
PyAero Documentation

Release v'2.1.6'

Andreas Ennemoser

Jun 15, 2023

Table of Contents

1	Introduction	5
2	Quick start guide	7
3	User Interface	9
3.1	Overview	9
3.2	Menus	9
3.3	Toolbar	13
3.4	Toolbox Functions	13
3.5	Tabbed Views	13
3.6	Zooming, Panning	14
3.7	Keyboard shortcuts	14
4	Loading Airfoils	17
4.1	Load via menu <i>Open</i>	17
4.2	Load via the inline file browser	17
4.3	Load via the <i>Toolbar</i>	18
4.4	Load a predefined airfoil	18
4.5	Load via drag and drop	19
5	Splining and Refining Airfoil Contours	21
6	Trailing Edge	27
7	Making Meshes	29
8	Batch mode (run from command line)	35
9	CFD_boundary_conditions	37
10	Aerodynamics (panel code)	39
11	Contour Analysis	41
12	Settings	43
13	Modify and extend GUI Functionality	45

14 Dependencies	47
15 License	49

Important: The documentation is not yet finished, but should be good enough to make meshes.

PyAero is an airfoil contour analysis and CFD meshing tool written in Python. PyAero is open-source and distributed under the MIT license, see [LICENSE](#).

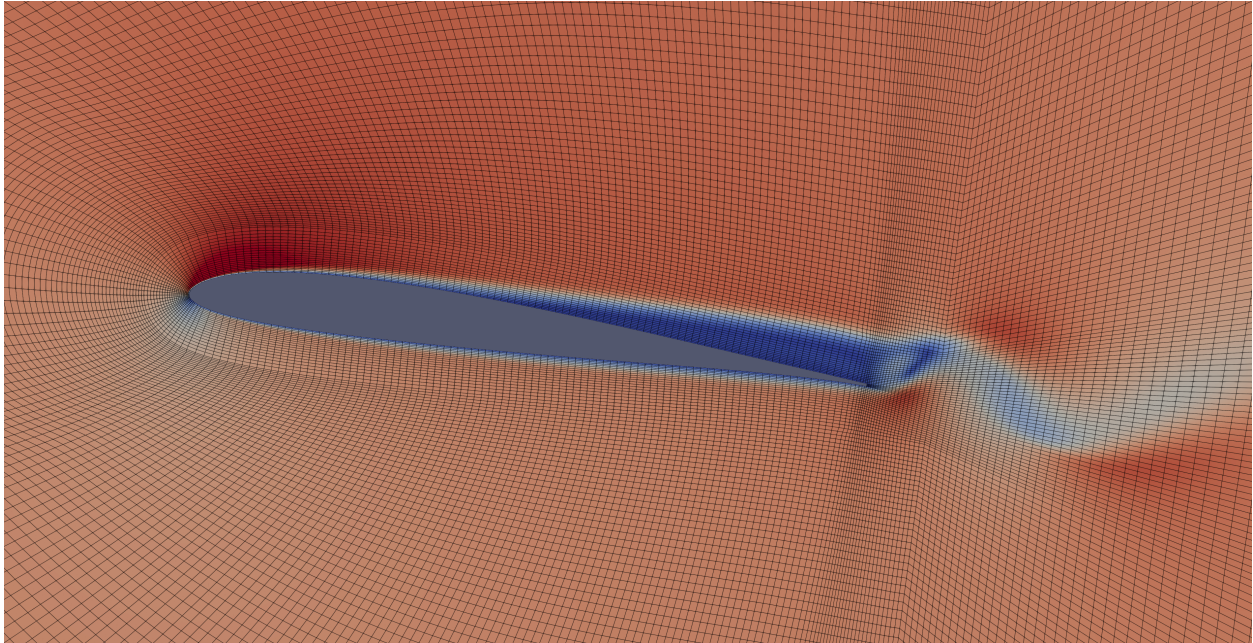


Fig. 1: PyAero generated mesh (Solver: [SU2](#), Visualization: [ParaView](#))

Above analysis result was obtained using the CFD code AVL-FIRE. It is an unsteady laminar 3D calculation of the RG14 airfoil. The mesh was thickened with several layers in spanwise direction in order to allow for turbulent fluctuations in all three dimensions. The calculation result shown is based on pure laminar settings (i.e. no turbulence model switched on). Later (result not shown) a LES calculation using the Kobayashi SGS model was done. With the applied mesh resolution both approaches lead to quite similar results.

Table 1: Aerodynamic coefficients, RG14 airfoil, $Re=330000$, $AOA\ 2^\circ$, 20 million cells

Model	Drag	Lift
Laminar	0.0079	0.371
LES	0.0078	0.362

Features

- Load and display airfoil contour files
- Airfoil splining and refining
 - Prepare contour for meshing
 - Splining is done to get a smooth contour and sufficient contour points
 - Refining allows to improve leading and trailing edge resolution
- Airfoil contour analysis
 - Analyze gradient, curvature, and curvature circle at the leading edge, i.e. leading edge radius

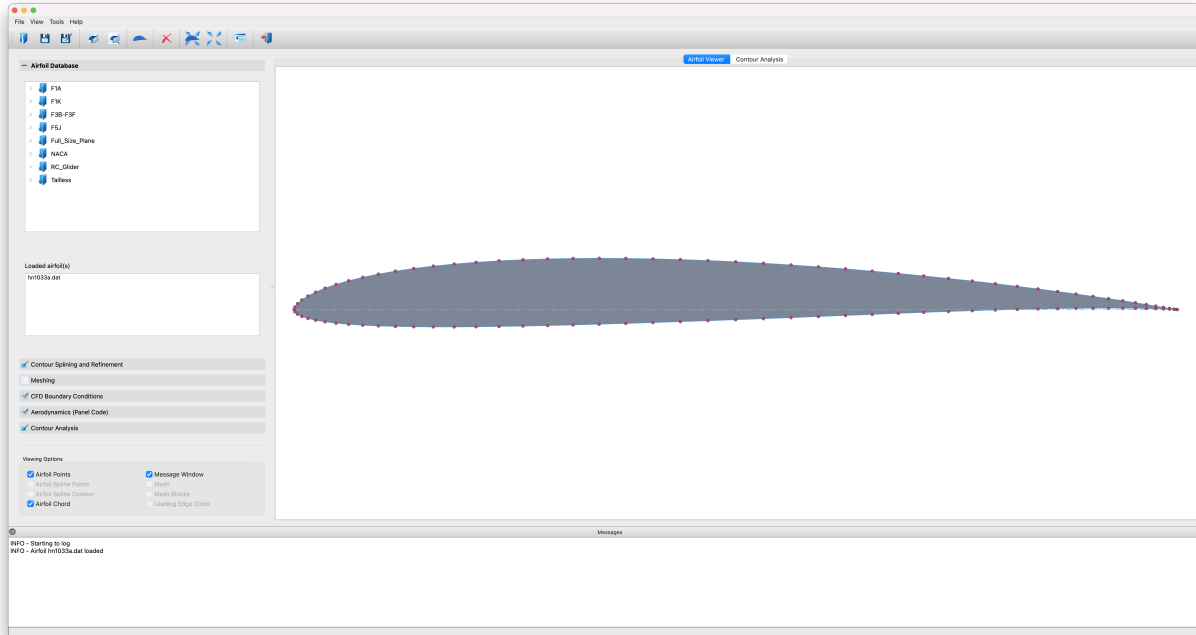


Fig. 2: PyAero user interface at a glance

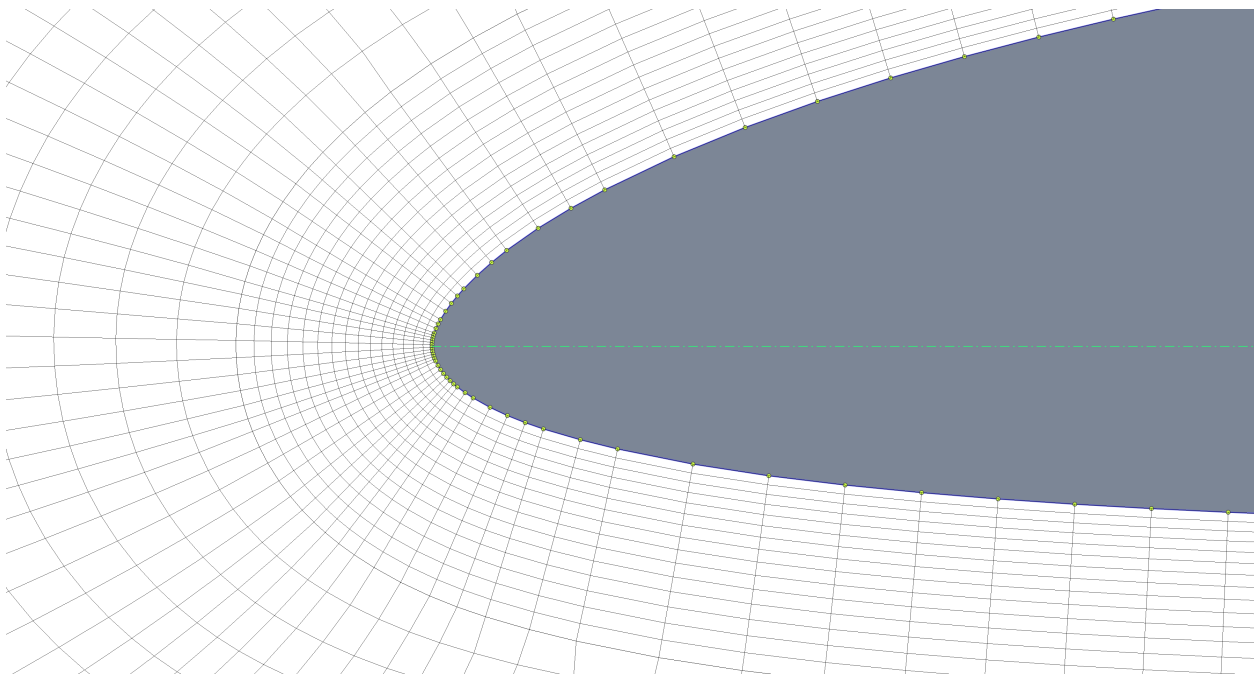


Fig. 3: Example mesh around HN1033 airfoil - Leading Edge

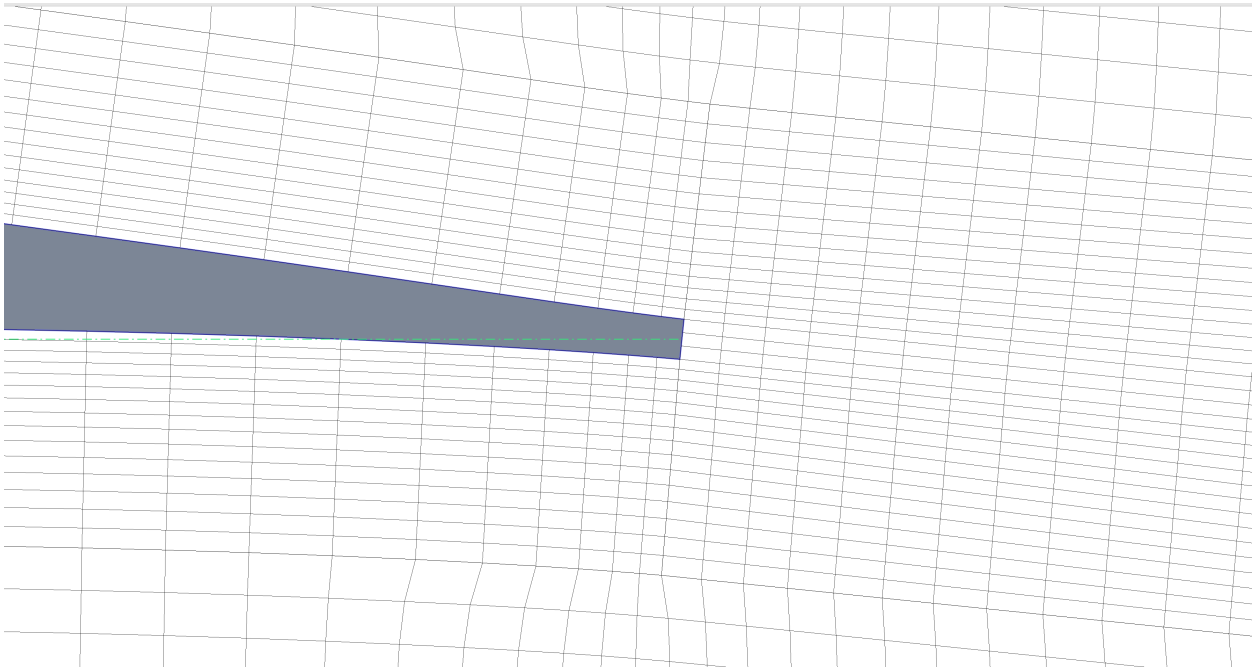


Fig. 4: Example mesh around HN1033 airfoil - Trailing Edge (with finite thickness)

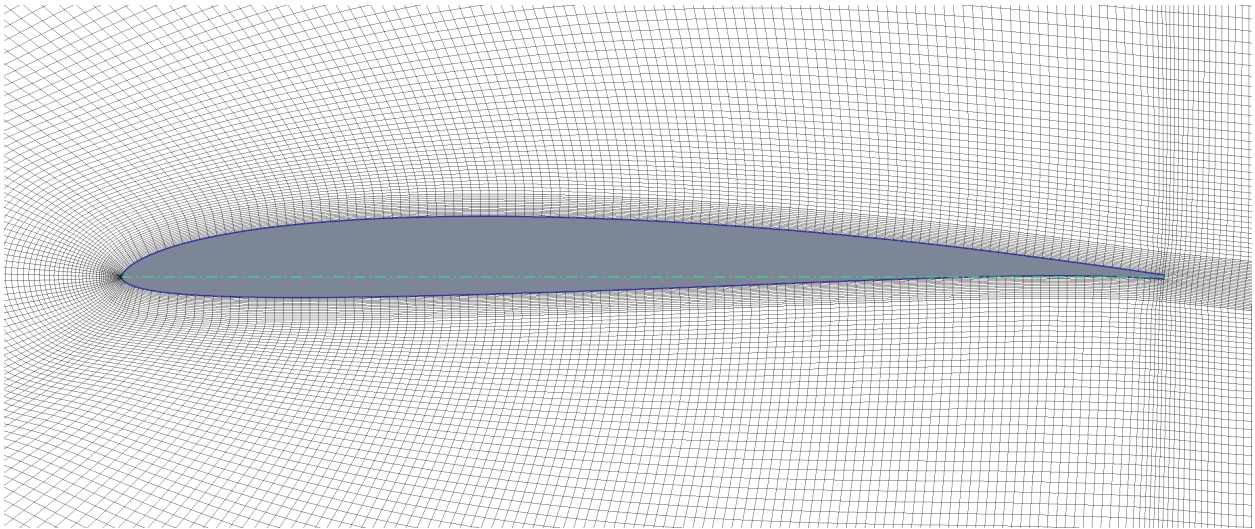


Fig. 5: Example mesh around HN1033 airfoil

Fig. 6: Example calculation result

- Trailing edge generation
 - Specification of the trailing edge thickness (blunt trailing edge)
 - Smart blending functions (arbitrary polynomial)
 - Independent blending for upper and lower contour (e.g. for strong cambered airfoils)
- Automatic generation of block-structured meshes for airfoils
 - Currently only single element C-type meshes are supported
- Mesh control
 - Boundary layer region
 - Wake region
 - Windtunnel
- Mesh export (most formats based on the [meshio](#) library)
 - [AVL FIRE](#) (*.flma)
 - [SU2](#) (*.su2)
 - [GMSH](#) (*.msh)
 - [VTK](#) (.vtk)
 - [CGNS](#) (.cgns)
 - [ABAQUS](#) (.inp)
- Simple aerodynamic analysis, i.e. panel methods
 - [AeroPython](#)

Quick start guide

Follow the [Quick start guide](#) tutorial to get a first impression of the key functionality.

Code repository

The code is hosted on GitHub: [PyAero source code](#)

CHAPTER 1

Introduction

PyAero is an open-source airfoil contour analysis and CFD meshing tool. The main intention of writing the software was to make an easy to use tool for 2D airfoil meshing. Meshes generated by **PyAero** are intended to be used in subsequent CFD analysis.

PyAero, at least at the moment, does not do the CFD calculation itself. At a later stage it might be possible that **PyAero** will be interfaced (i.e. export meshes in the respective format(s)) with existing open source CFD methods like **SU2** or similar.

As an initial step towards aerodynamic calculations, a panel method from **AeroPython** (©2014 Lorena A. Barba, Olivier Mesnard) has been implemented.

Airfoil contours (at least legacy airfoils) often are described through a limited number of points (approx. 60 points). When meshing such contours, if not interpolated by splines, the resulting mesh and numerical solutions based on it would end up with artefacts. These would deteriorate the quality of the analysis results.

Therefore, in addition to the mesh generation module, some additional features have been implemented. These features are intended to be able to analyze and improve the airfoil contour. The improvement process is supported by point insertion and spline interpolation techniques. First and second derivatives of the contour allow for control of the contour as well as curvature smoothness.

To reflect *real* shapes, an option for creating a trailing edge and blending it to the contour is implemented.

The following figures show an example mesh for the airfoil **RAE2822** and a magnification of leading edge and trailing edge sections.

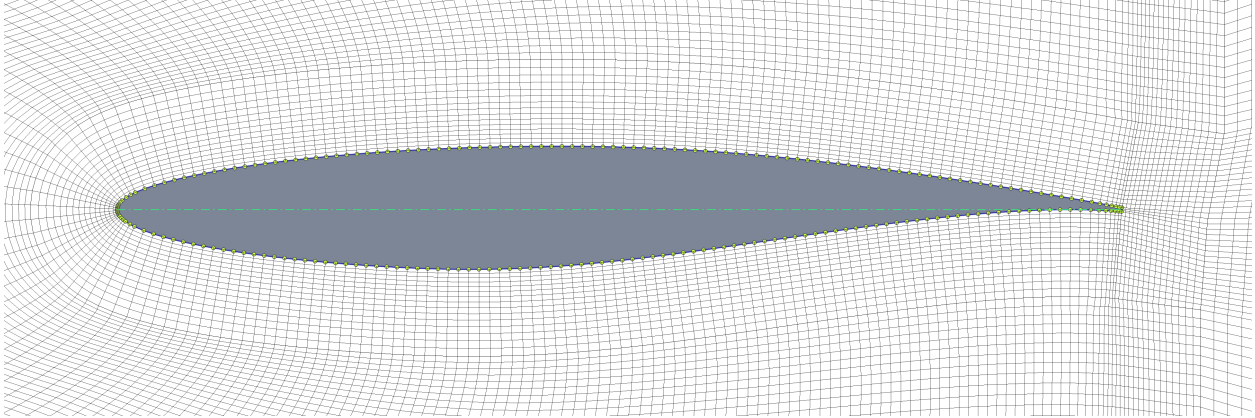


Fig. 1: Mesh around **RAE2822** airfoil

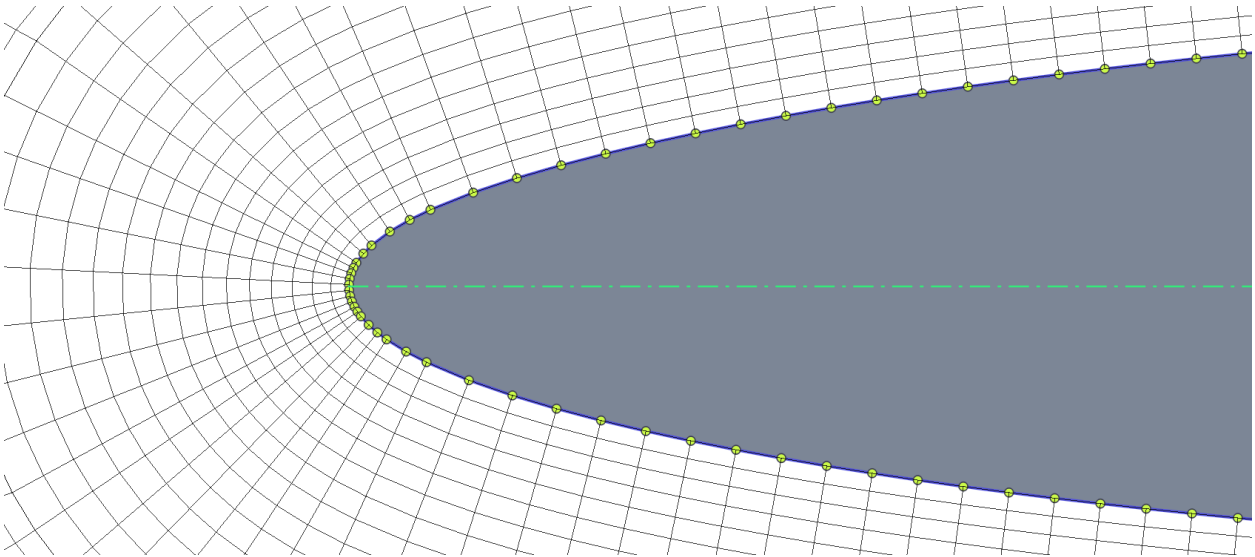


Fig. 2: Leading edge mesh view of **RAE2822** airfoil

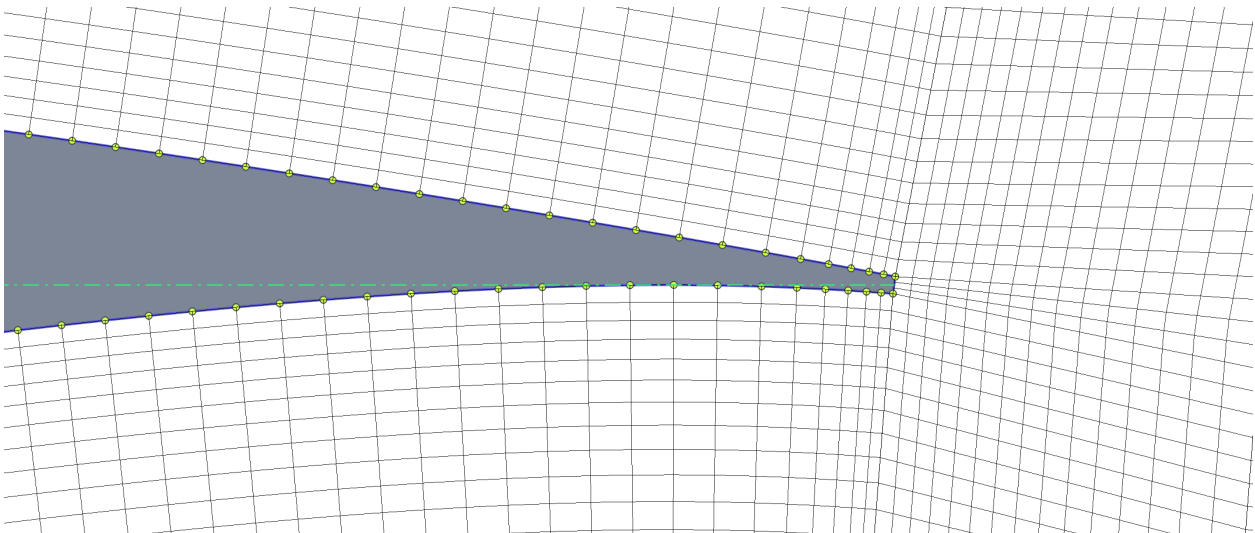


Fig. 3: Trailing edge mesh view of **RAE2822** airfoil

The general steps for mesh generation in [PyAero](#) can be explained as follows:

1. Load an airfoil contour file This is to get the raw data describing the airfoil contour.
2. Spline and refine the airfoil contour This is to update/improve the contour and prepare the mesh resolution along the airfoil.
3. Make a trailing edge with finite thickness This adds a so called blunt trailing edge to the contour. Skip this step if the trailing edge should be sharp.
4. Mesh the refined airfoil contour Start the meshing process.
5. Export the mesh in the required format Save the mesh in the specified format to the harddrive.

This is it.

Check the animation below, on how this looks in the graphical user interface (version 2.1.5).

Fig. 1: Step by step mesh generation with predefined airfoil contour

PyAero comes with a graphical user interface (GUI) written in Qt for Python aka Pyside6.

3.1 Overview

The layout of the user interface can be seen in the figure below. Different functional areas are bordered with blue lines. These areas are:

- Menubar
- Toolbar
- Toolbox
- Graphics view
- Viewing options
- Message window

Loading and saving geometry and meshes is done via the menus and the toolbar. Most operations during geometry preparation and meshing are done inside the toolbox.

3.2 Menus

Menus in PyAero try to behave much the same as in typical desktop software. For standard menus as *File* or *Print* the documentation will be kept short. See figure above for the location of the menubar in the GUI and the figure below for an overview of the menu structure.

The menus in the menubar and the tools in the toolbar (see Toolbar) are coded in a dynamic way. That is, all menus and toolbar items (and their respective handlers/callbacks) are read from XML files (see `PMenu.xml`, `PToolbar.xml` in the `data/Menus` folder of the installation). The graphical user interface is automatically populated using the entries of those files. With this structure in place, menus and toolbar items can easily be extended and customized.

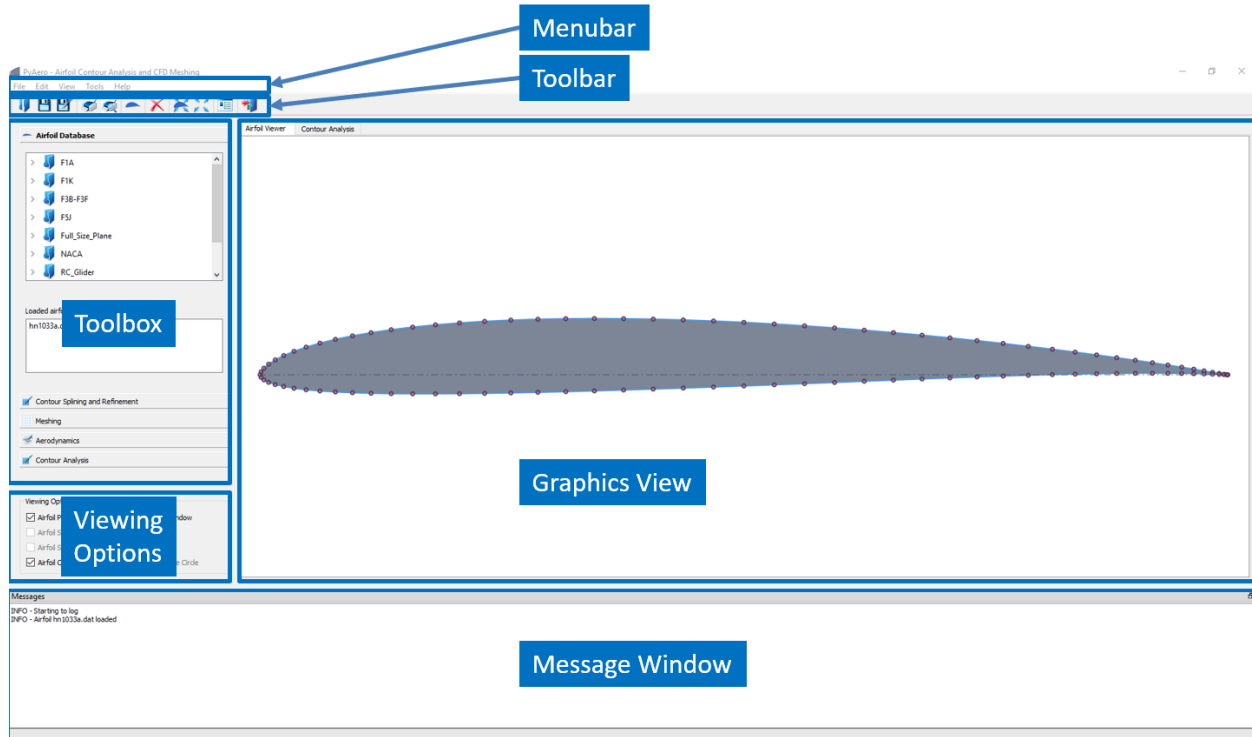


Fig. 1: Graphical user interface of PyAero

When adding new menus and thus functionality, it is required to provide corresponding functions in the code or handlers (in *Qt for Python* nomenclature so-called “slots”) to take care of the newly introduced functionality.

File	View	Tools	Help
<input type="checkbox"/> Open	<input type="checkbox"/> Fit Airfoil in View	<input type="checkbox"/> Calculator	<input type="checkbox"/> Manual (online)
<input type="checkbox"/> Save	<input type="checkbox"/> Fit all in View	<input type="checkbox"/> Settings	<input type="checkbox"/> Manual (Pdf)
<input type="checkbox"/> Save as ...	<input type="checkbox"/> Toggle Background		<input type="checkbox"/> Keyboard shortcuts
<input type="checkbox"/> Print	<input type="checkbox"/> Toggle Message Window		<input type="checkbox"/> About Qt
<input type="checkbox"/> Print Preview			<input type="checkbox"/> About PyAero
<input type="checkbox"/> Exit			

Fig. 2: PyAero menu structure

Note: Most probably, the XML files will be changed to JSON format sooner or later. This will not change the functionality.

3.2.1 Menu *File*

This menu is, as pointed out before, a typical desktop software menu. It consists of a set of dialogs for file manipulation, printing and alike.

Submenu *Open*

The *File* → *Open* submenu is used to load airfoil contour files or meshes.

At the moment the standard file format as used in many applications is supported.

“Coordinates in the standard format wrapping from upper surface trailing edge around the leading edge to the lower surface trailing edge”

—[UIUC airofil data site](#)

See figure below for the layout/format of such a file. Only the first and last few lines of the file are printed here.

```
#
#  RG-15 8.9% thickness
#
1.00000  0.0
0.99671  0.00054
0.98726  0.00229
0.97237  0.00514
...
...
...
0.97003  0.00097
0.98652  0.00064
0.99660  0.00021
1.00000  0.0
```

Fig. 3: Typical format of an airfoil contour file

This format is often used in software like [XFOIL](#) and similar.

In addition, lines starting with a hash sign # at the beginning are interpreted as comment lines (i.e. not used inside PyAero). This can be used for documentation or to add additional information.

Submenu *Save*

The *File* → *Save* submenu is used to save airfoil contour files or meshes. The file is saved using the name from the currently active airfoil. An extension with respect to the selected file type (e.g. .dat, .msh) is added automatically.

Submenu *Print*

The *File* → *Print* submenu allows for printing the content of the graphics view. Only graphics items as airfoil, coordinate cross, etc. will be plotted. The background gradient of the main graphics view will not be plotted. In case, that the background gradient shall be on the image, use the screenshot option.

Submenu *Print preview*

The *File* → *Print preview* submenu allows previewing the printout prior to printing. Actual printing then can be started directly from the print preview dialog, see Figure 2 3.

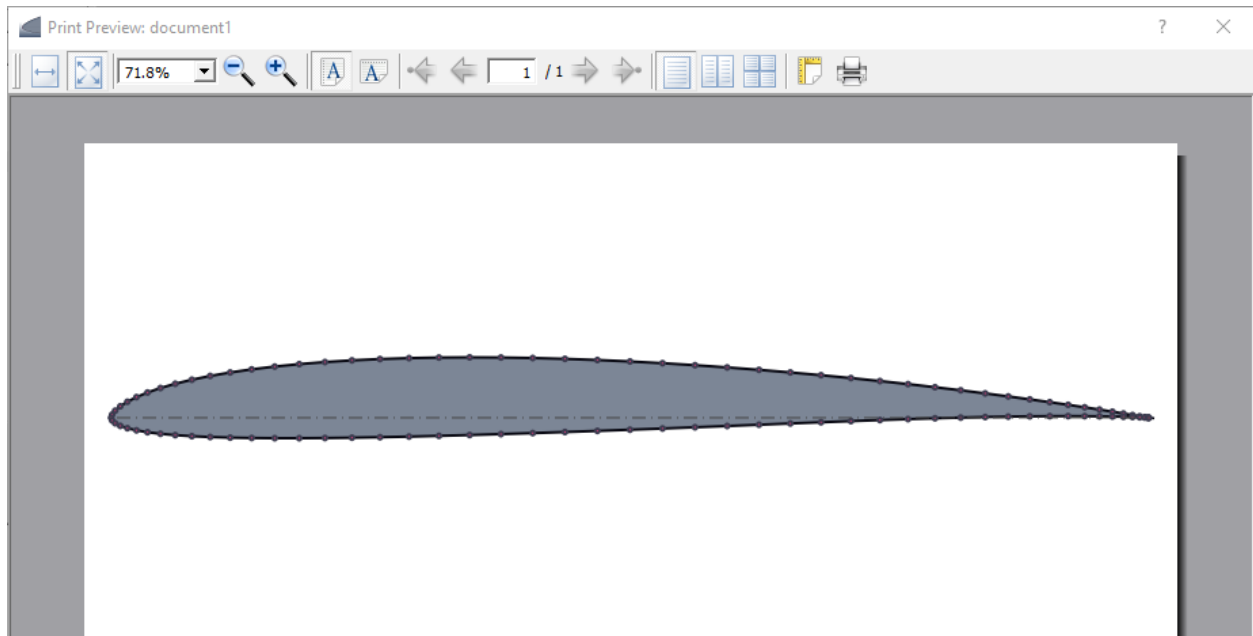


Fig. 4: Print preview dialog

Submenu *Exit*

The *File* → *Exit* submenu does what it says and closes all windows and exits [PyAero](#).

3.2.2 Menu *View*

This menu is used to set dedicated zoom levels or to toggle parameters of the view.

All submenus of the *View* menu can be activated by respective shortcuts or by using the right mouse button inside the graphics view.

Submenu *Fit airfoil in view*

The *View* → *Fit airfoil in view* does what it says.

It fits the complete airfoil into the graphics view. This is exactly the same zoom and position which is used after an airfoil has been loaded. The keyboard shortcut for this submenu is `CTRL-f`.

Submenu *Fit all in view*

The *View* → *Fit all in view* does also what it says. It fits loaded airfoils only (if no mesh exist yet) or the complete mesh of the windtunnel into the graphics view. The respective keyboard shortcut is `CTRL-SHIFT-f`.

Submenu *Toggle background*

The *View* → *Toggle background* can be used to toggle the background of the graphics view. The default background is a solid white background. Toggling switches over to a gradient filled background (white to dark grey, from top to bottom). The respective keyboard shortcut used is `CTRL-b`.

Submenu *Toggle message window*

The *View* → *Toggle message window* can be used to toggle the message window which is located below the graphics view (see also *Graphical user interface of PyAero*). Toggling hides the message window and after toggling once more it is brought back again. The respective keyboard shortcut used is CTRL-m.

The message window is a so called floating window which can also be detached from the GUI. It can be positioned anywhere on the screen and it can be reattached to the GUI again at any position (top, bottom, left, right).

3.2.3 Menu *Tools*

The *Tools* menu is planned to collect some useful helper tools. Currently there is a preparation for launching a calculator or for changing *PyAero* settings.

Note: At the moment both are not implemented.

3.2.4 Menu *Help*

This menu provides access to help on *PyAero*. Help is available via a link to the online [documentation](#) or to a PDF version of it.

The *Help* → *About PyAero* pops up a window showing basic information on author and credits. Further it lists the versions of the main Python packages used in *PyAero*.

3.3 Toolbar

The toolbar in *PyAero* allows fast access to actions which are otherwise triggered by menus. Each of the toolbar buttons launch a specific action. The toolbar can be customized by editing the file `$PYAEROPATH/data/PToolBar.xml`.

Fig. 5: Overview on toolbar options

3.4 Toolbox Functions

The toolbox functions are arranged at the left border of the GUI. A *toolbox* is a GUI element that displays a column of tabs one above the other, with the current item displayed below the current tab. The toolbox is the main working area when generating meshes with *PyAero*. The complete functionality like splining, refining, contour analysis and meshing are operated there. See the animation below to get an overview on the options available in the toolbox.

Fig. 6: Overview on toolbox options

3.5 Tabbed Views

The graphics view in *PyAero* and a set of other views (see figure below) are arranged via a tab bar. E.g., the views can be switched between the graphics view and the contour analysis view. The latter contains graphs for curvature analysis.

Fig. 7: Overview on tabbed views

Note: The look and feel of the tabs might change over time.

3.6 Zooming, Panning

When an airfoil is loaded it is displayed with a size that fits into the graphics view (leaving a small margin left and right). The contour can then be panned and zoomed in the following way:

3.6.1 Panning

In order to pan (drag) the contour or any other item press and hold `CTRL` (`CMD` on MacOS) and then press and hold the left mouse button and move the mouse in order to drag the contour.

Fig. 8: Drag the items in the view by pressing `CTRL` and moving the mouse (left button pressed)

3.6.2 Zooming

Zooming is activated by pressing and holding the left mouse button. While dragging the mouse, a rubberband rectangle is drawn. This rectangle indicates the area which will be zoomed when releasing the left mouse button. In order to avoid accidental zooming too deep, a minimum size rectangle has to show up. A valid zoom rectangle is indicated by changing its background to a transparent blueish color (the minimum allowed size can be set in `Settings.py` by changing the value of `RUBBERBANDSIZE`). In order to zoom in deeper, the rubberband rectangle can be subsequently used.

Zoom limits (`MINZOOM`, `MAXZOOM`) are set in the file `Settings.py`.

Fig. 9: Zoom the items in the view. Select a rectangle using the left mouse button.

Another natural possibility to zoom the view, is to use the scroll wheel. Thereby the geometry is zoomed with respect to the current mouse position.

Zooming can further be done using the `Page-Up` and `Page-Down` keys.

A reset to the initial (home) position can either be achieved by pressing the `HOME` key or by right clicking in the graphics view and selecting *Fit airfoil in view* from the pulldown menu.

3.7 Keyboard shortcuts

To speed up some operations, a set of keyboard shortcuts are defined. In some of the menus the shortcuts for the respective actions are defined at the right side of the menu. Furthermore, the shortcut `CTRL+k` on Windows and `CMD+k` on MacOS are used to access an overview of the keyboard shortcuts.

Note: The keyboard shortcuts are rendered as uppercase letters in the GUI. Nevertheless, always lowercase letters need to be used, unless the `SHIFT` key is a part of the shortcut.

Loading airfoils can be done in different ways.

4.1 Load via menu *Open*

The *File* → *Open* menu is the standard way to load airfoil contour data. The shortcut assigned to this menu is `CTRL+O`. When clicking this menu or applying the respective shortcut, a file dialog pops up. It allows to select files or browse directories.

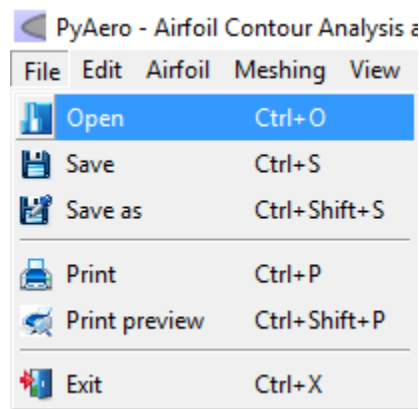


Fig. 1: *Open* menu to load an airfoil contour via the file browser

4.2 Load via the inline file browser

As outlined above there are more ways to load airfoils. A very handy way to browse airfoils is to use the implemented file browser. This browser is restricted in terms of navigation. Only files and folders below a predefined root path are

visible. The default root is the `data/Airfoils` subfolder from the standard installation. The root path for airfoils can be changed by the user in the file `src/Settings.py` by changing the value of the variable `AIRFOILDATA`.

The file browser is located in the toolbox on the left side of the application. It is the uppermost tab in the toolbox area.

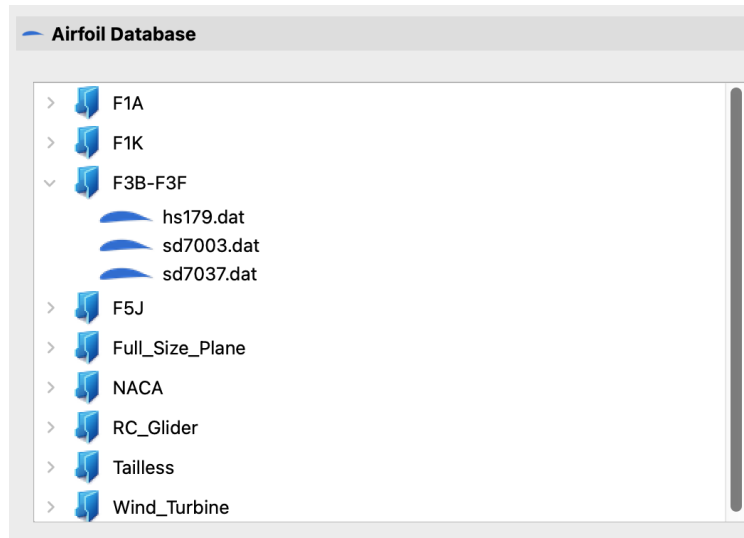


Fig. 2: File browser integrated in the *Toolbox*.

See also:

For more information on configuring the root path to airfoil data see [Settings](#).

4.3 Load via the *Toolbar*

Another way to open the file dialog is to click on the *Open* icon in the toolbar. The toolbar consists of a row of icons just below the menu bar. The toolbar and its icons can be customized by editing the file `data/PToolBar.xml`.

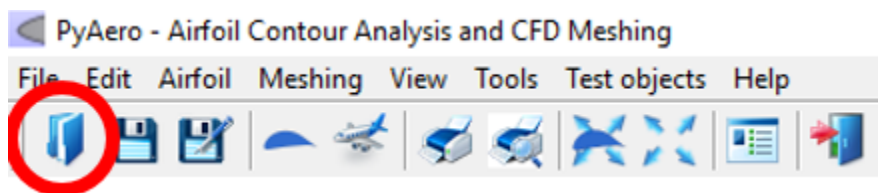


Fig. 3: Toolbar icon to load an airfoil contour via the file browser

See also:

For more information on configuring the menubar and the toolbar see [Settings](#).

4.4 Load a predefined airfoil

For testing purposes a predefined airfoil can be loaded without the need of a file dialog. The airfoil which is predefined can be configured.

See also:

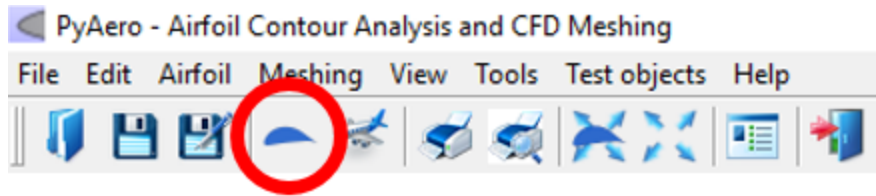


Fig. 4: Toolbar icon to load a predefined airfoil contour

See tutorial [Settings](#) on how to change the default airfoil.

4.5 Load via drag and drop

Last but not least, one or more airfoil(s) can be loaded via drag and drop. Just drag a couple of files, e.g. from the Explorer (Windows) or Finder (MacOS), to the graphics window. All files will be loaded, but only one file will be displayed. All the other files are shown (and can be activated by double-clicking on the name) in the toolbox area.

Fig. 5: Load multiple contours via drag and drop

Splining and Refining Airfoil Contours

The meshing process in [PyAero](#) relies on the point distribution on the airfoil contour. During meshing, there is a mesh constructed around the airfoil, which consists of mesh lines perpendicular to the airfoil contour. The mesh lines are starting at the individual airfoil contour points. So in order to be able to generate a proper mesh, the contour point distribution has to be adapted first. This is important, because particularly legacy wing sections have quite a coarse resolution (around 60 points). In the figure below, there is an animation of:

- the original airfoil contour
- the splined airfoil contour
- and the mesh lines parting from the splined contour

Fig. 1: Airfoil contour before and after splining

Two functions improve the contour before meshing. After clicking *Spline and Refine* in the respective toolbox function, the contour will at first be splined and in a second step refined.

The splining is done using B-splines via the Scipy function `scipy.interpolate.splprep`. This produces a spline representation through the initial (raw) airfoil contour. The number of points on the spline obviously can be set by *Number of points on spline*. Using an equal arc length the respective number of points is distributed homogeneously along the spline. This is the intended behaviour, as it guarantees constant size mesh cells around the airfoil (since the mesh is based on these points).

Obviously, at the leading and trailing edges some more care is necessary to produce the required mesh resolution. At the leading edge it is required to resolve the big pressure gradients which are produced by the shape of the airfoil nose.

A recursive refinement algorithm is used to resolve the contour until a certain criterion is met (see following figure). A B-spline is interpolated through the given raw contour points. At first, equidistant arc length segments are created on the spline (according to the prescribed number of points). During recursive refinement, the algorithm checks each pair of adjacent line segments if they match the criterion. The criterion is based on the angle included between adjacent segments. If the angle is less than a threshold specified by the user via the *Refinement tolerance* input, the algorithm adds two points. Each point is placed on the interpolated spline, half distance within each of the current segments. Then, new segments are created and angles are checked over and over again, until no pair of segments exists which

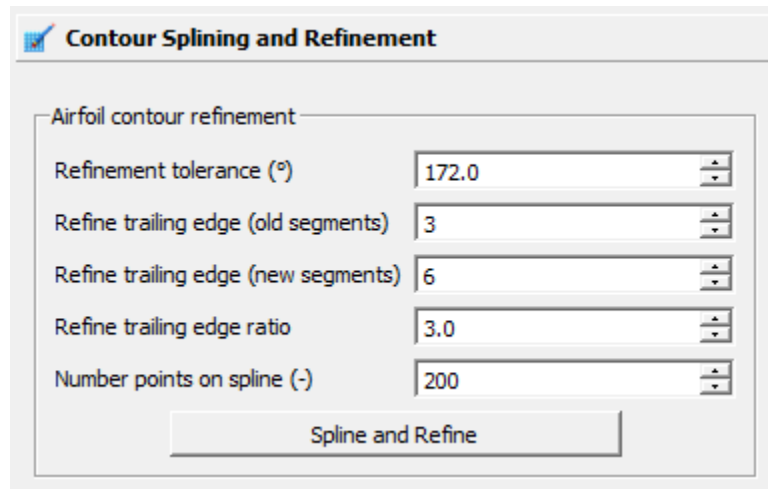


Fig. 2: Toolbox function for specifying spline and refining parameters

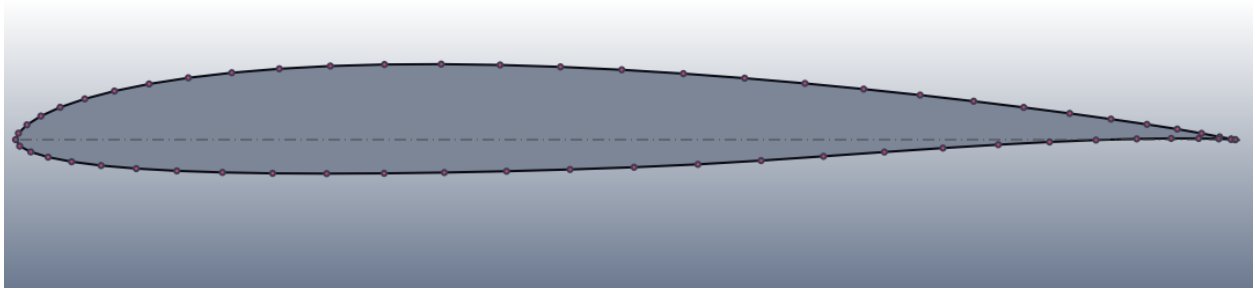


Fig. 3: Airfoil RG15: Points are as loaded from original file

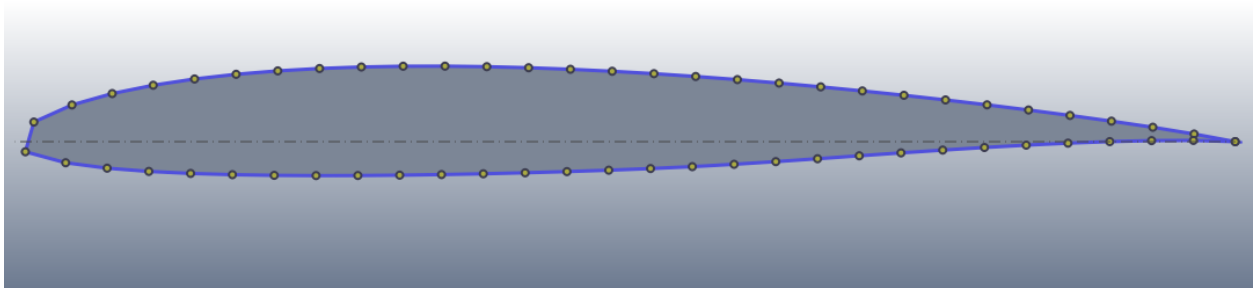


Fig. 4: Airfoil RG15: Contour after splining with 60 points, no refinements

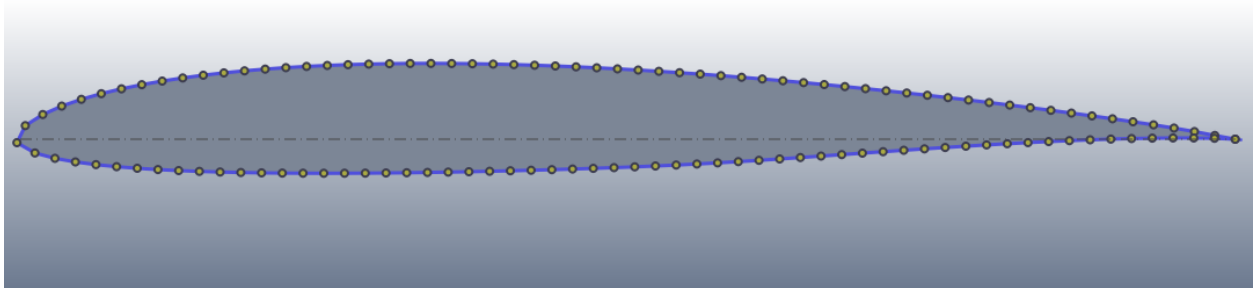


Fig. 5: Airfoil RG15: Contour after splining with 120 points, no refinements

include angles less than the threshold. This guarantees that the angle between adjacent mesh cells in the boundary layer is as uniform as possible. Thus, pressure gradients around the airfoil are resolved with the same quality.

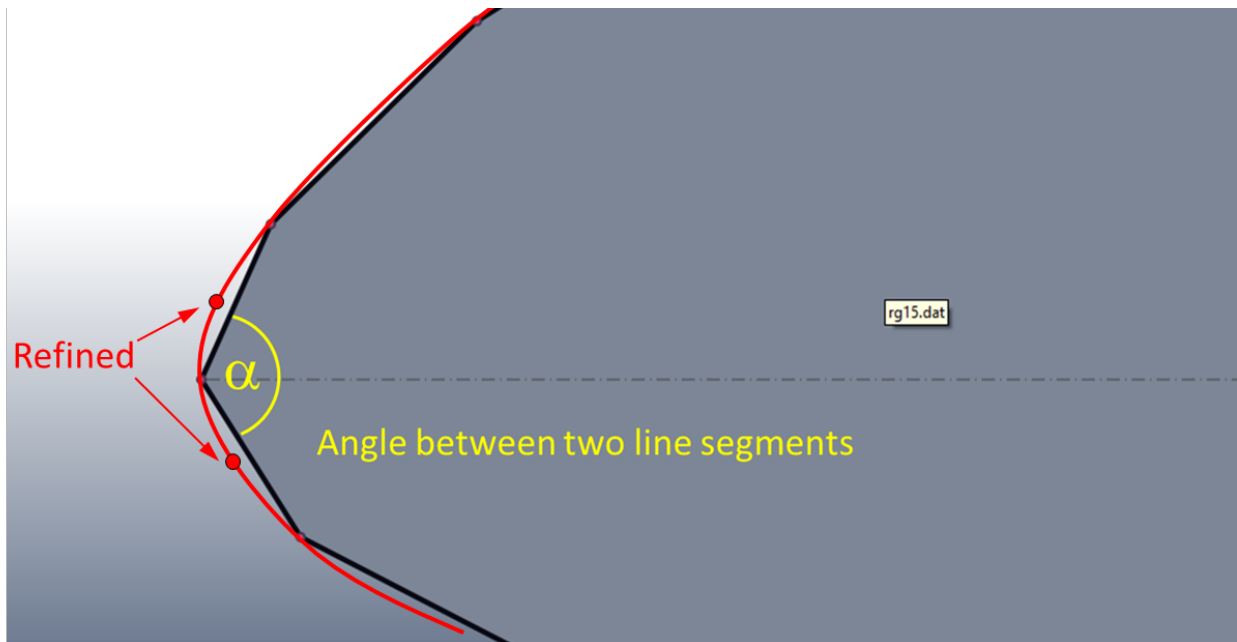


Fig. 6: Refinement algorithm

Angles between 170° and 173° already produce very well resolved leading edge contours (see following figures).

At the trailing edge again pronounced pressure gradients due to flow separation shall be resolved by a finer mesh.

The trailing edge refinement algorithm is somewhat simpler. The user specifies the number of segments to be refined at the trailing edge. If the number of *Refine trailing edge (old segments)* is 3, both, at the upper and lower sides of the contour, the last 3 segments are selected for refinement. The number of segments is then changed to *Refine trailing edge (new segments)*. If this number is 6 and the chosen compression rate *Refine trailing edge ratio* is 4, a distribution as depicted in figure *Airfoil RG15: Close-up, 60 points, LE refinement 3, 6, 4* is created.

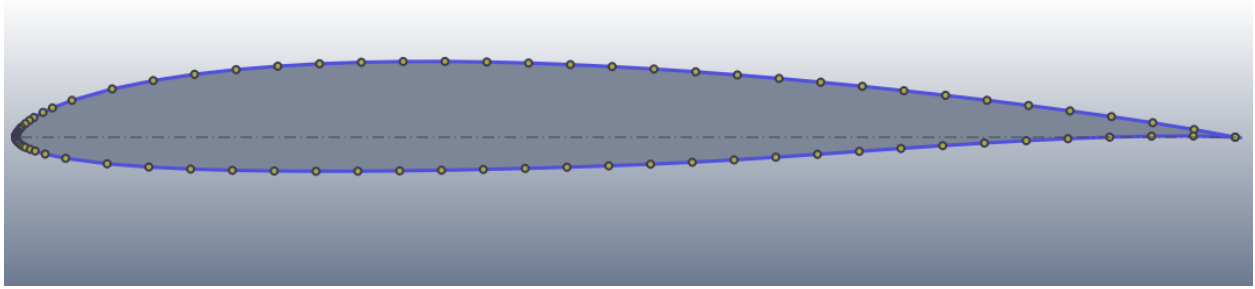


Fig. 7: Airfoil RG15: Contour after splining with 60 points, LE refinement 170°

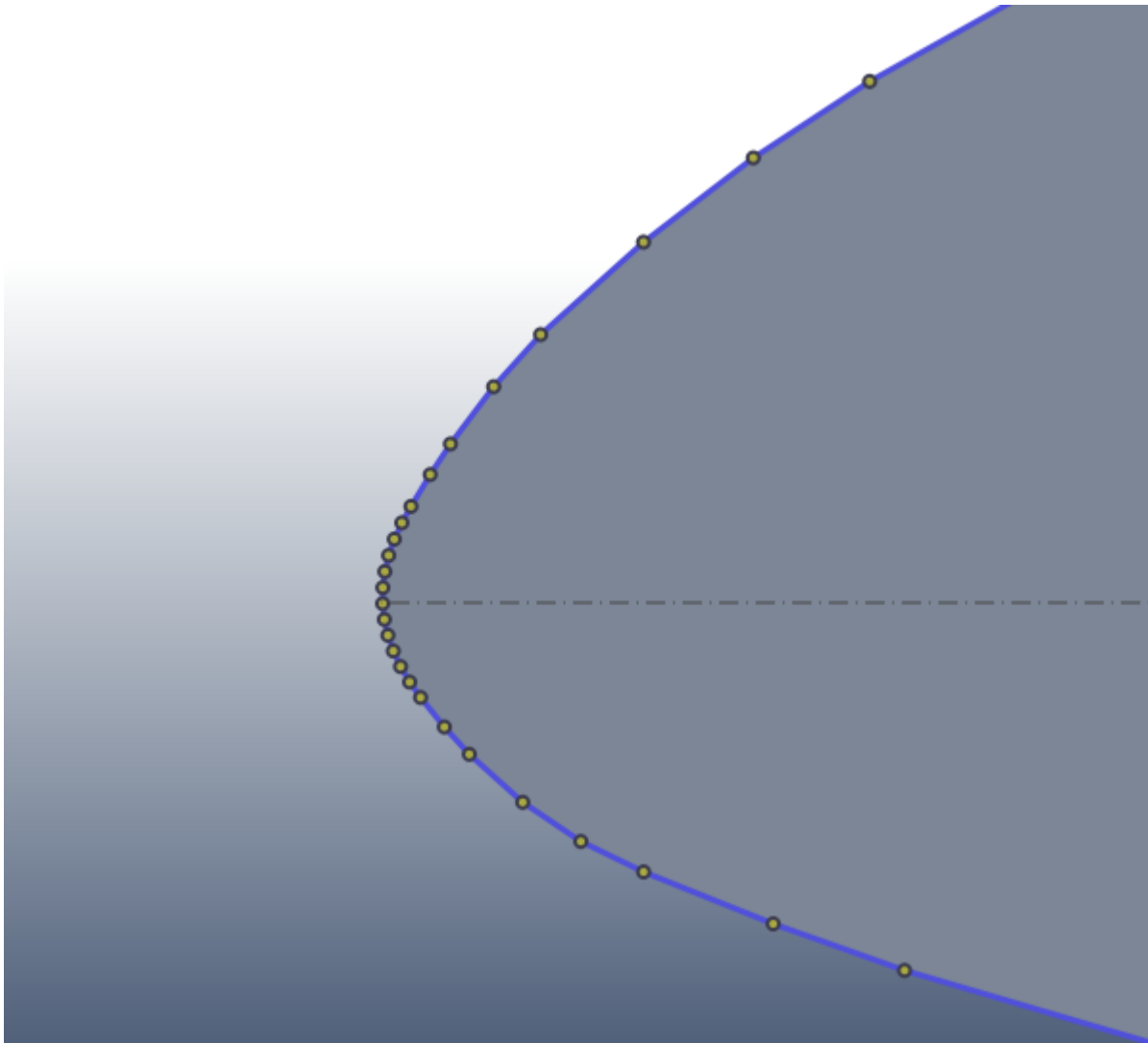


Fig. 8: Airfoil RG15: Close-up, 60 points, LE refinement 170°

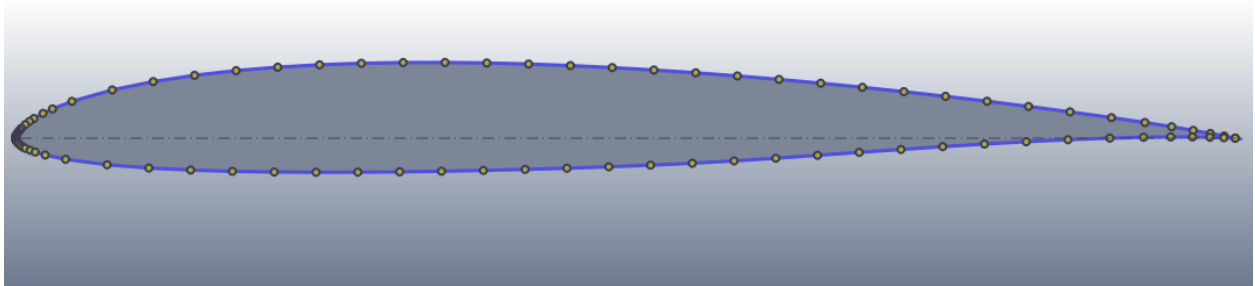


Fig. 9: Airfoil RG15: Contour after splining with 60 points, LE and TE refinements

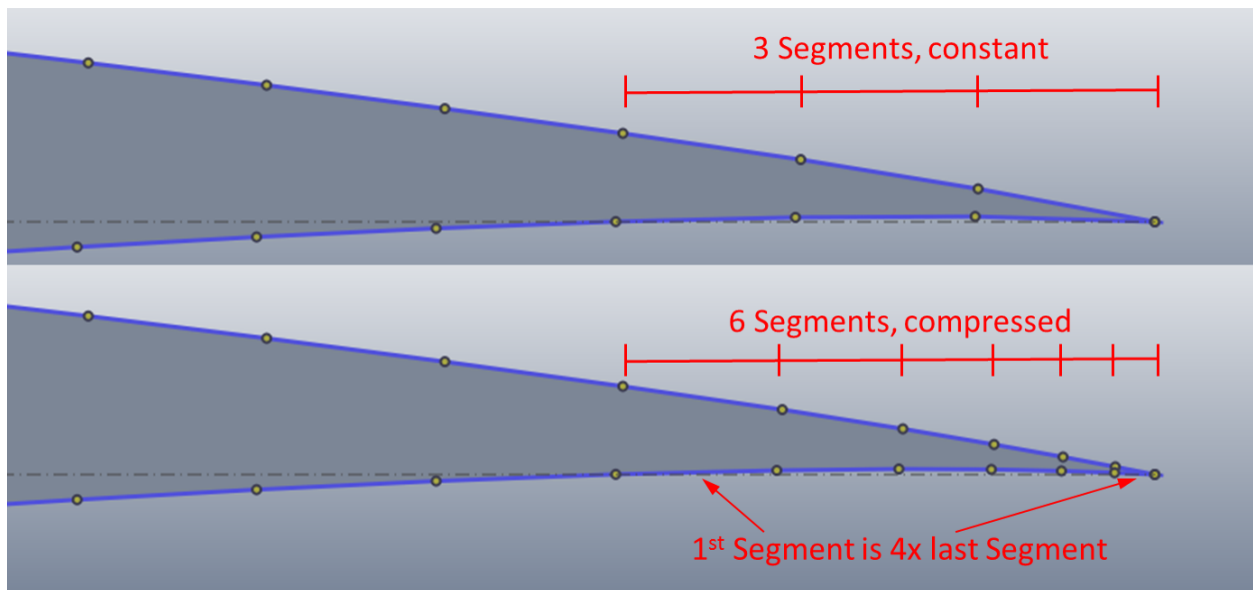


Fig. 10: Airfoil RG15: Close-up, 60 points, LE refinement 3, 6, 4

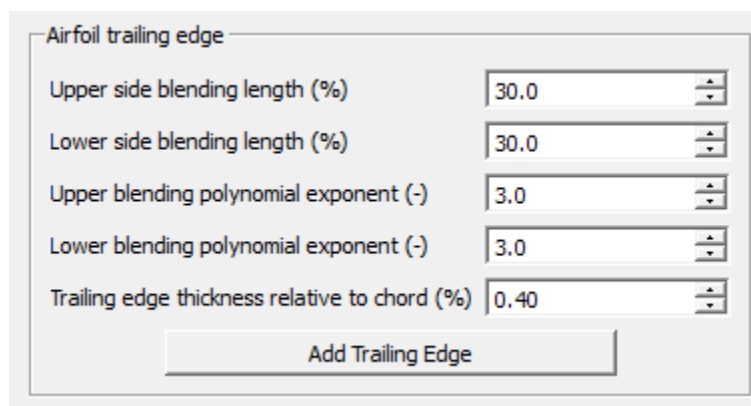
Trailing Edge

Note: If a sharp trailing edge is needed, this step can be skipped.

As outlined in the section before, the meshing process relies on the point distribution on the airfoil contour. Real airfoils, i.e. airfoils which are built as a hardware, have a trailing edge (TE) with a definite thickness, a *blunt trailing edge*. This is due to manufacturing and/or structural reasons. To be able to model this, [PyAero](#) has a dedicated function. The following figure shows an animation of a sharp trailing edge and a blunt trailing edge.

Fig. 1: Sharp and *blunt* trailing edges

The blunt trailing edge needs to be added to the original contour in a controlled manner. The parameters shown in the following figure can be used to control this process.



Airfoil trailing edge	
Upper side blending length (%)	30.0
Lower side blending length (%)	30.0
Upper blending polynomial exponent (-)	3.0
Lower blending polynomial exponent (-)	3.0
Trailing edge thickness relative to chord (%)	0.40
Add Trailing Edge	

Fig. 2: Toolbox function for specifying the blunt TE blending parameters

The trailing edge thickness itself can be specified relative to the unit chord. The thickening is done perpendicular to the camber line at the trailing edge. In order to prevent aerodynamic artefacts due to the blunt TE, a smooth controlled blend from the blunt TE vertices into the original (raw) contour needs to be done.

This is achieved by allowing to blend along a certain user specified length into the original contour. Furthermore, the degree of the blending curve can be specified. The blending length describes the fraction (in %) of the chord in which the blending is done. The polynomial exponent can be used to describe the blending curve degree. The blending can be controlled individually for the upper and lower sides of the contour. This is specifically useful for strongly camber airfoils.

To better understand the TE blending options the following figure depicts a visually exaggerated TE blending (by clicking several times on the *Add Trailing Edge* button). The upper side of the airfoil is blended over 50% of the chord with a linear blend, whereas the lower side is blended over 20% of the chord with a polynomial of degree three.

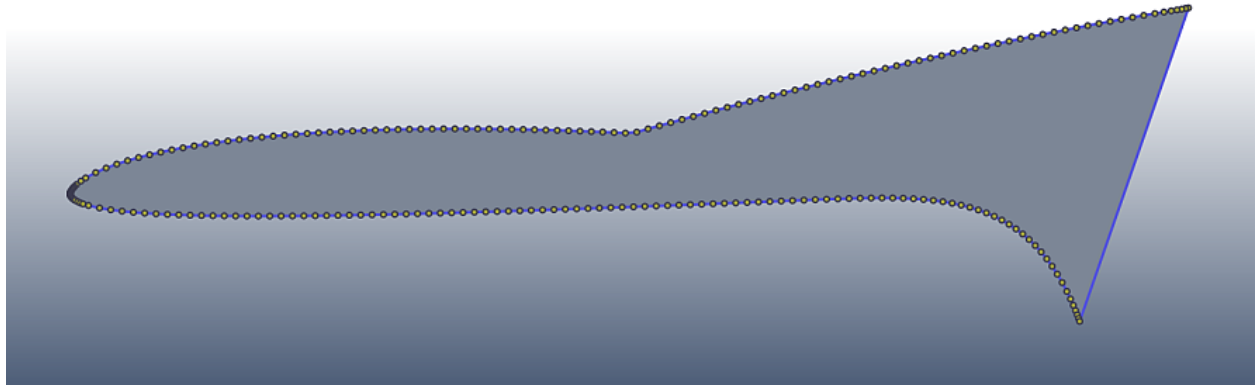


Fig. 3: Exaggerated TE blending: 50% linear (upper), 30% 3rd order polynomial (lower)

Playing with the settings and finding the best setup for blending is always best viewed by clicking two or three times on the *Add Trailing Edge* button. A reset to the original curve can be achieved by simply clicking on the *Spline and Refine* button in the menu above (see [Toolbox function for specifying spline and refining parameters](#)).

When a satisfying result is achieved for the contour in terms of splining, refining and trailing edge, the meshing process can be started.

Making Meshes

After *Splining and Refining Airfoil Contours* and optionally making a blunt *Trailing Edge*, the airfoil contour can be meshed which is the primary purpose of **PyAero**. As for splining and refining the meshing options are located in the toolbox area which is the left pane of the user interface (see *Overview on toolbox options*).

The default settings for the mesh generation process should be good enough to generate a mesh that can be used to do CFD RANS simulations.

The mesh is constructed from four individual blocks, each of which has its own configuration options. The mesh blocks are (listed below with the same name as in the GUI):

- Airfoil contour mesh (block 1)
- Airfoil trailing edge mesh (block 2)
- Windtunnel mesh around airfoil (block 3)
- Windtunnel mesh in the wake (block 4)

The main mesh block is the one directly attached to the airfoil contour. It is constructed by grid lines emerging perpendicular from the airfoil, starting at the points from the splined contour (see *Splining and Refining Airfoil Contours*). Another set of lines parallel to the airfoil contour complete the main mesh block. The default settings there implement a stretching away from the airfoil, so that the thinnest mesh layer is attached at the airfoil and further mesh layers are gradually thickened outwards.

The process of constructing the grid lines perpendicular and parallel to the contour guarantees a fully orthogonal mesh in the vicinity of the airfoil which is important for keeping numerical errors as low as possible in the region of interest. The mesh distribution settings for this block are depicted in the following figure (*Settings for the mesh around the airfoil (block 1)*).

The value for the number of `Gridpoints` along airfoil contour is grayed out. This value is taken from the number of points on the spline and is displayed here just for reference(see also *Toolbox function for specifying spline and refining parameters*). If a different number of grid points along the contour is required the spline has to be updated at first. Next the `Divisions` normal to airfoil allows to vary the number mesh layers normal to the contour within mesh block 1. The setting `1st cell layer thickness` specifies the dimension/length of block 1 normal to the contour in percentage of the airfoil chord. It is limited to 100% chord length, but typical values would be in the range 5% to 20%. The final parameter for block 1 is the `Cell thickness ratio (-)`. It specifies the ratio of the cell thickness of the outermost cell in the block (wrt to airfoil normal direction) over the cell thickness of the layer

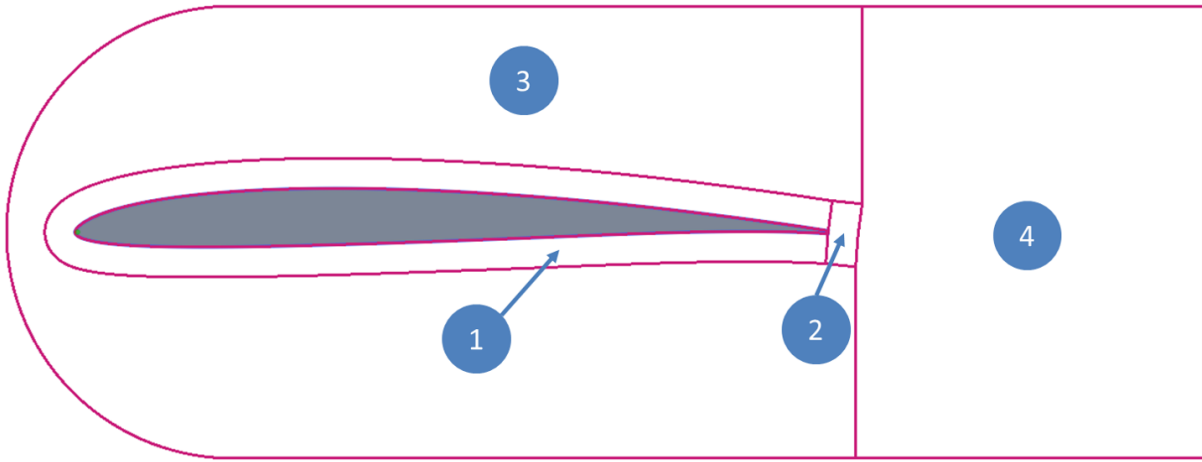


Fig. 1: Mesh blocking structure

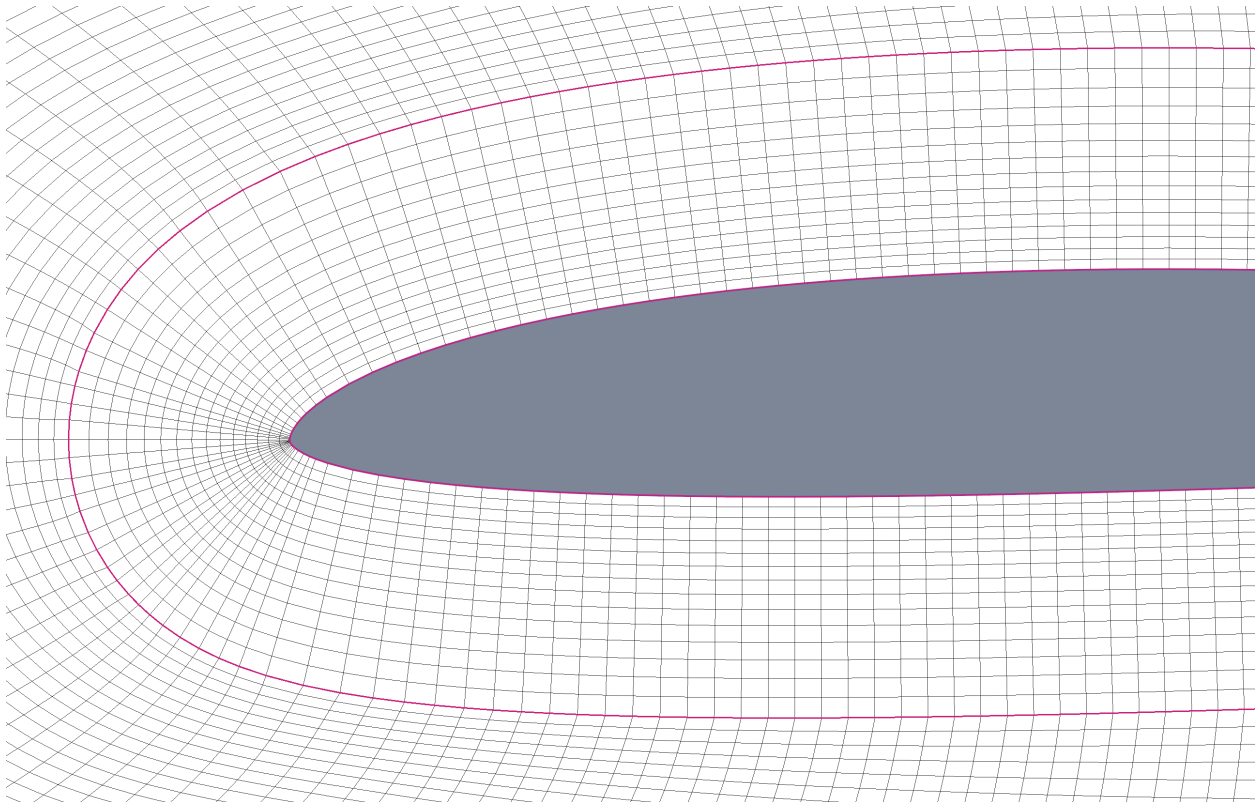


Fig. 2: Mesh around airfoil (block 1)

images/mesh_settings_airfoil_contour.png

Fig. 3: Settings for the mesh around the airfoil (block 1)

which is attached to the contour. So if for example the ratio is 3, the outer cell layer of block one is 3 times a thick as the cell layer at the airfoil (see [Mesh stretching ratio](#)).

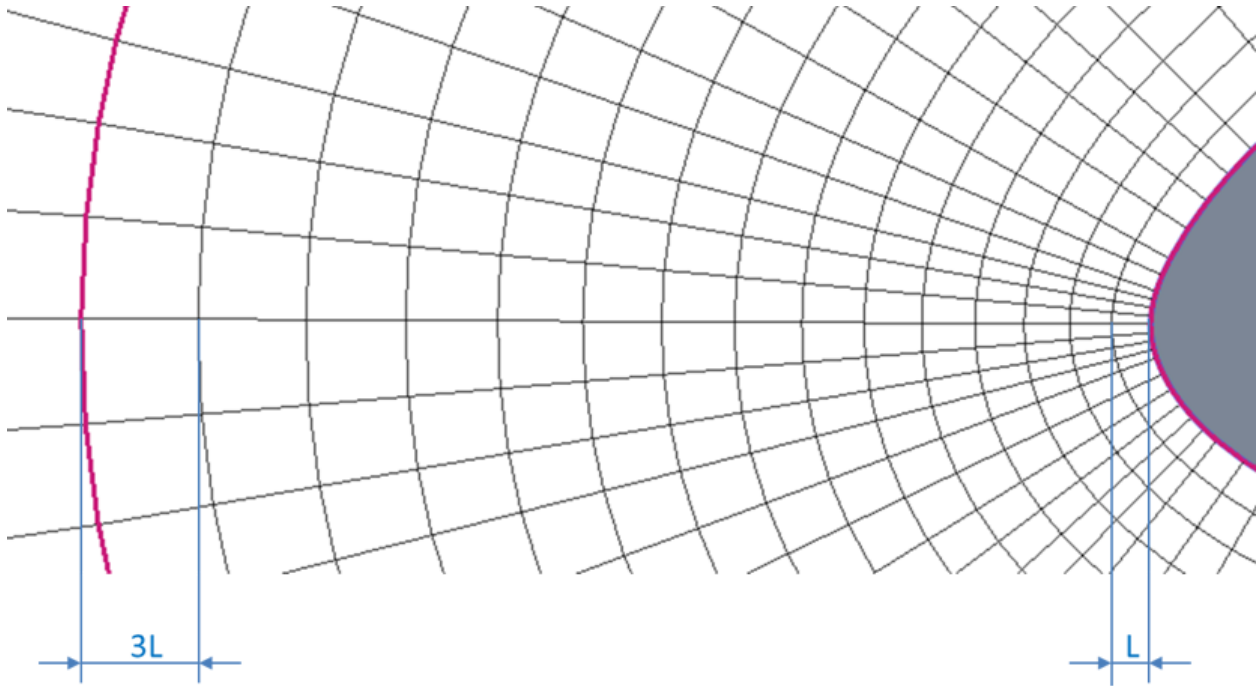


Fig. 4: Mesh stretching ratio

The trailing edge mesh is the region directly behind the airfoil (block 2, see [Mesh blocking structure](#)). This block has its own parameters in order to be able to fine control the grid resolution where upper and lower contour shear layers meet and interact(see figure_mesh_block_TE).

Fig. 5: Mesh at the trailing edge (block 2)

The parameter *Divisions at trailing edge* controls the number of subdivisions at the trailing edge (see blue circle in [Mesh at the trailing edge \(block 2\)](#)). If the airfoil trailing edge has a finite thickness (blunt trailing edge), these cells resolve the small vertical part of the trailing edge. *Divisions downstream trailing edge* is the number of subdivisions in the direction of the airfoil wake inside block 2. The *Length behind trailing edge (%)* is the length of block 2 in the same direction measured as fraction of the unit chord. The *Cell thickness ratio (-)* has the same effect on the grid line distribution as already depicted for the mesh around the airfoil (block 1).

In case of a sharp trailing edge, above parameters are not used. The cells of the airfoil upper and lower grid lines meet at the trailing edge and continue as one gridline downstream.

The next set of parameters specifies the grid distribution within block 3. The parameters are handled in the same way

Airfoil trailing edge mesh

Divisions at trailing edge	3
Divisions downstream trailing edge	6
Length behind trailing edge (%)	4.0
Cell Thickness ratio (-)	3.0

Fig. 6: Settings for the airfoil trailing edge mesh (block 2)

Fig. 7: Example mesh for a sharp trailing edge

as for block 1 and block 2. The distribution biasing is just an intermediate helper function and should be kept with its default value (see note below) for symmetric or slightly cambered airfoils. For airfoils with pronounced camber setting biasing to *lower* improves the mesh quality.

Important: The meshing algorithm in block 3 is not finished, rather it is a tweaked version of a transfinite interpolation. This will be updated with elliptic grid generation or similar.

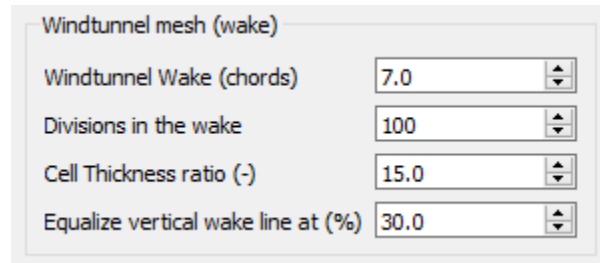
Windtunnel mesh (around airfoil)

Windtunnel Height (chords)	3.5
Divisions of Tunnel Height	100
Cell Thickness ratio (-)	10.0
Distribution biasing	symmetric

Fig. 8: Settings for the windtunnel around the airfoil (block 3)

The final mesh block (see block 4 in [Mesh blocking structure](#)) is the remainder of the windtunnel downstream. It copies the mesh distribution of blocks 1, 2 and 3 on its upstream side. Again the settings left over here should be self-explanatory, except *Equalize vertical wake line at (%)*. At the outlet of the windtunnel downstream all cells have equal width in the vertical direction. The setting just mentioned allows to specify at which percentage of the block 4 in downstream direction the cells will be of homogeneous size in the vertical direction (see [Mesh block 4 - equalizing trailing edge grid line distribution](#)). The dashed vertical line indicates the location from where the vertical grid line distribution is homogeneous.

The following figure shows the final mesh of an example airfoil (**hn1033a**).



Windtunnel mesh (wake)	
Windtunnel Wake (chords)	7.0
Divisions in the wake	100
Cell Thickness ratio (-)	15.0
Equalize vertical wake line at (%)	30.0

Fig. 9: Settings for the windtunnel in the wake (block 4)

Fig. 10: Mesh block 4 - equalizing trailing edge grid line distribution

Fig. 11: Final mesh around airfoil **hn1033a**

Batch mode (run from command line)

In order to be able to automate the mesh generation process, a batch utility is available. This allows to run the mesh generation on a large number of airfoils. A batch control file is used to specify the airfoils to be processed. The batch control file is a text file which contains the following information:

- Airfoils which should be meshed
- Settings to be used for the mesher
- Output formats in which the mesh(es) should be written

The command used to launch the batch processing is:

```
python src/PyAero.py -no-gui data/Batch/batch_control.json
```

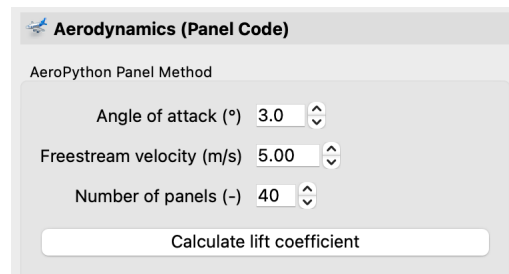
The batch control file can be located in an arbitrary folder and have any name. It just has to have the same format as the example file of the installation, see `data/Batch/batch_control.json`.

CHAPTER 9

CFD_boundary_conditions

Aerodynamics (panel code)

For fast lift coefficient calculations a panel method has been implemented. The code is based on an open source software developed by the [Barba group](#). The codes in [Aeropython](#) are part of a series of lessons of a university course. The specific module used here is based on a vortex-source panel method (code in [module 4](#)).



The screenshot shows a software interface titled "Aerodynamics (Panel Code)". Below the title is the subtitle "AeroPython Panel Method". There are three input fields with spinners: "Angle of attack (°)" set to 3.0, "Freestream velocity (m/s)" set to 5.00, and "Number of panels (-)" set to 40. At the bottom is a button labeled "Calculate lift coefficient".

Fig. 1: Settings for the panel method.

Clicking on *Calculate lift coefficient* will calculate the lift coefficient for the current configuration. The code runs very fast and is not influenced much by the number of panels. The code is using its own paneling method. Paneling is either based on the airfoil raw data or on the splined contour data (if available). A too high number of panels may lead to convergence issues.

A typical result/output of the method is shown below (the information is displayed in the message window):

```
INFO - Aerodynamic properties of Makarov KPS Long Root: INFO - Cl = 1.268 [-]
at Uinf = 5.000 [m/s], and AOA = 3.000 [degree]
```

Barba, Lorena A., and Mesnard, Olivier (2019). Aero Python: classical aerodynamics of potential flow using Python. Journal of Open Source Education, 2(15), 45, <https://doi.org/10.21105/jose.00045>

CHAPTER 11

Contour Analysis

CHAPTER 12

Settings

`PyAero` supports customizing its behaviour. The parameters which can be modified are stored in a file called `Settings.py`.

CHAPTER 13

Modify and extend GUI Functionality

CHAPTER 14

Dependencies

- Python 3
- Qt for Python (PySide6)
- Numpy
- Scipy
- meshio

CHAPTER 15

License

The MIT License (MIT)

Copyright (c) 2022 Andreas Ennemoser

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.