documentation, complete with hyperlinks to the source code.

## 4.1 Polar Class

A Polar object is meant to represent the variation in lift, drag, and pitching moment coefficient with angle of attack at a fixed Reynolds number. Generally, the methods of this class do not need to be used directly (other than the constructor), but rather are used by the `Airfoil` class.

**Class Summary:**

**class** `airfoilprep.`**`Polar`**(*Re*, *alpha*, *cl*, *cd*)

Constructor

**Parameters**

**Re** : float

Reynolds number

**alpha** : ndarray (deg)

angle of attack

**cl** : ndarray

lift coefficient

**cd** : ndarray

drag coefficient

**`blend`**(*other*, *weight*)

Blend this polar with another one with the specified weighting

**Parameters**

**other** : Polar

another Polar object to blend with

**weight** : float

blending parameter between 0 and 1. 0 returns self, whereas 1 returns other.

**Returns**

**polar** : Polar

a blended Polar

**`correction3D`**(*r_over_R*, *chord_over_r*, *tsr*, *alpha_max_corr=30*, *alpha_linear_min=-5*, *alpha_-linear_max=5*)

Applies 3-D corrections for rotating sections from the 2-D data.

**Parameters**

**r_over_R** : float

local radial position / rotor radius

**chord_over_r** : float

local chord length / local radial location

**tsr** : float

tip-speed ratio

**alpha_max_corr** : float, optional (deg)

maximum angle of attack to apply full correction

**alpha_linear_min** : float, optional (deg)

angle of attack where linear portion of lift curve slope begins

**alpha_linear_max** : float, optional (deg)

angle of attack where linear portion of lift curve slope ends

**Returns**

**polar** : Polar

A new Polar object corrected for 3-D effects

**Notes**

The Du-Selig method (Du and Selig, 1998) is used to correct lift, and the Eggers method (Eggers Jr et al., 2003) is used to correct drag.

**`extrapolate`**(*cdmax*, *AR=None*, *cdmin=0.001*, *nalpha=15*)

Extrapolates force coefficients up to +/- 180 degrees using Viterna's method (Viterna and Janetzke, 1982).

**Parameters**

**cdmax** : float

maximum drag coefficient

**AR** : float, optional

aspect ratio = (rotor radius / chord_75% radius) if provided, cdmax is computed from AR

**cdmin: float, optional** :

minimum drag coefficient. used to prevent negative values that can sometimes occur with this extrapolation method

**nalpha: int, optional** :

number of points to add in each segment of Viterna method

**Returns**

**polar** : Polar

a new Polar object

**Notes**

If the current polar already supplies data beyond 90 degrees then this method cannot be used in its current form and will just return itself.

If AR is provided, then the maximum drag coefficient is estimated as

```
>>> cdmax = 1.11 + 0.018*AR
```

**unsteadyparam**(*alpha_linear_min=-5*, *alpha_linear_max=5*)
compute unsteady aero parameters used in AeroDyn input file

**Parameters**

**alpha_linear_min** : float, optional (deg)

angle of attack where linear portion of lift curve slope begins

**alpha_linear_max** : float, optional (deg)

angle of attack where linear portion of lift curve slope ends

**Returns**

**aerodynParam** : tuple of floats

(control setting, stall angle, alpha for 0 cn, cn slope, cn at stall+, cn at stall-, alpha for min CD, min(CD))

## 4.2 Airfoil Class

An Airfoil object encapsulates the aerodynamic forces/moments of an airfoil as a function of angle of attack and Reynolds number. For wind turbine analysis, this class provides capabilities to apply 3-D rotational corrections to 2-D data using the Du-Selig method (Du and Selig, 1998) for lift, and the Eggers method (Eggers Jr et al., 2003) for drag. Airfoil data can also be extrapolated to +/-180 degrees, using Viterna's method (Viterna and Janetzke, 1982). This class also adds methods to read and write AeroDyn airfoil files directly.

**Class Summary:**

**class** airfoilprep.**Airfoil**(*polars*)
Constructor

**Parameters**

**polars** : list(Polar)

list of Polar objects

**blend**(*other*, *weight*)
Blend this Airfoil with another one with the specified weighting.

**Parameters**

**other** : Airfoil

other airfoil to blend with

**weight** : float

blending parameter between 0 and 1. 0 returns self, whereas 1 returns other.

**Returns**

**obj** : Airfoil

a blended Airfoil object

**Notes**

First finds the unique Reynolds numbers. Evaluates both sets of polars at each of the Reynolds numbers, then blends at each Reynolds number.

`correction3D` (*r_over_R*, *chord_over_r*, *tsr*, *alpha_max_corr=30*, *alpha_linear_min=-5*, *alpha_linear_max=5*)

apply 3-D rotational corrections to each polar in airfoil

**Parameters**

**r_over_R** : float

radial position / rotor radius

**chord_over_r** : float

local chord / local radius

**tsr** : float

tip-speed ratio

**alpha_max_corr** : float, optional (deg)

maximum angle of attack to apply full correction

**alpha_linear_min** : float, optional (deg)

angle of attack where linear portion of lift curve slope begins

**alpha_linear_max** : float, optional (deg)

angle of attack where linear portion of lift curve slope ends

**Returns**

**airfoil** : Airfoil

airfoil with 3-D corrections

**See Also:**

`Polar.correction3D`

apply 3-D corrections for a Polar

`createDataGrid` ()

interpolate airfoil data onto uniform alpha-Re grid.

**Returns**

**alpha** : ndarray (deg)

a common set of angles of attack (union of all polars)

**Re** : ndarray

all Reynolds numbers defined in the polars

**cl** : ndarray

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** : ndarray

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

**`extrapolate`**(*cdmax*, *AR=None*, *cdmin=0.001*)

apply high alpha extensions to each polar in airfoil

**Parameters**

**cdmax** : float

maximum drag coefficient

**AR** : float, optional

blade aspect ratio (rotor radius / chord at 75% radius). if included it is used to estimate cdmax

**cdmin: minimum drag coefficient** :

**Returns**

**airfoil** : Airfoil

airfoil with +/-180 degree extensions

**See Also:**

**`Polar.extrapolate`**

extrapolate a Polar to high angles of attack

**`getPolar`**(*Re*)

Gets a Polar object for this airfoil at the specified Reynolds number.

**Parameters**

**Re** : float

Reynolds number

**Returns**

**obj** : Polar

a Polar object

**Notes**

Interpolates as necessary. If Reynolds number is larger than or smaller than the stored Polars, it returns the Polar with the closest Reynolds number.

**classmethod `initFromAerodynFile`**(*aerodynFile*)

Construct Airfoil object from AeroDyn file

**Parameters**

**aerodynFile** : str

path/name of a properly formatted Aerodyn file

**Returns**
: **obj** : Airfoil

`interpToCommonAlpha`(*alpha=None*)
Interpolates all polars to a common set of angles of attack

**Parameters**
: **alpha** : ndarray, optional

common set of angles of attack to use. If None a union of all angles of attack in the polars is used.

`writeToAerodynFile`(*filename*)
Write the airfoil section data to a file using AeroDyn input file style.

**Parameters**
: **filename** : str

name (+ relative path) of where to write file

# Bibliography

Du, Z.; Selig, M. (Jan 1998). "A 3-D Stall-Delay Model for Horizontal Axis Wind Turbine Performance Prediction." *1998 ASME Wind Energy Symposium*. AIAA-1998-21.

Eggers Jr, A.J.; Chaney, K.; Digumarthi, R. (Jan 2003). "An Assessment of Approximate Modeling of Aerodynamic Loads on the UAE Rotor." *Aerospace Sciences Meeting and Exhibit*. AIAA-2003-0868.

Viterna, L.; Janetzke, D. (September 1982). *Theoretical and Experimental Power from Large Horizontal-Axis Wind Turbines*. NASA TM-82944, National Aeronautics and Space Administration, Cleveland, OH. Lewis Research Center.