European Conference on Computational Fluid Dynamics
ECCOMAS CFD 2006
P. Wesseling, E. Oñate and J. Périaux (Eds)

AERODYNAMIC SHAPE OPTIMIZATION BY MEANS OF SEQUENTIAL LINEAR PROGRAMMING TECHNIQUES

Giampietro Carpentieri*, Michel J.L. van Tooren† and Barry Koren††

Delft University of Technology, Faculty L&R,
Kluyverweg 1, 2629 HS Delft, The Netherlands
e-mail: G.Carpentieri@tudelft.nl

Key words: Aerodynamic Shape Optimization, Sequential Linear Programming, Unstructured Meshes, Adjoint equations

Abstract. *A Sequential Linear Programming technique, known as the method of centers, is the driver of an aerodynamic shape optimization framework on unstructured meshes. The first order information required by this technique, functional values and their gradients, are computed by a median–dual flow/adjoint solver which is coupled to an analytical shape parameterization. Functional and geometric constraints are easily handled by the algorithm which appears to be very effective in obtaining efficiently near–optimal designs. Shape optimization results are presented for transonic as well as supersonic flows involving appreciable shape deformations.*

1 INTRODUCTION

Gradient based shape optimization has received a boost since the introduction of the adjoint method in CFD.[3] Large scale designs involving Finite Volume solvers are nowadays possible at a reasonable cost, comparable to that of an additional flow solution for each design iteration.[4] The attractiveness of such benefits in terms of costs have motivated the large effort in the development of Adjoint solvers in the last decade.[10,12,14,15] Also, due to their flexibility with respect to the handling of complex geometry, unstructured flow solvers had received particular attention. Among others, median–dual (node centered) formulations were proven to be extremely efficient due to their "edge–based" data structure. Because of this, many median–dual implementations have appeared in the field.[6,7,8,9]

The present work is part of an effort that is dedicated to the development of an aerodynamic shape optimization framework for unstructured meshes. The development is horizontal, in the sense that not only CFD aspects are considered but also other aspects such as the shape parameterization and the optimization strategy. The latter are of equal importance for the Shape Optimization practitioner to succeed.

Up to now the capability of the framework is limited to the 2D Euler equations. In a previous work[1] the implementation of the median–dual finite volume solver was briefly

addressed. More in detail, the development and the implementation of the discrete adjoint solver was described. Some results, obtained by the coupling of the flow/adjoint solver with the shape parameterization and a Sequential Quadratic Programming optimization algorithm, were presented. Also, preliminary results from the application of a Sequential Linear Programming technique appeared to be promising.

Therefore, the present paper addresses briefly the flow/adjoint solver and focuses more on the optimization process from an application–oriented point of view. In section 2, after an introduction to the shape optimization problem, the shape parameterization is considered. The Sequential Linear Programming algorithm used to drive the optimization is presented at the end of the section. The algorithm, for the first time applied to a shape optimization problem, is also know as the method of centers. In section 3 the flow/adjoint solver is addressed whereas section 4 deals with the computation of the gradients. Compared to the last work,[1] the gradients are computed with the aid of Automatic Differentiation. In section 5 the coupling of the whole optimization framework is briefly addressed and, finally, numerical results are presented for shape optimization in transonic as well as supersonic flows.

2 SHAPE OPTIMIZATION

The present work focuses on a shape optimization problem in which the Drag coefficient (c_d) of the airfoil must be reduced and a certain number of constraints must be satisfied. The constraints are enforced on the Lift coefficient (c_l) and on geometric quantities such as the relative maximum thickness ($tc_{\max}$) of the section, the nose radius (r_u for the upper side and r_l for the lower side, see section 2.1) and the trailing edge angle (θ).

The Lift constraint is considered as an equality, necessary in order not to reduce the Drag coefficient at the expense of the Lift. The relative maximum thickness constraint, considered as inequality, is necessary for avoiding the section to become too thin. The other constraints, on the nose radius and on the trailing edge angle, are included to ensure the feasibility of the shape. For instance, very sharp nose and very thin trailing edge are not desirable from a point of view of manifacturability as well as off–design conditions. Moreover, apart from real life issues, unfeasible shapes can cause the break–down of the optimization process. In previous work,[1,2] without constraints on the nose and on the trailing edge angle, it was necessary to set bounds (by trial and error) on the design variables in order to avoid unfeasible shapes.

If bounds on the design variables are included, the problem is a bound–constrained optimization which, in general, can be formally stated as:

$$\begin{aligned} \min\ & f(\mathbf{x})\,, \\ & g_j(\mathbf{x}) \leq 0 \qquad (j = 1, n_g)\,, \\ & h_k(\mathbf{x}) = 0 \qquad (k = 1, n_h)\,, \\ & \mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U\,, \end{aligned} \tag{1}$$

where in this work $n_g = 4$, $n_h = 1$ and the objective function is $f = c_d/c_{dR}$ with c_{dR}

indicating a reference value (in the following the subscript R will indicate reference values). The thickness constraint is defined as $g_1 = 1 - tc_{\max}/tc_{\max R}$, the upper and lower nose radius constraints are $g_2 = 1 - r_u/r_{uR}$ and $g_3 = 1 - r_l/r_{lR}$ whereas the trailing edge angle constraint is $g_4 = 1 - \theta/\theta_R$. The Lift constraint, which is the only equality constraint considered here, is defined as $h = c_l/c_{lR} - 1$.

The reference values for the constraints have the function of defining the feasible design space. For the objective function, the reference value is only used for the purpose of scaling. In the following, the reference values are considered as initial values multiplied by a constant (the initial values are indicated with the subscript 0).

2.1 Parameterization of the shape

The shape of the airfoil is described using a Chebyshev polynomials parameterization in which the upper and the lower curves are independently approximated. For both curves the polynomials are made-up of orthogonal basis functions:[16]

$$f_d(x) = \sum_{k=0}^{N_D} d_k D_k(x)\,, \qquad D_k = T_k - T_{k+2}\,, \tag{2}$$
$$T_k(x) = \cos(k\theta(x))\,, \qquad \theta(x) = \cos^{-1}(2\sqrt{x} - 1)\,, \qquad (k \geq 0\,,\ 0 \leq x \leq 1)\,,$$

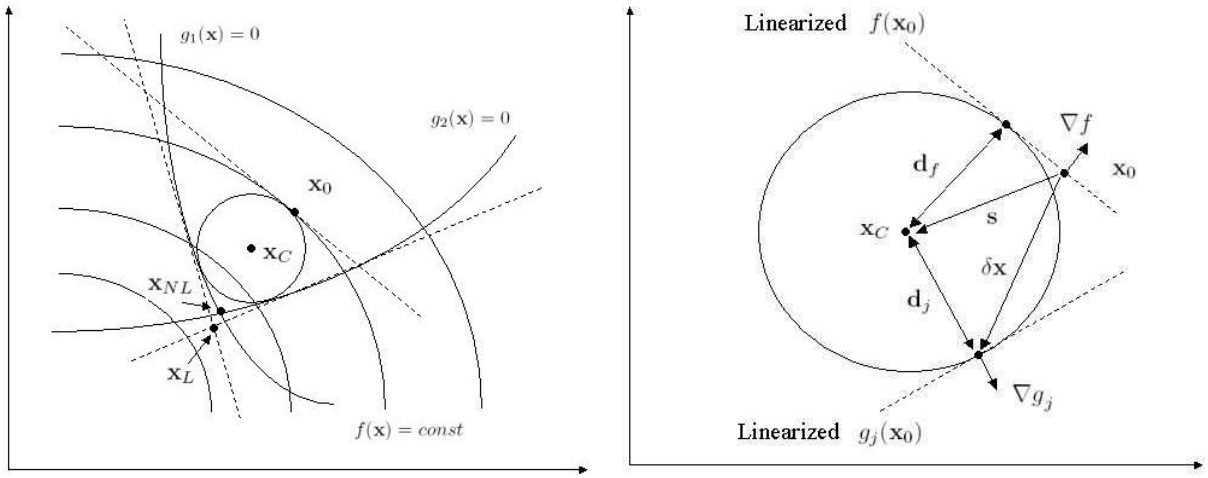
where N_D is the number of basis functions, D_k are the basis functions and d_k are the shape parameters or basis functions coefficients. Those parameters and their number are chosen by means of a minimization procedure which aims at reducing the error between the curves and the original airfoil coordinates.[2] A certain error is achieved in a way that there are no discrepancies between the flowfield around the original and the approximated airfoil geometry. In the present work the number of basis functions used for each curve is $N_D = 11$, since this number was found to be effective in approximating many airfoils accurately.

Therefore, the design variables vector $\mathbf{x} = [d_1, d_2, ..., d_{2N_D}]^T$ contains the shape parameters for the upper and the lower curves (22 design variables in total). Additionally, although not considered in this work, the angle of attack could also be included as design variable. Since the curves are defined analytically, it is possible to calculate the slope and the radius of curvature at any point (respectively obtained as $\arctan f_d'$ and $\sqrt{(1 + {f_d'}^3)}/f_d''$). They are used to compute the trailing edge angle θ and the radii of curvatures r_u and r_l. Two radii of curvature are computed at the leading edge due to a lack of continuity of the derivatives at that point. In fact, the derivatives of $f_d(x)$ in $x = 0$ and $x = 1$ can only be considered in the limit as $x \to 0$ and $x \to 1$.

2.2 Sequential Linear Programming: the method of centers

A Sequential Linear Programming algorithm has been already applied in a previous work.[1] The algorithm[5] is based upon the linearization of the non–linear optimization problem, which lead to another problem that can be easily solved by a Linear Programming

algorithm (e.g. the Simplex method). The solution of the Linear Programming problem is used as a starting point for a new linearization. The process is then iterated until convergence to the minimum of the non–linear problem. The algorithm was found effective but slightly inaccurate in terms of the constraints to be satisfied. Such a behavior is not surprising since, as can be seen from Fig. (1.a), the optimum of the Linearized problem is likely to lie outside of the feasible (convex) design space. Therefore, the optimum of the non–linear problem is approached from the unfeasible region.



(a) $\mathbf{x}_0$ = linearization point, $\mathbf{x}_c$ = center of the circle, $\mathbf{x}_L$ = optimum of the linearized problem, $\mathbf{x}_{NL}$ = optimum of the non–linear problem

(b) $\mathbf{s}$ = move toward the center of the circle, $\delta\mathbf{x}$ = move toward the constraint boundary

Figure 1: Geometric interpretation of the method of centers

In the present work another Sequential Linear Programming algorithm, known as the method of centers, is investigated. The idea behind this algorithm, which is depicted in Fig. (1.a), is to find the largest circle (hypersphere in multi dimension) that fits into the linearized design space. Once the latter has been found, a move in the center of the hypersphere will bring the design in a position which is closer to the optimum. A new linearization may be performed, a new hypersphere may be computed and again the design may be moved to its center. The process may be repeated until convergence to the non–linear optimum.

The interesting feature is that the optimimum is always approached from the feasible region. Moreover, unfeasible designs can be brought into the feasible region simply by computing an hypersphere in which the linearized function is neglected. The largest hypersphere is computed, as can be seen from Fig. (1.b), by means of some geometric arguments.[5] In fact, the hypersphere radius r should be the result of a maximization under the constraints that $d_f - r \geq 0$ and $d_j - r \geq 0$ for $j = 1, n_g$ with

$$d_f = -\frac{\nabla f^T \mathbf{s}}{|\nabla f|}, \qquad d_j = \frac{\nabla {g_j}^T (\delta\mathbf{x} - \mathbf{s})}{|\nabla g_j|} = -\frac{g(\mathbf{x}_0) + \nabla {g_j}^T \mathbf{s}}{|\nabla g_j|}, \tag{3}$$

where d_j is obtained considering that, being $\delta\mathbf{x}$ the move to the constraint boundary, one has $g_j(\mathbf{x}_0 + \delta\mathbf{x}) = g_j(\mathbf{x}_0) + \nabla {g_j}^T \delta\mathbf{x} = 0$. If equality constraints have to be included, the move $\mathbf{s}$ towards the center should not violate them, which means $h_k(\mathbf{x}_0 + \mathbf{s}) = h_k(\mathbf{x}_0) + \nabla {h_k}^T \delta\mathbf{x} = 0$ for $k = 1, n_h$. After some algebra, the maximization problem for the radius r of the hypersphere may be stated as follows:

$$
\begin{aligned}
\max\ & r\,, \\
\nabla f^T \mathbf{s} + |\nabla f| r &\leq 0\,, \\
\nabla {g_j}^T \mathbf{s} + |\nabla g_j| r &\leq -g_j(\mathbf{x}_0) \qquad (j = 1, n_g)\,, \\
\nabla {h_k}^T \mathbf{s} &= -h_k(\mathbf{x}_0) \qquad (k = 1, n_h)\,, \\
\mathbf{x}_L - \mathbf{x}_0 \leq \mathbf{s} &\leq \mathbf{x}_U - \mathbf{x}_0\,, \\
r &\geq 0\,.
\end{aligned}
\tag{4}
$$

Equation (4) is a Linear Programming problem for the vector $[\mathbf{s}, r]^T$. After the solution of the problem, the new design located in the center of the hypersphere can be obtained as $\mathbf{x}_{new} = \mathbf{x}_0 + \mathbf{s}$. However, as can be seen from Fig. (1.a), one can also take an additional move in the direction of the steepest descent in order to get closer to the optimum. In this case $\mathbf{x}_{new} = \mathbf{x}_0 + \mathbf{s} - \beta r \nabla f / |\nabla f|$ where $0 \leq \beta \leq 1$. Due to such additional move, also the line corresponding to the equality constraints in Eq. (4) should be slightly modified in order to have $h_k(\mathbf{x}_0 + \mathbf{s} - \beta r \nabla f / |\nabla f|) = 0$ for $k = 1, n_h$. Apart from the positive effect of accelerating the convergence to the optimum, the additional move can increase the risk of violating the constraints.

A negative aspect of this algorithm is the requirement of move limits[5] for those problems that are underconstrained. In fact, in such a case the linearization can lead to problems that are unbounded. Unfortunately, this is the case for the problems considered here. Therefore, move limits have to be defined together with a reduction factor. The latter, has to reduce the move limits when the optimum is approached. In practice, when move limits $\Delta\mathbf{x}$ are introduced the fifth line of Eq. (4) is replaced by $-\Delta\mathbf{x} \leq \mathbf{s} \leq \Delta\mathbf{x}$. One can think the move limits as a box in which the search is limited, an artificial bound for a problem that would be otherwise unbounded.

3 THE FLOW/ADJOINT SOLVER

The Flow solver[1] is based upon a median–dual type of discretization[7] suitable for unstructured hybrid meshes. This type of discretization allows the use of an edge-based data structure that make no distinction between 2D and 3D elements as well as between the different types of elements. The numerical fluxes are computed using a Roe's approximated Riemann solver. Second order accuracy is achieved trough a MUSCL–like reconstruction where the primitive variables are linearly extrapolated at the cell interfaces (edge mid–points). Elements of the reconstruction are a Least Squares gradient and a multidimensional type of limiter. Both are computed using the edge–based data structure of the solver. An implicit pseudo–time stepping, first order backward Euler scheme,

is used to solve the equations:

$$\left(\mathbf{D}_t + \frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{U}}\right)^n (\mathbf{U}^{n+1} - \mathbf{U}^n) = -\mathbf{R}^n \,, \tag{5}$$

where $\mathbf{R}$ is the residual vector, $\mathbf{U}$ is the vector of conservative variables, $\mathbf{D}_t$ is a diagonal matrix containing the cell volumes divided by their local time steps and $\widetilde{\partial \mathbf{R}}/\partial \mathbf{U}$ is an approximated Jacobian. The approximations are in terms of the spatial accuracy, which is limited to first order (the reconstruction contribution is neglected), and in terms of the Roe's fluxes in which the flux Jacobians are frozen when performing the linearization. The solution of the linear system arising at each time step is accomplished using a SSOR scheme implemented using the edge–based data structure of the solver. The scheme is fast and efficient and it has the additional feature of being optionally matrix–free. In spite of the time penalty, due to the recomputation of the Jacobian elements, with the matrix–free options the memory requirements are that of an explicit flow solver.

The discrete adjoint is the preferred approach for computing the adjoint variables, which are the sensitivity of the functionals with respect to the state (the residual vector $\mathbf{R}$). An exact discrete adjoint has been derived and implemented in a matrix-free fashion.[1] Exact means that the differentiation of all the components in the residual vector is made without any approximation. For a MUSCL-like implementation, the development of the discrete adjoint means the differentiation of the fluxes as well as the reconstruction operator. It is useful to consider a vector $\mathbf{H}$ containing the numerical fluxes evaluated on each cell interface (edge mid–point). Such a vector is dependent on the reconstructed left and right states on the interface, indicated respectively with $\mathbf{U}_L$ and $\mathbf{U}_R$. The length of the three vectors is equal to the numbers of interfaces, E, which in this case correspond to the number of edges in the mesh. The dependency of the residual vector on the conservative variables vector, both of length N equal to the number of nodes, can be expressed as $\mathbf{R}(\mathbf{U}) = \mathbf{R}\{\mathbf{H}[\mathbf{U}_L(\mathbf{U}), \mathbf{U}_R(\mathbf{U})]\}$. Applying the chain rule and transposition one has :

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T = \left(\frac{\partial \mathbf{U}_L}{\partial \mathbf{U}}^T \frac{\partial \mathbf{H}}{\partial \mathbf{U}_L}^T + \frac{\partial \mathbf{U}_R}{\partial \mathbf{U}}^T \frac{\partial \mathbf{H}}{\partial \mathbf{U}_R}^T\right) \frac{\partial \mathbf{R}}{\partial \mathbf{H}}^T \,, \tag{6}$$

where $\partial \mathbf{R}/\partial \mathbf{H}$ is a dummy matrix whose only non–zero elements have values 1 and -1, $\partial \mathbf{H}\ /\partial \mathbf{U}_L$ and $\partial \mathbf{H}\ /\partial \mathbf{U}_R$ are the differentiation of the numerical fluxes and $\partial \mathbf{U}_L/\partial \mathbf{U}$ and $\partial \mathbf{U}_R/\partial \mathbf{U}$ are the differentiation of the reconstruction operator. In practice none of these matrices is formed, but more likely a matrix–vector product involving $\partial \mathbf{R}/\partial \mathbf{U}^T$ is assembled directly on the edges in much the same way as for the residual vector.

The solution process for the adjoint equations is identical to that of the flow equations. The same first order backward Euler scheme is used, with the only difference that the approximated Jacobian is in this case transposed:

$$\left(\mathbf{D}_t + \frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{U}}^T\right)^n (\mathbf{\Lambda}^{n+1} - \mathbf{\Lambda}^n) = -\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{\Lambda} - \frac{\partial J}{\partial \mathbf{U}}^T\right)^n \,. \tag{7}$$

It can be argued that this solution method is not necessary for a linear system of equations. However, due to the poor diagonal dominance of the exact Jacobian, it can be an expensive and not always successful approach to use only a linear system solver. On the contrary, the use of the pseudo–time stepping (diagonal dominance is increased due to the approximated Jacobian and the time step contribution) gives a robust solver for the adjoint equations.[11] An additional feature of the pseudo-time stepping and the SSOR scheme adopted for the adjoint equations, is the possibility to solve for multiple right–hand sides. This means that multiple adjoints can be computed in only one-shot. This turns out to be very useful in constrained shape optimization since more than one functional is usually required. For the simultaneous solution of 3 functionals, up to 50% time saving has been noticed.

4 THE COMPUTATION OF THE GRADIENTS

Once the adjoint variables have been computed, the gradient of a functional J with respect to a shape-parameter x_i is obtained as:

$$\frac{dJ}{dx_i} = \frac{\partial J}{\partial x_i} - \mathbf{\Lambda}^T \frac{\partial \mathbf{R}}{\partial x_i}, \tag{8}$$

where $\partial J/\partial x_i$ and $\partial \mathbf{R}/\partial x_i$ can be computed, among other methods,[10,12,13] by finite difference or by means of Automatic Differentiation. The finite difference approach can be inefficient for a large number of parameters. Moreover, it needs a proper step size in order to reduce the cancellation and the truncation errors. On the contrary, Automatic differentiation relies on basic linearization rules and the exact differentiation of the code can be derived and executed efficiently.[13] More challenging is the derivation of the adjoint code with the reverse mode where storage and efficiency issues should be carefully considered.[12,13,14] For this reason, the adjoint code presented before has been hand–coded.

In this work the forward vectorial mode of Tapenade[17] has been adopted, which allows to compute $\partial \mathbf{R}/\partial x_i$ for the complete design vector $\mathbf{x}$ in one shot. Time saving is achieved compared to sequential evaluations since many quantities, having a value which is the same for all parameters, are computed only once.

In order to understand how the differentiated code is derived using Tapenade, the dependency of the residual vector $\mathbf{R}$ on a shape parameter x_i should be described first. It is convenient to start from the mesh coordinates $\mathbf{X}$, which are functions $\mathbf{X} = \mathbf{X}[\Delta\mathbf{X}_B(x_i)]$ of the shape parameter x_i. In fact, a change in x_i gives a displacement of the boundary $\Delta\mathbf{X}_B$ through the shape parameterization. In turn, the displacement implies a new repositioning of all the mesh coordinates $\mathbf{X}$. The latter can be achieved using a mesh deforming technique that compute the displacement $\Delta\mathbf{X}$ to be added to the original mesh points ($\mathbf{X}^{new} = \mathbf{X} + \Delta\mathbf{X}$). In this work a spring analogy is chosen, where the displacement of each point $\Delta\mathbf{X}_i$ is iteratively solved with Jacobi iterations:

$$\Delta\mathbf{X}_i{}^{k+1} = \frac{\sum_{i=1}^{N_i} k_{ij} \Delta\mathbf{X}_j{}^k}{\sum_{i=1}^{N_i} k_{ij}} \tag{9}$$

where N_i is the number of nearest neighbors of the node i and k_{ij} is the stiffness associated with the edge ij (its numerical value is chosen to be the inverse of the edge length). Therefore, the boundary displacement $\Delta\mathbf{X}_B$ is iteratively propagated in the whole domain.

The residual vector has a dependency $\mathbf{R} = \mathbf{R}[\mathbf{X}, \mathbf{N}(\mathbf{X}), \nabla\mathbf{U}(\mathbf{X}), \mathbf{\Phi}(\mathbf{X})]$ on the mesh points, where $\mathbf{N}$ are the grid metrics, integrated normal vectors in the grid and on the boundary, and $\nabla\mathbf{U}$ and $\mathbf{\Phi}$ are the least squares gradient and the limiter vector respectively. Applying the chain rule, the complete derivative read

$$\frac{\partial\mathbf{R}}{\partial x_i} = \left(\frac{\partial\mathbf{R}}{\partial\mathbf{X}} + \frac{\partial\mathbf{R}}{\partial\mathbf{N}}\frac{\partial\mathbf{N}}{\partial\mathbf{X}} + \frac{\partial\mathbf{R}}{\partial\nabla\mathbf{U}}\frac{\partial\nabla\mathbf{U}}{\partial\mathbf{X}} + \frac{\partial\mathbf{R}}{\partial\mathbf{\Phi}}\frac{\partial\mathbf{\Phi}}{\partial\mathbf{X}}\right)\frac{\partial\mathbf{X}}{\partial\Delta\mathbf{X}_B}\frac{\partial\Delta\mathbf{X}_B}{\partial x_i} \tag{10}$$

where, in practice, each of the derivative corresponds to a differentiated sub–routine. $\partial\Delta\mathbf{X}_B/\partial x_i$ is the differentiation of the shape parameterization routine with respect to the parameters, $\partial\mathbf{X}/\partial\Delta\mathbf{X}_B$ is the differentiation of the spring analogy routine with respect to the boundary displacement. $\partial\nabla\mathbf{U}/\partial\mathbf{X}$ and $\partial\mathbf{\Phi}/\partial\mathbf{X}$ are the reconstruction routines differentiated with respect to the mesh points. $\partial\mathbf{R}/\partial\mathbf{X}$, $\partial\mathbf{R}/\partial\mathbf{N}$, $\partial\mathbf{R}/\partial\nabla\mathbf{U}$ and $\partial\mathbf{R}/\partial\mathbf{\Phi}$ represent the residual routine which is differentiated with respect to mesh points, metrics, gradients and limiters.

A similar procedure has been implemented for the partial derivative of the functional $\partial J/\partial x_i$. In order to check the accuracy of the differentiated code, comparisons with finite difference has been made for all the differentiated routines. As in the case of the adjoint code, the differentiation of the code in forward mode is exact since no approximations have been made. It is interesting to note that the differentiation of the mesh displacement routine involves an iterative process. The routine has been differentiated properly by Tapenade, as noticed after the checking, without any particular problem.

5 NUMERICAL RESULTS

In order to perform shape optimization, the elements presented above are coupled together in a relatively simpler way. Once the initial geometry, the objective function and the constraints have been defined:

1. the flow solver module is used to compute the flow field $\mathbf{U}$;
2. the adjoint solver module is used to compute the adjoint variables $\mathbf{\Lambda}_L$ and $\mathbf{\Lambda}_D$ for the lift and the drag coefficient respectively;
3. a post–processing module receive as input the parameters, the flow and the adjoint variables. It compute the functionals (c_l and c_d) and the geometric quantities ($tc_{\max}$, r_u, r_l and θ) together with their gradient (∇c_l, ∇c_d, $\nabla tc_{\max}$, ∇r_u and ∇r_l) with respect to the shape parameters $\mathbf{x}$;
4. the optimization module, which implement the method of centers, compute a new improved design. Move limits are reduced as the optimum is approached. If some convergence criteria are met, the process is terminated;

5. the mesh is deformed according to the new values of the shape parameters and the process is repeated starting from step 1.

Each element is implemented as a Fortran 90 module at the exception of the Sequential Linear Programming module. The latter is a script implemented in the Matlab[18] language, which use Linear Programming libraries included in the Optimization Toolbox.

The framework has been employed in some shape optimization test cases. A mesh of approximatively 8000 nodes, as shown in Fig. (2.e), has been used for all the tests since different airfoils, starting from an initial airfoil, are obtained trough the application of the shape parameterization and the mesh deforming routines.

For each test case, the initial flow and adjoint field have been computed with a level of accuracy equal to 8 orders of magnitude reductions of the residual norm. During the optimization, both the flow and the adjoint, are converged to reach the same accuracy as the initial computation.

The initial design vector $\mathbf{x}$ at the beginning of the optimization is scaled in a way that $\mathbf{x}_0 = [1, 1, ..., 1]^T$. The move limits $\Delta\mathbf{x}$ assume the same value for each component of the design vector, they are set as $\Delta\mathbf{x} = b\,\mathbf{x}_0$. Therefore, in the following b will be referred as the move limit.

5.1 RAE2822

The RAE2822 airfoil is optimized at $M_\infty = 0.73$ and $\alpha = 2\,deg$ angle of attack. As can be seen on Fig. (2.c), a shock is present on the upper surface of the airfoil. The reference maximum thickness is equal to the initial maximum thickness of the airfoil ($tc_{\max R} = tc_{\max 0}$) whereas the reference values for the nose curvature and the trailing edge angle are set respectively to 70% and 90% of their initial values ($r_{uR} = 0.7\,r_{u0}$, $r_{lR} = 0.7\,r_{l0}$ and $\theta_R = 0.9\,\theta_0$). This choice of the reference values means that the thickness can't decrease, the trailing edge angle can only decrease 10% of its initial value and the two radii of curvature can only decrease 30% of their initial values. The lift is imposed to remain constant during the optimization, which means that for the equality lift constraint the reference value is the initial one ($c_{lR} = c_{l0}$). The objective function, which is the drag coefficient, is scaled by its initial value ($c_{dR} = c_{d0}$).

At the end of the optimization, as can be seen on Fig. (2.d), a shock–free airfoil is obtained. The constraints, although their trend is not shown, have been satisfied. The objective function, depicted in Fig. (2.a), has a non–smooth trend for a value of the move limit $b = 0.1$. Probably such value is too large and the design has the tendency to oscillate. In fact, with a move limit $b = 0.05$ the objective seems to behave smoothly, in spite of the reduced convergence rate.

When the objective function had become almost flat, the move limits have been reduced first by a factor 2 and then by a factor 10. In spite of the appreciable reduction of the objective in the first 10 iterations, the shock disappears completely only in the last 10 iterations where the objective appears to be flat. This inefficiency is due to the fact that

in proximity of the optimum the oscillations, although mitigated by the reduction factor for the move limit, are unlikely to be suppressed completely.

5.2 NACA0012 transonic

The NACA0012 is optimized at $M_\infty = 0.73$ and $\alpha = 2\,deg$ angle of attack. For such conditions the shape of the NACA0012 airfoil is clearly unsuitable. In fact, Fig. (3.c) shows a strong shock appearing on the upper side of the airfoil, almost at half of the chord. The reference values used for the geometric constraints are different from the previous case ($r_{uR} = 0.9\,r_{u0}$, $r_{lR} = 0.9\,r_{l0}$ and $\theta_R = 0.9\,\theta_0$) except for the thickness which is still not allowed to decrease ($tc_{\max R} = tc_{\max 0}$).

The move limits have been chosen to be $b = 0.3$ until the objective had become flat. After, they have been reduced as before. As can be seen on Fig. (3.a), the objective function smoothly reduced 90% with respect to the initial value. The constraints have been satisfied. The final shock–free airfoil is shown in Fig. (3.a). The flow conditions on the bottom side are almost sonic at the quarter of the chord.

5.3 NACA64A410 transonic

The NACA64A410 is optimized at $M_\infty = 0.75$ and $\alpha = 0\,deg$ angle of attack. A strong shock, see Fig. (4.c), appears at three quarters of chord on the upper side of the airfoil. The same reference values used for the NACA0012 have been used in this case.

The move limits, with the same values as in the NACA0012 case, are probably large for this case since the objective, see Fig. (4.a), does not seems to behave very smoothly. At the end of the optimization, as in the last cases, a shock–free airfoil has been obtained. However, as can be seen on Fig. (4.d), at this time the optimized shape appears to be rather complex if compared to the initial NACA64A410 shape.

5.4 NACA0012 supersonic

The NACA0012 has also been optimized at supersonic flow conditions, $M_\infty = 1.5$ and $\alpha = 2\,deg$ angle of attack. In the supersonic case the shape parameterization and the geometric constraints play a crucial role. It is well known that thin airfoil with sharp noses are likely to produce less wave drag. This is because if the nose is sharp the bow shock tends to be attached and oblique. On the contrary, a rounded nose produce a detached shock which is almost normal just ahead of the nose. The large wave drag which is noticed in this situation is due to the presence of the almost normal shock.

Therefore, an optimization in supersonic regime is expected to produce a sharp airfoil. The situation of a sharp nose cannot be handled properly with the shape parameterization used in this work. Moreover, real life airfoils do not have sharp noses but more likely noses with small radii of curvature. For this reason, a supersonic optimization is a good test case to see if the geometric constraints introduced in this work are effective.

In Fig. (5.c), a strong detached bow shock is visible on the original NACA0012 airfoil.

The NACA0012 airfoil has a 12% maximum thickness and a large radius of curvature at the nose. A value of the maximum thickness more suitable for supersonic flow could be 6% and therefore the reference thickness value is imposed to be half of the initial one ($tc_{\max R} = 0.6\, tc_{\max 0}$). Also, all the other reference geometric quantities has been imposed to be 10% of their initial values ($r_{uR} = 0.1\, r_{u0}$, $r_{lR} = 0.1\, r_{l0}$ and $\theta_R = 0.1\, \theta_0$). The latter means that, for instance, the nose radius curvature can decrease up to 90% of the initial value. Also in this case, the Lift coefficient must be kept constant ($c_{lR} = c_{l0}$) and the Drag coefficient,objective function, is scaled by the initial value ($c_{dR} = c_{d0}$). Move limits $b = 0.3$, as in the NACA0012 transonic case, have been used.

The objective function, see Fig. (5.a), smoothly reduced of 70%. Such large reduction in drag is due to the repositioning of the shock, which is almost attached to the nose at the end of the optimization. A magnification of the nose region, see Fig. (5.e), reveals that the shock is at less than 1% of the chord distance from the nose. The latter, appears to be rounded and smooth. The curvature of the nose is now 10% of the initial curvature since both the upper and lower curvature constraints are critical. The same is for the thickness constraint. The three constraints are depicted in Fig. (5.b), which shows how they approach criticality ($g_1, g_2, g_3 \rightarrow 0$). All the other constraints, although not shown, are satisfied. It is remarkable that also in this case, in spite of the challenging flow conditions and the large deformations involved, the whole framework proved to be effective and robust. Also, it is interesting to see that the final shape resembles that of lenticular airfoils, typically used in the supersonic regime.

6 CONCLUSIONS

Different shape optimization test cases have been successfully addressed by the framework presented in this work. The introduction of additional geometric constraints, nose curvature and trailing edge angle, has removed the ambiguity of setting the bounds on the shape parameters in order to avoid unfeasible shapes. In fact, the geometric constraints guarantee to have a smooth and feasible shape according to the prescribed reference values.

The optimization strategy, the method of centers, appear to be effective in obtaining the optimal shape. However, this method does not appear to be very efficient particularly in the final phase of the optimization. In fact, in the initial phase there is a large reduction in the objective in relatively few iterations. On the contrary, in the final phase the reduction is extremely small and perhaps more difficult to achieve. The difficulty is caused by the move limits which, on one side need to be reduced in order to avoid oscillations of the design and, on the other side need to be as large as possible in order to speed up the convergence. Clearly, the observed behavior is typical of a method that uses only first order information. Nevertheless, in spite of its simplicity, the algorithm seems to be a powerful tool for shape optimization purposes.

ACKNOWLEDGMENTS

This research is supported by the Dutch Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

REFERENCES

[1] G. Carpentieri, M.J.L. van Tooren and B. Koren, "Improving the Efficiency of Aerodynamic Shape Optimization on Unstructured Meshes." AIAA Paper 2006-298, Reno, NV, Jan. 2006.

[2] G. Carpentieri, M. J. L. van Tooren, M. Kelly and R. Cooper,"Airfoil Optimization Using an Analytical Shape Parameterization" CEIAT Paper 2005–0073, Belfast, UK, Aug. 2005.

[3] A. Jameson, "Aerodynamic Design via Control Theory." *Journal of Scientific Computing*, **3**, 233–260, 1988

[4] A. Jameson, "Optimum Transonic Wing Design Using Control Theory." Symposium Transsonicum IV, DLR Gottingen, Germany, Sep. 2002.

[5] G.N. Vanderplaats, *Numerical Optimization Techniques for Engineering Design*, 3^{th}. Edition, Vanderplaats Research & Development, 2001.

[6] J. A. Desideri and A. Dervieux, "Compressible Flow Solvers using Unstructured Grids." Lecture series 1988-05, Von Karman Institute for Fluid Dynamics 1988.

[7] T.J. Barth, "Aspects of unstructured grids and finite–volume solvers for the Euler and Navier–Stokes equations." Lecture series 1991-06, Von Karman Institute for Fluid Dynamics 1991.

[8] V. Selmin and L. Formaggia, "Unified Construction of Finite Element and Finite Volume Discretizations for Compressible Flows." *International Journal for Numerical Methods in Engineering*, **39**, 1–32, 1996.

[9] V. Venkatakrishnan and D. J. Mavriplis, "Implicit Solvers for Unstructured Meshes." *Journal of Computational Physics*, **105**, 83–91, 1993.

[10] E. J. Nielsen, "Aerodynamic Design Sensitivities on an Unstructured Mesh Using the Navier–Stokes Equations and a Discrete Adjoint Formulation." PhD Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1998.

[11] E. J. Nielsen, J. Lu, M. A. Park and D. L. Darmofal, "An Implicit, Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids." *Computers & Fluids*, **33**, 1131–1155, 2004.

[12] M. B. Giles, M. C. Duta, J.-D. Müller and N. A. Pierce, "Algorithm Developments for Discrete Adjoint Methods." *AIAA Journal*, **41**, 198–205, 2003.

[13] J.-D. Müller and P. Cusdin, "On the Performance of Discrete Adjoint CFD Codes using Automatic Differentiation." *Int. J. Numer. Meth Fluids*, **47**, 939–945, 2003.

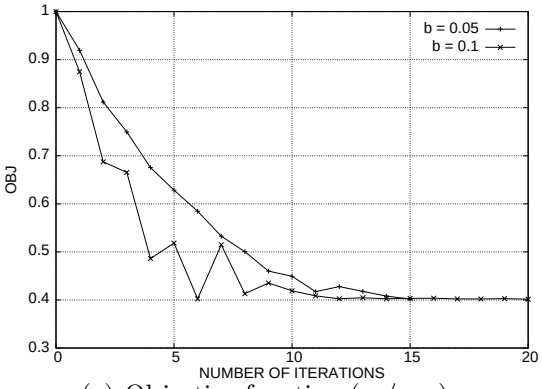
[14] B. Mohammadi, " A New Optimal Shape Design Procedure for Inviscid and Viscous Turbulent Flows." *International Journal for Numerical Methods in Fluids*, **25**, 183-203, 1997.

[15] D. J. Mavriplis, " A Discrete Adjoint–Based Approach for Optimization Problems on Three–Dimensional Unstructured Meshes ." AIAA Paper 2006-50, Reno, NV, Jan. 2005.

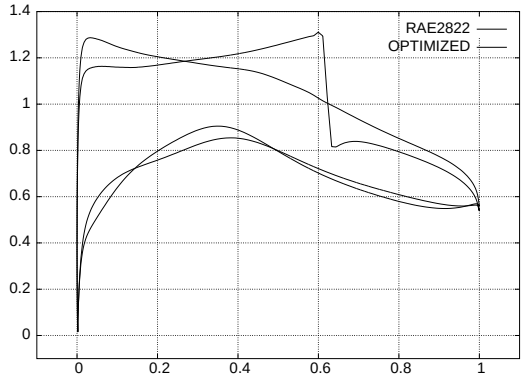
[16] A. Verhoff, D. Stooksberry and A. B. Cain, "An Efficient Approach to Optimal Aerodynamic Design Part 1: Analytic Geometry and Aerodynamic Sensitivities." AIAA Paper 93–0099, Reno, NV, Jan 1993.

[17] http://tapenade.inria.fr:8080/tapenade/index.jsp Software TAPENADE ©INRIA 2002, version 2.0

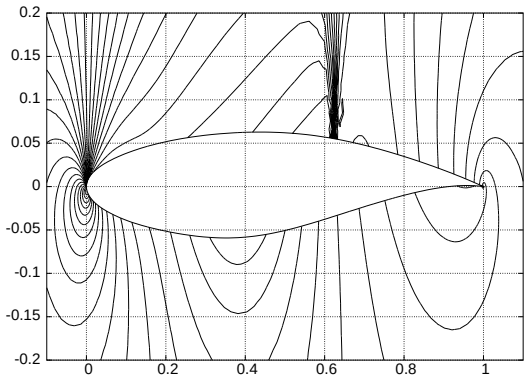
[18] Matlab 7, Optimization Toolbox



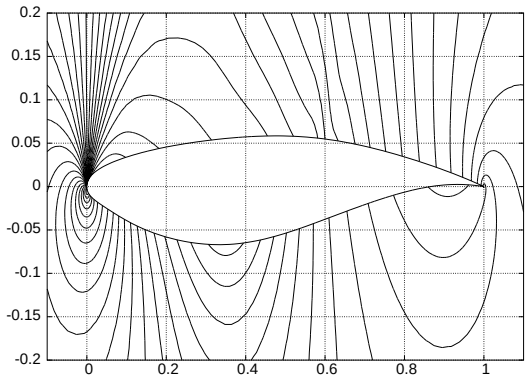
(a) Objective function (c_d/c_{dR}).



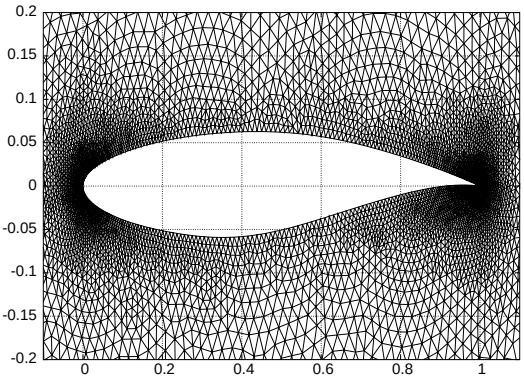
(b) Initial and final Mach number distribution.



(c) Pressure contours for the RAE2822.

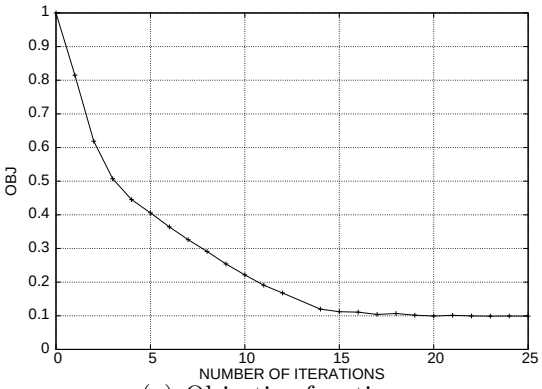


(d) Pressure contours of the optimized airfoil.

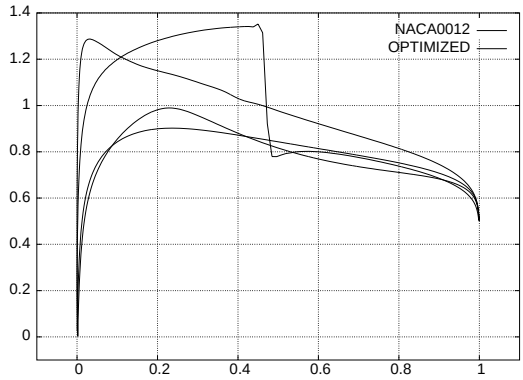


(e) 8K nodes unstructured mesh around the RAE2822.

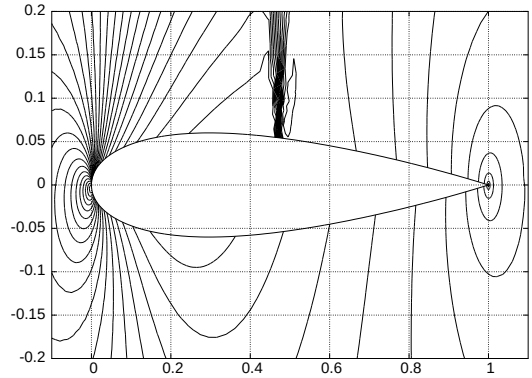
Figure 2: Optimization of the RAE2822 airfoil at $M_\infty = 0.73$ and $\alpha = 2\,deg$ angle of attack. At the end of the optimization: the equality lift constraint is $h = -5 \times 10^{-5}$, the thickness constraint is $g_1 = -0.011$, the upper radius of curvature and the lower radius of curvature constraints are respectively $g_2 = -0.357$ and $g_3 = -0.997$, the trailing edge constraint is $g_4 = -0.0641$.



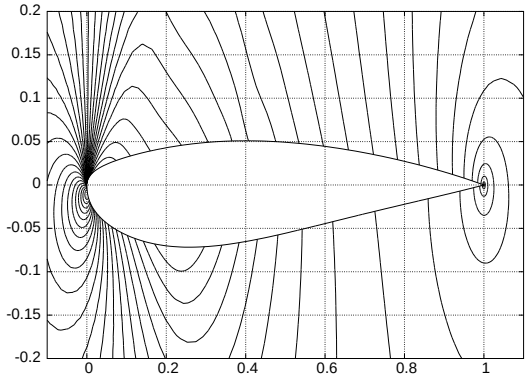
(a) Objective function.



(b) Initial and final Mach number distribution.

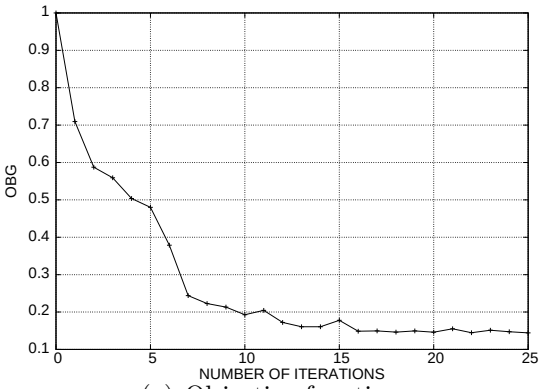


(c) Pressure contours of the NACA0012 airfoil.

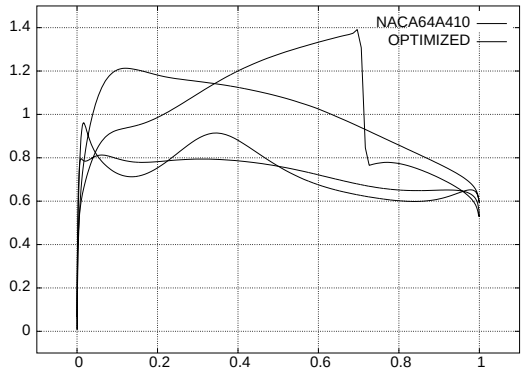


(d) Pressure contours of the optimized airfoil.

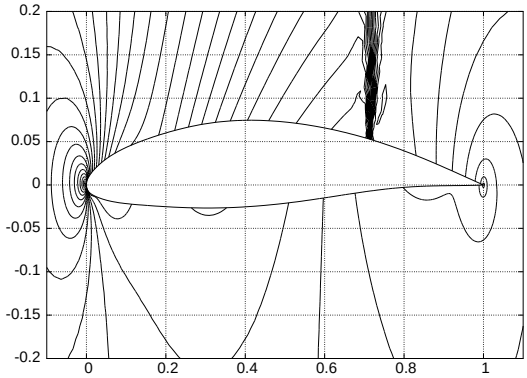
Figure 3: Optimization of the NACA0012 airfoil at initial condition $M_\infty = 0.75$ and $\alpha = 2\,deg$ angle of attack. The final constraints values are $h = -6.35 \times 10^{-6}$, $g_1 = -1.15 \times 10^{-3}$, $g_2 = -0.011$, $g_3 = -3.15$ and $g_4 = -0.44$.



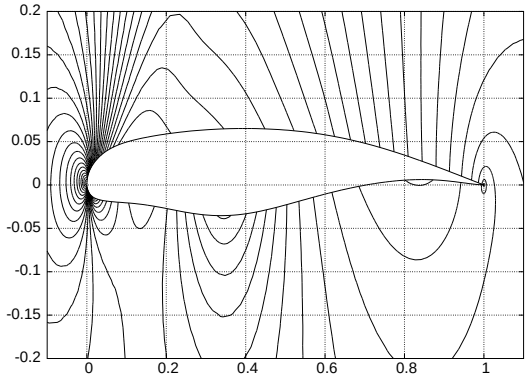
(a) Objective function.



(b) Initial and final Mach number distribution.

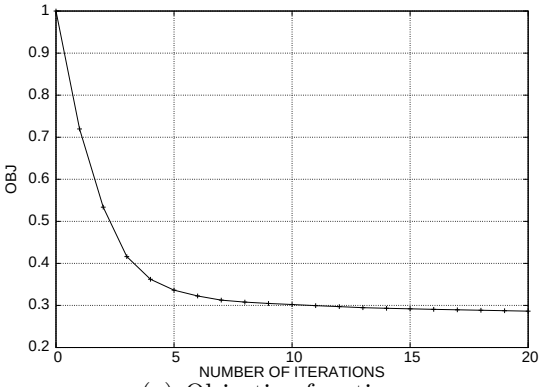


(c) Pressure contours of the NACA64A410.

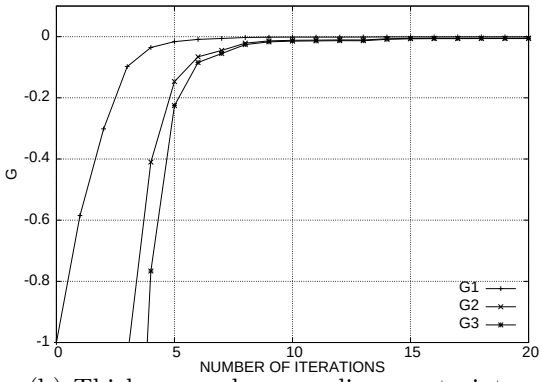


(d) Pressure contours of the optimized airfoil.

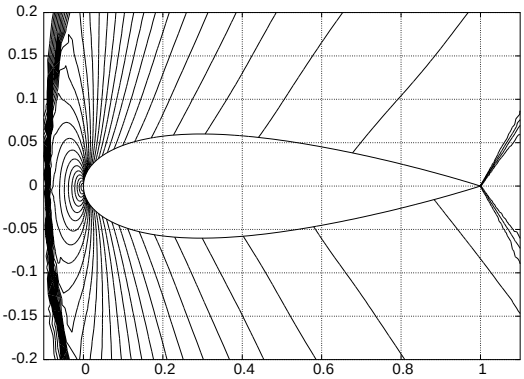
Figure 4: Optimization of the NACA64A410 airfoil at initial condition $M_\infty = 0.75$ and $\alpha = 0\,deg$ angle of attack. The final constraints values are $h = 6.5 \times 10^{-5}$, $g_1 = -2.3 \times 10^{-3}$, $g_2 = -4.415$, $g_3 = -0.1684$ and $g_4 = -0.0099$.



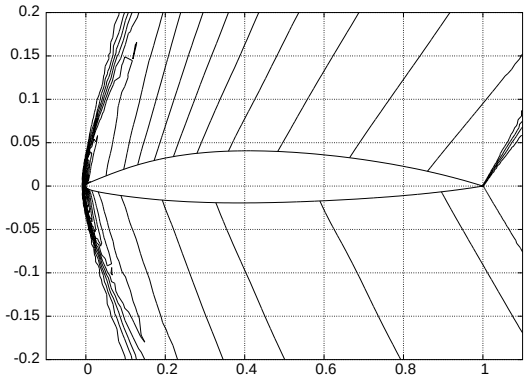
(a) Objective function.



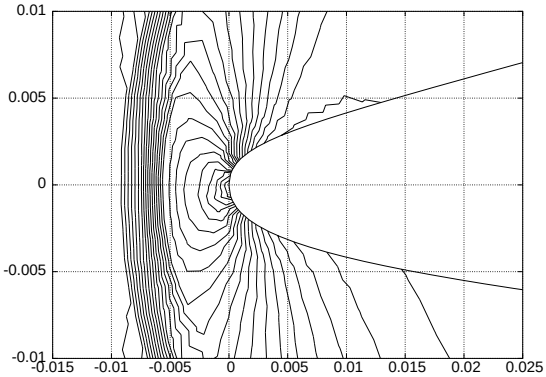
(b) Thickness and nose radius constraints.



(c) Pressure contours of the NACA0012.



(d) Pressure contours of the optimized airfoil.



(e) Magnification of the optimized airfoil nose.

Figure 5: Optimization of the NACA0012 airfoil at initial condition $M_\infty = 1.5$ and $\alpha = 2\,deg$ angle of attack. The final constraints values are $h = 8.8\times10^{-5}$, $g_1 = -6.5\times10^{-4}$, $g_2 = -5\times10^{-3}$, $g_3 = -6\times10^{-3}$ and $g_4 = -6.6$.