Airfoil Database

Release 1.4.3

May 07, 2021

Contents:

CHAPTER 1

Introduction

This is a software package for modelling airfoils (infinite wings). Using this package, you can generate a database of coefficients, create a set of approximating polynomials for the database, calculate coefficients at given conditions, and generate airfoil geometries. It is used extensively in the software MachUpX.

CHAPTER 2

Installation

2.1 Prerequisites

2.1.1 Getting Python

If you do not have Python installed on your machine, it can be downloaded from https://www.anaconda.com/distribution/.

2.1.2 Xfoil

AirfoilDatabase calculates section properties using a code called Xfoil, written by MIT. The Xfoil executable can be downloaded for Windows at http://web.mit.edu/drela/Public/web/xfoil/. To get this to work with AirfoilDatabase, you will need to make sure to add the Xfoil executable to your path environment variable. If you're not familiar with editing environment variables, you can find a quick tutorial here. Just follow the outlined procedure to add Xfoil to Path.

Xfoil is also available as a package on some Linux distributions and can be installed like any other package. For example on Mint:

```
$ sudo apt-get install xfoil
```

For other Unix (Linux/Mac) systems where a package is not available, compiling from the Fortran source code is required. Step by step instructions on how to do this can be found on the page *Compiling Xfoil on Unix*.

AirfoilDatabase must be able to send only the command `xfoil` to the system to run the Xfoil executable. To make sure Xfoil is properly configured, type "xfoil" into your terminal application and verify this runs the application (the interface for Xfoil is entirely terminal-based).

2.2 Getting the Source Code

You can either download the source as a ZIP file and extract the contents, or clone the AirfoilDatabase repository using Git. If your system does not already have a version of Git installed, you will not be able to use this second option unless you first download and install Git. If you are unsure, you can check by typing `git --version` into a command prompt. We recommend the second option as this makes updating the code much easier.

2.2.1 Downloading source as a ZIP file

1. Open a web browser and navigate to https://github.com/usuaero/AirfoilDatabase
2. Make sure the Branch is set to `Master`
3. Click the `Clone or download` button
4. Select `Download ZIP`
5. Extract the downloaded ZIP file to a local directory on your machine

2.2.2 Cloning the Github repository

1. From the command prompt navigate to the directory where AirfoilDatabase will be installed. Note: git will automatically create a folder within this directory called MachUpX. Keep this in mind if you do not want multiple nested folders called AirfoilDatabase.
2. Execute `git clone https://github.com/usuaero/AirfoilDatabase`

2.3 Installing

Navigate to the root (AirfoilDatabase/) directory and execute

```
pip install .
```

Please note that any time you update the source code, the above command will need to be run.

CHAPTER 3

Compiling Xfoil on Unix (Linux and Mac) Systems

The purpose of this document is to outline how to compile Xfoil on a Unix machine The system I am using to do this is Linux Mint 19.2 on an HP Z230 desktop. I cannot guarantee this process will work on your setup. It is possible to compile Xfoil using only the instructions in the READMEs (that's what I did), but hopefully this will be more clear.

3.1 Compilers

I use the GCC (GNU Compiler Collection). On Debian systems, this can be installed with

```
$ sudo apt-get install gcc
```

3.2 Getting the Source Code

The source code for Xfoil can be downloaded from http://web.mit.edu/drela/Public/web/xfoil/. Download the .tar.gz file which says it is for Unix. Open this with an archive manager and extract the contents into your home/ directory. Note that .tar.gz is a zip of a zip. Make sure you're extracting the contents of the .tar zip into your home/directory.

3.3 Build Sequence

At this point, we will simply be following the instructions given in the READMEs. Open a terminal window and navigate to the Xfoil/ directory.

3.3.1 Orr-Sommerfeld Database

Navigate to the orrs/ directory

```
$ cd orrs
```

Determine the absolute path to this directory by typing

```
$ pwd
```

Write this path down. Navigate to the src/ directory and open the osmap.f file using your favorite command-line text editor (mine's Vim).

```
$ cd src
$ vim osmap.f
```

We need to let the compiler know exactly where the osmap.dat file is. Around line 100 of osmap.f, edit the following line to contain the absolute path you got with the `pwd` command

```
DATA OSFILE / '/var/local/codes/orrs/osmapDP.dat' /
```

When done it should look something like this

```
DATA OSFILE / '/home/cory/Xfoil/orrs/osmapDP.dat' /
```

Around line 75, there are options for setting the precision (either double or single). We want to ensure double precision here, so make sure the following lines are not commented

```
    REAL RLSP, WLSP, HLSP,
    &     RINCR, WINCR, RL, WL, HL,
    &     A, AR, AW, AH, ARW, ARH, AWH, ARWH
```

Save and exit. Navigate to the Xfoil/bin directory and open the Makefile

```
$ cd ../bin
$ vim Makefile_DP
```

Lines 14-17 contain the compiler flags. You'll need to edit these to fit your compiler. I made changes to the first two lines to be

```
FC = gfortran
FLG = -O -f-default-real-8
```

Save and exit Makefile_DP.

Now it's time to compile the database! Type

```
$ make -f Makefile_DP osgen
$ make -f Makefile_DP osmap.o
$ cd ..
$ bin/osgen osmaps_ns.lst
```

You may get a couple warnings; that's okay.

3.3.2 Plot Library

Navigate to the plotlib/ directory

```
$ cd ../../plotlib
```

Open Makefile

```
$ vim Makefile
```

Edit line 72 to use the correct compiler

```
FC = gfortran
```

You will also need to set double precision here. Uncomment line 77 and change it to be

```
DP = -f-default-real-8
```

Save and exit. Do the same thing for config.make. Specify the compiler on line 54 and the double precision option on line 58. Save and exit. Now you can compile the plot library

```
$ make libPlt.a
```

3.3.3 Binaries

Navigate to the bin/ directory and open the Makefile

```
$ cd ../bin
$ vim Makefile
```

You will need to make changes to lines 102, 111-116 in the Makefile. Line 102 should be

```
FC = gfortran
```

Lines 111-116 should be

```
FFLAGS = -O -fdefault-real-8 -B
FFLOPT = -O -fdefault-real-8 -B
PLTOBJ = ../plotlib/libPlt.a

FFLAGS = -O -fdefault-real-8 -ftrapv -fdec
FFLOPT = -O -fdefault-real-8 -ftrapv -fdec
```

There are subtle differences between the above and what is originally in the file, so be careful. If you make mistakes, the compiler will usually let you know what it didn't understand. Save and exit.

Compile xfoil

```
$ make xfoil
```

You now need to edit a bit of the source code. Open the pplot.f file

```
$ vim ../src/pplot.f
```

And change line 57 to be

```
IF(LERR.ne.0) THEN
```

Save and exit. Now you can compile the rest of the files

```
$ make pplot
$ make pxplot
```

3.3.4 Running Xfoil

The Xfoil executable is located in the bin/ directory. Type

```
$ bin/xfoil
```

CHAPTER 4

Input Structure

As an initializer, the Airfoil class takes a JSON object (file) or Python dictionary. This object describes the geometry of the airfoil, the trailing flap (if present), and how its aerodynamics are to be predicted, whether using linear predictions, a database, or polynomial fits. The main purpose of this input structure is to allow native interface with MachUpX.

Generating or exporting a database cannot be specified through the input; this is done using the Airfoil class methods (see *Airfoil Class* for more information). If you wish to use the Airfoil class to generate a database, all that is required to input are the geometry specifications.

The following are keys which can be specified in the airfoil JSON object or dictionary.

"type" : string

The type of prediction to be used in determining aerodynamic properties. Can be "linear", "database", "poly_fit", or "functional". "linear" airfoils are defined by a linearization of airfoil parameters about the zero-lift angle of attack, with the exception of CD being a quadratic function of CL. All appropriate coefficients and derivatives should then be given. UNITS MAY NOT BE SPECIFIED BY THE USER FOR ANY AIRFOIL PARAMETERS. THESE VALUES MUST BE SPECIFIED IN THE UNITS GIVEN.

For the linear class of airfoils, we make corrections to section data due to flap deflection based on Phillips' corrections (see Phillips, *Mechanics of Flight*, 2010, pp. 39-46).

"database" airfoils are defined by an array of data, that is aerodynamic coefficients, determined at various angles of attack, Reynolds numbers, Mach numbers, and flap deflections. Such a database must be generated using this software. More information, see generate_database(), export_database(), and import_database() in *Airfoil Class*.

"poly_fit" airfoils are defined by polynomial functions of aerodynamic coefficients as a function of angle of attack, Reynolds number, Mach number, and flap deflection. These fits must be generated using this software. More information, see generate_polynomial_fits(), export_polynomial_fits(), and import_polynomial_fits() in *Airfoil Class*.

"functional" airfoils are defined by user-defined functions. This type can only be used if the airfoil input is declared as a dictionary within a Python script. The user defines three functions which give the coefficients of lift, drag, and moment as a function of various parameters.

"input_file" : str, optional

File to read database or fit data from. The file specified here should match the "type" of the airfoil specified (e.g. if "type" is "database", this should be the database file).

A note about database files: if the database was generated outside of AirfoilDatabase, be sure the angle of attack is stored in radians, not degrees.

"geometry" : dict, optional

Describes the geometry of the airfoil.

"outline_points" : str or array, optional

Path to a file containing airfoil outline points or array of outline points. The first column contains x-coordinates and the second column contains y-coordinates, where the x-axis originates at the leading edge and points back along the chord line and the y-axis points up. If a file, it should be comma-delimited or space-delimited. The points should begin at the trailing edge, wrap around the leading edge, and then end at the trailing edge. The airfoil will automatically determine the ordering of the points (whether top first or bottom first). Cannot be specified along with "NACA".

Experience has shown that airfoil outlines ending in a blunt tip at a closed trailing edge are poorly handled by AirfoilDatabase. We recommend having either an open trailing edge, or having the trailing edge taper to a smooth point.

"NACA" : str, optional

NACA designation for the airfoil. If given, the airfoil will automatically generate outline points using the NACA equations. Can only be NACA 4-digit series. Cannot be specified along with "outline_points".

"NACA_closed_te" : bool, optional

Whether to use the NACA 4-digit equations which force the trailing edge to be closed. Defaults to False.

"max_camber" : float, optional

Maximum camber of the airfoil as a fraction of the chord. Can be specified if "outline_points" and "NACA" are not specified. This and "max_thickness" affect the corrections for sweep applied in MachUpX. This has no effect on computations made within the Airfoil class. Defaults to 0.0.

"max_thickness" : float, optional

Maximum thickness of the airfoil as a fraction of the chord. Can be specified if "outline_points" and "NACA" are not specified. This and "max_camber" affect the corrections for sweep applied in MachUpX. This has no effect on computations made within the Airfoil class. Defaults to 0.0.

"trailing_flap_type" : str, optional

The shape of the flap for this airfoil. Can be "linear" or "parabolic". Defaults to "linear".

"trailing_flap_hinge_height" : float, optional

y location of the flap hinge, nondimensionalized by the chord length. Defaults to lie on the camber line.

The following keys are pertinent to the "linear" type.

"aL0" : float, optional

The zero-lift angle of attack in radians. Defaults to 0.0. Only for "linear".

"CLa" : float, optional

The lift slope in radians^-1. Defaults to 2pi Only for “linear”.

“CmL0” : float, optional

The quarter-chord moment coefficient at the zero-lift angle of attack. Defaults to 0.0. Only for “linear”.

“Cma” : float, optional

The moment slope in radians^-1. Defaults to 0.0. Only for “linear”.

“CD0” : float, optional

Constant coefficient in the quadratic fit of the CD/CL curve. Defaults to 0.0. Only for “linear”.

“CD1” : float, optional

Linear coefficient in the quadratic fit of the CD/CL curve. Defaults to 0.0. Only for “linear”.

“CD2” : float, optional

Quadratic coefficient in the quadratic fit of the CD/CL curve. Defaults to 0.0. Only for “linear”.

“CL_max” : float, optional

Maximum lift coefficient. Defaults to infinity. Only for “linear”.

The following keys are pertinent to the “functional” type.

“CL” : func

Function handle returning the coefficient of lift based on the given parameters. The function declaration should look like

def user_CL(**kwargs): … return CL

Kwargs may be given as floats or as 1-D numpy arrays. This function must be able to handle both (not difficult to implement). The following kwargs can be given to this function:

“alpha” : float

Angle of attack in radians.

“Rey” : float

Reynolds number.

“Mach” : float

Mach number.

“trailing_flap_deflection” : float

Trailing flap deflection in radians.

“trailing_flap_fraction” : float

Trailing flap fraction of the chord length.

This function should have defined defaults for each parameter in case they are not given.

“CD” : func

Function handle returning the coefficient of drag based on the given parameters. Same as “CL”.

“Cm” : func

Function handle returning the moment coefficient based on the given parameters. Same as “CL”.

CHAPTER 5

Airfoil Class

All functionality is wrapped in the Airfoil class. The Airfoil class can create geometries, create a database using Xfoil, export and import databases, curve fit a database, import and export curve fits, and calculate section properties. For an example of using the Airfoil class to generate and export a database, see the examples/ directory.

The following member functions are available:

class airfoil_db.**Airfoil**(*name*, *airfoil_input*, ***kwargs*)

A class defining an airfoil. If the airfoil geometry is defined using outline points, then when this class is initialized, a solver will be automatically run to determine the camber line and thickness distribution of the airfoil. The parameters "camber_relaxation", "le_loc", and "camber_termination_tol" then have bearing on this solver.

When using the member methods get_CL, get_CD, get_Cm, get_CLa, get_CLM, get_CLRe, and get_aL0, the default parameters are dependent upon the type of airfoil.

For all airfoil types except 'functional', 'alpha', 'trailing_flap_deflection', and 'trailing_flap_fraction' default to 0.0.

For 'functional' airfoils, the defaults are specified by the user.

For 'linear' airfoils, 'Mach' and 'Rey' have no effect on computations.

For 'database' airfoils, 'Mach' and 'Rey' default to the average value in the database, if the database is dependent on that variable. Otherwise, they default to the value specified as constant when the database was generated.

For 'poly_fit' airfoils, 'Mach' and 'Rey' default to the values given in the fit file. If export_polynomial_fits() is used, these values are the same as those for the 'database' type.

Parameters

- **name** (*str*) – Name of the airfoil.
- **airfoil_input** (*dict or str*) – Dictionary or path to JSON object describing the airfoil.
- **verbose** (*bool, optional*) – Whether to display information on the progress of parameterizing the geometry for an airfoil defined by a set of outline points. Defaults to False.

- **camber_relaxation** (*float, optional*) – A value between 0.0 and 1.0 that defines how much of the update at each iteration of the camber line solver should be accepted. Helpful for some poorly behaved cases. Defaults to 1.0 (full update).
- **le_loc** (*list, optional*) – Gives the location of the leading edge relative to the given points for an airfoil defined by a set of outline points. If this is given, the camber line will be forced to intersect this point. THIS POINT SHOULD LIE ON THE AIRFOIL OUTLINE. If not given, the camber line solver will try to iteratively find the leading edge (the point where the camber line intersects the front of the profile).
- **camber_termination_tol** (*float, optional*) – The tolerance below which the maximum approximate error in the camber line estimate must fall in order for the camber line solver to terminate. Defaults to 1e-10.
- **max_iterations** (*int, optional*) – Maximum number of iterations for the camber line solver. Defaults to 100.

check_against_NACA(*naca_des*)

Checks the error in the camber and thickness against that predicted by the NACA equations. This is recommended as a check for the user if unusual geometries are being imported. Checks against the open trailing edge formulation of the NACA equations.

Parameters **naca_des** (*str*) – NACA designation of the airfoil to compare against as a string. May only be 4-digit series.

export_database(***kwargs*)

Exports the database generated by generate_database().

Parameters **filename** (*str*) – File to export the database to.

export_linear_model(***kwargs*)

Exports the linear coefficients used to predict the behavior of the airfoil.

Parameters **filename** (*str*) – JSON file to export the model data to.

export_polynomial_fits(***kwargs*)

Save the polynomial fit to a JSON object.

Parameters

- **filename** (*str*) – JSON object to write polynomial fit data to.
- **write_limits** (*bool, optional*) – Whether to limit the polynomial fits based on the original range of data. Defaults to True.

generate_database(***kwargs*)

Makes calls to Xfoil to calculate CL, CD, and Cm as a function of each given degree of freedom.

Parameters

- **degrees_of_freedom** (*dict*) – A dict specifying which degrees of freedom the database should perturb. Allowable degrees of freedom are "alpha", "Rey", "Mach", "trailing_flap_deflection", and "trailing_flap_fraction".

  Each key should be one of these degrees of freedom. To specify a range for the degree of freedom, a dictionary with the following keys should be given:

  "range" [list] The lower and upper limits for this DOF.

  "steps" [int] The number of points in the range to interrogate.

  "index" [int] Index of the column for this degree of freedom in the database.

"log_step" [bool, optional] Whether the steps in this dof should be spaced linearly (False) or logarithmically (True). Defaults to False.

If instead of perturbing a variable, you want the database to be evaluated at a constant value of that variable, a float should be given instead of a dictionary. That variable will then not be considered a degree of freedom of the database and will not appear in the database.

An example is shown below:

dofs = {

"alpha" [{] "range" : [math.radians(-15.0), math.radians(15.0)], "steps" : 21, "index" : 1

}, "Rey" : 2900000.0, "trailing_flap_deflection" : {

"range" : [math.radians(-20.0), math.radians(20.0)], "steps" : 1, "index" : 0

}, "trailing_flap_fraction" : 0.25

}

The above input will run the airfoil through angles of attack from -15 to 15 degrees at a Reynolds number of 2900000.0 and through flap deflections from -20 to 20 degrees with a chord fraction of 25%.

If a float, the degree of freedom is assumed to be constant at that value.

If not specified, each degree of freedom defaults to the following:

"alpha" : 0.0

"Rey" : 1000000.0

"Mach" : 0.0

"trailing_flap_deflection" : 0.0

"trailing_flap_fraction" : 0.0

Please note that all angular degreees of freedom are in radians, rather than degrees.

Currently, the number of steps in Reynolds number multiplied by the number of steps in Mach number may not exceed 12. This is due to internal limitations in Xfoil. However, due to the weak dependence of airfoil properties on Reynolds number, we do not expect this to be a great hinderance.

- **N** (*int, optional*) – Number of panel nodes for Xfoil to use. Defaults to 200.
- **max_iter** (*int, optional*) – Maximum iterations for Xfoil. Defaults to 100.
- **x_trip** (*float or list, optional*) – x location, non-dimensionalized by the chord length, of the boundary layer trip position. This is specified for the top and bottom of the airfoil. If a float, the value is the same for the top and the bottom. If a list, the first list element is the top trip location and the second list element is the bottom trip location. Defaults to 1.0 for both.
- **N_crit** (*float or list, optional*) – Critical amplification exponent for the boundary layer in Xfoil. If a float, the value is the same for the top and the bottom. If a list, the first list element is for the top and the second list element is for the bottom. Defaults to 9.0 for both.
- **update_type** (*bool, optional*) – Whether to update the airfoil to use the newly computed database for calculations. Defaults to True.

- **show_xfoil_output** (*bool, optional*) – Display whatever Xfoil prints out. Defaults to False.
- **show_xfoil_plots** (*bool, optional*) – Display Xfoil plots. Defaults to True.
- **resize_xfoil_window** (*float, optional*) – resizes the xfoil window to screen size fraction. Xfoil defaults to 0.8 window/screen size. This variable defaults to None. Has no effect if show_xfoil_plots is False.
- **CD_type** (*str, optional*) – Which drag coefficient to read in. May be 'total', 'friction', or 'pressure'. Defaults to 'total'.
- **verbose** (*bool, optional*) – Defaults to True

generate_linear_model (***kwargs*)

Creates a linearized model of the airfoil coefficients in alpha. Pulls from the database or poly_fit information to generate this model. Cannot be used on a type "linear" airfoil.

linear_limits [list, optional] Limits in alpha for which the behavior of the airfoil can be considered linear. If not given, the user will be prompted to graphically select this region. Given in degrees.

update_type [bool, optional] Whether to change the type of the airfoil to "linear" once the model is determined. Defaults to True.

plot_model [bool, optional] Whether to display a polar of the linear region determined. Defaults to False.

Rey [float, optional] Reynolds number at which to evaluate the model.

Mach [float, optional] Mach number at which to evaluate the model.

generate_polynomial_fit (***kwargs*)

Generates a set of multivariable polynomials using least-squares regression to approximate the database. Note: This airfoil must have a database already for fits to be created.

Parameters

- **CL_degrees** (*dict or str, optional*) – If dict, order of fit polynomial for the coefficient of lift for each degree of freedom, formatted as

  { "<DOF1_NAME>" : <ORDER>, "<DOF2_NAME>" : <ORDER>, ...

  }

  Orders must be integers. Defaults to 1 for any not specified.

  Can also be specified as "auto". In this case, the fit degrees will be determined automatically using the method described in Morelli, "Global Nonlinear Aerodynamic Modeling Using Multivariate Orthogonal Functions," Journal of Aircraft, 1995. The algorithm tried to minimize the RMS error between the data and the prediction while also minimizing the degree of the fit. This will automatically determine the fit order for each degree of freedom.
- **CD_degrees** (*dict, optional*) – Same as CL_degrees.
- **Cm_degrees** (*dict, optional*) – Same as CL_degrees.
- **CL_kwargs** (*dict, optional*) – keyword arguments sent to the CL polynomial fit function

  When CL_degrees is specified as "auto" then CL_kwargs can be

  "max_order" [int, optional] gives the max order of polynomial for any one of the independent varialbes to try. Defaults to 6.

"tol" [float, optional] Gives the cut-off value for any polynomial coefficient to not be included in the final results. If a coefficient has an absolute value below tol, it won't be included. Defaults to 1e-12.

"sigma" [float, optional] value used to determine the trade off between how good of a fit to perform and how many terms to keep. Defaults to None, which causes the function to calculate sigma automatically using the mean squared of the difference of the independent variable values with respect to the mean independent variable value of the dataset

"sigma_multiplier" [float, optional] term multiplied onto sigma to change it's value. Allows using a multiple of the automatically determined sigma value. Defaults to 1.

Otherwise CL_kwargs could be

"interaction" [boolean, optional] value with default set to True. This variable determines whether or not interaction terms are included in the fit function. If set to True, interaction terms up the max order for each independent variable are included, i.e. if Nvec = [3,2] then the highest interaction term included is x_1^3*x_2^2. Specific interaction terms can be omitted using the constraints input

"sym" [list, optional] Defaults to an empty list. If used, the length should be V and each element should contain a boolean, True or False. The ith element determines if the ith independent variable is symmetric either even or odd, which is determined by the order given in Nvec. This will also remove the cooresponding interaction terms if they are enabled.

"sym_same" [list, optional] Defaults as an empty list. If used, the entries in the list should be tuples with two integers. The integers represent the independent variables that the "same" symmetry condition will be applied. The "same" symmetry ensures all interaction terms with exponents of the two independent variables that are either odd-odd or even-even to be forced to zero

"sym_diff" [= list, optional] Defaults as an empty list. Similar to "sym_same" except it enforces the "diff" symmetry condition which ensures all interaction terms with exponents of the two independent variables that are either odd-even or even-odd to be forced to zero

"zeroConstraints" [list, optional] Defaults as an empty list. Entries in the list contain integer tuples of length V. The integer values represent the powers of the independent variables whose coefficient will be forced to 0 before the best fit calculations are performed, allowing the user to omit specific interaction terms or regular polynomial terms

"constraints" [list, optional] Defaults to an empty list. Entries in the list contain tuples of length 2. The first entry is a list of integers that represent the powers of the independent variables whose coefficient will then be forced to be equal to the second entry in the tuple, which should be a float.

"percent" [boolean, optional] Default set to False. When set to True the least squares is performed on the percent error squared. This option should not be used if y contains any zero or near zero values, as this might cause a divide by zero error.

"weighting" [function, optional] Defaults to None. If given, weighting

should be a function that takes as arguments x, y, and p where x and y are the independent and dependent variables defined above and p is the index representing a certain data point. weighting should return a 'weighting factor' that determines how important that datapoint is. Returning a '1' weights the datapoint normally.

- **CD_kwargs** (*dict, optional*) – Same as CL_kwargs
- **Cm_kwargs** (*dict, optional*) – Same as CL_kwargs
- **update_type** (*bool, optional*) – Whether to update the airfoil to use the newly computed polynomial fits for calculations. Defaults to True.
- **verbose** (*bool, optional*) –

get_CD(***kwargs*)

Returns the coefficient of drag. note: all parameters can be given as numpy arrays, in which case a numpy array of the coefficient will be returned. to do this, all parameter arrays must have only one dimension and must have the same length.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.

Returns Drag coefficient

Return type float or ndarray

get_CL(***kwargs*)

Returns the coefficient of lift. Note: all parameters can be given as numpy arrays, in which case a numpy array of the coefficient will be returned. To do this, all parameter arrays must have only one dimension and must have the same length.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.

Returns Lift coefficient

Return type float or ndarray

get_CLM(***kwargs*)

Returns the lift slope with respect to Mach number using a forward-difference approximation. Simply returns 0 for a type 'linear' airfoil.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.
- **dx** (*float*) – Step size for finite-difference equation. Defaults to 0.05.

Returns Lift slope with respect to Mach number

Return type float or ndarray

get_CLRe (***kwargs*)

Returns the lift slope with respect to Reynolds number using a central-difference approximation. Simply returns 0 for a type 'linear' airfoil.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.
- **dx** (*float*) – Step size for finite-difference equation. Defaults to 1000.

Returns Lift slope with respect to Reynolds number

Return type float or ndarray

get_CLa (***kwargs*)

Returns the lift slope using a central-difference approximation.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.
- **dx** (*float, optional*) – Step size for finite-difference equation. Defaults to 0.001 radians.

Returns Lift slope

Return type float or ndarray

get_Cm(***kwargs*)
Returns the moment coefficient. note: all parameters can be given as numpy arrays, in which case a numpy array of the coefficient will be returned. to do this, all parameter arrays must have only one dimension and must have the same length.

Parameters

- **alpha** (*float, optional*) – Angle of attack in radians. Defaults to 0.0.
- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_moment_deriv** (*float or ndarray, optional*) – Change in section moment with respect to trailing flap deflection. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.

Returns Moment coefficient

Return type float or ndarray

get_aL0(***kwargs*)
Returns the zero-lift angle of attack, taking flap deflection into account.

Parameters

- **Rey** (*float, optional*) – Reynolds number.
- **Mach** (*float, optional*) – Mach number.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians. Defaults to 0.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord length. Defaults to 0.

Returns Zero-lift angle of attack

Return type float

get_camber(*x*)
Returns the y coordinate(s) of the camber line at the specified x location(s).

Parameters **x** (*float or ndarray*) – x location(s) at which to get the thickness.

Returns Camber line y coordinate(s) as a percentage of the chord.

Return type float or ndarray

get_max_camber()
Returns the maximum camber of the airfoil, divided by the chord length.

get_max_thickness()
Returns the maximum thickness of the airfoil, divided by the chord length.

get_outline_points(***kwargs*)
Returns an array of outline points showing the geometry of the airfoil.

Parameters

- **N** (*int, optional*) – The number of outline points to return. This function will not always return exactly this many but never more. Defaults to 200.

- **cluster** (*bool, optional*) – Whether to cluster points about the leading and trailing edges. Defaults to True.
- **trailing_flap_deflection** (*float, optional*) – Trailing flap deflection in radians (positive down). Defaults to zero.
- **trailing_flap_fraction** (*float, optional*) – Trailing flap fraction of the chord. Defaults to zero.
- **export** (*str, optional*) – If specified, the outline points will be saved to a file. Defaults to no file.
- **top_first** (*bool, optional*) – The order of the coordinates when exported. Defaults to going from the trailing edge along the top and the around to the bottom.
- **close_te** (*bool, optional*) – Whether the top and bottom trailing edge points should be forced to be equal. Defaults to False
- **plot** (*bool, optional*) – Whether a plot of the outline points should be displayed once computed. Defaults to False.
- **original_points** (*bool, optional*) – Whether this should simply return the original inputted points. If set to True, all other kwargs relating to the geometry will be ignored. Defaults to False.

Returns Outline points in airfoil coordinates.

Return type ndarray

get_thickness(*x*)

Returns the thickness normal to the camber line at the specified x location(s).

Parameters **x** (*float or ndarray*) – x location(s) at which to get the thickness.

Returns Thickness as a percentage of the chord.

Return type float or ndarray

import_database(***kwargs*)

Imports the specified database. Please note that if you have generated your own database not using AirfoilDatabase, angle of attack should be stored in radians, rather than degrees.

Parameters

- **filename** (*str*) – File to import the database from
- **update_type** (*bool, optional*) – Whether to update the airfoil to use the newly imported database for calculations. Defaults to True.

import_polynomial_fits(***kwargs*)

Read in polynomial fit data from a JSON object.

Parameters

- **filename** (*str*) – JSON object to read polynomial fit data from.
- **update_type** (*bool, optional*) – Whether to update the airfoil to use the newly imported polynomial fits for calculations. Defaults to True.

read_pacc_file(*filename*, *CD_type='total'*)

Reads in and formats an Xfoil polar accumulation file.

Parameters

- **filename** (*str*) – File to read in.

- **CD_type** (*str, optional*) – Which drag coefficient to read in. May be 'total', 'friction', or 'pressure'. Defaults to 'total'.

Returns

- **alpha** (*list*) – List of angles of attack from the accumulation polar.
- **CL** (*list*) – Coefficient of lift at each alpha.
- **CD** (*list*) – Coefficient of drag at each alpha.
- **Cm** (*list*) – Moment coefficient at each alpha.
- **Re** (*float*) – Reynolds number for the polar.
- **M** (*float*) – Mach number for the polar.

run_xfoil (***kwargs*)

Calls Xfoil and extracts the aerodynamic coefficients at the given state.

Parameters

- **alpha** (*float or list of float*) – Angle(s) of attack to calculate the coefficients at in radians. Defaults to 0.0.
- **Rey** (*float or list of float*) – Reynolds number(s) to calculate the coefficients at. Defaults to 1000000.
- **Mach** (*float or list of float*) – Mach number(s) to calculate the coefficients at. Defaults to 0.0.
- **trailing_flap_deflection** (*float or list of float*) – Flap deflection(s) to calculate the coefficients at in radians. Defaults to 0.0.
- **trailing_flap_fraction** (*float or list of float*) – Flap fraction(s) to calculate the coefficients at in radians. Defaults to 0.0.
- **N** (*int, optional*) – Number of panels for Xfoil to use. Defaults to 200.
- **max_iter** (*int, optional*) – Maximum iterations for Xfoil. Defaults to 100.
- **x_trip** (*float or list, optional*) – x location, non-dimensionalized by the chord length, of the boundary layer trip position. This is specified for the top and bottom of the airfoil. If a float, the value is the same for the top and the bottom. If a list, the first list element is the top trip location and the second list element is the bottom trip location. Defaults to 1.0 for both.
- **xycm** (*list, optional*) – x-y coordinates, non-dimensionalized by the chord length, of the reference point for determining the moment coefficient. Defaults to the quater-chord.
- **N_crit** (*float or list, optional*) – Critical amplification exponent for the boundary layer in Xfoil. If a float, the value is the same for the top and the bottom. If a list, the first list element is for the top and the second list element is for the bottom. Defaults to 9.0 for both.
- **show_xfoil_output** (*bool, optional*) – Display whatever Xfoil outputs from the command line interface. Defaults to False.
- **show_xfoil_plots** (*bool, optional*) – Display Xfoil plots. Defaults to True.
- **resize_xfoil_window** (*float, optional*) – resizes the xfoil window to screen size fraction. Xfoil defaults to 0.8 window/screen size. This variable defaults to None. Has no effect if show_xfoil_plots is False.

- **CD_type** (*str, optional*) – Which drag coefficient to read in. May be 'total', 'friction', or 'pressure'. Defaults to 'total'.
- **verbose** (*bool, optional*) –

Returns

- **CL** (*ndarray*) – Coefficient of lift. First dimension will match the length of alpha, second will match Rey, etc.
- **CD** (*ndarray*) – Coefficient of drag. Dimensions same as CL.
- **Cm** (*ndarray*) – Moment coefficient. Dimensions same as CL.

set_err_state(***kwargs*)

Sets the error state for the airfoil. Each may be specified as 'raise' or 'ignore'.

Parameters **poly_fit_bounds** (*str, optional*) – How to handle PolyFitBoundsError. Defaults to 'raise'.

set_type(*database_type*)

Determines how the aerodynamic coefficients will be calculated.

Parameters **database_type** (*str*) – "linear", "functional", "database", or "poly_fit". Airfoil will automatically check if it has the necessary to perform the given type of computation and throw a warning if it does not.

set_verbosity(*verbosity*)

Sets the verbosity of the airfoil.

CHAPTER 6

Error Handling

The following custom errors are defined in AirfoilDatabase

class airfoil_db.**DatabaseBoundsError**(*airfoil*, *exception_indices*, *inputs_dict*)

An exception thrown when the inputs to the airfoil database fall outside the database bounds.

airfoil

The name of the airfoil for which this exception occurred.

Type str

inputs_dict

The arguments passed to the airfoil.

Type dict

exception_indices

The indices at which the arguments fell outside the database bounds.

Type list

message

A message about the error.

Type str

Python Module Index

a

Index