Aerodynamic Shape Optimization Using Symbolic Sensitivity Analysis

Elham, Ali; van Tooren, Michel

DOI
10.2514/6.2017-0359

Publication date
2017

Document Version
Accepted author manuscript

Published in
58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference

Postprint of: Ali Elham,. "Aerodynamic Shape Optimization Using Symbolic Sensitivity Analysis", 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum, (AIAA 2017-0359)
http://dx.doi.org/10.2514/6.2017-0359

Aerodynamic Shape Optimization Using Symbolic Sensitivity Analysis

Ali Elham *
Delft University of Technology, 2629HS Delft, The Netherlands
Michel J.L. van Tooren †
University of South Carolina, Columbia, SC 29201 USA

The least-squares finite element method is used to solve the compressible Euler equations around airfoils in transonic regime. The symbolic analysis method is used to generate the element stiffness and force matrices. The equations of the element matrices are derived symbolically based on the flow primitive variables and the position of the element nodes. The symbolic analysis is also used to compute the exact derivatives of the residuals with respect to both design variables (e.g. the airfoil geometry) and the state variables (e.g. the flow velocity). The symbolic analysis allows to compute the exact Jacobian of the governing equations in a computationally efficient way, which is used for Newton iteration. Besides, using the symbolic analysis the sensitivities of the outputs, such as the airfoil drag, with respect to the design variables, such as the airfoil geometry, are computed using the discrete adjoint method without the need for automatic differentiation. This makes the analysis and optimization computationally more efficient.

I. Introduction

An aerodynamic shape optimization is performed by a combined use of computational fluid dynamics (CFD) and numerical optimization techniques. Such an optimization includes hundreds of design variables. Although applications of heuristic optimization algorithms for aerodynamic shape optimization can be found in literature,[1,2] gradient based algorithms are still the most efficient methods for shape optimizations based on high fidelity CFD analysis including large number of design variables. However a sensitivity analysis is required for using gradient based algorithms.

Adjoint methods for sensitivity analysis have been widely used in recent years for aerodynamic shape optimization. Both continuous[5–7] and discrete adjoint[3,4] methods have been used. Application of the continuous adjoint faces some difficulties. In the continuous adjoint first the flow equations are differentiated and then discretized. Therefore there can be a mismatch between the gradient computed using the continuous adjoint and the discrete gradient. For example the continuous adjoint results in erroneous gradient close to the wing sharp trailing edge.[6] On the other hand discrete adjoint applies the differentiation to the discredited flow equations. However computing the partial derivatives, that are requited to use the discrete adjoint method, is a challenge. Automatic differentiation (AD) in the reverse mode is used to solve this issue.[3,4] Although using AD enormously reduces the human effort in developing adjoint CFD codes, its memory requirements negatively affects the computational efficiency.[8]

Recent development in software like Mathematica, Matlab, Maple and Mathcad allows symbolic analysis. Using this capability, analysis such as integration or differentiation can be done symbolically. In this research symbolic analysis is used to develop a CFD code including discrete adjoint sensitivity analysis. All the partial derivatives required for such a purpose were initially derived using symbolic analysis and then hard-coded in the code. Using this approach no AD is required, which makes the analysis and optimization more efficient.

*Postdoc Researcher, Flight Performance and Propulsion, Faculty of Aerospace Engineering, AIAA Member.
†Professor, Ronald E. McNair Center for Aerospace Innovation and Research, AIAA Member.

II. Finite element formulation

The least-squares finite element method is used to solve the compressible Euler equations, as proposed by Jiang and Carey.[9] Two dimensional unsteady compressible Euler equations can be presented in the following matrix form:

$$\frac{dU}{dt} + A_1 \frac{dU}{dx} + A_2 \frac{dU}{dy} = 0 \tag{1}$$

where $U = [\rho, u, v, p]^T$ is the vector of the primitive variables and the A_1 and A_2 matrices are as follows:

$$A_1 = \begin{bmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \rho^{-1} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{bmatrix} \tag{2}$$

$$A_2 = \begin{bmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \rho^{-1} \\ 0 & 0 & \gamma p & v \end{bmatrix} \tag{3}$$

where u and v are the velocity components, ρ is the density, p is the pressure and γ is the specific heat ratio. Equation (1) can be linearized by setting $A^n = A(U^n)$. An implicit time differentiation results in the following equation:

$$R = U^{n+1} - U^n + \Delta t A_1^n \frac{\partial U^{n+1}}{\partial x} + \Delta t A_2^n \frac{\partial U^{n+1}}{\partial y} = 0 \tag{4}$$

Using a least-square finite element method,[13] the L^2-norm of the residual R is minimized, i.e.:

$$\min \quad \Phi = \int_\Omega R^T R \; dxdy \tag{5}$$

In a finite element approach the primitive variables inside each element are approximated using the so-called shape functions as follows:

$$\tilde{U} = \sum_{i=1}^{nn} N_i U_i \tag{6}$$

where nn is the number of nodes for each element, N_i is the i^{th} shape function and U_i is the value of the primitive variables at node i. Using the least-squares finite element method the following element stiffness and force matrices are obtained:

$$K_{ij}^e = \int_{\Omega_e} (LN_i)^T (LN_j) dxdy \tag{7}$$

$$F_{ij}^e = \int_{\Omega_e} (LN_i)^T U^n dxdy \tag{8}$$

where

$$LN_i = N_i I + \Delta t \frac{\partial N_i}{\partial x} A_1^n + \Delta t \frac{\partial N_i}{\partial y} A_2^n \tag{9}$$

I is the identity matrix. In finite element analysis the element matrices are usually computed using numerical integration techniques such as Gauss Quadrature.[10] This approach requires multiple time calculation of the integral function and also introduces numerical error. In this research the symbolic analysis technique is used to derive the exact solution of the integrals shown in Eqs. (7) and (8).

In this research three nodes linear triangular elements are used, where the shape functions are defined based on the local coordinates L_1, L_2 and L_3 as shown in Eq. (10).

$$L_1 = \frac{A_1}{A} \qquad (10)$$
$$L_2 = \frac{A_2}{A}$$
$$L_3 = \frac{A_3}{A}$$

where A is the area of the triangle, and A_1, A_2, and A_3 are the areas of the three sub-triangles as shown in Fig.1. From this figure one can observe that $A_1 + A_2 + A_3 = A$ and $L_1 + L_2 + L_3 = 1$. The shape functions for this type of element are defined as:

$$N_1 = L_1 \qquad (11)$$
$$N_2 = L_2$$
$$N_3 = 1 - L_2 - L_2$$

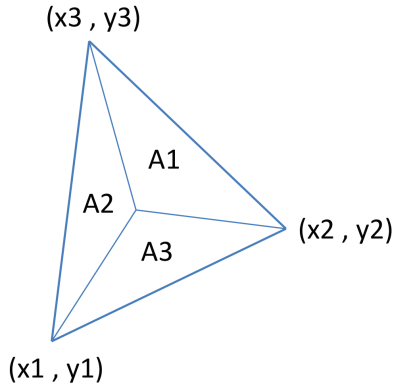


Figure 1: Triangular element local coordinates.

To derive the equations of K^e and F^e using symbolic analysis the x and y positions of the element nodes (for 2D elements) as well as the flow variables ρ, u, v and p are defined as symbols. Matlab symbolic analysis toolbox is used to derive the solution of Eqs. (7) and (8) symbolically. Lines 1 to 62 of the Matlab code shown in Appendix are used to derive K^e and F^e.

The resulting symbolic equations for K and F were subsequently hard-coded in a function for computing the element matrices. This function receives the x and y positions of the nodes (x_1, y_1 to x_3, y_3 in Appendix) and the values of the flow variables at each node (ρ_1, u_1, v_1, p_1 to ρ_3, u_3, v_3, p_3 in Appendix) as well as the time step (dt in Appendix) and returns the stiffness and the force matrices for each element. Using the proper assembly the global stiffness and force matrices are generated.

Two types of boundary conditions are used for airfoil analysis; the far field boundary condition, where the pressure, velocity and density are given, and the wall boundary condition, where for inviscid flow the normal velocity to the wall is set to zero. Applying the first type of boundary condition is easy. The values of the variables of the nodes placed on this boundary are defined as fixed degrees of freedom. However the second type of boundary condition requires some attention. In order to apply the wall condition the local angle of the wall at each node, θ, is first determined from the airfoil geometry, see Section IV. A transformation matrix, T is defined based on θ of the nodes on the walls. In order to compute K and F for the elements with at least one node on the wall, the coordinate of the velocity variables of the node(s) on the wall are rotated in such a way to have tangential and normal velocity to the wall instead of u and v in x and y directions respectively. This transformation is done using the T matrix. The normal velocity of the nodes on the wall is set to zero and defined as fixed degree of freedom.

III. Solving the nonlinear system of equations

Both Picard and Newton methods are used to solve the nonlinear system $R(U) = K(U)U - F(U) = 0$. The Newton method has a larger rate of convergence compared to the Picard method, but it has a smaller radius of convergence.[11] Therefore the iteration is started using the Picard method and after a few iteration, the Newton method is used.

Using the Picard iteration method, in this case, is identical to using the Newton method to solve the linearized system $R^{n+1} = K(U^n)U^{n+1} - F(U^n) = 0$, where $\partial R/\partial U$ is equal to $K(U^n)$. In such an approach first the stiffness and force matrices are evaluated using the U vector from the previous iteration (or the initial guess at the first iteration). Then ΔU (the change in U) is computed as:

$$\Delta U = -\frac{\partial R}{\partial U}^{-1} R(U) \tag{12}$$

where as mentioned earlier $\partial R/\partial U$ is approximated to be equal to K, and R is evaluated as $R(U) = K(U^n)U^n - F(U^n)$.

The next step is to solve the nonlinear system $R^{n+1} = K(U^{n+1})U^{n+1} - F(U^{n+1}) = 0$ using the Newton method. In this approach the exact derivative of R with respect to U is required, which includes the partial derivatives of K and F with respect to U. The symbolic analysis is used to derive $\partial R^e/\partial U^e$ as a function of the nodes position as well as the value of the primitive variables at each node, see lines 66 to 94 of the Appendix. The equations for R^e and $\partial R^e/\partial U^e$ are hard-coded and called to evaluate the residual and its derivatives with respect to the primitive variables for each element. Using this approach R^e is directly evaluated, therefore the need for calculating and storing the stiffness and force matrices is eliminated. It improves the speed as well as memory requirement of the code. Using a proper assembly the values of R and $\partial R/\partial U$ of the whole system are computed.

IV. Shape parametrization and mesh deformation

The airfoil geometry can be parametrized using different methods. In this case the Class Shape Transformation (CST) method[14] is used. The CST method is symbolically described as follows:

$$\zeta(\psi) = C_{N2}^{N1}(\psi) \cdot S(\psi) \tag{13}$$

where ψ and ζ are normalized x and y positions respectively. C_{N2}^{N1} is the class function and $S(\psi)$ present the shape function. The class function expresses the basis class of shapes:

$$C_{N2}^{N1}(\psi) = (\psi)^{N1}\ (1-\psi)^{N2} \tag{14}$$

$N1$ and $N2$ are equal to 0.5 and 1 respectively for an airfoil with round leading edge and sharp trailing edge. The shape function is a Bernstein polynomial representation which describes the permutation around this basic shape:

$$S(\psi) = \sum_{i=0}^{p} \bar{P}_i B_{i,p}(\psi) \tag{15}$$

where p is the order of the Bernstein polynomial, $\bar{P}_i$ is the vector of control points and $B_{i,p}(\psi)$ are the Bernstein polynomials of degree p. The values of $\bar{P}_i$ are defined as design variables and referred to as the CST modes. Defining the values of the CST modes, the position of the nodes on the airfoil wall as well as the derivative of the x and y position of the wall nodes with respect to the CST coefficients are computed analytically.

The spring analogy is used to deform the mesh. Defining the position of the nodes on the wall using the CST method, the position of the other nodes are determined iteratively using this equation:

$$\Delta x_i^{m+1} = \frac{\sum_{j=1}^{N_i} k_i j \Delta x_j^m}{\sum_{j=1}^{N_i} k_i j} \tag{16}$$

where N_i is the number of the nodes connected to node i, and k_{ij} is the inverse of the length of the edge ij. Using Eq. (16) the sensitivity of the position of each node with respect to the CST modes is determined analytically.

V. Adjoint method for sensitivity analysis

Beside computing the exact Jacobian of the residuals, symbolic analysis is also used for sensitivity analysis required for optimization. The derivative of a function of interest I, for example the drag coefficient, with respect to a design variable X (not to be confused with x position of the nodes) is as follows:

$$\frac{dI}{dX} = \frac{\partial I}{\partial X} + \frac{\partial I}{\partial U}\frac{dU}{dX} \tag{17}$$

In the discrete adjoint approach the derivatives of the residuals with respect to the design variables are used to eliminate dU/dX from Eq. (17):

$$\frac{dR}{dX} = \frac{\partial R}{\partial X} + \frac{\partial R}{\partial U}\frac{dU}{dX} = 0 \tag{18}$$

$$\frac{dU}{dX} = -\frac{\partial R}{\partial U}^{-1}\frac{\partial R}{\partial X} \tag{19}$$

Substituting Eq. (19) in Eq. (17), dI/dX is calculated as follows:

$$\frac{dI}{dX} = \frac{\partial I}{\partial X} - \lambda^T\frac{\partial R}{\partial X} \tag{20}$$

where the adjoint vector, λ, is calculated from the following equation:

$$\frac{\partial R}{\partial U}^T\lambda = \frac{\partial I}{\partial U} \tag{21}$$

In an airfoil optimization I can be the lift coefficient C_l, the drag coefficient C_d and the pitching moment coefficient C_m. The design variables X can be the airfoil geometry as well as the angle of attack α and the free stream Mach number M.

Beside $\partial R/\partial U$, the partial derivatives of R and I with respect to the design variables X and the partial derivative of I with respect to U are required to compute the total derivatives dI/dX. The symbolic analysis is used to derive the partial derivatives of the element residuals with respect to the x and y positions of the element nodes, i.e. $\partial R^e/\partial x^e$ and $\partial R^e/\partial y^e$, see lines 96 to 181 of the code shown in Appendix. These equations were hard-coded in a function that receives the nodes positions as well as the value of the primitive variables at each node and returns $\partial R^e/\partial x^e$ and $\partial R^e/\partial y^e$. Using a proper assembly the partial derivatives of the global residuals with respect to the position of the nodes are computed.

The partial derivative $\partial R/\partial X$ is then computed as:

$$\frac{\partial R}{\partial X} = \frac{\partial R}{\partial x}\frac{\partial x}{\partial X} + \frac{\partial R}{\partial y}\frac{\partial y}{\partial X} + \frac{\partial R}{\partial \theta}\frac{\partial \theta}{\partial X} \tag{22}$$

where x and y are the positions of the nodes, θ is the slope of the wall at the position of the node and X is the vector of CST modes. The terms $\partial R/\partial\theta$ and $\partial\theta/\partial X$ are computed using the transformation matrix T, and the airfoil geometry (the CST formulation) respectively.

The angle of attack and the free stream Mach number only affect the values of u and v for the nodes defined as inlet boundary condition directly. Therefore $\partial U/\partial\alpha$ and $\partial U/\partial M$ are nonzero only for the nodes on the inlet boundary. For these nodes $\partial U/\partial\alpha$ and $\partial U/\partial M$ are as follows:

$$\frac{\partial U^e}{\partial \alpha} = [0 \quad -Msin(\alpha) \quad Mcos(\alpha) \quad 0]^T \tag{23}$$

$$\frac{\partial U^e}{\partial M} = [0 \quad cos(\alpha) \quad sin(\alpha) \quad 0]^T \tag{24}$$

The partial derivatives of R^e with respect to α and M are derived as follows:

$$\frac{\partial R^e}{\partial \alpha} = \frac{\partial R^e}{\partial U^e}\frac{\partial U^e}{\partial \alpha} \tag{25}$$

$$\frac{\partial R^e}{\partial M} = \frac{\partial R^e}{\partial U^e}\frac{\partial U^e}{\partial M} \tag{26}$$

By a proper assembly $\partial R/\partial\alpha$ and $\partial R/\partial M$ are computed.

The airfoil aerodynamic coefficients C_l, C_d and C_m are computed by a proper integration of the pressure over the airfoil surface. The pressure coefficient on the airfoil surface is a function of U of the nodes on the airfoil surface (defined as wall boundary condition) and the free stream Mach number. The equations of the partial derivatives $\partial C_l/\partial U$, $\partial C_d/\partial U$ and $\partial C_m/\partial U$ for the nodes on the airfoil surface are derived symbolically (manually not by Matlab symbolic analysis toolbox) and hard-coded. These derivatives for the nodes other than those on the airfoil surface are zero. Similar equations for derivatives of the airfoil coefficients with respect to the free stream Mach number are derived. In the same way the symbolic equations for the partial derivatives of the airfoil aerodynamic coefficients with respect to the x and y positions of the nodes on the surface of the airfoil are derived. Eventually the partial derivatives of I with respect to X is computed based on $\partial I/\partial x$, $\partial I/\partial y$, $\partial x/\partial X$ and $\partial y/\partial X$.

VI. Validation

A Matlab code named FEMflow is developed based on the mentioned method. The code is able to use parallel computing for solving massive CFD problems. In such an approach, the total domain is divided into n sub-domains, where n is the number of available computing nodes. The R vector as well as the matrix of $\partial R/\partial U$ for each domain is computed in parallel. A proper assembly is used at the end of generate the R vector and $\partial R/\partial U$ matrix for the whole domain.

In order to validate the code, the results of FEMflow for four different test cases were compared to experimental data from AGARD report[15] and shown in Table 1. Figure 2 shows the pressure contours for the four test cases shown in Table 1.

Table 1: Validation results of FEMflow.

			FEMflow		AGARD	
Airfoil	M	α	C_l	C_d	C_l	C_d
RAE2822	0.75	3	1.0713	0.0497	1.1058 ± 0.0322	0.0467 ± 0.0056
NACA0012	0.95	0		0.1078		0.1082 ± 0.0008
NACA0012	1.2	0		0.0948		0.0953 ± 0.0014
NACA0012	1.2	7	0.5170	0.1535	0.5211 ± 0.0142	0.1538 ± 0.0012

To verify the sensitivity analysis method the NACA0012 airfoil is parametrized using 20 CST modes (10 for the upper and 10 for the lower surfaces). Figure 3 shows the comparison between the derivatives of C_l, C_d and C_m with respect to the CST modes computed by the adjoint method and the finite differencing.

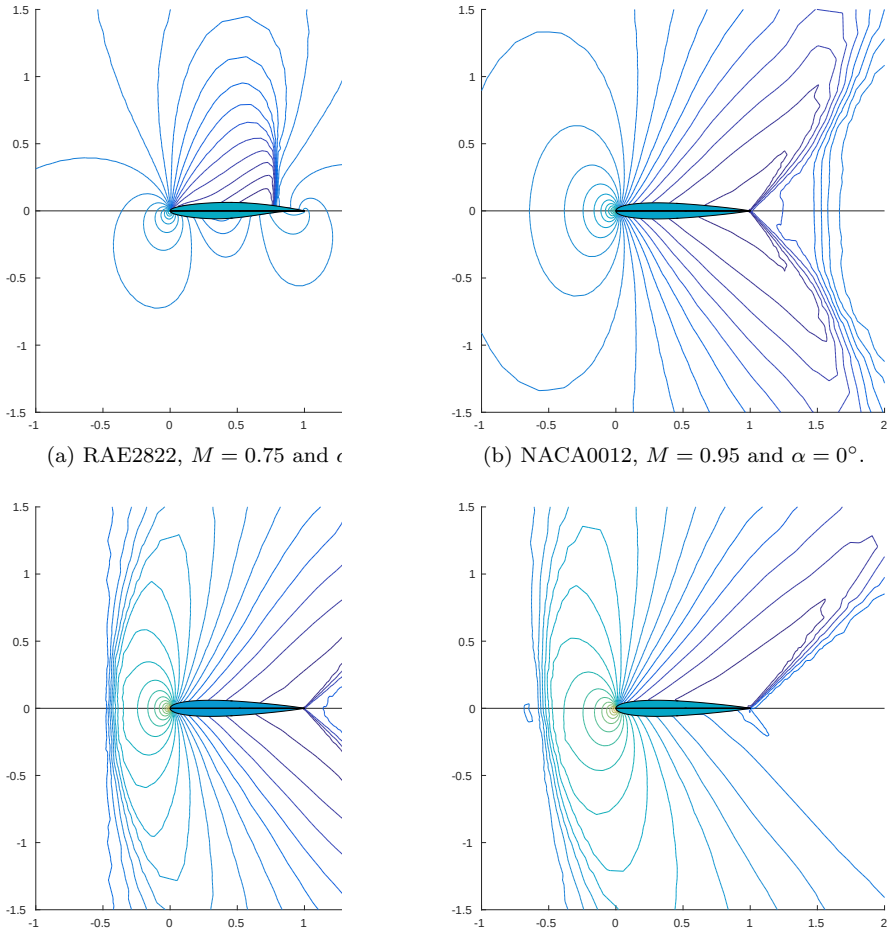
Besides, computing the exact value of dR/dU allows to use the full benefit of the quadratic convergence rate of the Newton method. Figure 4 shows the convergence history of analysis of the NACA0012 airfoil in Mach number of 0.7 and angle of attack of 2 degrees. First the Jacobian of the residuals is approximated by assuming a linear equation $R = KU - F$, where $dR/dU = K$. Then the exact Jacobian of the nonlinear system derived using symbolic analysis is used. From Fig. 4 one can observe that using the exact Jacobian, the Newton method reduced the norm of residuals by 9 order of magnitude in 10 iterations.

VII. Mesh adaptation

The least-squares residuals of the Euler equations can also be used as an error indicator for mesh adaptation.[12] Using such an approach the element error is defined based on the least squares residuals of the steady Euler equations:

$$E = \frac{1}{\Omega_e} U^e \int_{\Omega_e} \left[\frac{\partial N_i}{\partial x} A_1 + \frac{\partial N_i}{\partial y} A_2\right]^T \left[\frac{\partial N_j}{\partial x} A_1 + \frac{\partial N_j}{\partial y} A_2\right] dxdy \;\; U^e \tag{27}$$

Using this error the elements that need to be refined are indicated. An automatic mesh adaptation code is included in FEMflow. Two options are used for mesh refinement. In the first option the elements marked for refinement are divided into three elements. This is done by connecting each of the three element nodes to the point at the center of the element as shown in Fig. 5a. In the second option the elements marked for



(a) RAE2822, $M = 0.75$ and α

(b) NACA0012, $M = 0.95$ and $\alpha = 0°$.

(c) NACA0012, $M = 1.2$ and α

(d) NACA0012, $M = 1.2$ and $\alpha = 7°$.

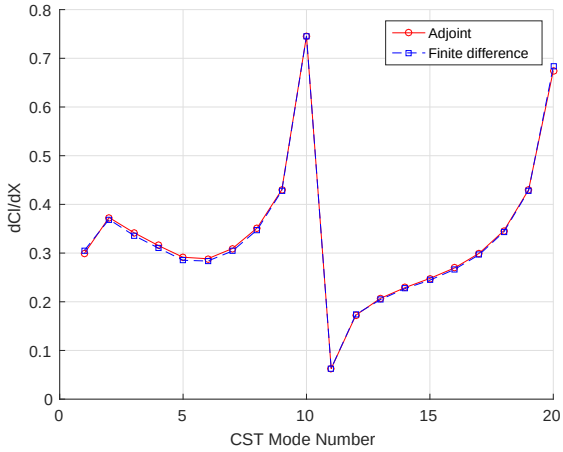
Figure 2: Pressure contours for the FEMflow validation test cases.

refinement are divided into two elements. The longest edge of each element is divided into two parts from the middle point of the edge. This point is then connected to the third node of the element. In order to make the mesh conformal, the neighbor element is also divided into two elements as shown in Fig. 5b.

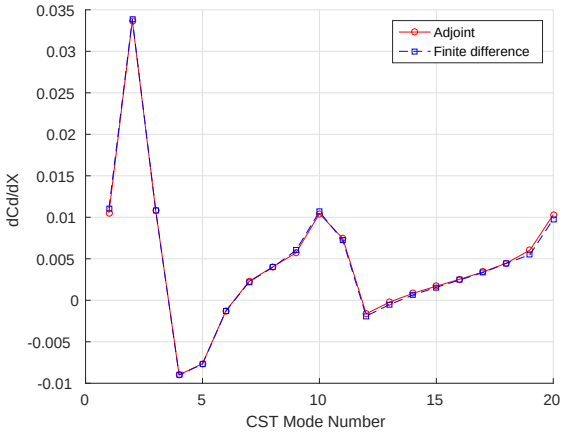
Figure 6 shows an example of mesh refinement in FEMflow. The RAE2822 airfoil in Mach 0.73 and $\alpha = 2°$ is considered in this case. Figure 6 shows the mesh before and aftere refinement using the type two mesh refinement method. The initial mesh has 10216 elements and 5233 nodes, while the adapted mesh has 16262 elements and 8282 nodes. The C_p distributions over the airfoil for two different meshes are plotted in Fig. 7.

VIII. Test case optimization

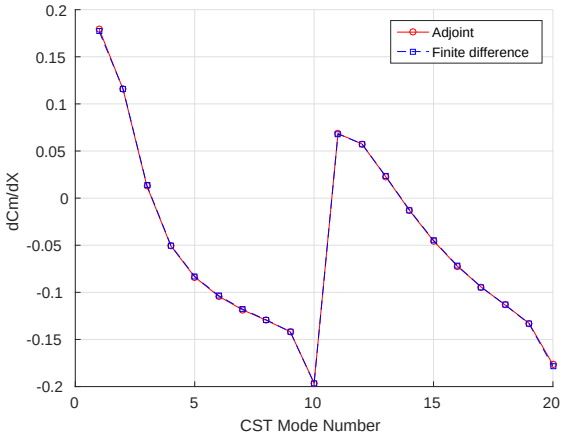
In the first test case optimization of the NACA0012 airfoil at $M = 0.75$ is considered. The optimization is formulated as follows:



(a) dCl/dX.



(b) dCd/dX.



(c) dCm/dX.

Figure 3: Verification of the sensitivity analysis. Results are for NACA0012 airfoil at $M = 0.7$ and $\alpha = 2°$.

$$
\begin{aligned}
min \quad & C_d(X) && && (28)\\
& X_i = \text{CST Modes} && i = 1..10 &\\
s.t. \quad & C_l - C_{l_t} = 0 && && (29)\\
& C_{m_t} - C_m \leq 0 && && (30)\\
& t_{max_{init}} - t_{max} \leq 0 && && (31)
\end{aligned}
$$

The airfoil geometry is parametrized using 10 CST modes, 5 for each surface. The airfoil lift coefficient is constrained to be equal to the lift coefficient of the initial airfoil at 2 degrees angle of attack. The airfoil pitching moment is constrained to be equal to or larger (less negative value) than the initial airfoil pitching moment. The airfoil maximum thickness to chord ratio is also constrained to be equal or higher than the thickness to chord ratio of the initial airfoil. The optimization is executed using a mesh with 10216 elements, see Fig. 8. The mesh adaptation has not been used during the optimization. The SNOPT optimization algorithm[16] is used as the optimizer.

Table 2 summarizes the results of the optimization. Figure 9 shows the convergence history of the objective function. The geometry and the pressure distribution of the initial and the optimized airfoils are compared in Fig. 10.

From Fig. 10b one can observe that the optimizer managed to eliminate the shock wave from the upper surface of the airfoil. This resulted in more than 65% reduction in the airfoil inviscid drag. The airfoil

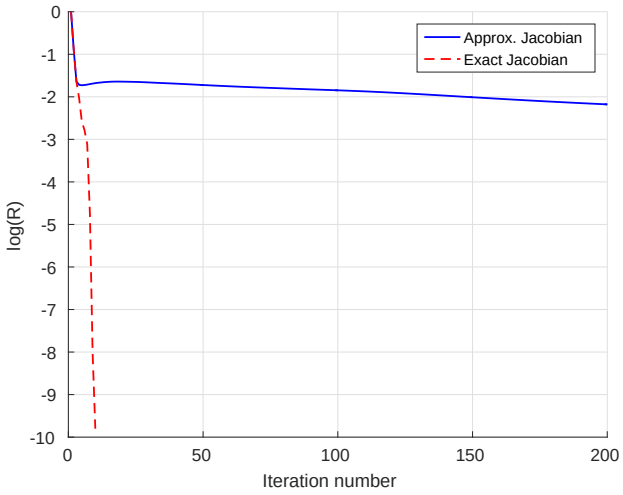
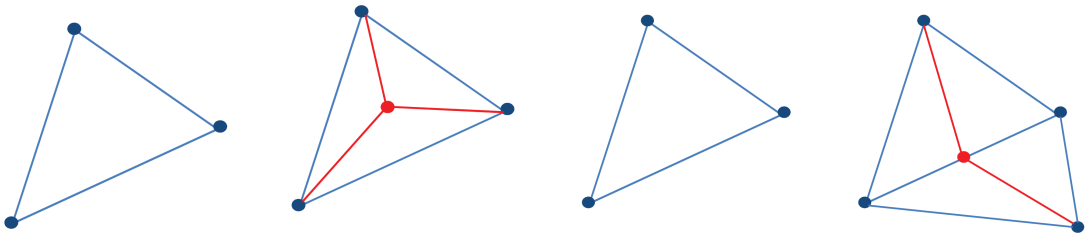
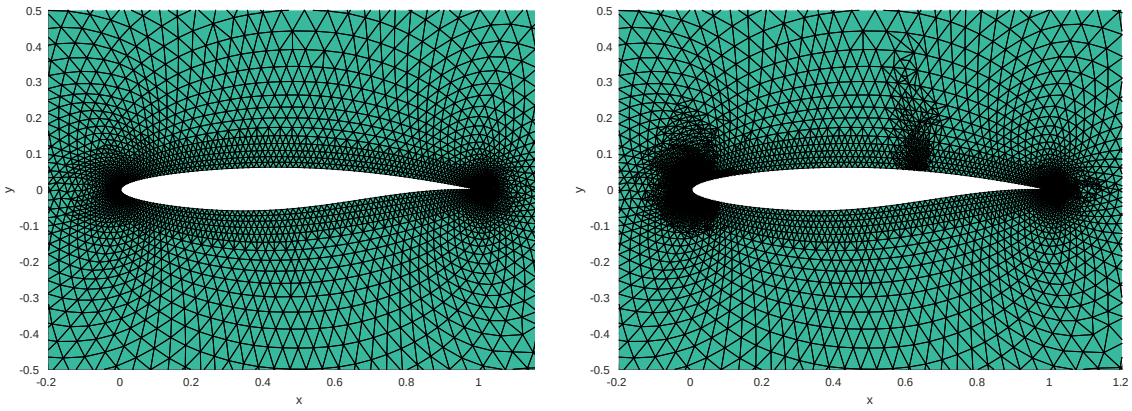


Figure 4: The effect of exact Jacobina on the convergence history of NACA0012 airfoil in $M = 0.7$ and $\alpha = 2°$.



(a) Type one. (b) Type two

Figure 5: Dividing elements into smaller ones for mesh refinement.



(a) Before refinement. (b) After refinement

Figure 6: Unstructured mesh around RAE2822 airfoil before and after refinement.

maximum thickness, the lift coefficient and the pitching moment coefficient remained the same as the initial airfoil.

As the second test case, optimization of the RAE2822 airfoil at Mach number of 0.73 is considered. The same constraints on airfoil lift (at 2 degrees angle of attack), pitching moment and maximum thickness as the previous case are applied. The same mesh with 10216 elements is used for this optimization as well.

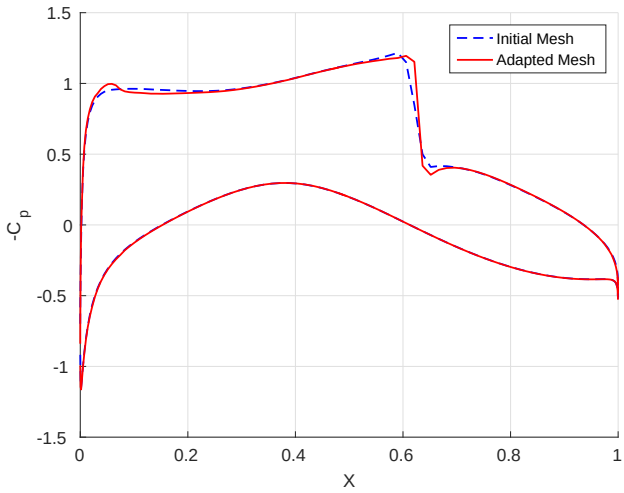


Figure 7: Effect of mesh adaptation on airfoil pressure distribution.

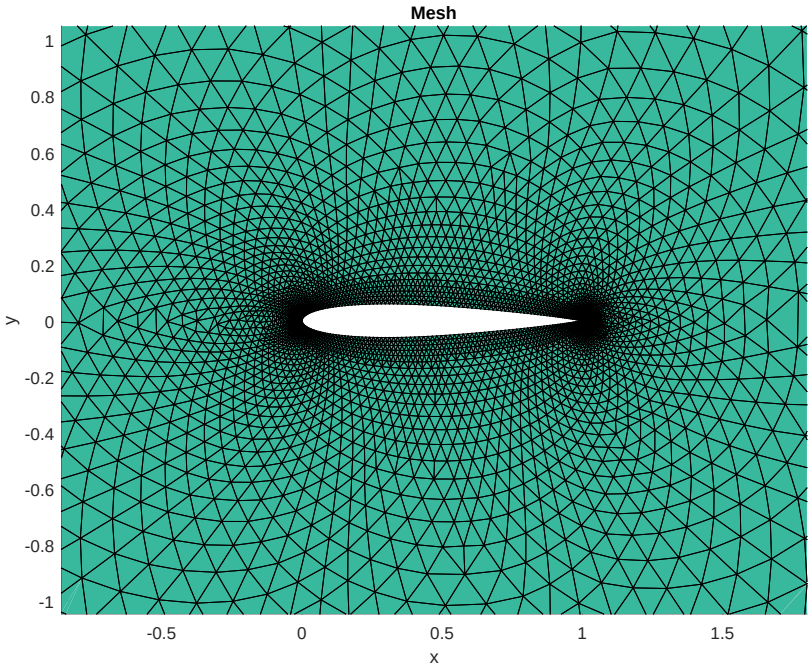


Figure 8: Unstructured mesh around NACA0012 airfoil with 10216 elements.

Table 2: Results of NACA0012 airfoil optimization

	α [deg]	C_l	C_d	C_m	$\left(\frac{t}{c}\right)_{max}$
Initial	2	0.4202	0.0121	-0.0125	0.12
Optimized	1.73	0.4204	0.0042	-0.0125	0.12

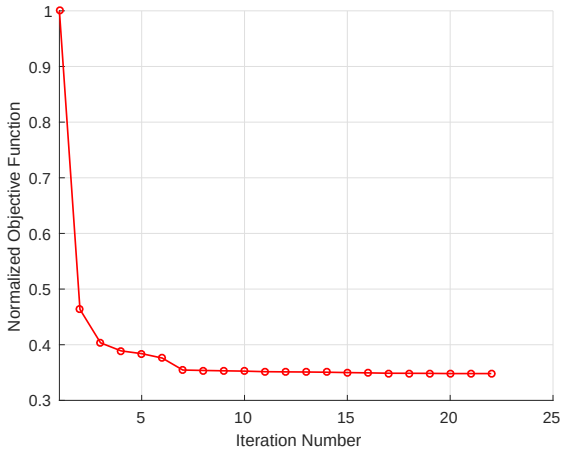
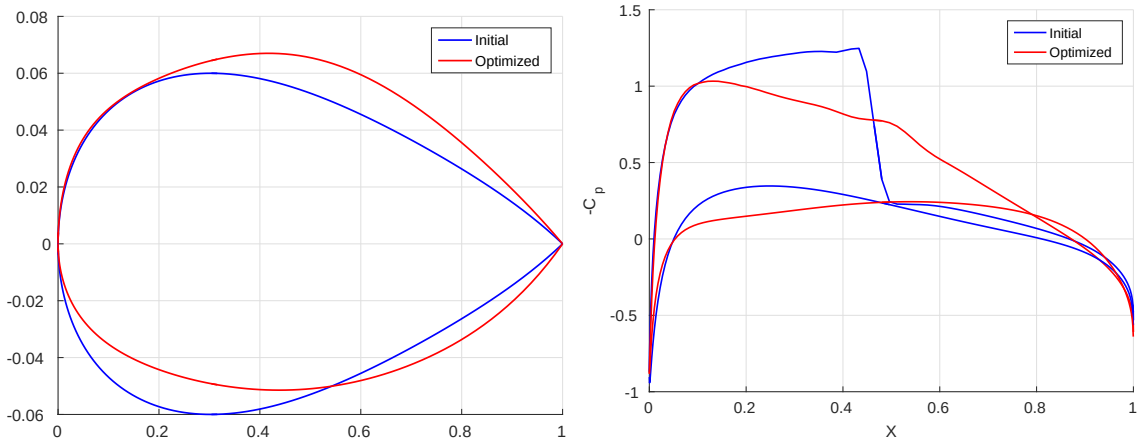


Figure 9: Convergence history of NACA0012 airfoil optimization.



(a) Airfoils geometry.

(b) Pressure coefficient.

Figure 10: Comparison of the initial and optimized airfoils for NACA0012 airfoil optimization.

The results of this optimization are shown in Figs. 11 and 12. Similar to the previous case, the optimizer managed to eliminate the shock wave from the upper side of the airfoil, while keeping the airfoil lift, pitching moment and maximum thickness the same as the initial airfoil. The inviscid drag of the airfoil was reduced by more than 16%. Table 3 summarizes the results of this optimization.

Table 3: Results of RAE2822 airfoil optimization

	α [deg]	C_l	C_d	C_m	$\left(\frac{t}{c}\right)_{max}$
Initial	2	0.7691	0.0163	-0.1311	0.12
Optimized	1.84	0.7691	0.0136	-0.1311	0.12

IX. Conclusions

This paper presented a successful application of symbolic analysis to aerodynamic shape optimization. The use of symbolic analysis resulted in a couple of advantages. The first advantage is that using the symbolic analysis the equations of the residuals and the derivative of the residuals with respect to the flow primitive variables are directly derived. Hard-coding these equations allows direct computation of the vector of the residuals R without the need for computing and storing the stiffness matrix and the force vector. This accelerates the computation and reduces the memory requirement.

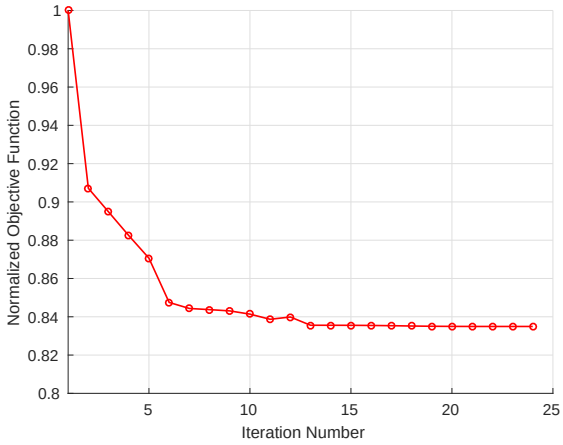
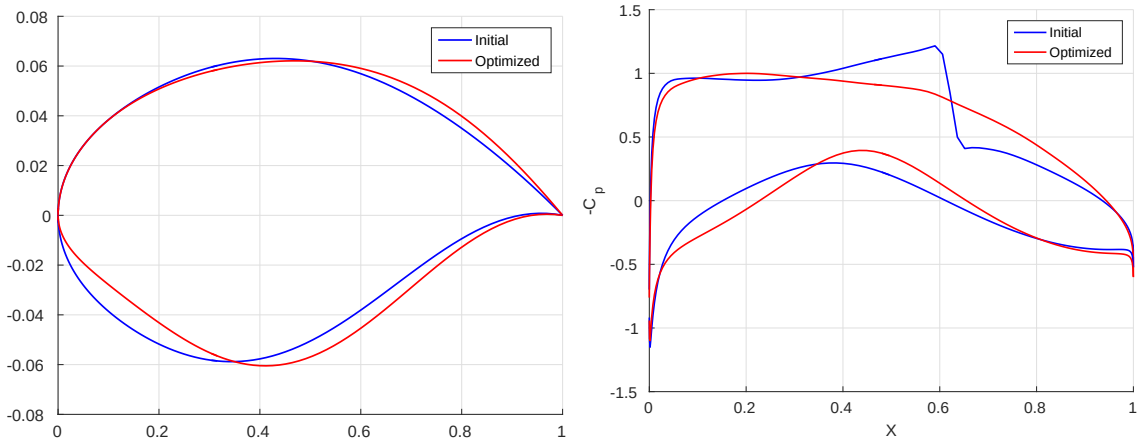


Figure 11: Convergence history of RAE2822 airfoil optimization.



(a) Airfoils geometry.

(b) Pressure coefficient.

Figure 12: Comparison of the initial and optimized airfoils for RAE2822 airfoil optimization.

Another advantage of using symbolic analysis is to compute exact derivatives of any function of interest with respect to the design variables without using automatic differentiation or finite differencing method. Applying the finite differencing method increases the computational time and also introduces errors in sensitivity analysis. Using forward mode of AD is time consuming and the reverse mode needs high memory requirement. However using the symbolic analysis the difficulties associated with memory requirement, computational time and accuracy are eliminated.

The least-squares finite element method is used to solve the Euler equations for inviscid compressible flow. A code named FEMflow is developed based on the mentioned method and the symbolic analysis to analysis and optimize airfoil shape at transonic flow regime. The accuracy of the code for computing lift and drag has been validated using experimental data, and the sensitivity analysis has been verified using finite differencing. The results of two airfoil optimization test cases have been used to demonstrate the ability of the proposed method for efficient aerodynamic shape optimization.

Appendix

```
%% Defining symbols

syms L1 L2
syms x1 x2 x3 y1 y2 y3
```

```
syms r u v p
syms r1 u1 v1 p1 r2 u2 v2 p2 r3 u3 v3 p3
syms dt

%% Defining shape functions for linear triangular elements

L3 = 1 - L1 - L2;

S1 = L1;
S2 = L2;
S3 = L3;

%% derivatives of the shape functions

dx_dL1 = x1 - x3;
dx_dL2 = x2 - x3;
dy_dL1 = y1 - y3;
dy_dL2 = y2 - y3;

J = [dx_dL1  dy_dL1;  dx_dL2  dy_dL2];

detJ = (x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3);

dS1 = J\[diff(S1,L1); diff(S1,L2)];  dS1dx = dS1(1);  dS1dy = dS1(2);
dS2 = J\[diff(S2,L1); diff(S2,L2)];  dS2dx = dS2(1);  dS2dy = dS2(2);
dS3 = J\[diff(S3,L1); diff(S3,L2)];  dS3dx = dS3(1);  dS3dy = dS3(2);

%% Euler equations

g = 1.4;

A1 = [u  r  0  0
      0  u  0  1/r
      0  0  u  0
      0  g*p 0  u];

A2 = [v  0  r  0
      0  v  0  0
      0  0  v  1/r
      0  0  g*p  v];

E = eye(4);


Ln1 = E*S1 + dt*A1*dS1dx + dt*A2*dS1dy;
Ln2 = E*S2 + dt*A1*dS2dx + dt*A2*dS2dy;
Ln3 = E*S3 + dt*A1*dS3dx + dt*A2*dS3dy;

L = [Ln1 Ln2 Ln3];
f = [r; u; v; p];

%% Matrices

SK = transpose(L)*L;
K = int(int(SK*detJ,L2,0,1-L1),L1,0,1);

SF = transpose(L)*f;
F = int(int(SF*detJ,L2,0,1-L1),L1,0,1);

%% Deriving R and dR/dU

U = [r1; u1; v1; p1; r2; u2; v2; p2; r3; u3; v3; p3];
R = K*U-F;

dRdr = diff(R,r);
dRdu = diff(R,u);
dRdv = diff(R,v);
dRdp = diff(R,p);

dRdr1 = diff(R,r1) + dRdr*S1;
```

```
dRdr2 = diff(R,r2) + dRdr*S2;
dRdr3 = diff(R,r3) + dRdr*S3;

dRdu1 = diff(R,u1) + dRdu*S1;
dRdu2 = diff(R,u2) + dRdu*S2;
dRdu3 = diff(R,u3) + dRdu*S3;

dRdv1 = diff(R,v1) + dRdv*S1;
dRdv2 = diff(R,v2) + dRdv*S2;
dRdv3 = diff(R,v3) + dRdv*S3;

dRdp1 = diff(R,p1) + dRdp*S1;
dRdp2 = diff(R,p2) + dRdp*S2;
dRdp3 = diff(R,p3) + dRdp*S3;

dRdU1 = [dRdr1 dRdu1 dRdv1 dRdp1];
dRdU2 = [dRdr2 dRdu2 dRdv2 dRdp2];
dRdU3 = [dRdr3 dRdu3 dRdv3 dRdp3];

dRdU  = [dRdU1 dRdU2 dRdU3];

%% dR/dx and dR/dy for a point at the center of the element i.e. x = (x1+x2+x3)/3 and y = (y1+y2+y3)/3

syms x y

S1 = (x2*y3 - x3*y2)/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) - (y*(x2 - x3))/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) + (x*(y2 - y3))/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2);
S2 = (y*(x1 - x3))/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) - (x1*y3 - x3*y1)/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) - (x*(y1 - y3))/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2);
S3 = (x1*y2 - x2*y1)/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) - (y*(x1 - x2))/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2) + (x*(y1 - y2))/ ...
(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - x3*y2);

dS1_dx1 = diff(S1,x1) + diff(S1,x)*1/3;
dS2_dx1 = diff(S2,x1) + diff(S2,x)*1/3;
dS3_dx1 = diff(S3,x1) + diff(S3,x)*1/3;

dS1_dx2 = diff(S1,x2) + diff(S1,x)*1/3;
dS2_dx2 = diff(S2,x2) + diff(S2,x)*1/3;
dS3_dx2 = diff(S3,x2) + diff(S3,x)*1/3;

dS1_dx3 = diff(S1,x3) + diff(S1,x)*1/3;
dS2_dx3 = diff(S2,x3) + diff(S2,x)*1/3;
dS3_dx3 = diff(S3,x3) + diff(S3,x)*1/3;


dr_dx1 = r1*dS1_dx1 + r2*dS2_dx1+ r3*dS3_dx1 ;
dr_dx2 = r1*dS1_dx2 + r2*dS2_dx2+ r3*dS3_dx2 ;
dr_dx3 = r1*dS1_dx3 + r2*dS2_dx3+ r3*dS3_dx3 ;

du_dx1 = u1*dS1_dx1 + u2*dS2_dx1+ u3*dS3_dx1 ;
du_dx2 = u1*dS1_dx2 + u2*dS2_dx2+ u3*dS3_dx2 ;
du_dx3 = u1*dS1_dx3 + u2*dS2_dx3+ u3*dS3_dx3 ;

dv_dx1 = v1*dS1_dx1 + v2*dS2_dx1+ v3*dS3_dx1 ;
dv_dx2 = v1*dS1_dx2 + v2*dS2_dx2+ v3*dS3_dx2 ;
dv_dx3 = v1*dS1_dx3 + v2*dS2_dx3+ v3*dS3_dx3 ;

dp_dx1 = p1*dS1_dx1 + p2*dS2_dx1+ p3*dS3_dx1 ;
dp_dx2 = p1*dS1_dx2 + p2*dS2_dx2+ p3*dS3_dx2 ;
dp_dx3 = p1*dS1_dx3 + p2*dS2_dx3+ p3*dS3_dx3 ;


dRdx1 = diff(R,x1) + diff(R,r)*dr_dx1 + diff(R,u)*du_dx1 + diff(R,v)*dv_dx1 +diff(R,p)*dp_dx1 ;
dRdx2 = diff(R,x2) + diff(R,r)*dr_dx2 + diff(R,u)*du_dx2 + diff(R,v)*dv_dx2 +diff(R,p)*dp_dx2 ;
dRdx3 = diff(R,x3) + diff(R,r)*dr_dx3 + diff(R,u)*du_dx3 + diff(R,v)*dv_dx3 +diff(R,p)*dp_dx3 ;

dRdx = [dRdx1  dRdx2  dRdx3];
```

```
145
146 %
147 dS1_dy1 = diff(S1,y1) + diff(S1,y)*1/3;
148 dS2_dy1 = diff(S2,y1) + diff(S2,y)*1/3;
149 dS3_dy1 = diff(S3,y1) + diff(S3,y)*1/3;
150
151 dS1_dy2 = diff(S1,y2) + diff(S1,y)*1/3;
152 dS2_dy2 = diff(S2,y2) + diff(S2,y)*1/3;
153 dS3_dy2 = diff(S3,y2) + diff(S3,y)*1/3;
154
155 dS1_dy3 = diff(S1,y3) + diff(S1,y)*1/3;
156 dS2_dy3 = diff(S2,y3) + diff(S2,y)*1/3;
157 dS3_dy3 = diff(S3,y3) + diff(S3,y)*1/3;
158
159
160 dr_dy1 = r1*dS1_dy1 + r2*dS2_dy1+ r3*dS3_dy1 ;
161 dr_dy2 = r1*dS1_dy2 + r2*dS2_dy2+ r3*dS3_dy2 ;
162 dr_dy3 = r1*dS1_dy3 + r2*dS2_dy3+ r3*dS3_dy3 ;
163
164 du_dy1 = u1*dS1_dy1 + u2*dS2_dy1+ u3*dS3_dy1 ;
165 du_dy2 = u1*dS1_dy2 + u2*dS2_dy2+ u3*dS3_dy2 ;
166 du_dy3 = u1*dS1_dy3 + u2*dS2_dy3+ u3*dS3_dy3 ;
167
168 dv_dy1 = v1*dS1_dy1 + v2*dS2_dy1+ v3*dS3_dy1 ;
169 dv_dy2 = v1*dS1_dy2 + v2*dS2_dy2+ v3*dS3_dy2 ;
170 dv_dy3 = v1*dS1_dy3 + v2*dS2_dy3+ v3*dS3_dy3 ;
171
172 dp_dy1 = p1*dS1_dy1 + p2*dS2_dy1+ p3*dS3_dy1 ;
173 dp_dy2 = p1*dS1_dy2 + p2*dS2_dy2+ p3*dS3_dy2 ;
174 dp_dy3 = p1*dS1_dy3 + p2*dS2_dy3+ p3*dS3_dy3 ;
175
176
177 dRdy1 = diff(R,y1) + diff(R,r)*dr_dy1 + diff(R,u)*du_dy1 + diff(R,v)*dv_dy1 +diff(R,p)*dp_dy1 ;
178 dRdy2 = diff(R,y2) + diff(R,r)*dr_dy2 + diff(R,u)*du_dy2 + diff(R,v)*dv_dy2 +diff(R,p)*dp_dy2 ;
179 dRdy3 = diff(R,y3) + diff(R,r)*dr_dy3 + diff(R,u)*du_dy3 + diff(R,v)*dv_dy3 +diff(R,p)*dp_dy3 ;
180
181 dRdy = [dRdy1  dRdy2  dRdy3];
```

References

[1]Antunes, A.P., Azevedo, J.L.F., "Studies in Aerodynamic Optimization Based on Genetic Algorithms," JOURNAL OF AIRCRAFT, Vol. 51, No. 3, 2014, pp.1002-1012.

[2]Elham, A., van Tooren M.J.L., "Weight Indexing for Wing-Shape Multi-Objective Optimization," AIAA JOURNAL, Vol. 52, No. 2, 2014, pp. 320-337.

[3]Mader, C.A., Martins. J.R.R.A., Alonso, J.J., van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," AIAA JOURNAL Vol. 46, No. 4, 2008, pp. 863-873.

[4]Carpentieri, G., Koren, B., van Tooren, M.J.L., "Adjoint-based aerodynamic shape optimization on unstructured meshes," Journal of Computational Physics, Vol. 224, 2007, pp. 267-287.

[5]Brezillon, J., Gauger, N.R., "2D and 3D aerodynamic shape optimisation using the adjoint approach," Aerospace Science and Technology Vol. 8, 2004, pp. 715-727.

[6]Palacios, F., Economon, T.D., Wendorf, A.D., and Alonso, J.J., "Large-scale aircraft design using SU2," 53rd AIAA Aerospace Sciences Meeting, 5-9 January 2015, Kissimmee, Florida, USA.

[7]Baysal, O., and Ghayour, K., "Continuous Adjoint Sensitivities for Optimization with General Cost Functionals on Unstructured Meshes," AIAA JOURNAL Vol. 39, No. 1, January 2001, pp. 48-55.

[8]Muller, J.D., Cusdin, P., "On the performance of discrete adjoint CFD codes using automatic differentiation," INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, Vol. 47, 2005, pp.939-945.

[9]Jiang, B.N., Carey, G.F., "Least-Squares Finite Element Methods for Compresible Euler Equations," INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, Vol. 10, 1990, pp. 557-568.

[10]Krishnamoorthy, C.S., " Finite element analysis, theory and programming," second edition, Mc Graw Hill Education, New Delhi, India, 1994.

[11]Reddy, J.N., Gartling, D.K., " The Finite Element Method in Heat Transfer and Fluid Dynamics," CRC Press, Boca Raton, Florida, USA, 2001.

[12]Lefebvre, D., Peraire, J., Morgan, K., " Finite Elmenet Least Squares Solution of The Euler Equatsions Using Linear and Quadratic Approximations," International Journal of Computational Fluid Dynamics, Vol. 1, pp. 1-23, 1993.

[13]Bochev, P.B., Gunzburger, M.D., "Least-Squares Finite Element Methods," Springer, New York, NY, USA, 2009.

[14]Kulfan, B., "Universal parametric geometry representation method," Journal of Aircraft , Vol. 45, No. 1, 2008, pp. 142-158.

[15]Viviand, H., "Numerical solutions of two-dimensional reference test cases, in test cases for inviscid flow field methods," AGARD-AR-211, May 1985.

[16]Gill, P., Murray, W., and Saunders, M., "SNOPT: An SQP Algorithm for Large-scale Constrained Optimization," SIAM Review, Vol. 47, No. 1, 2005, pp. 99-131.