An Interactive Airfoil Analysis and Design Tool in Matlab

Krzysztof J. Fidkowski*
University of Michigan, Ann Arbor, MI, 48188

This paper presents software design aspects, capabilities, and possible extensions of mfoil, an open-source subsonic airfoil analysis code written as a single-file Matlab class. The code uses mostly the same physical models as XFOIL to perform coupled inviscid-viscous analyses, with changes in the solver, stagnation-point treatment, and transition prediction. However, it differs in code layout, modularity, and choice and scope of variables. Furthermore, while written in an interpreted language, it takes advantage of Matlab's powerful linear algebra capabilities to prevent excessive computational costs. A graphical user interface reduces the learning curve, whereas the class structure enables direct embedding in other scripts. Examples in this paper demonstrate these capabilities, offer directions for extensions of the code, and present one such extension: adaptive paneling driven by an output error estimate.

I. Introduction

Out of many airfoil analysis and design codes that have been studied over the last several decades [1–4], XFOIL [5] has grown to be the code of choice for research and educational uses. It boasts a high degree of robustness, comes with a powerful user interface, has minimal dependencies, and has pre-built packages for different platforms. However, as any other code in such a spotlight, it is not without a wishlist. First, while it is efficiently written in Fortran, many years of changes and revisions have made it difficult for new developers to grasp the code. Short variable naming conventions, global variable scope, and a low degree of modularity make implementing changes or extensions difficult. Often, instead of linking directly to another framework, coupling with XFOIL happens inefficiently through scripts and wrappers. Such a coupling is not ideal for in-depth studies, e.g. optimization and uncertainty quantification, that may require more data from the code, such as sensitivities, access to models, or understanding how/where they are implemented. Furthermore, in education, aerodynamics students are having, at least from anecdotal observations over multiple years of instruction, a more difficult time using XFOIL due to installation issues on newer operating systems and an unfamiliar user interface. Lack of customization, e.g. in plotting, is another drawback.

In previous work, we introduced an alternative Matlab-based analysis code, mfoil [6], not to be confused with the MFOIL graphical user interface for PROFOIL [7]. The mfoil code is inspired by XFOIL and its associated publications [5, 8–10]. Whereas out previous paper introduced the tool and dove into technical details of models and the solver, this paper focuses on the software aspects of mfoil, for improving understanding of how to use the code, embed it in scripts, and extend it. The intent is to rekindle interest in airfoil analysis and design from both educational an research fronts. We therefore also highlight areas for improvement, including solver speed and robustness, model calibration, and sensitivity calculation, and we present one example of an extension into adjoint-driven adaptive paneling.

II. Code Design

The mfoil code is written as a single-file Matlab class with organized properties, variables, and functions. This section describes the capabilities and structure of the code.

A. Capabilities

Based on the methods and models in XFOIL, mfoil features similar capabilities, which are as follows:

- Coupled inviscid panel and viscous integral boundary layer equation discretization and solution, using a Newton-Raphson solver with analytical derivatives and sparse-matrix Jacobian storage.
- A nonlinear, Kármán-Tsien compressibility correction that extends the code's applicability to subcritical compressible flows.
- Geometry definition using NACA 4 and 5-digit airfoils, or airfoils from provided coordinates.

*Professor, Department of Aerospace Engineering, AIAA Associate Fellow.

- Basic geometry manipulation functionality, including the deflection of a flap, addition of a camber perturbation, and airfoil derotation.
- Inviscid or viscous solutions at a prescribed angle of attack or at a prescribed lift coefficient.
- Viscous solutions at a given Reynolds number, including transition and the possibility of separation bubbles.
- Initialization of boundary-layer variables or reuse from previous solutions.
- Access, via modular structures, to all solution and post-processing variables.
- Functions for plotting results, including viscous distributions.

B. Structure

At the highest level, the mfoil class consists of nine property structures: (1) geometry, (2) airfoil panels, (3) wake panels, (4) operating conditions, (5) inviscid solution variables, (6) viscous solution variables, (7) global system variables, (8) post-processing quantities, and (9) parameters. Each property structure consists of its own variables, which can be scalars, strings, vectors, or arrays. For example, the panel structure contains the node coordinate and spline information at the panel nodes. The split between airfoil and wake panels is made because only viscous solutions require wake panels. Similarly, the inviscid and viscous solution variables are split into separate structures since the viscous variables are not active in inviscid solutions.

The class contains over a dozen methods, starting with a constructor that builds the airfoil geometry and creates the panels. Other methods change the geometry, set operating conditions, solve the system, post-process the results, and display data. There is also a “pinging” method that verifies the analytical derivatives functions at a unit level, for example in the calculation of specific correlations, at an intermediate level, for example in the calculation of the residual on a transition panel, and at a global level after constructing the entire fully-coupled system. The verification uses second-order finite difference comparisons and checks the rate of convergence of the error.

For an mfoil class instance “m”, the listing below presents how some of the common variables and parameters are accessed through the structures.

```
% post-processed quantities are in m.post; these include
m.post.cp;     % cp distribution
m.post.cpi;    % inviscid cp distribution
m.post.cl;     % lift coefficient
m.post.cm;     % moment coefficient
m.post.cdpi;   % near-field pressure drag coefficient
m.post.cd;     % total drag coefficient
m.post.cdf;    % skin friction drag coefficient
m.post.cdp;    % pressure drag coefficient

m.post.th;     % theta = momentum thickness distribution
m.post.ds;     % delta* = displacement thickness distribution
m.post.sa;     % amplification factor/shear lag coeff distribution
m.post.ue;     % edge velocity (compressible) distribution
m.post.uei;    % inviscid edge velocity (compressible) distribution
m.post.cf;     % skin friction distribution
m.post.Ret;    % Re_theta distribution
m.post.Hk;     % kinematic shape factor distribution

% the inviscid solution (vorticity distribution) is stored in m.isol
m.isol.AIC;    % aero influence coeff matrix
m.isol.gamref; % 0,90-deg alpha vortex strengths at airfoil nodes
m.isol.gam;    % vortex strengths at airfoil nodes (for current alpha)
m.isol.xi;     % distance from the stagnation at all points

% the viscous BL solution, residual, and Jacobian are stored in m.glob
m.glob.U;      % primary states (th,ds,sa,ue) [4 x Nsys]
m.glob.dU;     % primary state update
m.glob.dalpha; % angle of attack update
m.glob.R;      % residuals [3*Nsys x 1]
m.glob.R_U;    % residual Jacobian w.r.t. primary states
m.glob.R_x;    % residual Jacobian w.r.t. xi (s-values) [3*Nsys x Nsys]
```

III. Usage

Airfoil analyses with mfoil can be performed at the Matlab command line, embedded into scripts, or through the graphical user interface. Each call to the constructor instantiates an airfoil object, which comes with its own geometry, panels, operating conditions, parameters, etc. This section presents several usage examples that demonstrate key capabilities of mfoil.

A. Command Line

When an mfoil class is instantiated, the default operating condition is zero angle of attack, inviscid flow, and no compressibility correction. These can be changed using the `setoper` function. After running the solver, a default results plot appears showing the pressure distribution on the airfoil, basic information about the run, and the airfoil geometry. The appearance of this plot can be suppressed by turning off the `doplot` parameter.

Figure 1 shows an example of an inviscid analysis of a NACA 2412 airfoil with 199 panels (200 points), first at $\alpha = 2°$ and then at a target lift coefficient of $c_\ell = 0.5$. The pressure distribution is for the prescribed lift coefficient case.

```
% start with a 4-digit NACA airfoil
m = mfoil('naca','2412', 'npanel',199);

% set angle of attack to 2 degrees
m.setoper('alpha',2);

% run the solver, plot the results
m.solve

% now set a constant lift coefficient
m.setoper('cl',0.5);

% run the solver, plot the results
m.solve
```

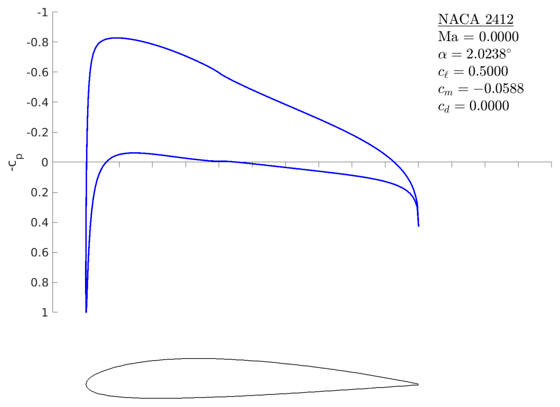


Fig. 1 Inviscid analysis of a NACA 2412 airfoil.

Next, Figure 2 demonstrates a viscous analysis of the same airfoil at $\alpha = 2°$, and Reynolds number $Re = 10^6$. Setting the Reynolds number turns on the viscous mode, which includes laminar, transition, and turbulence models. To go back to inviscid mode, the key-value "visc = false" can be specified when setting operating conditions, or the flag can be directly changed in the operating condition structure: `m.oper.viscous=false`. The viscous results plot now shows values for additional coefficients: the skin friction and pressure drag coefficients. Also, the pressure coefficient on the results plot shows two sets of results, one from an inviscid analysis as a dotted line, and one from a viscous analysis as a solid line. The surfaces are color coded: red for the lower, blue for the upper, and black for the wake. The airfoil plot also now depicts the boundary layer through lines offset from the geometry by the displacement thickness.

The airfoil can also be loaded from a coordinate file, as shown in Figure 3. The file contains the (x, y) pairs of points on the airfoil, starting and ending at the trailing edge and ordered either clockwise or counterclockwise. If there is no gap, the first/last points should be duplicated in the coordinate file. In this example, after loading the airfoil, a flap is applied by calling the `geom_flap` function, which takes in the hinge point and flap deflection angle, in degrees, positive down. To keep the chordline horizontal, the geometry is derotated after the flap deflection, through a call to `geom_rotate`.

Calling `m.geom_addcamber` changes the vertical position of each point on the airfoil surface by an interpolation of the provided (x, z) camberline coordinates. If the airfoil already has camber, the camberline is incremented, not replaced. Figure 4 demonstrates such a modification of an airfoil that starts out as a NACA 0008.

The coupled inviscid-viscous solver uses a Newton-Raphson method that can fail when the initial guess, which comes from an uncoupled initialization, is not very good. This generally happens at high angles of attack or low Reynolds numbers, when the flow is in danger of separation. Using a converged solution that is close to the desired one (e.g. at a lower angle of attack) can help, as shown in Figure 5. To enable reuse of the boundary-layer solution, the boundary-layer state initialization can be turned off via `m.oper.initbl = false`.

```
% start with a 4-digit NACA airfoil
m = mfoil('naca','2412', 'npanel',199);

% set angle of attack and Reynolds
    number
m.setoper('alpha',2, 'Re',10^6);

% run the solver, plot the results
m.solve
```

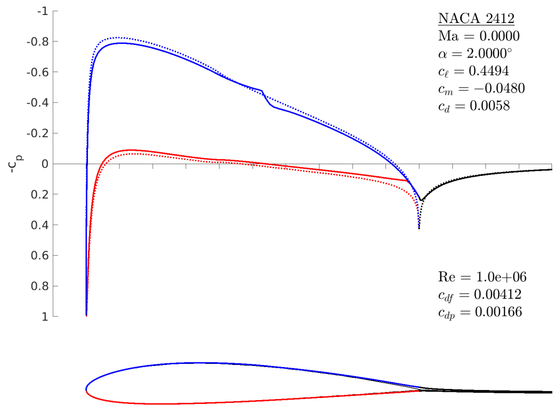


Fig. 2 Viscous analysis of a NACA 2412 airfoil.

```
% import RAE 2822 airfoil coordinates
X = load('rae.txt');

% make/panel an airfoil out of these
    coordinates
m = mfoil('coords', X, 'npanel',199)

% introduce a flap at x=0.8, y=0,
    10deg down
m.geom_flap([0.8,0], 10)

% derotate: make the chord line
    horizontal
m.geom_derotate

% set operating conditions
m.setoper('alpha',1, 'Re',10^6);

% run the solver, plot the results
m.solve
```

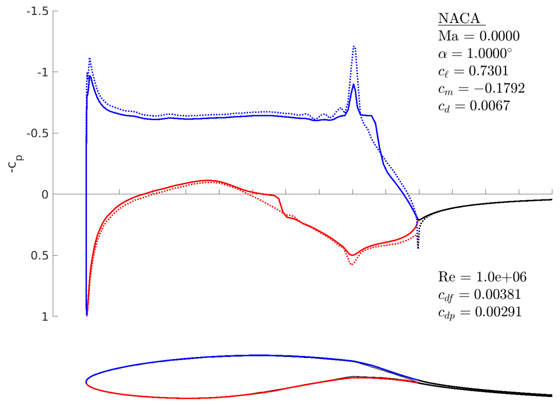


Fig. 3 RAE 2822 coordinate file input and flap analysis.

```
% start with a symmetric NACA airfoil
m = mfoil('naca','0008', 'npanel',199);

% make points for a camber line
    addition
x = linspace(0,1,101);
z = 0.03*sin(2*pi*x);

% add the camber to the airfoil
m.geom_addcamber([x;z]);

% set operating conditions
m.setoper('alpha',2, 'Re',10^6)

% run the solver, plot the results
m.solve
```

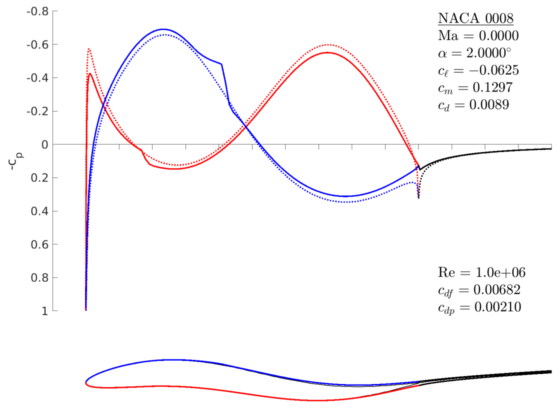


Fig. 4 Camber modification of a thin airfoil.

```
% start with a 4-digit NACA airfoil
m = mfoil('naca','2412', 'npanel',199);

% set a difficult operating condition
m.setoper('alpha',9, 'Re',10^6);

% try the solver ... does not converge
    (see right, top)
m.solve

% set a related easier operating
    condition
m.setoper('alpha',5, 'Re',10^6);

% run the solver
m.solve

% turn off BL initialization
m.oper.initbl = false;

% set the difficult operating condition
m.setoper('alpha',9, 'Re',10^6);

% run the solver: now converges (see
    right, bottom)
m.solve
```

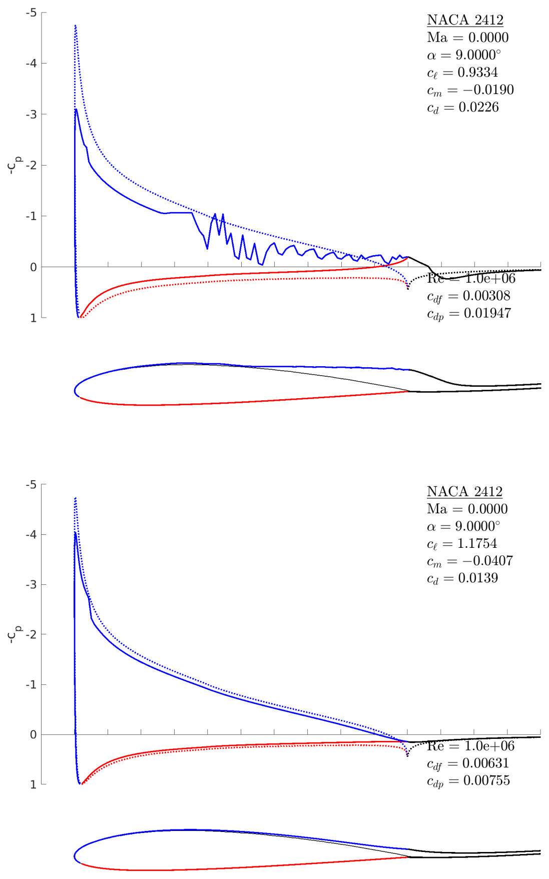
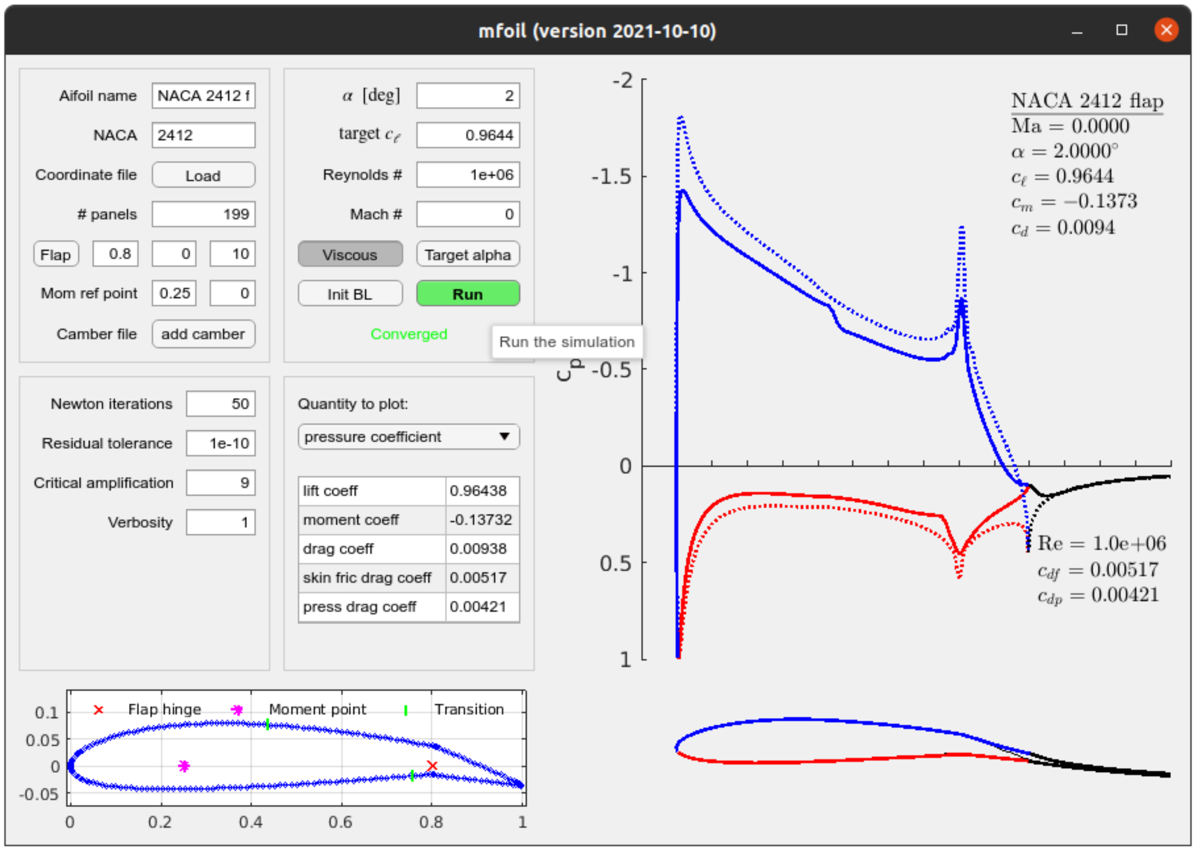


Fig. 5 Addressing convergence problems for a difficult operating condition through boundary-layer solution continuation.

B. Graphical User Interface

A graphical user interface has also been created for working with the mfoil class, using Matlab's `uifigure` capability. The layout of the user-interface (UI) figure can be customized and can include components such as labels, editable fields, buttons, output texts, and graphs. Run-time hints and tips are provided through text displayed by hovering the mouse over buttons. A single mfoil class is instantiated when the UI figure is created, and call-back functions follow button presses and field edits to call methods of the class.

Figure 6 shows the layout used for the mfoil interface. The top-left panel pertains to the airfoil geometry and paneling. NACA digits can be specified or a coordinate file, clockwise or counterclockwise order, can be loaded. A flap can be introduced by specifying the hinge location and deflection angle, and camber modifications can be loaded from a file. Next is the operating-conditions tab, in which the angle of attack, lift coefficient, Reynolds number, and Mach number are specified. The user can switch between viscous and inviscid modes, and whether the simulation is run at a given angle of attack or a fixed lift coefficient. At each run, the boundary layer can be initialized or re-used from the previous simulation, and this is controlled through a toggle button.


mfoil (version 2021-10-10)

Aifoil name	NACA 2412 f
NACA	2412
Coordinate file	Load
# panels	199
Flap	0.8 0 10
Mom ref point	0.25 0
Camber file	add camber

α [deg]	2
target c_ℓ	0.9644
Reynolds #	1e+06
Mach #	0
Viscous	Target alpha
Init BL	Run

Converged

Run the simulation

Newton iterations	50
Residual tolerance	1e-10
Critical amplification	9
Verbosity	1

Quantity to plot: pressure coefficient

lift coeff	0.96438
moment coeff	-0.13732
drag coeff	0.00938
skin fric drag coeff	0.00517
press drag coeff	0.00421

NACA 2412 flap
$\mathrm{Ma} = 0.0000$
$\alpha = 2.0000^\circ$
$c_\ell = 0.9644$
$c_m = -0.1373$
$c_d = 0.0094$
$\mathrm{Re} = 1.0e{+}06$
$c_{df} = 0.00517$
$c_{dp} = 0.00421$

Fig. 6 Graphical user interface layout and a converged result.

The lower-left panel contains editable fields in which some key solver parameters are specified. Many more are "under-the-hood", but still accessible through the class. These can be added to the user interface as needed in the future. The last panel pertains to outputs: it provides a text table of basic scalar quantities computed from the current solution, and a drop-down selection of which quantity to plot on the main plotting window, on the right of the UI figure. The baseline pressure coefficient plot mimics the display of XFOIL: it shows inviscid and viscous pressure coefficients overlaid and color-coded by surface, text of the operational and output scalars, and an airfoil plot with boundary-layer displacement thickness lines. Additional quantities, such as viscous distributions, can also be plotted.

Finally, the bottom-left portion of the interface displays the airfoil paneling and key points, such as the moment reference point, the flap hinge, and location(s) of transition. Just as mfoil itself, the graphical user interface is open-source and can be customized to add or remove components to better suit a particular workflow. For example, more geometry manipulation functions can be added for a design application, or more parameters could be exposed for trade or uncertainty quantification studies.

IV. Post-Processing

Outside of the graphical user interface, post-processing can be done manually by directly accessing desired variables from the class, or through certain built-in plotting functions that are part of the class. The latter include the baseline pressure-coefficient and results plot shown in the previous usage figures, airfoil geometry and panel plots, boundary-layer plots, velocity vectors (`plot_velocity`) and streamfunction contours (`plot_stream`) shown in Figure 7, and a viscous distribution plot (`plot_distributions`) shown in Figure 8.

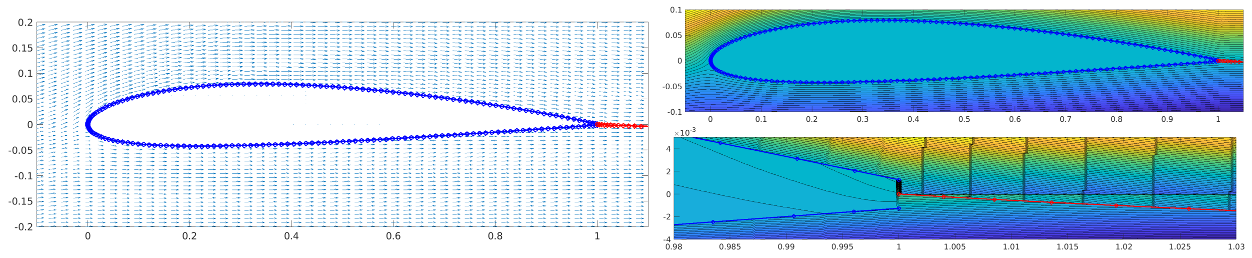


Fig. 7 Sample velocity vector and streamfunction plots.

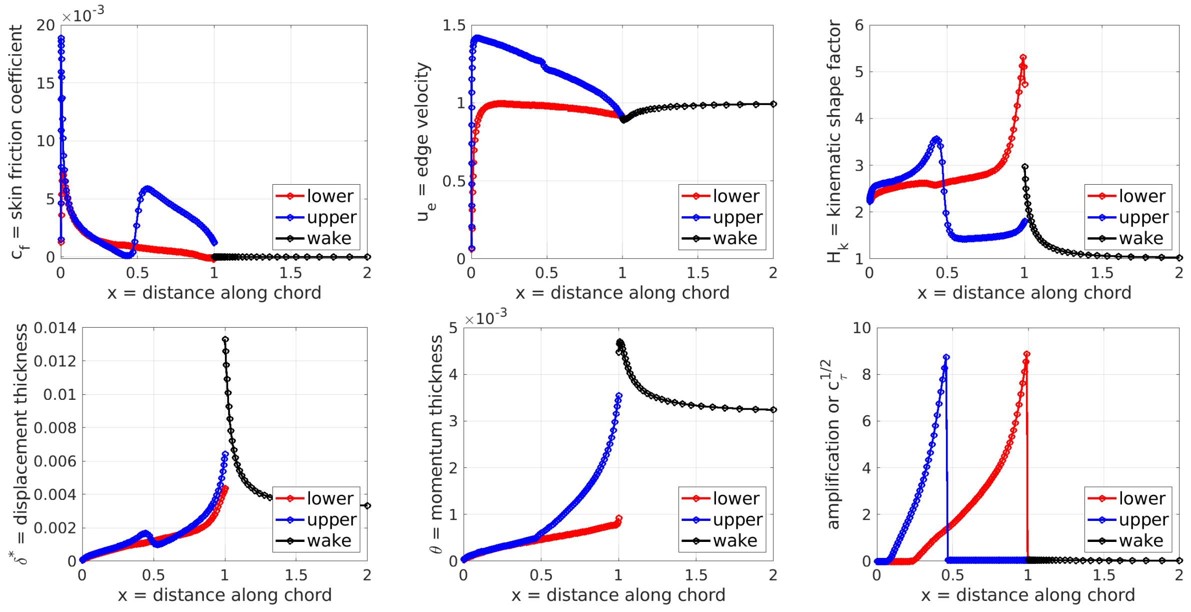


Fig. 8 Viscous graphs generated by mfoil's default plotting function.

V. Extensions

The mfoil code comes with a solver, basic geometry creation and manipulation capabilities, and post-processing functions. Its modular structure makes extensions possible, ideas for some of which are outlined below.

1) More accurate discretization: the current code assumes a wake of prescribed length whose position is determined by the inviscid solution. Using the coupled solution to define the wake in an iterative manner could improve the accuracy of viscous simulations at high angles of attack.
2) Adjoints: the code already has analytical derivatives of the residuals for the Newton-Raphson solver, and hence an adjoint solver, using the transpose of the residual Jacobian matrix, would be a relatively straightforward

extension. The adjoint could be used for gradient-based shape optimization, uncertainty quantification, and error estimation.

3) Adaptive paneling: airfoil panels are currently positioned based on a user-prescribed number of panels and geometry curvature constraints. A large number of panels increases solution costs and can make initial solutions difficult to obtain. The panels could instead be placed adaptively, e.g. with increased resolution near an interesting location such as transition, to improve accuracy. This adaptation could be driven by adjoint-based error estimates.
4) Solver robustness: convergence of the Newton-Raphson coupled solver depends on the initial state, particularly for the boundary-layer variables, and current initialization techniques can lead to solver failure. More sophisticated initialization strategies, including parameter, solution, and mesh continuation, can be investigated to improve the solver robustness.
5) Uncertainty quantification and modeling: many of the current models are based on experimental and numerical data available when XFOIL was written. Data from new experiments and high-resolution computations could be used to perform uncertainty quantification studies, recalibrate models, or add new ones, especially for transition prediction and turbulent shear stress evolution.

In this paper we investigate one of these extensions, adaptive meshing using adjoint-weighted residuals.

VI. Adjoints and Adaptive Paneling

Output-based adaptation techniques using adjoints have been investigated in many previous works, primarily in the context of finite-volume and finite-element discretizations of partial-differential equations [11–13]. They offer accuracy improvements through discretization error estimates, and error control through localized adaptive indicators. The resulting adapted meshes yield more accurate outputs at lower computational costs compared to uniform refinement or heuristic adaptive strategies.

Compared to finite-volume or finite-element discretizations, panel methods are much less computationally demanding. However, their computational cost is not trivial for coupled inviscid-viscous solutions, particularly when running with a fine paneling in an interpreted-language implementation, as is the case with mfoil. The computational cost is also amplified in a many-query setting, such as optimization or design-space exploration. An output-based adaptive approach can reduce these costs while providing an estimate of the output error.

Adaptive discretizations are of limited use when the error is mostly evenly distributed on a uniform mesh. In panel methods, the panel sizes are generally set using curvature-based criteria, in order to reduce errors in the geometry approximation. Such an approach is usually sufficient for the pure potential-flow case, in which the interesting leading and trailing edge areas are refined well with curvature-based criteria. However, in the viscous case, additional areas that may require refinement include separation, where the shape factor changes rapidly, and transition to turbulence. The panel size requirements for laminar and turbulent regions may also be different. For these reasons, we presently focus on the viscous discretization in the adaptation.

A. Adjoints

For a linear system, a discrete adjoint solution can be computed by solving the transposed system driven by the linearization of a chosen output. In the nonlinear case, the same procedure can be applied to the linearized system. In the coupled inviscid-viscous solver, the nonlinear system can be written as [6]

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{4N}$ is the state, $\mathbf{R} \in \mathbb{R}^{4N}$ is the residual, and $N = N_a + N_w$ is the total number of nodes on the airfoil, N_a, and wake, N_w. At each node i, the state vector consists of four quantities, $\mathbf{u}_i = [\theta, \delta^*, \tilde{n} \text{ or } c_\tau^{1/2}, u_e]^T$: the momentum thickness, displacement thickness, amplification factor or shear stress, and edge velocity. To interpret the third state, a separate flag indicates whether a node is laminar or turbulent. The system of discrete residuals, Eqn. 1, comes from a finite-difference discretization of the governing boundary-layer equations, together with a coupling to the inviscid panel solver through the edge velocity, as explained in [6].

For a scalar output J computed from the state vector $\mathbf{U}$, the discrete adjoint vector $\mathbf{\Psi}$ satisfies

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{\Psi} = \frac{\partial J}{\partial \mathbf{U}}^T. \tag{2}$$

The discrete adjoint can be used for efficient calculation of derivatives, for example in optimization, when a gradient is required with respect to geometry parameters. However, for error estimation and adaptation, we require an adjoint that can be interpreted consistently across discretizations, i.e. one that emulates the continuous adjoint, and this is not the case when the residuals come from finite-difference approximations. For a set of differential equations, $\mathbf{r}(\mathbf{u}) = \mathbf{0}$, the continuous adjoint comes from a variational formulation. We define the augmented Lagrangian as

$$\mathcal{L} \equiv J - \int_\Omega \boldsymbol{\psi}^T \mathbf{r}(\mathbf{u})\, d\Omega, \tag{3}$$

where Ω is the computational domain. Stationarity of the Lagrangian with respect to state variations then gives a differential equation and boundary conditions for the continuous adjoint, $\boldsymbol{\psi}$. While we do not use a variational formulation, we can emulate one in the finite-difference method by writing a discrete approximation of the integral in the Lagrangian,

$$\mathcal{L} \equiv J - \sum_i \widetilde{\boldsymbol{\Psi}}_i^T \mathbf{R}_i\, \Delta\Omega_i, \tag{4}$$

where the summation is over the discrete unknowns and $\Delta\Omega_i$ accounts for the portion of the domain volume attributed to unknown i. In this equation, $\widetilde{\boldsymbol{\Psi}}_i$ corresponds to an approximation of the continuous adjoint $\boldsymbol{\psi}$ at node i, and this correspondence relies on the discrete residuals $\mathbf{R}_i$ being point-wise approximations of $\mathbf{r}(\mathbf{u})$.

In the specific case of the boundary-layer equations in the coupled solver, the residuals, and hence adjoints, are associated with panels. We denote the size of panel i, the equivalent of $\Delta\Omega_i$, by $\Delta\xi_i$, as ξ is the streamwise direction coordinate along the airfoil surface. Since the discrete adjoint equation, Eqn. 2, arises from a Lagrangian without this interval size weight, the mapping between the discrete and approximate continuous adjoint components is $\widetilde{\boldsymbol{\Psi}}_i = \boldsymbol{\Psi}_i / \Delta\xi_i$.

The summation in the approximate Lagrangian in Eqn. 4 is over the airfoil and wake panels, with three boundary-layer equations per panel. We do not account for the inviscid-viscous coupling residuals, which amount to one extra equation per node, in the Lagrangian. This coupling is present in the solution of the discrete adjoint problem via Eqn. 2, but the coupling residuals are not included in the Lagrangian or error estimate. This is because the potential flow equations are not in the form of a differential equation system, and hence they do not lend themselves directly to a continuous adjoint analogy.

As a test case, we consider the NACA 2410 airfoil at angle of attack $\alpha = 3°$ and Reynolds number $Re = 10^5$. The output of interest is the total drag coefficient, $J = c_d$. Figure 9 shows the primal solution, in the form of the skin friction distribution, and the three components of the adjoint solution, $\widetilde{\boldsymbol{\Psi}}_i$, corresponding to the three boundary-layer equation residuals per panel. The adjoints are relatively smooth, and the normalization by the panel size makes the adjoint fields mesh independent. Notable features of the adjoint are (1) a sudden jump in the amp/lag adjoint component at transition, due to the change in the equations from the amplification factor to the turbulent shear lag equation; and (2) large magnitude values of the momentum and shape-factor adjoint components on the upper surface, near the point of separation.

B. Output Error Estimation and Adaptation

The Lagrangian in Eqn. 4 gives an equation for the output error estimate in the presence of nonzero residuals,

$$\delta J = \sum_i \widetilde{\boldsymbol{\Psi}}_i^T \delta\mathbf{R}_i\, \Delta\xi_i. \tag{5}$$

This is the adjoint-weighted-residual (AWR) formula. The nonzero residuals arise from an analysis of the converged state prolongated into a finer space: here obtained by uniform refinement of each panel. Only the three boundary-layer equations per panel participate in this residual calculation, and no coupled solution is performed on this fine space – i.e. residuals are evaluated, but not driven to zero. In addition, we only consider the airfoil panels for the output error estimate, as for the meshing parameters chosen, we found that the wake resolution was sufficient to keep the wake discretization error an order of magnitude smaller than the airfoil discretization error.

For the purpose of adaptation, in the AWR formula, we place absolute value signs around the contributions of each panel to the total error, in order to obtain a conservative error estimate and an adaptive indicator,

$$\epsilon_i = |\widetilde{\boldsymbol{\Psi}}_i^T \delta\mathbf{R}_i\, \Delta\xi_i|. \tag{6}$$

Figure 10 shows the resulting indicator and the adapted paneling. We see that the adaptive indicator reaches large values at the leading/trailing edges, near the point of separation, and at/after the point of transition. This error indicator is then

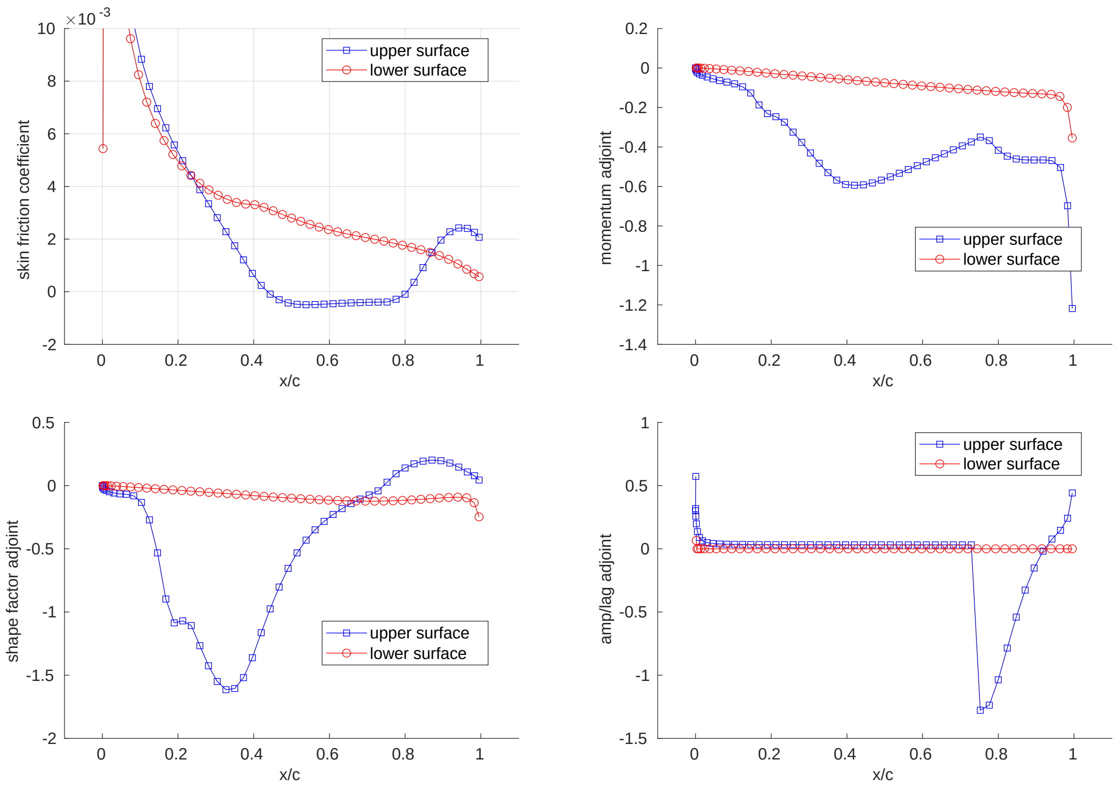


Fig. 9 Skin-friction coefficient primal solution and boundary-layer residual components of the drag coefficient adjoint, for the adaptive case.

used to redistribute the nodes forming the airfoil panels. This redistribution is similar to the temporal discretization adaptation presented in [14]. Specifically, an a priori error convergence rate is assumed, with a rate of 2.0, in order to convert the adaptive indicator on each panel into a desired panel size, employing error equidistribution and a fixed number of panels. The adapted paneling in Figure 10 shows refinement in the expected areas based on the adaptive indicator.

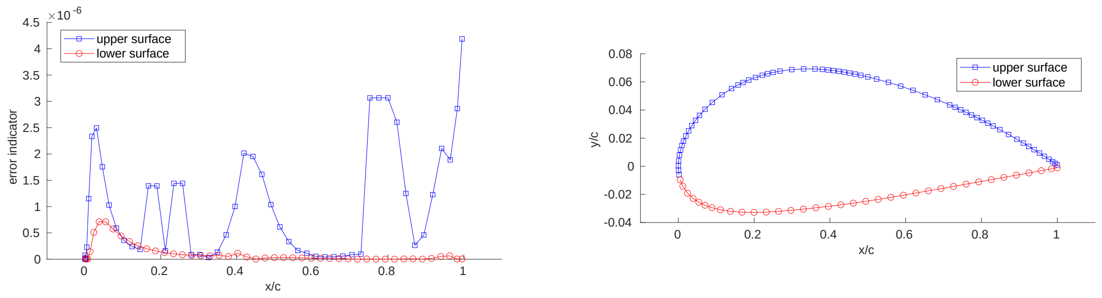


Fig. 10 Error indicator and adapted panels for the adaptive case.

Figure 11 presents the analysis results of the airfoil with the original and adapted panelings. Both discretizations contain 99 panels, which translates to 100 points, on the airfoil. A truth value for the drag coefficient for this case is computed using 400 points on the airfoil, and this value is $c_{d,\text{truth}} = 0.01471$. We see that the adaptation decreases the drag coefficient error, as the drag coefficient changes from .014837 on the un-adapted mesh to .014746 on the adapted one. This is almost a four-fold reduction in the output error, in one adaptive iteration at a fixed discretization size.

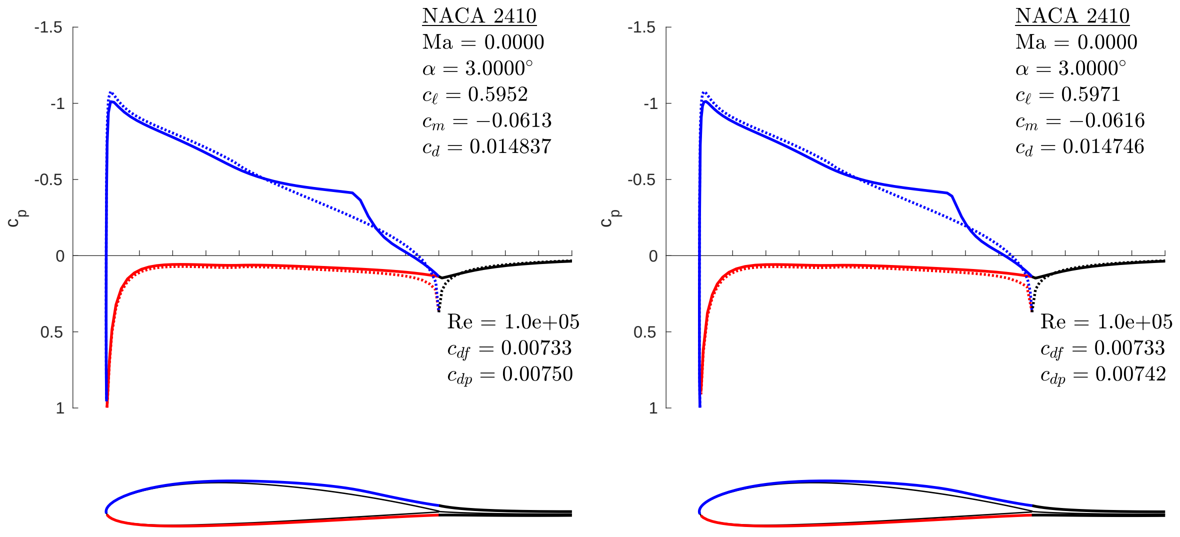


Fig. 11 Results from the original and adapted panelings, using N=99 panels.

VII. Summary and Conclusions

This paper describes the software design aspects of mfoil, a Matlab implementation of a coupled inviscid-viscous airfoil solver for low-speed aerodynamics. Implemented as a single Matlab class and made open-source available, mfoil is inspired by XFOIL and provides an accessible and extensible implementation in a popular interpreted programming environment. Usage modes can range from basic analysis via the graphical user interface to direct embedding in more complex scripts and frameworks, with access to inner structures and variables of the mfoil code. The present examples demonstrate these modes, discuss ideas for extensions, and presents one of these: adaptive paneling driven by an adjoint-weighted-residual error estimate.

References

[1] Bauer, F., Garabedian, P., Korn, D., and Jameson, A., "Supercritical Wing Sections II," *Lecture Notes in Economics and Mathematical Systems*, Vol. 108, Springer-Verlag Berlin Heidelberg New York, 1975, pp. 1–289. https://doi.org/10.1007/978-3-642-48912-9.

[2] Eppler, R., and Somers, D. M., "A Computer Program for the Design and Analysis of Low-Speed Airfoils," NASA Technical Memorandum 80210, 1980.

[3] Eppler, R., and Somers, D. M., "Supplement to: A Computer Program for the Design and Analysis of Low-Speed Airfoils," NASA Technical Memorandum 81862, 1980.

[4] Melnik, R., Brook, J., and Mead, H., "GRUMFOIL: A Computer Code for the Computation of Viscous Transonic Flow over Airfoils," AIAA Paper 87-0414, 1987. https://doi.org/10.2514/6.1987-414.

[5] Drela, M., "XFOIL: An analysis and design system for low Reynolds number airfoils," *Low Reynolds Number Aerodynamics, Lecture Notes in Engineering*, Vol. 54, edited by T. Mueller, Springer Berlin Heidelberg, 1989, pp. 1–12. https://doi.org/10.1007/978-3-642-84010-4_1.

[6] Fidkowski, K. J., "A Coupled Inviscid-Viscous Airfoil Analysis Solver, Revisited," *AIAA Journal*, Vol. 60, No. 5, 2022, pp. 2961–2971. https://doi.org/10.2514/1.J061341.

[7] Ragheb, A. M., and Selig, M. S., "Multi-Element Airfoil Configurations for Wind Turbines," AIAA Paper 2011-3971, 2011. https://doi.org/10.2514/6.2011-3971.

[8] Drela, M., and Giles, M. B., "Viscous-Inviscid Analysis of Transonic and Low Reynolds Number Airfoils," *AIAA Journal*, Vol. 25, No. 10, 1987, pp. 1347–1355. https://doi.org/10.2514/3.9789.

[9] Giles, M. B., and Drela, M., "Two-Dimensional Transonic Aerodynamic Design Method," *AIAA Journal*, Vol. 25, No. 9, 1987, pp. 1199–1206. https://doi.org/10.2514/3.9768.

[10] Drela, M., and Giles, M., "ISES: A Two-Dimensional Viscous Aerodynamic Design and Analysis Code," AIAA Paper 87-0424, 1987. https://doi.org/10.2514/6.1987-424.

[11] Becker, R., and Rannacher, R., "An optimal control approach to a posteriori error estimation in finite element methods," *Acta Numerica*, edited by A. Iserles, Cambridge University Press, 2001, pp. 1–102.

[12] Venditti, D. A., and Darmofal, D. L., "Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows," *Journal of Computational Physics*, Vol. 187, No. 1, 2003, pp. 22–46.

[13] Fidkowski, K. J., and Darmofal, D. L., "Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics," *AIAA Journal*, Vol. 49, No. 4, 2011, pp. 673–694. https://doi.org/10.2514/1.J050073.

[14] Kast, S. M., and Fidkowski, K. J., "Output-based Mesh Adaptation for High Order Navier-Stokes Simulations on Deformable Domains," *Journal of Computational Physics*, Vol. 252, No. 1, 2013, pp. 468–494. https://doi.org/10.1016/j.jcp.2013.06.007.