Xoptfoil-JX is a modified version of Xoptfoil version 1.11.1 – the airfoil optimizer by Daniel Prosser

Xoptfoil-JX Description of Modifications

Jochen Guenzel, November 2019

Target of this work was to overcome some of the hurdles when developing a „ready for use airfoil“ with Xoptfoil. It‘s still work in progress. As only ‘particle swarm' optimization based on ‘hicks-henne' shape functions was used for optimization all the other use cases and variations of Xoptfoil were not evaluated with the extensions described below.

Aerodynamic target values

In the current approach Xoptfoil optimizes all design variables (e.g. the cd value at a certain operating point) to their best value being part of the overall sum of design variables forming the ‘objective function’. The way to control the ongoing “fight” between the design variables for the best value is to tweak the ‘weighting‘ of a design variable.

When it comes to the final optimization of an airfoil, the tweaking of weightings can become quite challenging and time consuming. One percent less weight of a design variable at one end of the airfoil polar can improve the value of a design variable at the other end of a polar by 10 percent and vice versa.

To ease this final optimization process „aerodynamic targets“ were introduced. With the new optimization types ‘target-drag’ and ‘target-moment‘ one can define a certain ‘target_value‘ for an operating point. With this the Xoptfoil optimization tries to get to the value of the design variable as close as possible to the defined target value. As with the normal optimization types the gravitational force towards the target value is controlled by the weighting of the operating point.

It turned out that „aerodynamic targets“ are quite cool. Taken to extreme you may define a polar with target values for all operating points, give it to Xoptfoil and you will get the desired corresponding airfoil… (if the defined target values are part of the solution space… so care must be taken to define realistic targets values. Otherwise a design variable will move only towards a target but never reach it).

A more realistic use case could be: Improve the polar of my airfoil at higher cl-values above 0.8, leave the mid-range as it is and allow some deterioration at cl below 0.1.

To support such a use case a special ‘target_value= -1‘ can be defined which will take the current value of the seed airfoil as the target value for optimization. This will result in a simple „Try to keep the current value as it is“.

The optimization type ‘target-moment‘ was implemented when optimizing airfoils for flying wings where the moment of an airfoil highly determines the final flight characteristics. Although the evaluation of the cm value is not the prime domain of xfoil the resulting airfoils are looking quite „realistic“. It seems that the best way to influence the moment polar, is to define a ‘target_moment‘ at mid cl values of 0.3 to 0.5 which will lift or lower the whole moment polar to the desired value.

The aerodynamic targets are defined within the Xoptfoil input file like this (example)

```
op_mode(7) = 'spec-cl'
op_point(7) = 0.8
optimization_type(7) = 'target-drag'
target_value(7) = 0.0142
reynolds(7) = 146E+03
weighting(7) = 0.7
```

```
op_mode(4) = 'spec-cl'
op_point(4) = 0.4
optimization_type(4) = 'target-moment'
target_value(4) = -0.005
reynolds(4) = 172E+03
weighting(4) = 1.7
```

Geometric target values

Xoptfoil allows to control the geometry of an airfoil within certain limits with constraints – for example with ‘min_thickness’ and ‘max_thickness’ of the airfoil. If there is the need that the final airfoil should have a certain thickness e.g. the foil at the root of a wing, the way to go is to bring the seed airfoil to the desired thickness and then define max and min thickness constraints close above and below this value. For the optimization process this is only the second best approach as it narrows the solution space dramatically (constraints act like a wastebasket for all designs exceeding the constrain) and leads to a slow convergence towards the best solution.

For such an use case an approach, where thickness of the airfoil is not a constraint but a target is much more natural for the optimization process. Watching the visualization, it’s nice to see how such a geometric target may be in competition with an aerodynamic optimization.

Another use case came up, when I tried to have a certain thickness at the back part of the foil near the trailing edge. In Xoptfoil the parameter ‘addthick min(n)’ could be used for this requirement. But using this parameter an unwanted side effect arose: A thickness parameter will “link” the top and the bottom side of an airfoil. If the objective function tells that the bottom side should be cambered a little more then the upper side will be cambered in the same way through the thickness link. But what if the top side was already ideal for best trip location of turbulent transition… In this case it was preferable that the lower side “pays the bill” for a minimum thickness of the airfoil. To control this behavior two geometric targets, z-value bottom side and z-value top side, were implemented.

The approach of additional geometric targets harmonizes quite well with the existing aerodynamic design variables. The influence of geometric targets is controlled by their weighting. Xoptfoils input file is extended with following parameters (example)

```
&geometric_targets

  ngeotargets = 2

  x_Pos(1) = 0.7
  target_type(1) = 'zBot'
  target_geo(1) = 0.012
  weighting_geo(1) = 0.5

  target_type(1) = 'Thickness'
  target_value(1) = 0.076
  weighting(1) = 1.2
```

Assessment of airfoils surface quality

In my first tries with Xoptfoil I was surprised by the visual surface quality of the final airfoil – especially having a picture in mind that the airfoil should look like a car standing outside after a hailstorm of ‘Hicks Henne shapes’ applied to the seed airfoil. When I had a closer look on the pressure/velocity distribution over x (for example with Xflr5), these little and micro bumps you could see when zooming in, were the result of all shapes applied on the surface. I thought.

This picture of a hailstorm turned out to be wrong.

But the other thing on airfoil surface quality was really annoying: The curve reversals reported by Xoptfoil for the seed airfoil when starting a new optimization. And even worse, Xoptfoil reported curve reversals which I couldn't see at all.

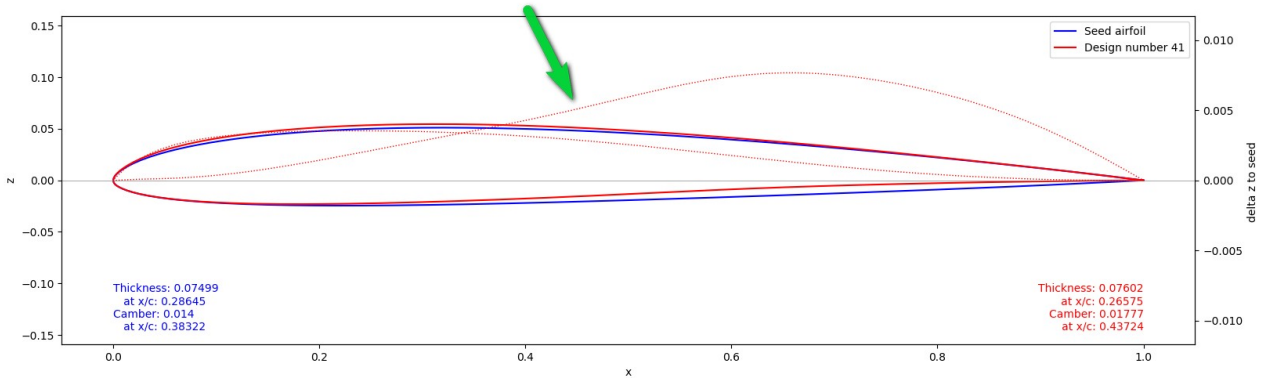
In my wish to have a smooth, continuous surface I had reduced the 'curv_threshold' value more and more. The result was a disease of "curve reversals" messages. I tried several methods to get rid off these reversals of the seed airfoil:

- xfoil (xflr5) re-paneling of the airfoil with a very low number of panels
- deleting coordinates from the airfoil definition file
- reverse design in xflr5 using splines

The more I did this, the more I had the feeling to follow the wrong approach. I wanted to take a seed airfoil and give it right away to Xoptfoil for optimization. My assumption was that there must be something wrong with the reversal detection within Xoptfoil.

The further dive in into this topic showed I was wrong again – at least half wrong.

So I decided to have a closer look what's going on with the surface of the airfoil when Xoptfoil is doing its work. With a few additional lines of code in the 'xoptfoil visualizer' it's quite easy to show the difference between the z coordinates of seed and the current optimized airfoil. This difference equals to the sum of all shapes applied to the seed airfoil.



The thin dotted line shows the z-value Xoptfoil adds/subtracts currently to/from the seed airfoil. This value is the sum of all shape functions generated so far during this optimization (approx. 300 iterations). Obviously Xoptfoil is modifying the airfoil very smoothly...

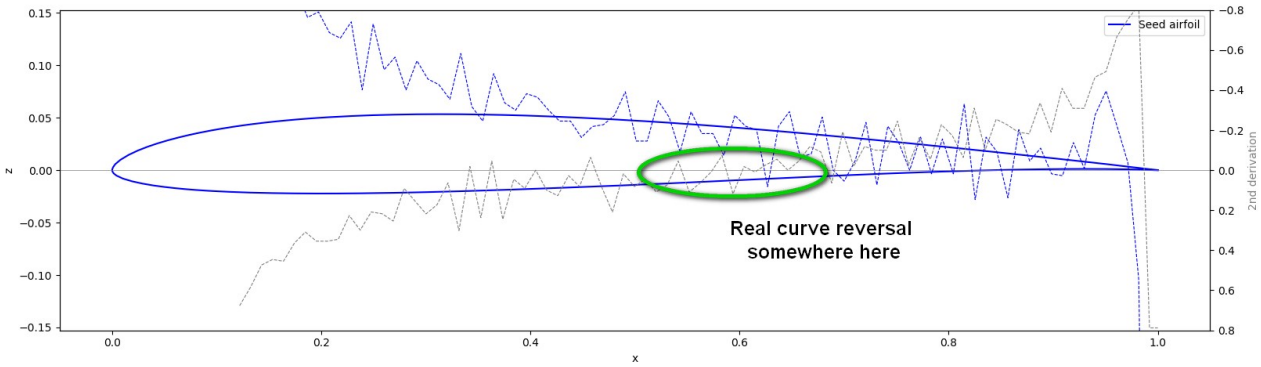
Looking at this I had another silent conversation with Dan telling him "You sun of a gun!". With a little guidance the overall shape applied to the airfoil is very smooth and continuous. Just perfect.

So – where could be the core of the problem? If it's not Xoptfoil, it must be the seed airfoil.

To assess a polyline like the surface of an airfoil (which are the coordinate points connected by straight lines) a look at the first, second and third derivative of the polyline is helpful. This is done by a "finite difference" approach (see Wikipedia) which also Xoptfoil implements to detect curve reversals. The finite differences have to be adapted due to the non-regular distribution of the x-coordinates of an airfoil.

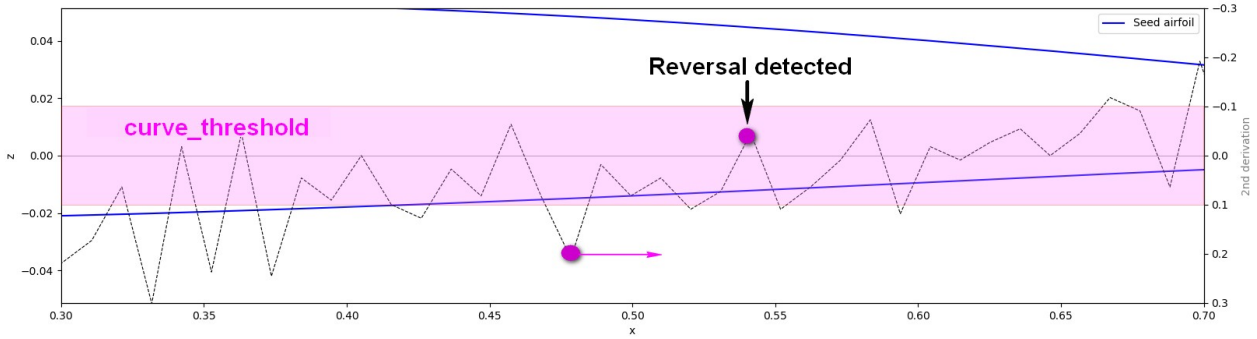
The newly calculated derivatives are handed from Xoptfoil to the 'xoptfoil_visualizer' via the "design coordinates file".

Zooming into the plot of the 2nd derivative of a typical "human made" airfoil the plot looks much more like the altitude profile of the Black Forest than the surface of a high end airfoil



The 2nd derivative of the surface polyline. When crossing the x-axis, there is a curve reversal. The bottom side of this airfoil has a real reversal at x = 0.6 which is not easy to detect (grey dashed line)

Zooming a little more allows to show Xoptfoils approach to detect curve reversals.



Curve reversal detection within Xoptfoil. Going from left to right, the z-value of a coordinate has to change from outside the 'curve_threshold' to the "other side" of the x-axis.

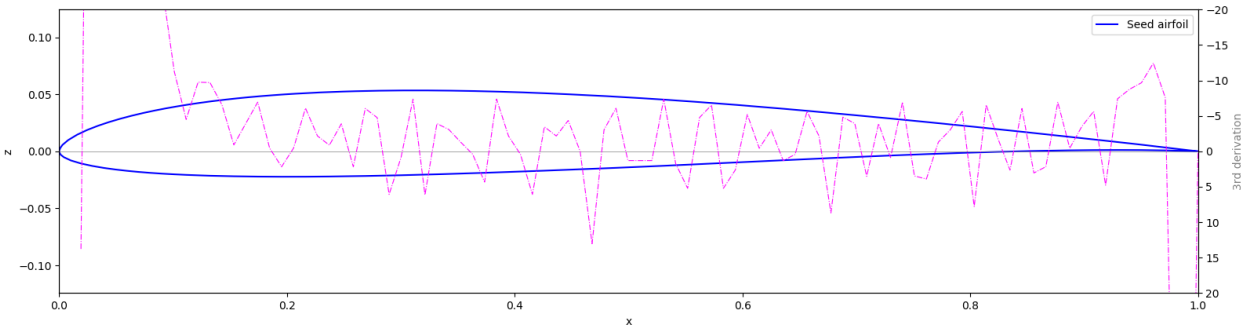
The Xoptfoil approach is tricky to detect real reversals if the "threshold area" is wide enough to absorb all the up and downs of the 2nd derivative. Reducing the threshold value will produce a lot of "false detections" caused by all this zigzag. With an increased threshold all real reversals within the "threshold area" won't be detected leading to surface with bumps...

At this point I realized my hopeless struggle against Xoptfoils "curve reversal messages". Having removed one "zig" another "zag" would pop up immediately...

And a more fundamental question arises: Are reversals the only parameter for a good surface of an airfoil?

Looking at this "Black Forest profile" it is obvious, that the reversals detected by the threshold are just one aspect. What about this many little zig zags and this highs and lows of the overall curve?

The little zig zags, I called "spikes", will get dominant in the 3rd derivative of the surface polyline. Because of their high amplitude they are easy to detect just in the same way curve reversals are detected in the 2nd derivative.



The 3rd derivation of the bottom side polyline amplifies the “zigzag” of the 2nd derivation

With some little functions implemented in Xoptfoil the number of reversals, highs and lows and spikes are counted. A little extension of the console output shows the development of these figures and a pseudo graphical view of the top and bottom side during optimization

```
     Top      P=0.83  0R  2HL  2r  5hl   ------------------------------------l---h-------------l---------h------------H--------rl---L--r---
     Bottom   P=0.65  1R  1HL  1r 15hl   ----------------------h-l--h-l-----h-l---h--l-----h--l-Rh-----l--h---l---------L-----r-------------
Obj:  0.6556        + Geo:  0.2601        + Pen:   0.000        = New:  0.9157         Improv:   8.4297%

     Top      P=1.49  0R  4HL  4r  5hl   ------------------------------------l---h-------------Hr-L-----r-h------------H--------rl---L--r---
     Bottom   P=1.20  1R  3HL  1r 14hl   ----------------------h-l--h-l-----h-l---h--l-----h--lR-h-----l-Lh-H----------l-----Lr-------------
Obj:  0.6556        + Geo:  0.2622        + Pen:   0.000        = New:  0.9178         Improv:   8.2183%

     Top      P=1.43  0R  4HL  2r  5hl   ------------------------------------l---h------------Hl-L-------h------------H--------rl---L-r----
     Bottom   P=1.35  1R  3HL  1r 17hl   ----------------------h-l--h-l-----h-l---h--l-----h--lR-h-----l-Lh-H-l----------L----r--l---h------
Obj:  0.6546        + Geo:  0.2642        + Pen:   0.000        = New:  0.9188         Improv:   8.1171%

     Top      P=1.49  0R  4HL  4r  5hl   ------------------------------------l---h-------------Hr-L-----r-h------------H--------rl---L--r---
     Bottom   P=1.58  1R  4HL  3r 15hl   ----------------------h-l--h-l-----h-l---h--l-----h--lR-h-----lL-r-H----------rl-----Lr-----h---H---
Obj:  0.6578        + Geo:  0.2631        + Pen:   0.000        = New:  0.9209         Improv:   7.9080%
```

Console output during optimization with additional information about the current surface quality on top and bottom side of the airfoil

This extended output is nice to look at - but doesn’t really help getting rid of seed airfoils reversals or even better, helps to improve the surface quality of the optimized airfoil. So the next step is nearly inevitable …

Smoothing of the airfoil surface

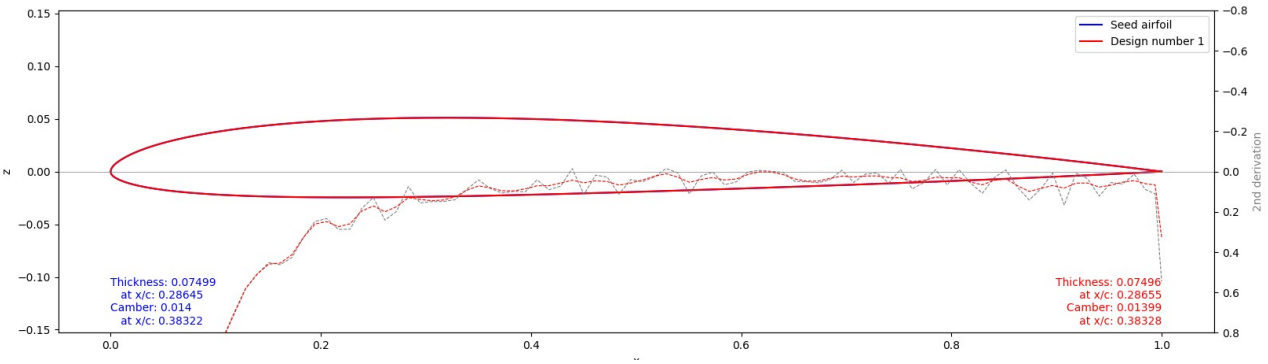
Xfoil, being used by Xoptfoil, already has some build in smoothing approaches. The most common is re-paneling, working on the coordinates, but there is also “Smoothing” or “Hanning Filter” based on QSpec distribution with a reverse transformation to the geometry. The experiences I made with re-paneling showed only in very few cases an improvement of the surface quality (which equals to the number of high/lows and spikes). The problem with Xfoils smoothing or filtering on the other side is, that both tend to perturbate the surface of the airfoil too much.

One challenge if smoothing should also be applied during Xoptfoils optimization, is to avoid a too strong interception with the shape perturbation work of the particle swarm. Smoothing should not smooth away the shapes…

The first tries were based on a simple “moving average” implementation. It looked promising but it showed out, that “moving average”

- leads to discontinuities at the start and end of the smoothed area
- is too “aggressive” to the high curvature around the leading edge
- leads to a shift of the curve from left to right …

By accident I found an article about polyline approximation with the “Chaikin or Corner Cutting algorithm”. Being a fan off simple approaches, I thought “That’s it!” and implemented this algorithm in Xoptfoil. With minor modifications to take special care of the leading edge area, the Chaikin algorithm works astonishing well. Typically after 2 to 4 loops with each having 5 Chaikin iterations, the seed airfoil is ready to go for optimization. Now Xoptfoils curvature reversal detection pops up only if there are “real” reversals…



The red dashed line shows the 2nd derivative of the smoothed airfoil having no curve reversals in opposite to the seed airfoil (dashed grey line)

Smoothing is controlled with additional parameters in the Xoptfoil input file. Mainly these are two thresholds for spike and high/low detection.

It turned out, that the Chaikin algorithm is quite careful not to perturbate the overall surface leading to a change of the aerodynamic characteristics. But it is quite efficient reducing spikes and smooth highs and lows of the curvature.

Reaching this point the final step had to be made: Integrating the assessment of the surface quality into Xoptoil optimization process. It was obvious, that Xoptfoils shape functions can’t do anything again “spikes” as their width (bump width) is typically much higher than the width of a spike. The spike reduction has to be done by smoothing which is called from time to time during optimization.

But maybe the shapes applied by the swarm could help to reduce the highs and lows of the curvature leading to a more continuous final surface (which is the sum of the “bumby” seed and the overall shapes applied)?

For this the sum of all reversals, high/lows and spikes of a surface is taken as a new design variable within the new “geometric objective function” (see above) – and here we go …

Still I’m in a testing phase with tweaking the parameters. But the results look very promising in achieving a “well curved” airfoil with very little spikes and reduced “bumps or waves”.

For the optimization process it would be better to measure the height of each curvature “high or low” instead of just dectecting the number based on a threshold. This could further reduces waves.

After all this details the tip of the day for those how got enough of “curve reversals messages” when starting a new Xoptfoil project: Just take a generated NACA airfoil as seek and let Xoptfoil do its work … ;-)

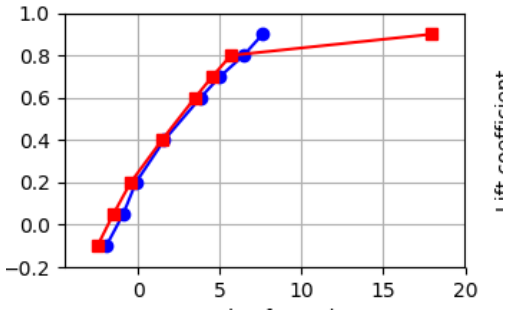
Aerodynamic issues and minor things

In the end some minor things which came along.

Flip of cd or cl during optimization

When doing optimizations with operating points at higer cl-values, it randomly occurs, that xfoil calculates an unrealistic value for cd which can be extremely low or high. I called this situations during optimization “flips” as they normally “flop” back to a normal value in the next successful iteration.

In Xoptfoil there is some clever code to detect and repair such a “flip situation”. Unfortunately it doesn’t catch all situations. Problem arise when for example the cd flipped to a very low value bringing the overall objective function to an all time best. Right after this the optimization will be caught in this pseudo minimum of the objective function.



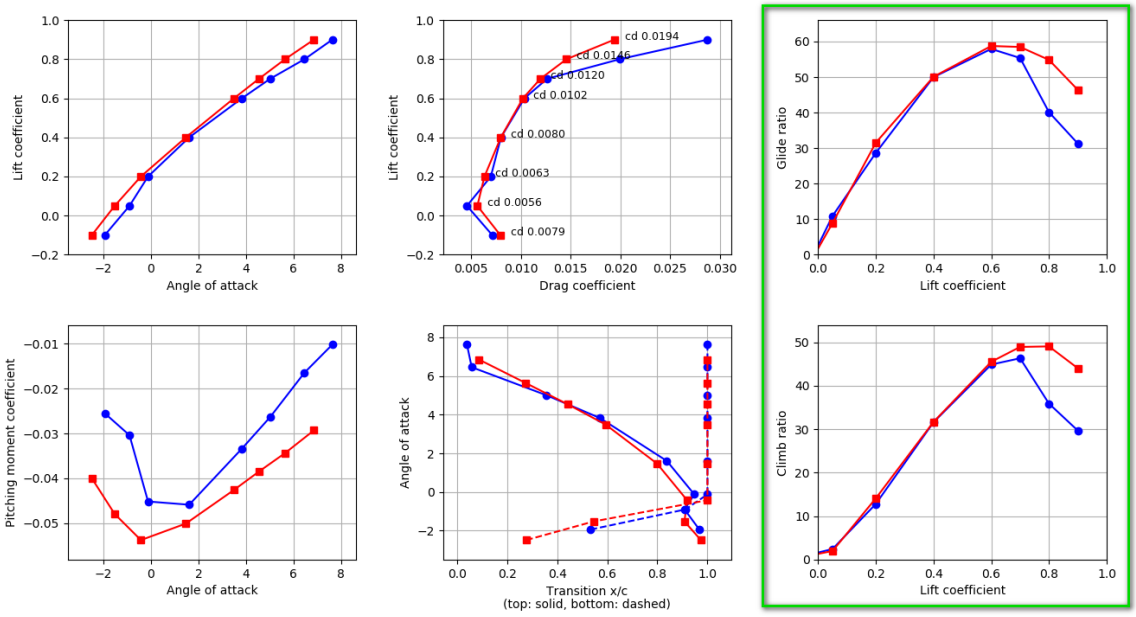
Example of a “cd flip” during optimization

I tried to improve the “flip” detection and also penalize all changes of the design variable which a greater then 20%. This strongly improved the probability of a healthy optimization run.

When using ‘xoptfoil_visualizer.py’ these “flips” still appear from time to time on the screen because the data to ‘xoptfoil_visualizer.py’ is recalculated when writing to the file and no “detect and repair” mechanism is active at that moment.

Extensions within the Visualizer

To get an earlier indication whether the optimized airfoil will have the desired aerodynamic properties the polar window of the visualizer is complemented by two sub views for glide ratio and climb ratio. Together with the printed cd values in the cl/cd diagram it helps a lot y helpful for early assessment during optimization.



Finally some remarks on software modifications in Xoptfoil

It was a funny but also challenging experience for me to step again into software development after some 15 years or so. The last Fortran coding I made with real punch cards long long time ago.

Maybe the biggest hurdle was in the beginning to set up a development environment with Visual Studio Code (very easy) and all the make and compiler tools (more tricky). After this is done it is simple to start with some mini changes, compile it and be proud that it really works.

Super easy are changes in ‘xoptfoil_visualizer.py’ written in Python. Load the program into an editor, change it, save it and restart the visualizer. Just try it...